Chapter II. Controlling Cars on a Bridge

1 Introduction

The intent of this chapter is to introduce a complete example of a small system development. During this development, you will be made aware of the systematic approach we are using: it consists in developing a series of more and more accurate models of the system we want to construct. This technique is called *refinement*. The reason for building consecutive models is that a unique one would be far too complicated to reason about. Note that each model does not represent the programming of our system using a high level programming language, it rather formalizes what an *external observer* of this system could perceive of it.

Each model will be analyzed and proved, thus enabling us to establish that it is *correct* relative to a number of criteria. As a result, when the last model will be finished, we shall be able to say that this model is *correct by construction*. Moreover, this model will be so close to a final implementation that it will be very easy to transform it into a genuine program.

The correctness criteria alluded to above will be made completely clear and systematic by giving a number of *proof obligation rules* which will be applied on our models. After applying such rules, we shall have to prove formally a number of statements. To this end, we shall also give a reminder of the classical *rules of inference of the sequent calculus*. Such rules concern propositional logic, equality, and basic arithmetic. The idea here is to give the reader the possibility to manually prove the statements as given by the proof obligation rules. Clearly, such proofs could easily be discharged by a theorem prover, but we feel that it is important at this stage for the reader to exercise himself before using an automatic theorem prover. Notice that we do not claim that a theorem prover would perform these proofs the way it is proposed here: quite often, a tool does not work like a human being does.

This chapter is organized as follows. Section 2 contains the requirement document of the system we would like to develop: for this, we shall use the principles we have explained in the previous chapter. Section 3 explains our refinement strategy: it essentially assigns the various requirements to the various development steps. The four remaining sections are devoted to the development of the initial models and that of the three subsequent refinements.

2 Requirements Document

The system we are going to build is a piece of software, called the *controller*, connected to some equipment, called the *environment*. There are thus two kinds of requirements: those concerned with the functionalities of the controller labeled FUN, and those concerned with the environment labeled ENV.

Note that the model we are going to build is a *closed model* comprising the controller *as well as its environment*. The reason is that we want to define with great care the assumptions we are making concerning the environment. In other words, the controller we shall build eventually will be *correct* as long as these assumptions are fulfilled by the environment: outside these assumptions, the controller is not guaranteed to perform correctly. We shall come back to this in section 7.

Let us now turn our attention to the requirements of this system. The main function of this system 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 system is equipped with two traffic lights.

The system is equipped with two traffic lights with two colors: green and red	ENV-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	ENV-2

Drivers are supposed to obey the traffic lights 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	ENV-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	ENV-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 in the island respectively.

The sensors are used to detect the presence of a car entering or leaving the bridge: "on" means that a car is willing to enter the bridge or to leave it	ENV-5

The pieces of equipment which have been described are illustrated Fig. 1.The system has two main additional 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

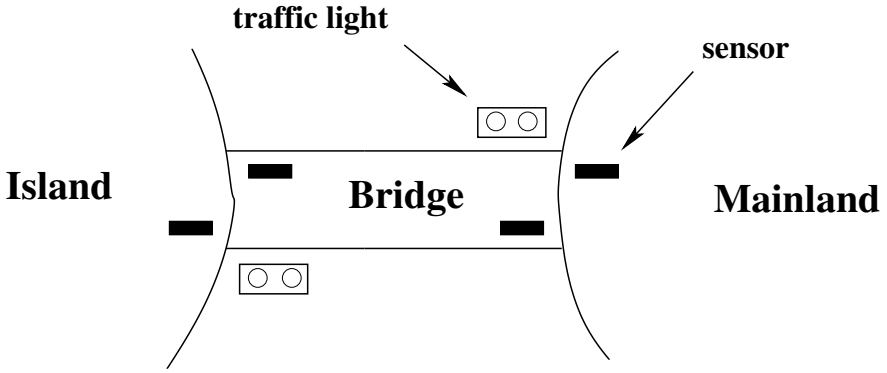


Fig. 1. The Bridge Control Equipment

3 Refinement Strategy

Before engaging in the development of such a system, it is profitable to clearly identify what our design strategy will be. This is done by listing the order in which we are going to take account of the various requirements we proposed in the requirement document of the previous section. Here is our strategy:

- We start with a very simple model allowing us to take account of requirement FUN-2 concerned with the maximum number of cars on the island and the bridge (section 4)
- Then the bridge is introduced into the picture, and we thus take account of requirement FUN-3 telling us that the bridge is one way or the other (section 5).
- In the next refinement, we introduce the traffic lights. This corresponds to requirements ENV-1, ENV-2, and ENV-3 (section 6).
- In the last refinement, we introduce the various sensors corresponding to requirements ENV-4 and ENV-5 (section 7). In this refinement, we shall also introduce eventually the *architecture* of our closed model made of the controller, the environment, and the communication channels between the two.

You may have noticed that we have not been speaking of requirement FUN-1 telling us what the main function of this system is. This is simply because it is fairly general. It is in fact taken care of at each development step.

4 Initial Model: Limiting the Number of Cars

4.1 Introduction

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, they 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.

This is a very frequent approach to be taken. We start to build a model which is *far more abstract* than the final system we want to construct. The idea is to take account initially of very few constraints only.

This is because we want to be able to reason about this system in a simple way, considering in turn each requirement.

As a useful analogy, we suppose to observe the situation from very high in the sky. Although we cannot see the bridge, we suppose however that we can "see" the cars on the island-bridge compound 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 Fig. 2.

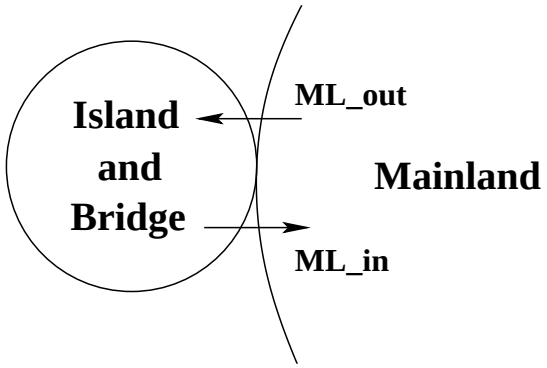


Fig. 2. The Mainland and the Island-Bridge

Our first task is to formalize the *state* of this simple version of our system (section 4.2). We shall then formalize the two *events* we can observe (section 4.3).

4.2 Formalizing the State

The state of our model is made up of two parts: the *static part* and the *dynamic part*. The static part contains the definition and axioms associated with some *constants*, whereas the dynamic part contains the *variables* which are modified as the system evolves. The static part is also called the *context* of our model.

The context of our first model is very simple. It contains a single constant d, which is a natural number denoting the maximum number of cars allowed to be on the island-bridge compound at the same time. The constant d has a simple axiom: it is a natural number. As can be seen below, we have given this axiom the name **axm0_1**:

constant: d

axm0_1: $d \in \mathbb{N}$

The dynamic part is made up of a single variable n, which denotes the actual number of cars in the island-bridge compound at a given moment. This is simply written as shown in the following boxes:

variable: n

inv0_1: $n \in \mathbb{N}$
inv0_2: $n \leq d$

Variable n is defined by means of two conditions which are called the *invariants*. They are named **inv0_1** and **inv0_2**. The reason to call them invariants is straightforward: despite the changes over time in the value of n, these conditions remain always true. Invariant **inv0_1** says that n is a natural number. And 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.

The labels **axm0_1**, **inv0_1**, and **inv0_2** we have used above are chosen in a systematic fashion. The prefix **axm** stands for the *axiom* of the constant d whereas the prefix **inv** stands for the *invariant* of the variable n. The **0**, as in **axm0_1** or **inv0_2**, stands for the fact that these conditions are introduced in the initial model. Subsequent models will be the first refinement, the second refinement, and so on. They will be numbered **1**, **2**, etc. Finally, the second number, as **2** in **inv0_2**, is a simple serial number. In the sequel, we shall use such a systematic labeling scheme for naming our state conditions. Sometimes, but rarely, we shall change the prefixes **axm** and **inv** for others. We found this naming scheme convenient, but, of course, any other naming scheme can be used provided it is systematic.

4.3 Formalizing the Events

At this stage, we can observe two transitions, which we shall call *events* in the sequel. They correspond to cars entering the island-bridge compound or leaving it. On Fig. 3 there is an illustration of the situation just before and just after an occurrence of the first event, ML_out (the name ML_out stands for "going out of mainland"). As can be seen, the number of cars in the compound is incremented as a result of this event:

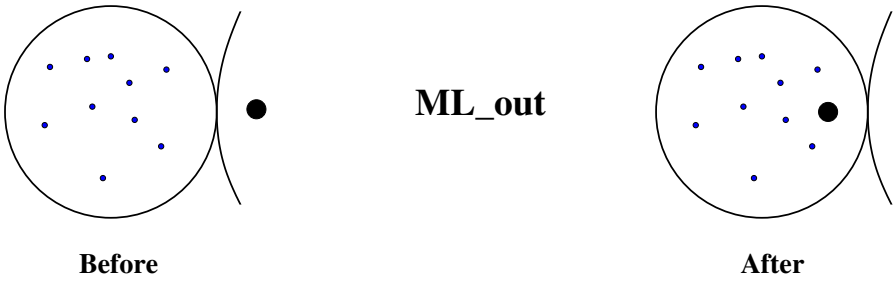


Fig. 3. Event ML_out

Likewise, Fig. 4 shows the situation just before and just after an occurrence of the second event, ML_in (the name ML_in stands for "getting in the mainland"). As can be seen, the number of cars in the compound is decremented as a result of this event:

In a *first approximation*, we define the events of the initial model in a simple way as follows:

ML_out	ML_in
$n := n + 1$	$n := n - 1$

An event has a *name*: here ML_out and ML_in. It contains an *action*: here $n := n + 1$ and $n := n - 1$. These statements can be read as follows: "n becomes equal to $n + 1$" and "n becomes equal to $n - 1$". Such statements are called *actions*. It is important to notice that in writing these actions *we are*

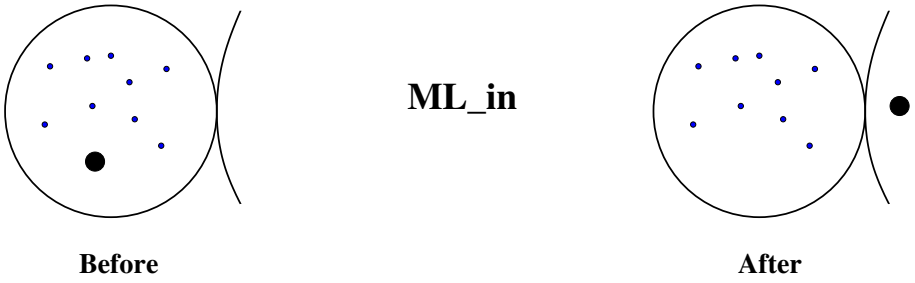


Fig. 4. Event ML_in

not programming. We are just formally representing what can be observed in discrete evolutions of our system. We are giving a formal representation to our observation.

You might have noticed that we have said above that the two events are proposed "in a first approximation". There are two reasons for writing this:

1. Our model observation is done in an *incremental fashion*. In other words, we are not defining immediately the final state and events of our system. Remember, *we are not programming*, we are defining models of the system we want to construct, and these models cannot be defined at once in full generality: this requires some gradual introduction of state components and transitions.

2. We propose here a state and various events, but we are not yet sure that these elements are consistent: this will have to be proved formally, and in doing this we might discover that what we have proposed is not correct.

4.4 Before-after Predicates

In this section, we present the concept of *before-after predicate*. This concept will be helpful to define the proof obligation rules in subsequent sections.

To each event defined as an action there corresponds a, so-called, before-after predicate. The before-after predicate associated with an action denotes the relationship that exists between the value of the concerned variable *just before* and *just after* the transition. This is indicated as shown below:

Events	ML_out $n := n + 1$	ML_in $n := n - 1$
Before-after Predicates	$n' = n + 1$	$n' = n - 1$

As can be seen, the before-after predicate is easily obtained from the action: the variable on the left-hand side of the action symbol ":=" is primed, the action symbol ":=" is changed to an equality symbol "=", and finally, the expression on the right-hand side of the action symbol is taken as is.

In a before-after predicate, a primed variable such as n' denotes, *by convention*, the value of the variable n *just after* the transition has occurred, whereas n represents its value *just before*. For instance, just after

an occurrence of the event ML_out, the value of the variable n is equal to the value it had just before plus one, that is $n' = n + 1$.

The before-after predicates we have presented here have got very simple shapes, where the primed value is equal to some expression depending on the non-primed value. Of course, more complicated shapes can be encountered but in this example, which is *deterministic*, we shall not encounter more complicated cases.

4.5 Proving Invariant Preservation

When writing the actions corresponding to the events ML_in and ML_out, we did not necessarily take into account invariants **inv0_1** and **inv0_2**, because we only concentrated in the way the variable n was modified. As a consequence, there is no reason a priori for these invariants to be preserved by these events. In fact, *it has to be proved in a rigorous fashion*. The purpose of this section is thus to define precisely what we have to prove in order to ensure that the invariants are indeed invariant!

The statement to be proved is generated in a systematic fashion by means of a rule, called INV, which is defined once and for all. Such a rule is called a *proof obligation* rule or a *verification condition* rule.

Generally speaking, suppose that our constants are collectively called c. And let $A(c)$ denote the axioms associated with these constants c. More precisely, $A(c)$ stands for the list: $A_1(c), A_2(c), \ldots$ of axioms associated with the constants. In our example model, $A(c)$ is reduced to a list consisting of the single element **axm0_1**. Likewise, let v denote the variables and let $I(c, v)$ denote the invariants of these variables. As for the axioms of the constants, $I(c, v)$ stands for the list $I_1(c, v), I_2(c, v), \ldots$ of invariants. In our example, $I(c, v)$ is reduced to a list consisting of the two elements **inv0_1** and **inv0_2**. Finally, let $v' = E(c, v)$ be the before-after predicate associated with an event. The invariant preservation statement which we have to prove for this event and for a specific invariant $I_i(c, v)$ taken from the set of invariants $I(c, v)$ is then the following:

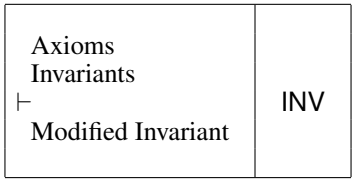
$A(c),\ I(c,v) \ \vdash \ I_i(c, E(c, v))$	INV

This statement, which is called a *sequent* (wait till next section for a precise definition of sequents), can be read as follows: "Hypotheses $A(c)$ and hypotheses $I(c, v)$ *entail* predicate $I_i(c, E(c, v))$". This is what we have to prove for each event and for each invariant $I_i(c, v)$. It is easy to understand: just before the transition, we can assume clearly that each axiom of the set $A(c)$ holds. We can also assume that each invariant of the set $I(c, v)$ holds too. As a consequence, we can assume $A(c)$ and $I(c, v)$. Now, just after the transition, where the value of v has been changed to $E(c, v)$, then the invariant statement $I_i(c, v)$ becomes $I_i(c, E(c, v))$, and it must hold too since it is claimed to be an *invariant*.

To simplify writing and ease reading, we shall write sequents vertically when there are several hypotheses. In the case of our rule INV, this yields the following:

$A(c)$ $I(c, v)$ $\vdash$ $I_i(c, E(c, v))$	INV

As this formulation of proof obligation rule INV might seem a bit difficult to remember, let us rewrite it in another way, which is less formal:



Proof obligation rule INV states *what we have to formally prove* in order to be certain that the various events maintain the invariants. But we have not yet defined what we mean by "formally prove": this will be explained in sections 4.8 to 4.11. We shall also explain how we can construct formal proofs in a systematic fashion. Finally, note that such sequents which correspond to applying rule INV is generated easily by a tool, which is called a *Proof Obligation Generator*.

4.6 Sequent

In the previous section, we introduced the concept of sequent in order to express our proof obligation rule. In this section, we give more information about such a construct[1]. As explained above, a statement of the following form is called a *sequent*:

The symbol ⊢ is named the *turnstile*. The part situated on the left hand part of the turnstile, here **H**, denotes a finite set of predicates called the *hypotheses* (or *assumptions*). Notice that the set **H** can be empty. The part situated on the right hand side of the turnstile, here **G**, denotes a predicate called the *goal* (or *conclusion*).

The intuitive meaning of such a statement is that the goal **G** is provable *under* the set of assumptions **H**. In other words, the turnstile can be read as the verb "entail", or "yield": the assumptions **H** yield the conclusion **G**.

In the sequel, we shall always generate such sequents (and try to prove them) in order to analyze our models. We shall also give rules to prove sequents in a formal way.

4.7 Applying the Invariant Preservation Rule

Coming back to our example, we are now in a position to clearly state what we have to prove: this is what we are going to do in this section. The proof obligation rule INV we have given in section 4.5 yields several sequents to prove. We have one application of this proof obligation rule per event and per invariant: in our case, we have two events, namely ML_out and ML_in and we have two invariants, namely **inv0_1** and **inv0_2**. This makes four sequents to prove: two sequents for each of the two events.

In order to remember easily what proof obligations we are speaking about, we are going to give *compound names* to them. Such a proof obligation name first mentions the event that is concerned, then the invariant, and we finally put the label INV in order to remember that it is an invariant preservation proof obligation (as there will be some other kinds of proof obligations[2]). In our case, the four proof obligations are thus named as follows:

[1] Sequents and the sequent calculus are reviewed in a more formal way in section 1 of Chapter 9.
[2] All proof obligations are reviewed in section 2 of chapter 5.

ML_out / **inv0_1** / INV

ML_in / **inv0_1** / INV

ML_out / **inv0_2** / INV

ML_in / **inv0_2** / INV

Let us now apply proof obligation rule INV to the two events and the two invariants. Here is what we have to prove concerning event ML_out and invariant **inv_01**:

Axiom **axm0_1**
Invariant **inv0_1**
Invariant **inv0_2**
⊢
Modified invariant **inv0_1**

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

ML_out / **inv0_1** / INV

Remember that event ML_out has the before-after predicate $n' = n + 1$. This is why predicate $n \in \mathbb{N}$ corresponding to invariant **inv0_1** in the assumptions has been replaced by predicate $n + 1 \in \mathbb{N}$ in the goal. Here is what we have to prove concerning event ML_out and invariant **inv0_2**:

Axiom **axm0_1**
Invariant **inv0_1**
Invariant **inv0_2**
⊢
Modified invariant **inv0_2**

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

ML_out / **inv0_2** / INV

Here is what we have to prove concerning event ML_in and invariant **inv0_1** (remember that the before-after predicate of event ML_in is $n' = n - 1$):

Axiom **axm0_1**
Invariant **inv0_1**
Invariant **inv0_2**
⊢
Modified invariant **inv0_1**

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

ML_in / **inv0_1** / INV

Here is what we have to prove concerning event ML_in and invariant **inv0_2**:

Axiom **axm0_1**
Invariant **inv0_1**
Invariant **inv0_2**
⊢
Modified invariant **inv0_2**

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

ML_in / **inv0_2** / INV

4.8 Proving the Proof Obligations

We know exactly which sequents we have to prove. Our next task is thus now to prove them: this is the purpose of the present section. The formal proofs of the previous sequents are done by applying some *transformations* on sequents, yielding one or several other sequents to prove, until we reach sequents that are considered being proved without any further justification. The transformation of one sequent into new ones corresponds to the idea that proofs of the latter are sufficient to prove the former. For example, our first sequent, namely:

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \vdash \\ n+1 \in \mathbb{N} \end{array} \qquad (1)$$

can be simplified by removing some *irrelevant hypotheses* (clearly hypotheses $d \in \mathbb{N}$ and $n \leq d$ are useless for proving our goal $n + 1 \in \mathbb{N}$), yielding the following simpler sequent:

$$n \in \mathbb{N} \vdash n+1 \in \mathbb{N} \qquad (2)$$

What we have admitted here as a step in the proof is the fact that a proof of sequent (2) is *sufficient* to prove sequent (1). In other words, if we succeed now proving sequent (2), then we have also a proof of sequent (1). The proof of the sequent (2) is reduced to nothing. In other words, it is accepted as being proved without further justification. It says that, under the assumption that n is a natural number, then $n + 1$ is also a natural number.

4.9 Rules of Inference

In the previous section, we applied informally some rules either to transform a sequent into another one or to accept a sequent without further justification. Such rules can be rigorously formalized, they are called *rules of inference*: this is what we do in the present section. The first rule of inference we have used can be stated as follows:

$$\frac{\mathbf{H1} \vdash \mathbf{G}}{\mathbf{H1}, \mathbf{H2} \vdash \mathbf{G}} \ \mathsf{MON}$$

Here is the structure of such a rule. On top of the horizontal line, we have a set of sequents (here just one). These sequents are called the *antecedents* of the rule. Below the horizontal line, we always have a single sequent called the *consequent* of the rule. On the right of the rule we have a name, here MON: this is the name of the inference rule. In this case, this name stands for *monotonicity* of hypotheses.

The rule is to be read as follows: in order to prove the consequent, it is sufficient to have proofs of each sequent in the antecedents. In the present case, it says that in order to have a proof of goal **G** under the

two sets of assumptions **H1** and **H2**, it is sufficient to have a proof of **G** under **H1** only. We have indeed obtained the effect we wanted: the removing of possibly irrelevant hypotheses **H2**.

Note that in applying this rule we do not require that the subset of assumptions **H2** is entirely situated after the subset **H1** as is strictly indicated in the rule. In fact, the subset **H2** is to be understood as *any subset* of the hypotheses. For example, in applying this rule to our proof obligation (1) in the previous section we removed the assumption $d \in \mathbb{N}$ situated before assumption $n \in \mathbb{N}$, and assumption $n \leq d$ which was situated after it.

The second rule of inference can be stated as follows:

$$\frac{}{\mathbf{H},\ \mathbf{n} \in \mathbb{N} \ \vdash\ \mathbf{n}+1 \in \mathbb{N}}\ \mathsf{P2}$$

Here we have a rule of inference with no antecedent. For this reason, it is called an *axiom*. This is the second Peano Axiom for natural numbers, hence the name P2. It says that in order to have a proof of the consequent it is not necessary to prove anything. Under the hypothesis that **n** is a natural number then **n**+1 is also a natural number. Notice that the presence of the additional hypotheses **H** is optional here. This is so because it is always possible to remove the additional hypotheses using rule MON. Thus the rule could have been stated more simply as follows (this is the convention we shall follow in the sequel):

$$\frac{}{\mathbf{n} \in \mathbb{N} \ \vdash\ \mathbf{n}+1 \in \mathbb{N}}\ \mathsf{P2}$$

A similar but more constraining rule will be used in the sequel. It concerns decrementing a natural number:

$$\frac{}{0 < \mathbf{n} \ \vdash\ \mathbf{n}-1 \in \mathbb{N}}\ \mathsf{P2}'$$

This rule says that $\mathbf{n} - 1$ is a natural number under the assumption that **n** is *positive*. We shall also use two other rules of inference called INC and DEC

$$\frac{}{\mathbf{n} < \mathbf{m} \ \vdash\ \mathbf{n}+1 \leq \mathbf{m}}\ \mathsf{INC}$$

Rule of inference INC says that $\mathbf{n} + 1$ is smaller than or equal to **m** under the assumption that **n** is strictly smaller than **m**.

$$\frac{}{\mathbf{n} \leq \mathbf{m} \ \vdash\ \mathbf{n}-1 < \mathbf{m}}\ \mathsf{DEC}$$

Rule of inference DEC says that $\mathbf{n} - 1$ is smaller than **m** under the assumption that **n** is already smaller than or equal to **m**. We shall clearly need more rules of inference dealing with elementary logic and also with natural numbers. But for the moment, we only need those we have just presented in this section.

Notice that it is well known that inference rules P2', INC and DEC are *derived* inference rules: it simply means that such inference rules can be deduced from more basic inference rules such as P2 above and others. But in this presentation, we are not so much interested in this: we want just to construct a *library* of useful inference rules

4.10 Meta-variables

You might have noticed that the various identifiers, namely **H1**,**H2**, **G**, **n**, **m** we have used in the rules of inferences proposed in the previous section, were emphasized: we have not used the standard mathematical font for them, namely $H1, H2, G, n, m$. This is because such variables are not part of the mathematical language we are using. They are called *meta-variables*.

More precisely, each proposed rule of inference stands for a schema of rules corresponding to all the possible *matches* we could ever perform. For example rule P2:

$$\frac{}{\mathbf{n} \in \mathbb{N} \vdash \mathbf{n}+1 \in \mathbb{N}}\ \text{P2}$$

describes the second Peano axiom in very general terms: it can be applied to the sequent

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

by *matching* meta-variable **n** to the mathematical language expression $a+b$.

4.11 Proofs

Equipped with a number of rules of inference, we are now ready to perform some elementary formal proofs: this is the purpose of this section. A proof is just a sequence of sequents connected by the name of the rule of inference which allows one to go from one sequent to the next in the sequence. The sequence of sequents ends with the name of a rule of inference with no antecedent. We shall see in section 4.24 that proofs have a more general shape but that one is sufficient for the moment.

For example, the proof of our first proof obligation ML_out / **inv0_1** / INV is the following. It corresponds exactly to what has been said informally in section 4.8, namely removing some useless hypotheses and then accepting the second sequent without any further proof:

$$\frac{\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \vdash \\ n+1 \in \mathbb{N} \end{array}}{\text{MON}}\ \frac{\begin{array}{l} n \in \mathbb{N} \\ \vdash \\ n+1 \in \mathbb{N} \end{array}}{\text{P2}}$$

The next proof, corresponding to proof obligation ML_out / **inv0_2** / INV, fails because we cannot apply rule INC on the final sequent as we do not have an assumption telling us that $n < d$ holds as required by rule INC: we only have the weaker assumption $n \leq d$. For this reason, we have put a "**?**" at the end of the sequence of sequents:

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \vdash \\ n+1 \leq d \end{array} \quad \text{MON} \quad \begin{array}{l} n \leq d \\ \vdash \\ n+1 \leq d \end{array} \quad ?$$

Likewise, the proof of ML_in / **inv0_1** / INV fails. Here we cannot apply inference rule P2' on the last sequent because we do not have the required assumption $0 < n$, only the weaker one $n \in \mathbb{N}$:

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \vdash \\ n-1 \in \mathbb{N} \end{array} \quad \text{MON} \quad \begin{array}{l} n \in \mathbb{N} \\ \vdash \\ n-1 \in \mathbb{N} \end{array} \quad ?$$

The last proof, that of ML_in / **inv0_2** / INV, succeeds:

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \vdash \\ n-1 \leq d \end{array} \quad \text{MON} \quad \begin{array}{l} n \leq d \\ \vdash \\ n-1 < d \ \lor \ n-1 = d \end{array} \quad \text{OR\_R1} \quad \begin{array}{l} n \leq d \\ \vdash \\ n-1 < d \end{array} \quad \text{DEC}$$

Notice that in the second step we felt free to replace $n - 1 \leq d$, that is $n - 1$ is smaller than *or* equal to d, by the equivalent formal statement $n - 1 < d \ \lor \ n - 1 = d$, where $\lor$ is the disjunctive operator "or". Then, we apply inference rule OR_R1 (see next section) in order to obtain the goal $n - 1 < d$ which is easily proved using rule DEC.

4.12 More Rules of Inference

At the end of previous section, we apply our first logical rule of inference named OR_R1[3]. In the sequel, there will be more logical rules of inference, but this one is the first one we need. We present it together with the companion rule OR_R2:

$$\frac{\mathbf{H} \ \vdash \ \mathbf{P}}{\mathbf{H} \ \vdash \ \mathbf{P} \lor \mathbf{Q}} \quad \text{OR\_R1} \qquad \frac{\mathbf{H} \ \vdash \ \mathbf{Q}}{\mathbf{H} \ \vdash \ \mathbf{P} \lor \mathbf{Q}} \quad \text{OR\_R2}$$

[3] All rules of inference are reviewed in Chapter 9.

Both rules state obvious facts about proving a disjunctive goal **P** $\vee$ **Q** involving two predicates **P** and **Q**. For proving their disjunction, it is sufficient to prove one of them: **P** in the case of rule OR_R1 and **Q** in the case of rule OR_R2.

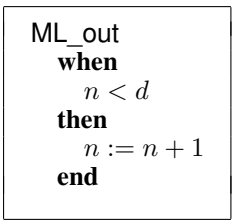
Notice that in the _R1 or _R2 suffixes of these rule names, the R stands for "right": it means that this rule transforms a goal, that is a statement situated on the *right* hand side of the turnstile. Other logical inference rules presented below will be named using the suffix L when the transformed predicate is an hypothesis, that is a predicate situated on the *left* hand side of the turnstile.

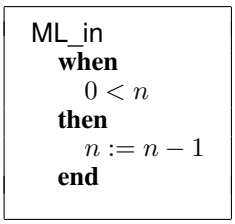
4.13 Improving the two Events: Introducing Guards

Coming back to our example, we have now to do some modifications in our model due to the fact that some proofs have failed. We figure out that proving has the same effect as debugging. In other words, *a failed proof reveals a bug*.

In order to correct the deficiencies we have discovered while carrying out the proof, we have to add *guards* to our events. Together, these guards denote the *necessary conditions* for an event to be enabled. More precisely, when an event is enabled, it means that the transition corresponding to the event can take place. On the contrary, when an event is not enabled (that is when at least one of its guards does not hold) it means that the corresponding transition cannot occur.

For event ML_out to be enabled, we shall require that n be strictly smaller than d, that is $n < d$. And for event ML_in to be enabled, we shall require that n be strictly positive, that is $0 < n$. Notice that such guarding conditions are exactly the conditions that were missing in the sequents we had to prove in the previous section: we have been guided by the failure of the proofs. Adding guards to events is done as follows:

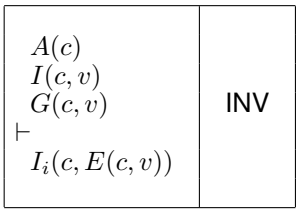


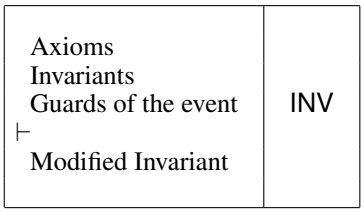


As can be seen, we have a simple piece of syntax here: the guard is situated between the keywords **when** and **then**, whereas the action is situated between the keywords **then** and **end**. Note that we may have several guards in an event, although it is not the case here.

4.14 Improving the Invariant Preservation Rule

When dealing with a guarded event with the set of guards denoted by $G(c, v)$ and before-after predicate of the form $v' = E(c, v)$ (where c denotes the constants and v the variables as introduced in section 4.5), our previous proof obligation rule INV has to be modified by adding the set of guards $G(c, v)$ to the hypotheses of the sequent. This yields the following more general proof obligation for events that have guards:





4.15 Reproving Invariant Preservation

The statements to prove by applying the amended proof obligation rule INV are modified accordingly and are now easily provable. Here is the sequent to prove for ensuring that event ML_out preserves invariant **inv0_2**:

Axiom **axm0_1**
Invariant **inv0_1**
Invariant **inv0_2**
Guard of ML_out
⊢
Modified invariant **inv0_2**

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

ML_out / **inv0_2** / INV

Here is what we have to prove concerning event ML_in and invariant **inv0_1**

Axiom **axm0_1**
Invariant **inv0_1**
Invariant **inv0_2**
Guard of ML_in
⊢
Modified invariant **inv0_1**

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

ML_in / **inv0_1** / INV

The two missing proofs can now be performed easily:

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

MON

$$n < d$$
$$\vdash$$
$$n + 1 \leq d$$

INC

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

MON

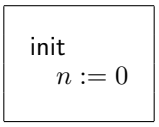
$$0 < n$$
$$\vdash$$
$$n - 1 \in \mathbb{N}$$

P2'

Notice that the proofs we have already done for proof obligations ML_out / **inv0_1** / INV and ML_in / **inv0_2** / INV in section 4.11 need not to be redone. This is because we are just adding a new hypothesis to these proof obligations: remember inference rule MON (section 4.9) which says that for proving a goal under certain hypotheses **H**, it is sufficient to perform the proof of the same goal under *less* hypotheses. So, conversely adding an hypothesis to a proof which is already done does not invalidate that proof.

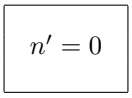
4.16 Initialization

In the previous sections, we defined the two events ML_in and ML_out and the invariants of the model, namely **inv0_1** and **inv0_2**. We also proved that these invariants were preserved by the transition defined by the events. But we have not defined what happens at the beginning. For this, it is necessary to define a special initial event that is systematically named init. In our case, this event is the following:



As can be seen, the initializing event corresponds to the observation that initially there are no cars in the compound. Notice that this event has no guard: this must always be the case with the initializing event init. In other words, initialization must always be possible!

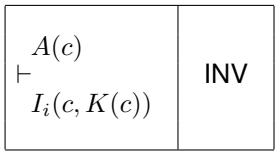
Also note that the expression situated on the right hand sign of $:=$ in the action of the event init cannot refer to any variable of the model. This is because we are *initializing*. In our case, the variable n cannot be assigned an expression depending on n. The corresponding before-after predicate of init is thus always rather an "after predicate" only. In our case, it is the following (as can be seen, this predicate does not mention n, only n'):

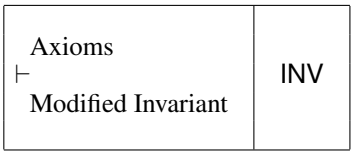


4.17 Invariant Establishment Rule for the Initializing Event init

Event init cannot preserve the invariants because before init the system state "does not exist". Event init must just *establish the invariant* for the first time. In this way, other events, which by definition are only observable after initialization has taken place, can be enabled in a situation where the invariants hold.

We have thus to define a proof obligation rule for this invariant establishment. It is almost identical to the proof obligation rule INV we have already presented in section 4.5. The difference is that in the proof obligation rule presented now the invariants are not mentioned in the hypotheses of the sequent. More precisely, given a system with constants c, with a set of axioms $A(c)$, with variables v and an invariant $I_i(c, v)$, and an initializing event with after predicate $v' = K(c)$, then the proof obligation rule INV for invariant establishment is the following:





4.18 Applying the Invariant Establishment Rule

The application of the previous rule to our initializing event init yields the following to prove:

Axiom **axm0_1**
$\vdash$
Modified Invariant **inv0_1**

$$d \in \mathbb{N} \vdash 0 \in \mathbb{N}$$

inv0_1 / INV

Axiom **axm0_1**
$\vdash$
Modified Invariant **inv0_2**

$$d \in \mathbb{N} \vdash 0 \leq d$$

inv0_2 / INV

4.19 Proving the Initialization Proof Obligations: More Inference Rules

The proof obligations of the previous section cannot be formally proved without other inference rules. The first one is named P1: it is the first Peano axiom, which says that 0 is a natural number. Notice that the sequent that defines the consequent of this inference rule has no assumption:

$$\frac{}{\vdash\ 0 \in \mathbb{N}}\ \text{P1}$$

The second rule of inference we need is a consequence of the third Peano axiom which says that 0 is not the successor of any natural number. We can prove that this can be re-phrased as follows: 0 is the smallest natural number. In other words, under the assumption that **n** is a natural number then 0 is smaller than or equal to **n**. For convenience, we shall name it P3:

$$\frac{}{\mathbf{n} \in \mathbb{N} \vdash 0 \leq \mathbf{n}}\ \text{P3}$$

We leave it as an exercise to the reader to now prove the two previous initialization proof obligations.

4.20 Deadlock Freedom

Since our two main events ML_in and ML_out are now guarded, it means that our model might *deadlock* when both guards are false: none of the events would be enabled, the system would be blocked. Sometimes this is what we want, but certainly not here, where this is to be avoided. As a matter of fact, we discover that this non-blocking property was *missing* in our requirement document of section 2. So we edit this document by adding this new requirement:

Once started, the system should work for ever	FUN-4

4.21 Deadlock Freedom Rule

Given a model with constants c, set of axioms $A(c)$, variables v and set of invariants $I(c, v)$, we have thus to prove a proof obligation rule called DLF (for deadlock freedom) stating that one of the guards $G_1(c, v)$, $\ldots$, $G_m(c, v)$ of the various events is always true. In other words, in our case, cars can always either enter the compound or leave it. This is to be proved under the set of axioms $A(c)$ of the constants c and under the set of invariants $I(c, v)$. The proof obligation rule can be stated as follows in general terms:

$A(c)$ $I(c, v)$ $\vdash$ $G_1(c, v) \ \vee \ \ldots \ \vee \ G_m(c, v)$	DLF

Axioms Invariants $\vdash$ Disjunction of the guards	DLF

Note that the application of this rule is *not mandatory*: not all systems need to be deadlock free.

4.22 Applying the Deadlock Freedom Proof Obligation Rule

Here is what we have to prove according to rule DLF:

Axiom **axm0_1** Invariant **inv0_1** Invariant **inv0_2** $\vdash$ Disjunction of the guards	$d \in \mathbb{N}$ $n \in \mathbb{N}$ $n \leq d$ $\vdash$ $n < d \ \vee \ 0 < n$	DLF

4.23 More Inference Rules

The previous deadlock freedom proof obligation cannot be proved without more rules of inference. The first one is a logical rule, which corresponds to the classical technique of a *proof by cases*. It is named OR_L as it has to do with the "or" symbol $\vee$ situated in the "Left" hand assumption part of a sequent. Notice that the antecedent of this rule has two sequents. More precisely, in order to prove a goal under a disjunctive assumption $\mathbf{P} \vee \mathbf{Q}$, it is sufficient to *prove independently* the same goal under assumption **P** and also under assumption **Q**:

$$\frac{\mathbf{H}, \mathbf{P} \ \vdash \ \mathbf{R} \qquad \mathbf{H}, \mathbf{Q} \ \vdash \ \mathbf{R}}{\mathbf{H}, \ \mathbf{P} \vee \mathbf{Q} \ \vdash \ \mathbf{R}} \ \text{OR\_L}$$

For the sake of completeness, we provide again the two logical inference rule we have already presented in section 4.12:

$$\frac{\mathbf{H} \ \vdash \ \mathbf{P}}{\mathbf{H} \ \vdash \ \mathbf{P} \vee \mathbf{Q}} \ \text{OR\_R1} \qquad \frac{\mathbf{H} \ \vdash \ \mathbf{Q}}{\mathbf{H} \ \vdash \ \mathbf{P} \vee \mathbf{Q}} \ \text{OR\_R2}$$

Our last logical rules have to do with the essence of a sequent.The first rule, HYP says that when the goal of a sequent is present as an assumption of that sequent then the sequent is proved. The second rule, FALSE_L, says that a sequent with a false assumption is proved. We denote false with the symbol $\bot$.

$$\frac{}{\mathbf{P} \vdash \mathbf{P}}\ \text{HYP} \qquad \frac{}{\bot \vdash \mathbf{P}}\ \text{FALSE\_L}$$

Our next two rules of inference are dealing with equality. They explain how one can exploit an assumption which is an equality.

$$\frac{\mathbf{H(F)},\ \mathbf{E} = \mathbf{F} \vdash \mathbf{P(F)}}{\mathbf{H(E)},\ \mathbf{E} = \mathbf{F} \vdash \mathbf{P(E)}}\ \text{EQ\_LR} \qquad \frac{\mathbf{H(E)},\ \mathbf{E} = \mathbf{F} \vdash \mathbf{P(E)}}{\mathbf{H(F)},\ \mathbf{E} = \mathbf{F} \vdash \mathbf{P(F)}}\ \text{EQ\_RL}$$

In rule EQ_LR, we have a sequent with a goal **P(E)**, which is a predicate *depending* on the expression **E**. We also have a set of hypotheses **H(E)** depending of **E**. Finally, we have the equality hypothesis **E = F**. The rule says that we can then replace this sequent by another one where all occurrences of **E** in **P(E)** and in **H(E)** have been replaced by **F**. The label LR is to remember that we apply the equality from "Left" to "Right". Rule EQ_RL exploits the same equality by applying it now from right to left, that is replacing **F** by **E** in **P(F)** and **H(F)** yielding **P(E)** and **H(E)**. Note that we have not formally explained exactly what we mean when we say that a predicate "depends" on an expression **E**. This will be made more precise later.

Our last rule, dealing with equality, says that any expression **E** is equal to itself. This rule in not used immediately in the next proof but it is quite natural to introduce it now:

$$\frac{}{\vdash\ \mathbf{E} = \mathbf{E}}\ \text{EQL}$$

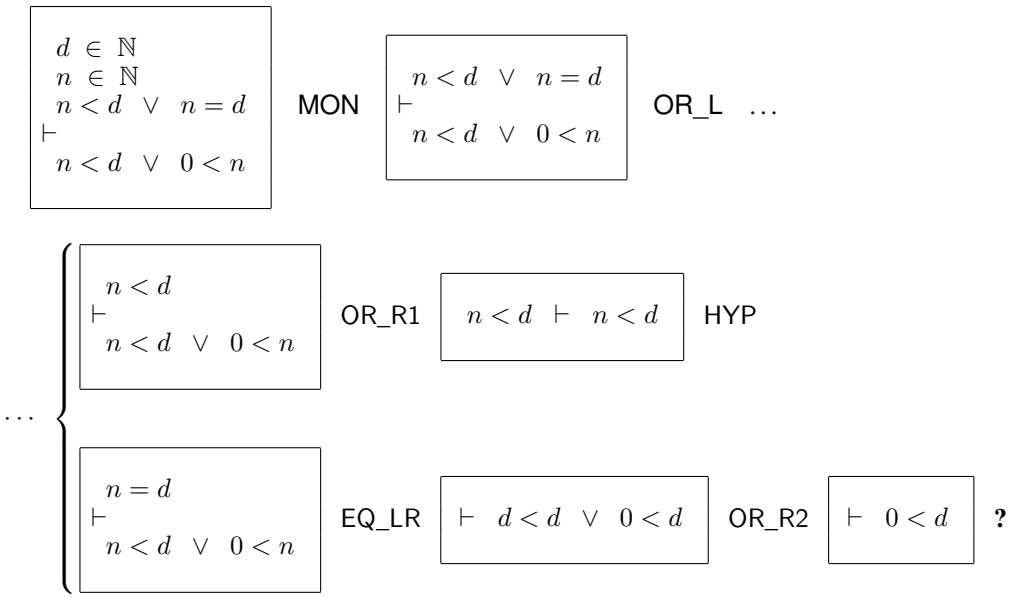
4.24 Proving the Deadlock Freedom Proof Obligation

Coming back to our example, we are going to give a tentative proof of our deadlock freedom proof obligation DLF, which we repeat below:

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \vdash \\ n < d \ \lor \ 0 < n \end{array}\qquad \text{DLF}$$

Here, we are going to apply inference rule OR_L because the assumption $n \leq d$ is in fact equivalent to the disjunctive predicate $n < d \ \lor \ n = d$. As can be seen, the usage of inference rule OR_L induces a

tree shape to the proof. This is because we have two antecedents in this rule. This tree shape is the normal shape of a proof:



We now discover that the last sequent cannot be proved. We have thus to add the following axiom, named **axm0_2**, which was obviously forgotten:

axm0_2: $0 < d$

We notice that this additional axiom allows us to have a more precise requirement FUN-2:

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

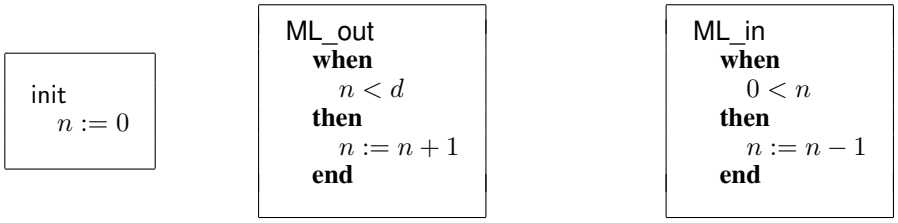
Adding this axiom to avoid deadlock is very intuitive because when d is equal to 0 then the system is deadlocked right from the beginning since no car can ever enter the compound. Again, note that adding this new axiom for the constant d does not invalidate the proofs we have already made so far: this is a consequence of the monotonicity rule MON introduced in section 4.9.

4.25 Conclusion and Summary of the Initial Model

As we have seen, the proofs (or rather the failed proof attempts) allowed us to discover that our events were too naive (we had to add guards in section 4.13) and also that one axiom was missing for constant d (previous section). This is quite frequently the case: proofs help discovering inconsistencies in a model. In fact, this is the heart of the modeling method! Here is the final version of the state for our initial model:

constant: d	**axm0_1:** $d \in \mathbb{N}$ **axm0_2:** $d > 0$	**variable:** n	**inv0_1:** $n \in \mathbb{N}$ **inv0_2:** $n \leq d$

And here are the final version of the events of our initial model:



5 First Refinement: Introducing the One Way Bridge

5.1 Introduction

We are now going to proceed with a *refinement* of our initial model. A refinement is a more precise model than the initial one. It is more precise but it should not contradict the initial model. Therefore we shall certainly have to *prove* that the refinement is consistent with the initial model. This will be made clear in this section.

In this first refinement, we introduce the bridge. This means that we are able to observe more accurately our system. 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 Fig. 5.

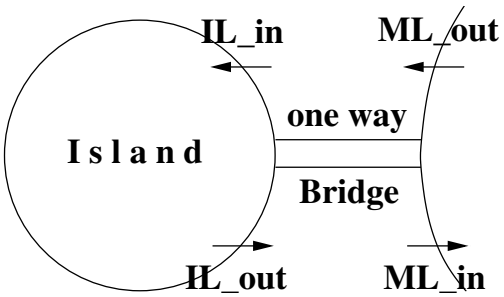


Fig. 5. Island and Bridge

5.2 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 variables*. This is because now we can see cars on the bridge and in 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 in the island, and variable c denotes the number of cars on the bridge and going to the mainland. This is illustrated in Fig. 6.

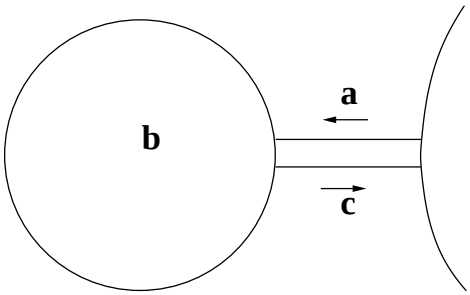


Fig. 6. The Concrete State

The state of the initial model is called the *abstract state* and the state of the refined model is called the *concrete state*. Likewise, variable n of the abstract state is called an *abstract variable* whereas variables a, b, and c of the concrete state are called *concrete variables*.

The concrete state is represented by a number of invariants, which we call the *concrete invariants*. First, variables a, b, and c are all natural numbers. This is stated in invariants **inv1_1**, **inv1_2**, and **inv1_3** below.

variables: a, b, c

inv1_1: $a \in \mathbb{N}$

inv1_2: $b \in \mathbb{N}$

inv1_3: $c \in \mathbb{N}$

inv1_4: $a + b + c = n$

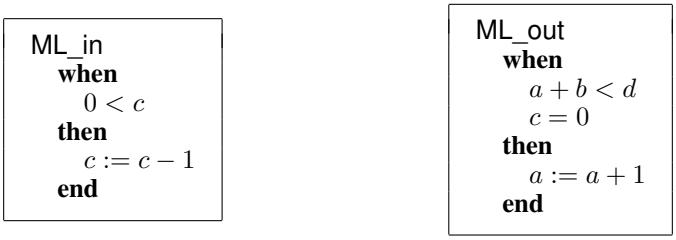
inv1_5: $a = 0 \ \lor \ c = 0$

Then we express that the sum of these variables is equal to the previous abstract variable n which has disappeared. This is expressed in invariant **inv1_4**. This invariant relates the concrete state represented by the three variables a, b, and c, to the abstract state represented by the variable n. And finally, we state that the bridge is one-way: this is our basic requirement FUN-3. This is expressed by saying that a or c are equal to zero. Clearly they cannot be both positive since the bridge is one-way, but they can be both equal to zero if the bridge is empty. This one-way property is expressed in invariant **inv1_5**.

Notice that among the concrete invariants some are dealing with the concrete variables only, these are **inv1_1**, **inv1_2**, **inv1_3**, and **inv1_5**, and one is dealing with concrete and abstract variables, this is **inv1_4**.

5.3 Refining the Abstract Events

The two abstract events ML_out and ML_in now have to be *refined* as they are not dealing with the abstract variable n anymore but with the concrete variables a, b, and c. Here is the proposed *concrete versions* (sometimes also called refined versions) of events ML_in and ML_out:



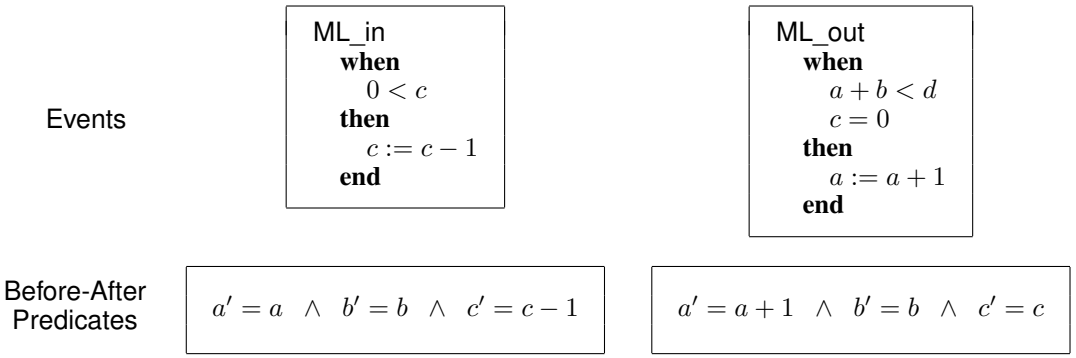
Notice that event ML_out has two guards, namely $a + b < d$ and $c = 0$. Also notice that although our refined model has now three variables a, b, and c, only one of them is mentioned in each event: c in event ML_in and a in event ML_out. In fact, the other two are, in each case, implicitly mentioned as being *left unchanged*.

It is easy to understand what these events are doing. In event ML_in, the action decrements variable c as there will be one car less on the bridge, and this can be done if there are some cars on the bridge going to the mainland, that is when $0 < c$ holds (notice that we are sure that there are no cars on the bridge going to the island as a must be equal to 0 since c is positive).

In event ML_out, the action increments variable a as there will be one more car on the bridge. But this is possible only if c is equal to 0, because of the one-way constraint of the bridge. Moreover, this will also be possible only if the new car entering the bridge does not break the constraint that there are a maximum number of d cars in the compound, that is when $a + b + c < d$ holds, which reduces to $a + b < d$ since c is equal to 0.

5.4 Revisiting the Before-after Predicates

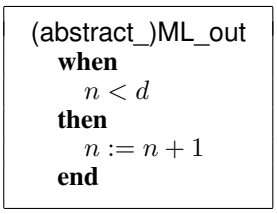
The observation concerning the actions of the concrete events in the previous section makes us be more precise now concerning the construction of the before-after predicate. The before-after predicate of the action corresponding to an event situated in a model containing *several variables* must mention explicitly that the missing variables in the action are not modified: this is done by stating that their primed after-values are equal to their non-primed before-values. In the case of our example, the following before-after predicates are associated with the corresponding event actions as follows:

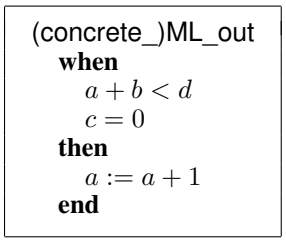


As can be seen, the before after-predicate of the action $c := c - 1$ contains $c' = c - 1$ as expected but also $a' = a$ and $b' = b$. We have similar equalities corresponding to the action $a := a + 1$.

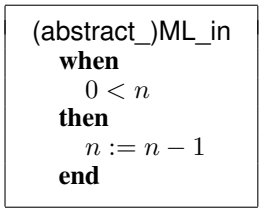
5.5 Informal Proofs of Refinement

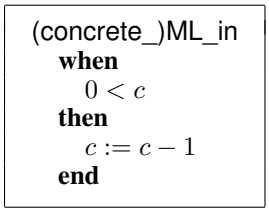
In the next section, we shall clearly define what is required for the concrete version of an event *to refine its abstraction*. In this section, we are just presenting some informal arguments. For this, we are going to compare the abstract and concrete versions of our two events respectively. Here are the two versions of the event ML_out:





As can be seen, the concrete version of this event has guards which are completely different from that of the abstraction. We can already "feel" that the concrete version is *not contradictory* with the abstract one. When the concrete version is enabled, that is when its guards hold, then certainly the abstract one is enabled too. This is so because the two concrete guards $a+b < d$ and $c = 0$ together imply $a+b+c < d$, that is $n < d$, which is the abstract guard, according to invariant **inv1_4** which states that $a + b + c$ is equal to n. Moreover, when a is incremented and the other variables left unchanged as stated in the action $a := a+1$ in the concrete version, this clearly corresponds to the fact that n is incremented in the abstract one, according again to the invariant **inv1_4**. Likewise, here are the two versions of the event ML_in:





A similar informal "proof" can be conducted on these two versions of event ML_in showing that the concrete one does not contradict the abstract one.

5.6 Proving the Correct Refinements of Abstract Events

In this section, we are going to give systematic rules defining exactly what we have to prove in order to ensure that a concrete event indeed refines its abstraction. In fact, we have to prove two different things. First a statement concerning the guards, and second a statement concerning the actions.

Guard Strengthening We first have to prove that the concrete guard is *stronger* than the abstract one. The term "stronger" means that the concrete guard *implies* the abstract guard. In other words, it is not possible to have the concrete version enabled whereas the abstract one would not. Otherwise, it would be possible to have a concrete transition with no counterpart in the abstraction. This has to be proved under the abstract axioms of the constants, the abstract invariants and the concrete invariants. We shall see in section 5.16 that we cannot strengthen the refined guards too much because it might result in unwanted deadlocks.

In more general terms, let c denote the constants, $A(c)$ the set of constant axioms, v the abstract variables, $I(c, v)$ the set of abstract invariants, w the concrete variables, $J(c, v, w)$ the set of concrete invariants. Let an abstract event have the set of guards $G(c, v)$. In other words, $G(c, v)$ stands for the list $G_1(c, v), G_2(c, v), \ldots$. Let the corresponding concrete event have the set of guard $H(c, w)$. We have then to prove the following for each concrete guard $G_i(c, v)$:

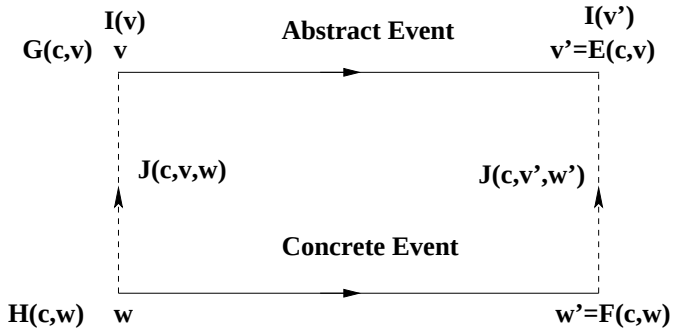
$A(c)$ $I(c, v)$ $J(c, v, w)$ $H(c, w)$ $\vdash$ $G_i(c, v)$	GRD

Axioms Abstract invariants Concrete invariants Concrete guards $\vdash$ Abstract guard	GRD

Notice again that the set of concrete invariants denoted by $J(c, v, w)$ contain some elementary invariants dealing with concrete variables w only, while others are dealing with both abstract and concrete variables v and w. This is the reason why we collectively denote this set of concrete invariants by $J(c, v, w)$.

Also note that it is possible to introduce new constants in a refinement. But we have not stated this in the concrete invariants $J(c, v, w)$ in order to keep the formulae small.

Correct Refinement We have to prove that the concrete event transforms the concrete variables w into w', in a way which does not contradict the abstract event. While this transition happens, the abstract event changes the concrete variables v, which are related to w by the concrete invariant $J(c, v, w)$, into v', which must be related to w' by the modified concrete invariant $J(c, v', w')$. This is illustrated in the following diagram:



With our usual conventions, this leads to the following proof obligation rule named INV, where $J_j(c, v, w)$ denotes a single invariant of the set of concrete invariants $J(c, v, w)$.

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

Axioms Abstract invariant Concrete invariant Concrete guard $\vdash$ Modified concrete invariant	INV

5.7 Applying the Refinement Rules

Coming back to our example, we apply rule GRD to both refined events ML_out and ML_in. This leads to some sequents, which look complicated but are easy to prove.

Applying guard strengthening to Event ML_out. Here is what we have to prove for event ML_out:

axm0_1
axm0_2
inv0_1
inv0_2
inv1_1
inv1_2
inv1_3
inv1_4
inv1_5
Concrete guards of ML_out

$\vdash$

Abstract guards of ML_out

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

ML_out / GRD

This huge impressive sequent can be dramatically simplified. First, we can remove useless hypotheses by applying MON. Then we can apply the equality present in hypothesis, that is $c = 0$, thus transforming hypothesis $a+b+c = n$ first into $a+b+0 = n$ and then into $a+b = n$. Then, we can apply this equality, thus replacing hypothesis $a + b < d$ by the simpler one $n < d$. And now we discover that this is exactly the goal we wanted to prove: we can apply HYP. More formally, this leads to the following successive transformations of our original sequent (obtained after applying rule MON):

$a + b + c = n$
$a + b < d$
$c = 0$
$\vdash$
$n < d$

EQ_LR

$\underline{a + b + 0 = n}$
$\underline{a + b < d}$
$\vdash$
$n < d$

ARI

$a + b = n$
$a + b < d$
$\vdash$
$n < d$

EQ_LR

$n < d$
$\vdash$
$n < d$

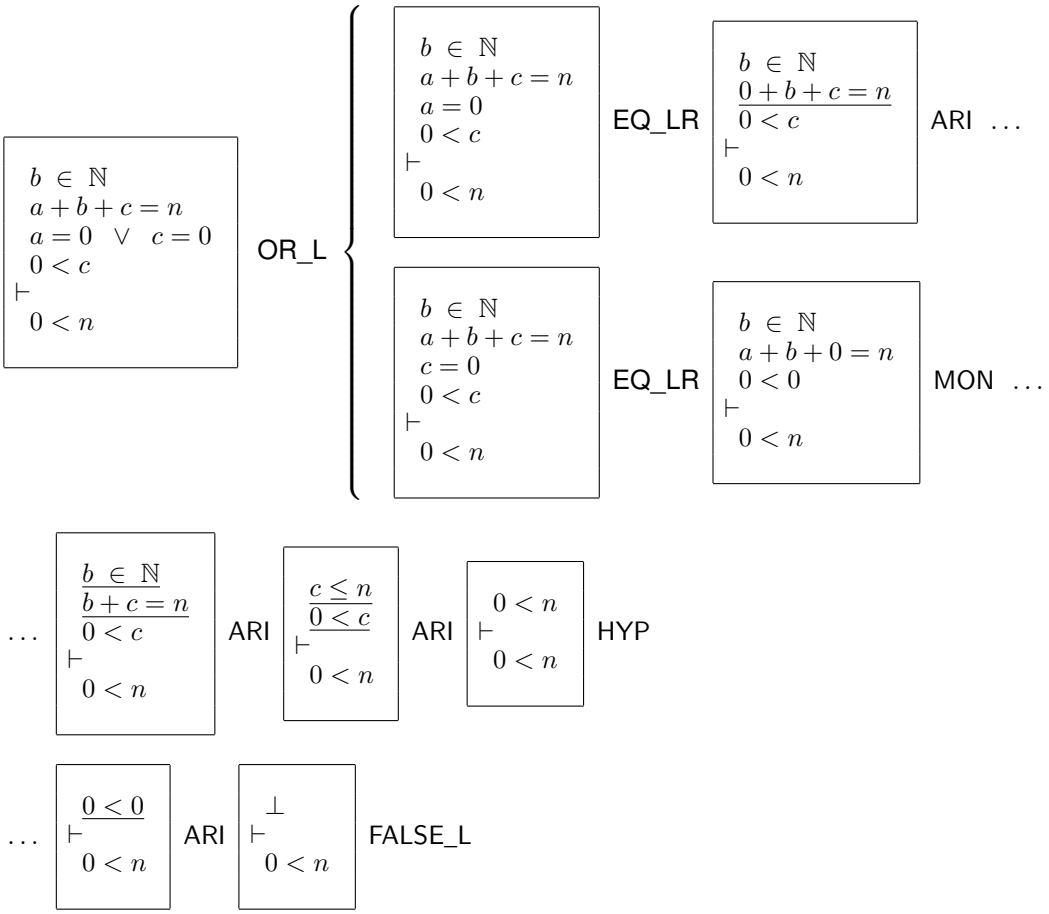
HYP

As can be seen, we have introduced a generic rule of inference named ARI. This is to say in a less formal way that the justification is based on simple arithmetic properties. We could have defined corresponding specific rules of inference but there is no point in doing this here for the moment. In order to make clear which arithmetic properties we are thinking of, the relevant assumptions or goal are underlined in the sequent situated to the left of this ARI justification.

Applying guard strengthening to Event ML_in. After applying MON, we obtain the following sequent:

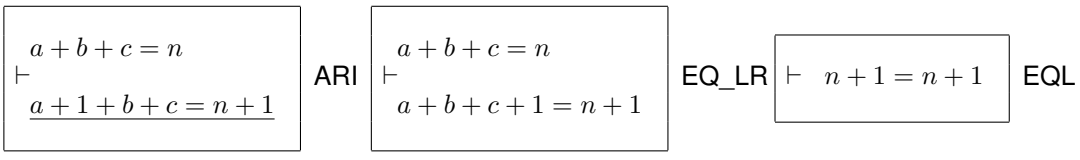
$b \in \mathbb{N}$
$a + b + c = n$
$a = 0 \ \lor \ c = 0$
$0 < c$
$\vdash$
$0 < n$

The disjunctive assumption suggests doing a proof by cases. Then we may apply the equalities, thus simplifying the sequents to prove. The proof is finished by applying a number of simple arithmetic transformations. Here is the proof of this sequent:

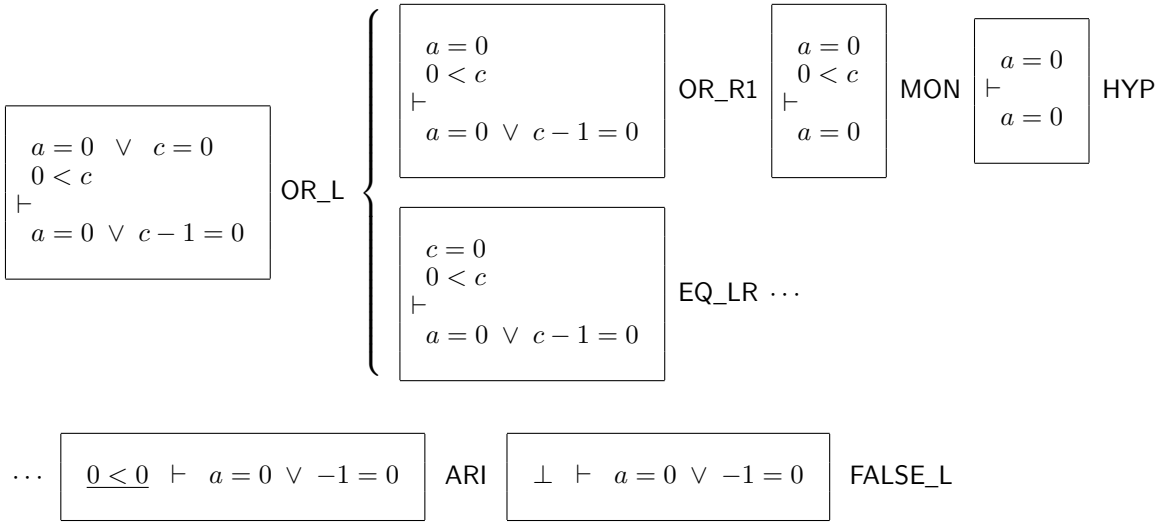


Applying proof obligation rule INV leads to ten predicates to prove since we have two events and five concrete invariants. We shall only present some of them, the other being left as exercises to the reader.

Invariant inv1_4 preservation with Event ML_out. After applying MON, here is the proof we obtain:

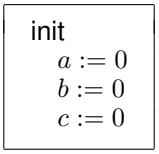


Invariant inv1_5 preservation with Event ML_in. After applying MON, here is the proof we obtain:



5.8 Refining the Initialization Event init

We also have to define the refinement of the special event init. This event can be stated obviously as follows:

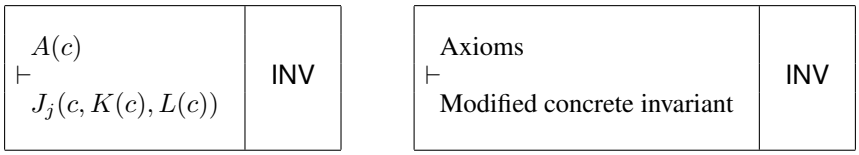


Here we see for the first time a *multiple action*. The corresponding before-after predicate is what we expect:

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

5.9 Correct Proof Obligation Refinement Rule for the Initialization Event init

The proof obligation rule we have to apply in the case of the init event is a special case of the proof obligation rule INV. It is also called INV. If the abstract initialization has an after predicate of the form $v' = K(c)$ and the concrete initialization has an after predicate of the form $w' = L(c)$, then the proof obligation rule is the following:



Notice that we have no proof obligation rule for guard strengthening since, by definition, the initialization event is not guarded.

5.10 Applying the Initialization Proof Obligation Refinement Rule

The application of the proof obligation rule introduced in the previous section is straightforward in our example. Out of the five predicates, we give only the most important ones. Here is what we have to prove concerning invariant **inv1_4**:

axm0_1
axm0_2
$\vdash$
Modified concrete invariant **inv1_4**

$d \in \mathbb{N}$
$d > 0$
$\vdash$
$0 + 0 + 0 = 0$

inv1_4 / INV

Here is what we have to prove concerning invariant **inv1_5**:

axm0_1
axm0_1
$\vdash$
Modified concrete invariant **inv1_5**

$d \in \mathbb{N}$
$d > 0$
$\vdash$
$0 = 0 \quad \lor \quad 0 = 0$

inv1_5 / INV

The proofs of these sequents are left to the reader.

5.11 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:

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

We have here again some *multiple actions*. These actions are incomplete however since in the first one variable c is missing, whereas variable a is missing in the second one. Such multiple actions are associated with the following before-after predicates:

$a' = a - 1 \quad \land \quad b' = b + 1 \quad \land \quad c' = c$

$a' = a \quad \land \quad b' = b - 1 \quad \land \quad c' = c + 1$

It is also easy to understand what these events are doing. Event IL_in corresponds to a car leaving the bridge and entering the island. The action thus decrements the number a of cars on the bridge and simultaneously increments the number b of cars on the island. But this can only be done when there are cars on the bridge, that is when the condition $0 < a$ holds.

Event IL_out corresponds to a car leaving the island and entering the bridge. The action clearly decreases b and simultaneously increases c. But this can be done only if there are cars on the island, that is if the condition $0 < b$ holds. A second condition for event IL_out to be enabled is that there are no cars on the bridge going to the island (remember, the bridge is one-way), that is if the condition $a = 0$ holds.

5.12 The Empty Action skip

As we shall explain in the next section, the new events have to be proved to refine a "dummy event" that is non-guarded and *does nothing* in the abstraction. Such a void action is denoted by means of the empty action skip.

It is very important to note that the before-after predicate of skip depends on the state of the model in which it is located. In the present case, we shall speak first of such an empty action in the initial model which has the single variable n. Its before-after predicate is thus the following:

$$n' = n$$

But a skip action residing in the concrete state, where we have the three variables a, b, and c, would be associated with the following different before-after predicate:

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

5.13 Proving that the New Events are Correct

The new events that we have introduced in section 5.11 were not visible in the abstraction. Although not visible, it does not mean that they did not exist and occur. When you are looking through a microscope you can see things that were not visible without the microscope. By analogy, refinement is the same thing as looking a system through a microscope. The transitions corresponding to the new events IL_in and IL_out were not visible in the abstraction but, again, they existed. Formalizing this idea consists in saying that the new events refine a non-guarded event with the empty action skip. As a consequence, we can use proof obligation rule INV to prove that our new events are correct. Here is what we have to prove for event IL_in and concrete invariant **inv1_4**:

axm0_1
axm0_2
inv0_1
inv0_2
inv1_1
inv1_2
inv1_3
inv1_4
inv1_5
Concrete guards of IL_in
$\vdash$
Modified Invariant **inv1_4**

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

IL_IN / **inv1_4** / INV

Notice that n is not modified as the event refines a non-guarded event with a skip action in the abstraction. After applying MON, we obtain the following proof:

$a + b + c = n$
$\vdash$
$\underline{a - 1 + b + 1 + c = n}$

ARI

$a + b + c = n$
$\vdash$
$a + b + c = n$

HYP

Here is what we have to prove for event IL_in and concrete invariant **inv1_5**:

$\ldots$
inv1_1
inv1_2
inv1_3
inv1_4
inv1_5
Concrete guards of IL_in
$\vdash$
Modified Invariant **inv1_5**

$\ldots$
$a \in \mathbb{N}$
$b \in \mathbb{N}$
$c \in \mathbb{N}$
$a + b + c = n$
$a = 0 \ \lor \ c = 0$
$0 < a$
$\vdash$
$a - 1 = 0 \ \lor \ c = 0$

IL_IN / **inv1_5** / INV

After applying MON, we obtain the following proof:

$a = 0 \ \lor \ c = 0$
$0 < a$
$\vdash$
$a - 1 = 0 \ \lor$
$c = 0$

OR_L

$a = 0$
$0 < a$
$\vdash$
$a - 1 = 0 \ \lor$
$c = 0$

EQ_LR

$\underline{0 < 0}$
$\vdash$
$-1 = 0 \ \lor$
$c = 0$

ARI

$\bot$
$\vdash$
$-1 = 0 \ \lor$
$c = 0$

FALSE_L

$c = 0$
$0 < a$
$\vdash$
$a - 1 = 0 \ \lor \ c = 0$

OR_R2

$c = 0$
$0 < a$
$\vdash$
$c = 0$

MON

$c = 0$
$\vdash$
$c = 0$

HYP

We leave it as an exercise to the reader to state and prove the remaining sequents corresponding to the other predicates and the other new event IL_out.

5.14 Proving the Convergence of the New Events

In the case where we introduce some new events, we have to prove something else, namely that they *do not diverge*. In other words, the new events must not be indefinitely enabled. Should it be the case, then the concrete versions of the existing events, here ML_out and ML_in, could be postponed indefinitely, which is certainly something we want to avoid since such events could possibly occur in the abstraction. For proving this, we have to exhibit a natural number expression called a *variant* and prove that it is *decreased by all new events*. This leads to two proof obligation rules: one states that the proposed variant is a natural number and the other states that the variant is decreased by the new events.

In the first proof obligation rule, called NAT, we prove that the exhibited variant $V(c, w)$ is a natural number. This is to be done assuming the axioms $A(c)$ of the constants c, the abstract invariants $I(c, v)$, the concrete invariants $J(c, v, w)$ and the guards of each new event $H(c, w)$. The guard $H(c, w)$ is put in the antecedent as we are not interested in proving that the exhibited variant is a natural number when the guard of a new event does not hold (it could be negative in this case).

$A(c)$ $I(c, v)$ $J(c, v, w)$ $H(c, w)$ $\vdash$ $V(c, w) \in \mathbb{N}$	NAT

Axioms Abstract invariants Concrete invariants Concrete guards of a new event $\vdash$ Variant $\in \mathbb{N}$	NAT

The second proof obligation rule states that the variant $V(c, w)$ is decreased. This has to be proved for each new event with guards $H(c, w)$ and before-after predicate $w' = F(c, w)$:

$A(c)$ $I(c, v)$ $J(c, v, w)$ $H(c, w)$ $\vdash$ $V(c, F(c, w)) < V(c, w)$	VAR

Axioms Abstract invariants Concrete invariants Concrete guards of a new event $\vdash$ Modified Variant $<$ Variant	VAR

Note that the variant is unique. In other words, the *same variant* has to be decreased by each new event. Sometimes the variant might be a little more complicated than a simple natural number expression, but in this example we only need a natural number variant.

5.15 Applying the Non-divergence Proof Obligation Rules

In our case, the proposed variant is the following:

$\mathbf{variant\_1}: \quad 2 * a + b$

Applying proof obligation rule VAR on event IL_in leads to the following large but obvious sequent to prove:

axm0_1 **axm0_2** **inv0_1** **inv0_2** **inv1_1** **inv1_2** **inv1_3** **inv1_4** **inv1_5** Concrete guards of IL_in ⊢ Modified variant $<$ Variant	$d \in \mathbb{N}$ $0 < d$ $n \in \mathbb{N}$ $n \leq d$ $a \in \mathbb{N}$ $b \in \mathbb{N}$ $c \in \mathbb{N}$ $a + b + c = n$ $a = 0 \ \lor \ c = 0$ $0 < a$ ⊢ $2 * (a - 1) + b + 1 < 2 * a + b$	IL_in / VAR

Next is the application of proof obligation rule VAR to event IL_out:

axm0_1 **axm0_2** **inv0_1** **inv0_2** **inv1_1** **inv1_2** **inv1_3** **inv1_4** **inv1_5** Concrete guards of IL_out ⊢ Modified variant $<$ Variant	$d \in \mathbb{N}$ $0 < d$ $n \in \mathbb{N}$ $n \leq d$ $a \in \mathbb{N}$ $b \in \mathbb{N}$ $c \in \mathbb{N}$ $a + b + c = n$ $a = 0 \ \lor \ c = 0$ $0 < b$ $a = 0$ ⊢ $2 * a + b - 1 < 2 * a + b$	IL_out / VAR

Both these sequents are proved by simple arithmetic calculations. Proofs are omitted.

5.16 Relative Deadlock Freedom

Finally, we have to prove that all concrete events (old and new together) do not deadlock more often than the abstract events. We have thus to prove that the disjunction of the abstract guards $G_1(c, v), \ldots, G_m(c, v)$ imply the disjunction of the concrete guards $H_1(c, w), \ldots, H_n(c, w)$. This proof obligation rule is called DLF:

$A(c)$ $I(c, v)$ $J(c, v, w)$ $G_1(c, v) \ \lor \ \ldots \ \lor \ G_m(c, v)$ ⊢ $H_1(c, w) \ \lor \ \ldots \ \lor \ H_n(c, w)$	DLF

Axioms Abstract invariants Concrete invariants Disjunction of abstract guards ⊢ Disjunction of concrete guards	DLF

5.17 Applying Relative Deadlock Freedom Proof Obligation

The application of proof obligation rule DLF leads to the following to prove. Notice that we have removed the disjunction of the abstract guards in the antecedent of the implication because we have already proved them in the initial model:

axm0_1 **axm0_2** **inv0_1** **inv0_2** **inv1_1** **inv1_2** **inv1_3** **inv1_4** **inv1_5** $\vdash$ Disjunction of concrete guards	$d \in \mathbb{N}$ $0 < d$ $n \in \mathbb{N}$ $n \leq d$ $a \in \mathbb{N}$ $b \in \mathbb{N}$ $c \in \mathbb{N}$ $a + b + c = n$ $a = 0 \ \lor \ c = 0$ $\vdash$ $(a + b < d \ \land \ c = 0) \ \lor$ $c > 0 \ \lor$ $a > 0 \ \lor$ $(b > 0 \ \land \ a = 0)$	DLF

5.18 More Inference Rules

In the previous sequent, we had to prove the *disjunction* of various predicates. A convenient way to do this (using commutativity and associativity of disjunction) is to apply the following rule that moves the negation of one of the disjuncts of the goal into the assumptions of the sequent. Here is the corresponding rule:

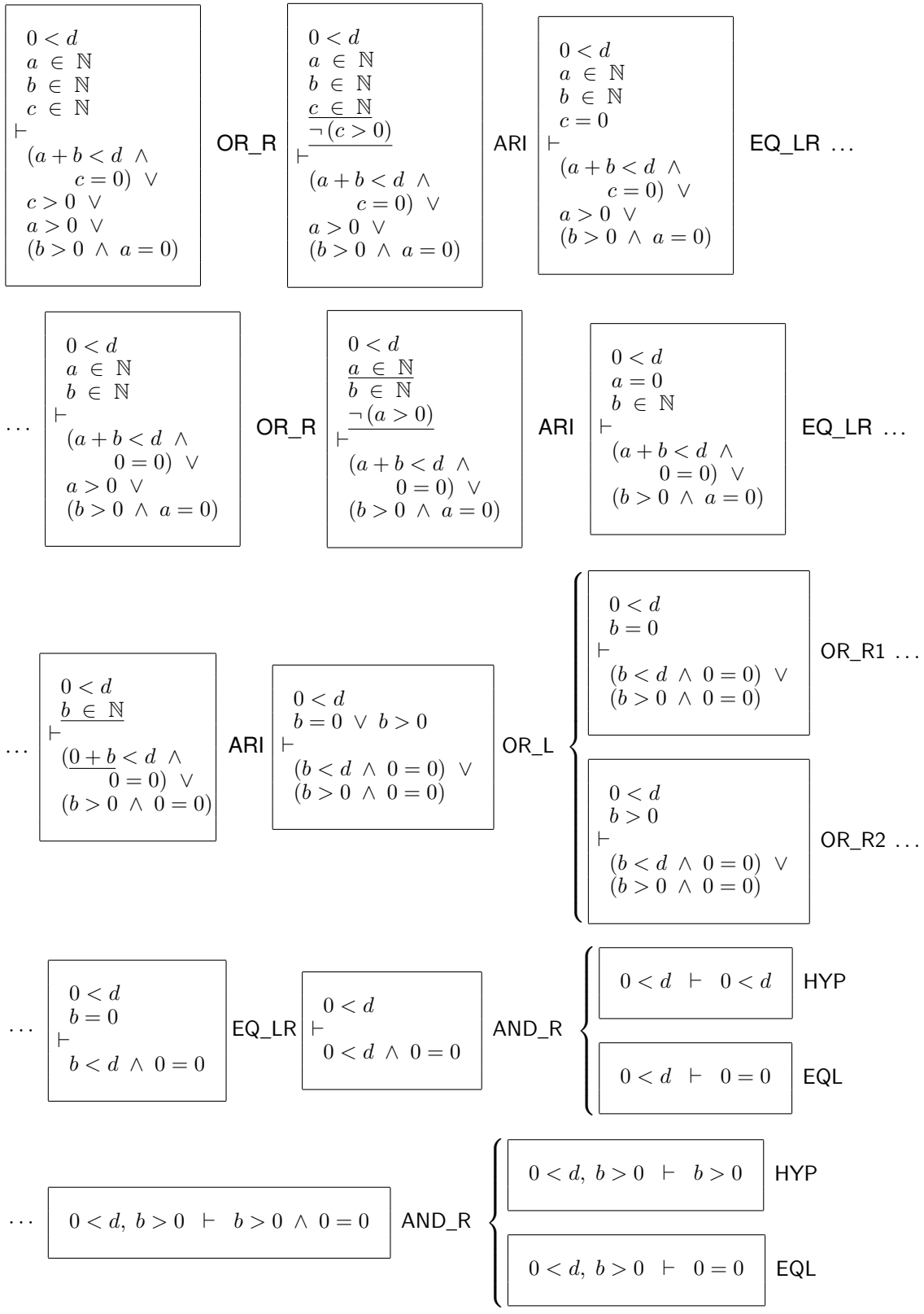
$$\frac{\mathbf{H}, \neg \mathbf{P} \ \vdash \ \mathbf{Q}}{\mathbf{H} \ \vdash \ \mathbf{P} \lor \mathbf{Q}} \quad \text{OR\_R}$$

The next two rules allow to simplify conjunctive predicates appearing either in the assumptions or in the goal of a sequent:

$$\frac{\mathbf{H}, \mathbf{P}, \mathbf{Q} \ \vdash \ \mathbf{R}}{\mathbf{H}, \ \mathbf{P} \land \mathbf{Q} \ \vdash \ \mathbf{R}} \quad \text{AND\_L}$$

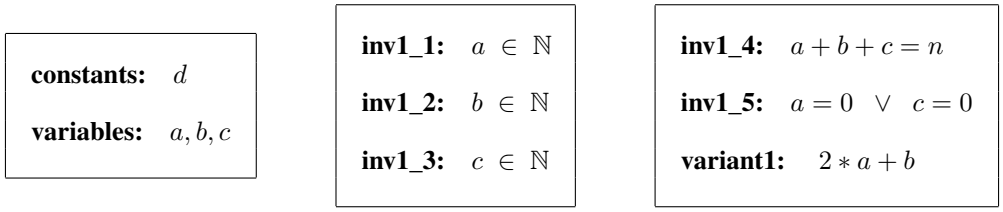
$$\frac{\mathbf{H} \ \vdash \ \mathbf{P} \qquad \mathbf{H} \ \vdash \ \mathbf{Q}}{\mathbf{H} \ \vdash \ \mathbf{P} \land \mathbf{Q}} \quad \text{AND\_R}$$

Proving the Deadlock Freedom Proof Obligation. Equipped with these new inference rules, we may now prove the deadlock freedom proof obligation (after applying MON):

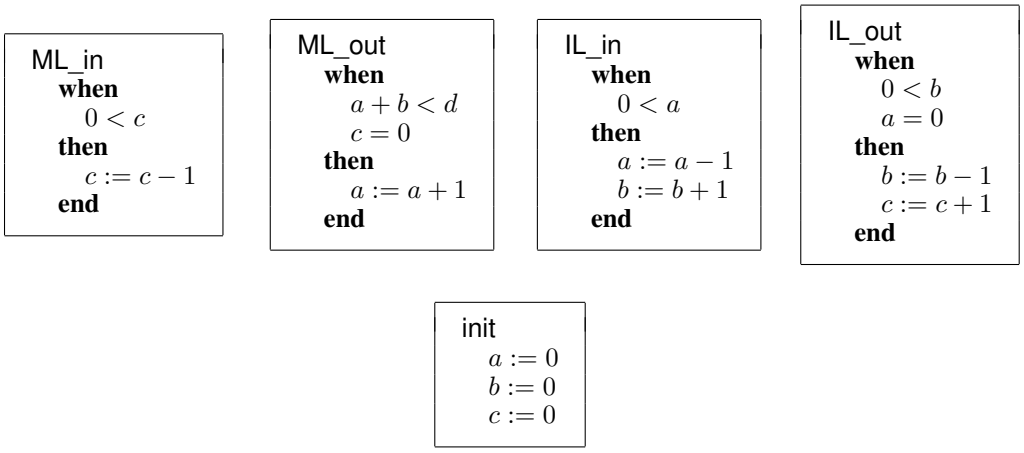

$0 < d$
$a \in \mathbb{N}$
$b \in \mathbb{N}$
$c \in \mathbb{N}$
$\vdash$
$(a + b < d \ \wedge$
$c = 0) \ \vee$
$c > 0 \ \vee$
$a > 0 \ \vee$
$(b > 0 \ \wedge \ a = 0)$
OR_R
$\neg(c > 0)$
ARI
$c = 0$
EQ_LR ...
...
$a \in \mathbb{N}$
$b \in \mathbb{N}$
$0 = 0) \ \vee$
$a = 0$
$b \in \mathbb{N}$
$b \in \mathbb{N}$
$\neg(a > 0)$
$0 = 0) \ \vee$
$b = 0$
$b \in \mathbb{N}$
$0 = 0) \ \vee$
$b = 0 \ \vee \ b > 0$
OR_L
$b = 0$
OR_R1 ...
$b > 0$
OR_R2 ...
HYP
EQL
AND_R
$0 < d, \ b > 0 \ \vdash \ b > 0$
$0 < d, \ b > 0 \ \vdash \ 0 = 0$

5.19 Summary of the First Refinement

Here is a summary of the state of the first refinement:



Here is a summary of the events of the first refinement



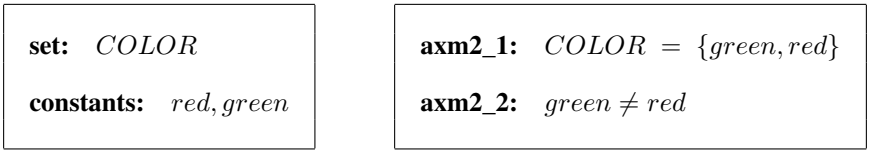
6 Second Refinement: Introducing the Traffic Lights

In its present form, the model of the bridge appears to be a bit magic. It seems, from our observation, 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). This means they can observe the state of the system. Clearly, this is not realistic. In reality, as we know, the drivers follows the indication of some traffic lights, they clearly do not count the cars!

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 can change the colors of the traffic lights. Fig. 7 illustrates the new physical situation which can be observed in this refinement.

6.1 Refining the State

At this stage, we must extend our set of constants by first introducing the set $COLOR$ and its two distinct values red and $green$. It is done as follows:



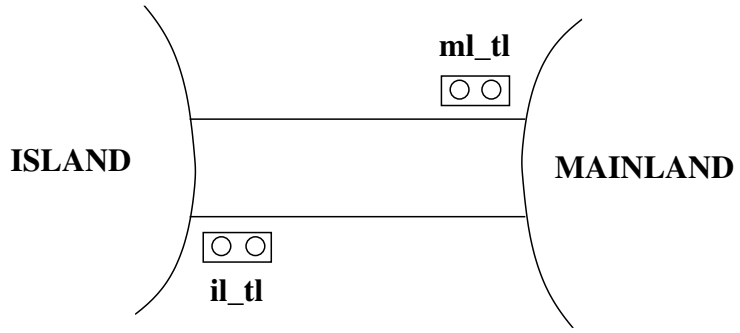


Fig. 7. The Traffic Lights

Two new variables are then introduced, namely ml_tl (for mainland traffic light) and il_tl (for island traffic light). These variables are defined as colors: this is formalized in invariants named **inv2_1** and **inv2_2** below. Since drivers are allowed to pass only when traffic lights are green, we better ensure, by two *conditional invariants* named **inv2_3** and **inv2_4**, that when ml_tl is green then the abstract guard of event ML_out holds, and that when il_tl is green then the abstract guard of event IL_out holds. Notice that we are here taking account of requirements ENV-1, ENV-2, and ENV-3. Here are the refined variables:

variables: ... ml_tl il_tl

inv2_1: $ml\_tl \in COLOR$

inv2_2: $il\_tl \in COLOR$

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

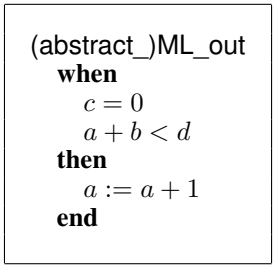
inv2_4: $il\_tl = green \Rightarrow 0 < b \ \wedge \ a = 0$

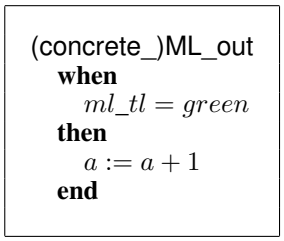
Note again that invariants **inv2_3** and **inv2_4** are *conditional invariants*. Such invariants are introduced by means of the logical implication operator "⇒". Clearly, we shall need inference rules dealing with this logical operator. We shall introduce such inference rules in section 6.6.

At this point, it seems that we are in a situation which is a bit different from the one we had in the previous refinement, where concrete variables a, b, and c were replacing the more abstract variable n. Here we are just adding two new variables ml_tl and il_tl and we are keeping the abstract variables a, b and c. Such a special refinement scheme is called a *superposition*. We shall see in section 6.4 that superposition refinement requires an additional proof obligation rule.

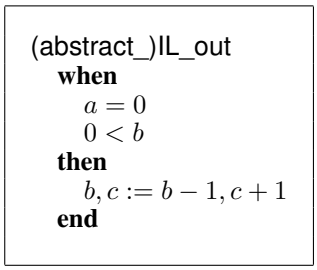
6.2 Refining Abstract Events

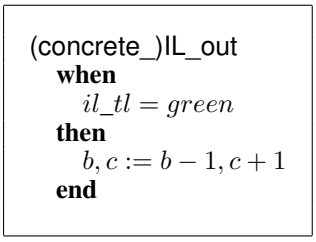
Events ML_out and IL_out are now refined by changing their guards to the test of the green value of the corresponding traffic lights. This is here where we implicitly assume that drivers obey the traffic lights, as indicated by requirements ENV-3. Note that events IL_in (entering the island from the bridge) and ML_in (entering the mainland from the bridge) are not modified in this refinement. Here is the new version of event ML_out presented together with its abstraction:





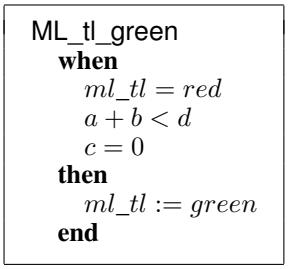
Here is the new version of event IL_out presented together with its abstraction:

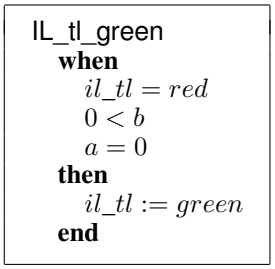




6.3 Introducing New Events

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





6.4 Superposition: Adapting the Refinement Rule

In this section, we depart again from our example and explain superposition in some general terms. When we have a case of superposition where some abstract variables are kept in the concrete state we have to adapt the refinement proof obligation rule INV. The other refinement rules need not be adapted.

Suppose we have variables u and v in the abstract state and variables v and w in the concrete state. Variables v are thus common to the abstract and concrete states. Let $I(c, u, v)$ denote the abstract invariants and $J(c, u, v, w)$ denote the concrete invariants. In order to be able to apply the proof obligation rule INV, the abstract and concrete states must be *completely disjoint*, and this is clearly not the situation we have in the present case of superposition.

In order to get back to a disjoint situation, we can rename the variables in the concrete state, changing v to, say, $v1$ and *adding the additional concrete invariant* $v1 = v$. Suppose now that the before-after

predicates of an event in the abstract state are $u' = E(c, u, v)$ and $v' = M(c, u, v)$. Suppose that the before-after predicates of the corresponding concrete event are $v' = N(c, v, w)$ and $w' = F(c, v, w)$ in the concrete state. Suppose the guards of this concrete event are denoted by $H(c, v, w)$. Applying the proof obligation refinement rule INV yields two kinds of sequent to prove:

Axioms of constants	$A(c)$	$A(c)$
Abstract invariants	$I(c, u, v)$	$I(c, u, v)$
Concrete invariants	$J(c, u, v1, w)$	$J(c, u, v1, w)$
	$v1 = v$	$v1 = v$
Concrete guards	$H(c, v1, w)$	$H(c, v1, w)$
$\vdash$	$\vdash$	$\vdash$
Modified invariants	$J_j(c, E(c, u, v), M(c, u, v), F(c, v1, w))$	$M(c, u, v) = N(c, v1, w)$

Applying now the equality $v1 = v$, that is replacing every occurrence of $v1$ by v in the previous sequents, leads to the following where $v1$ has disappeared. This is the adaptation we wanted to perform on the proof obligation refinement rule INV. We can interpret this adaptation as adding to the basic proof obligation rule INV another proof obligation rule which says that the abstract and concrete expressions assigned to the common variables in the abstract and concrete states are equal under the assumption of the concrete invariants $J(c, u, v, w)$:

$A(c)$	$A(c)$
$I(c, u, v)$	$I(c, u, v)$
$J(c, u, v, w)$	$J(c, u, v, w)$
$H(c, v, w)$	$H(c, v, w)$
$\vdash$	$\vdash$
$J(c, E(c, u, v), M(c, u, v), F(c, v, w))$	$M(c, u, v) = N(c, v, w)$

The first sequent corresponds to proof obligation rule INV as before and the second, and new one named SIM, simply states the equality of the abstract and concrete expressions assigned to the common variables:

$A(c)$ $I(c, u, v)$ $J(c, u, v, w)$ $H(c, v, w)$ $\vdash$ $M(c, u, v) = N(c, v, w)$	SIM

Axioms of constants Abstract invariants Concrete invariants Concrete guards $\vdash$ Equality of the expressions assigned to the common variables

6.5 Proving that the Events are Correct

In order to prove that the concrete old events refine their abstractions, we now have to apply three proof obligations: GRD, SIM, and INV.

Events IL_in and ML_in are identical to their abstraction, so proof obligations GRD and SIM applied to them do not imply anything to prove. It is easy to prove that proof obligation rule INV applied to them leads to statements that are trivial: one has to prove that the concrete invariants **inv2_3** and **inv2_4** are preserved.

Proof obligation SIM applied to events events IL_out and ML_out is also trivial because the abstract and concrete actions of these events are exactly the same. Applying proof obligation rule GRD to these events leads to simple proofs: one has to use invariants **inv2_3** and **inv2_4** to prove that the guards are strengthened. This is what we did informally in section 6.2.

What remains to be done are therefore the proofs obtained by applying proof obligation rule INV to events IL_out and ML_out. We shall see below that this raises some difficulties.

6.6 More Logical Inference Rules

In this section, we add some new logical rules of inference which will be needed in order to perform our proofs. The next two rules allow to simplify implicative predicates appearing either in the assumptions or in the goal of a sequent. We also present a rule dealing with a negative assumption[4]:

$$\frac{\mathbf{H}, \mathbf{P}, \mathbf{Q} \ \vdash \ \mathbf{R}}{\mathbf{H}, \ \mathbf{P}, \ \mathbf{P} \Rightarrow \mathbf{Q} \ \vdash \ \mathbf{R}} \ \text{IMP\_L} \qquad \frac{\mathbf{H}, \mathbf{P} \ \vdash \ \mathbf{Q}}{\mathbf{H} \ \vdash \ \mathbf{P} \Rightarrow \mathbf{Q}} \ \text{IMP\_R} \qquad \frac{\mathbf{H}, \mathbf{P} \ \vdash \ \neg \mathbf{Q}}{\mathbf{H}, \ \neg \mathbf{P} \ \vdash \ \mathbf{Q}} \ \text{NOT\_L}$$

6.7 Tentative Proofs and Solutions

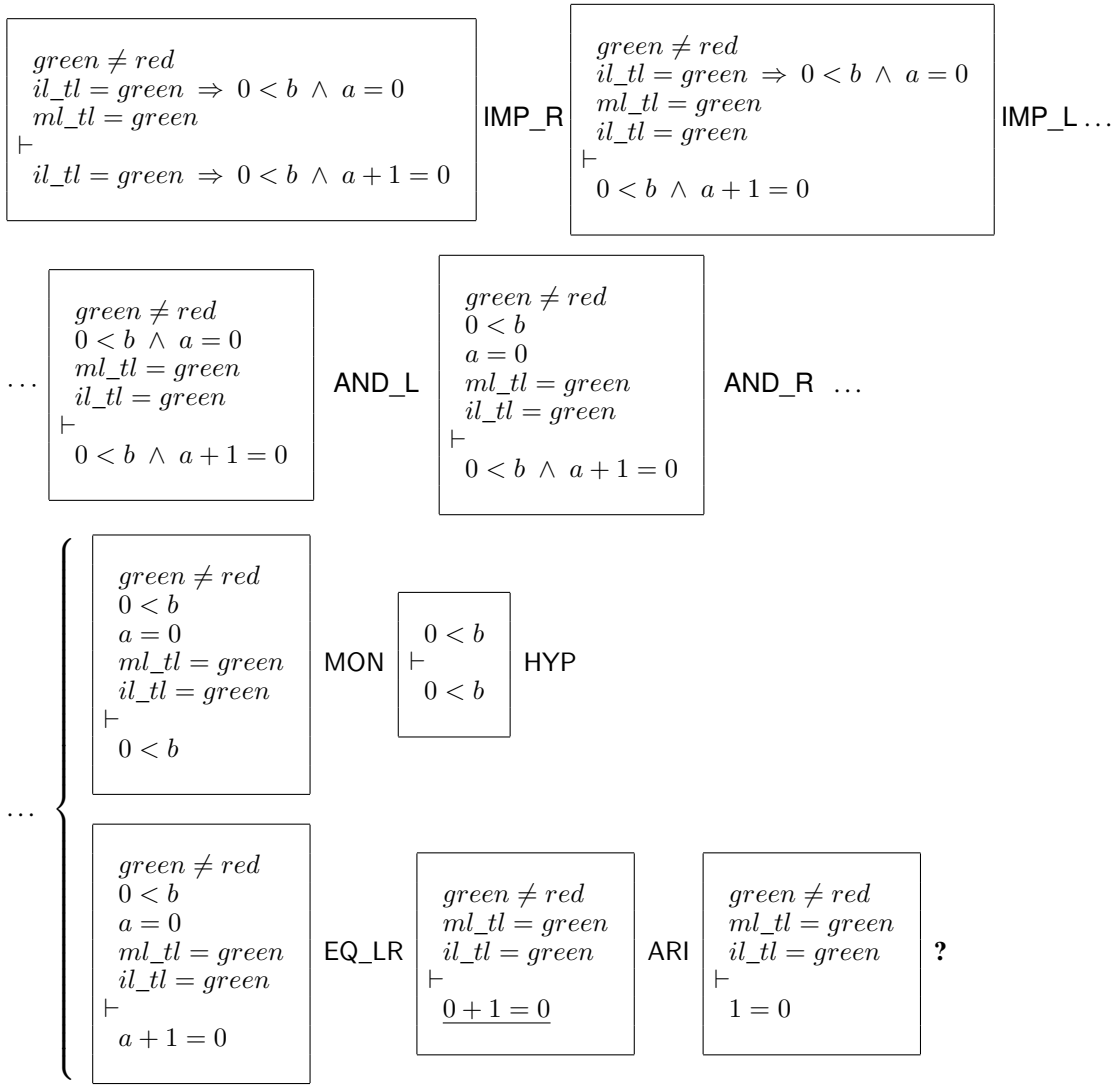
Proving that event ML_out preserves invariant inv2_4. Here is the sequent to prove:

axm0_1	$d \in \mathbb{N}$
axm0_2	$0 < d$
axm2_1	$COLOR = \{green, red\}$
axm2_2	$green \neq red$
inv0_1	$n \in \mathbb{N}$
inv0_2	$n \leq d$
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	$a = 0 \ \lor \ c = 0$
inv2_1	$ml\_tl \in \{red, green\}$
inv2_2	$il\_tl \in \{red, green\}$
inv2_3	$ml\_tl = green \ \Rightarrow \ a + b < d \ \wedge \ c = 0$
inv2_4	$il\_tl = green \ \Rightarrow \ 0 < b \ \wedge \ a = 0$
Guard of ML_out	$ml\_tl = green$
$\vdash$	$\vdash$
Modified **inv2_4**	$il\_tl = green \ \Rightarrow \ 0 < b \ \wedge \ a + 1 = 0$

ML_out / **inv2_4** / INV

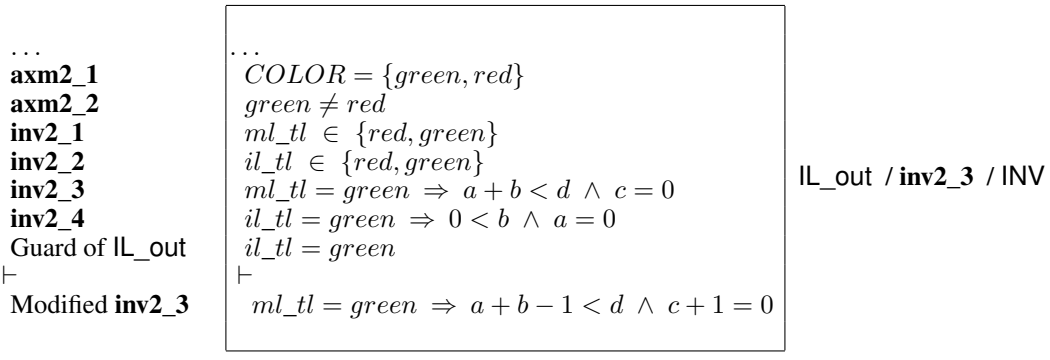
Here is a tentative proof of this sequent (after applying MON):

[4] We remind the reader that all rules of inference are reviewed in chapter 9.

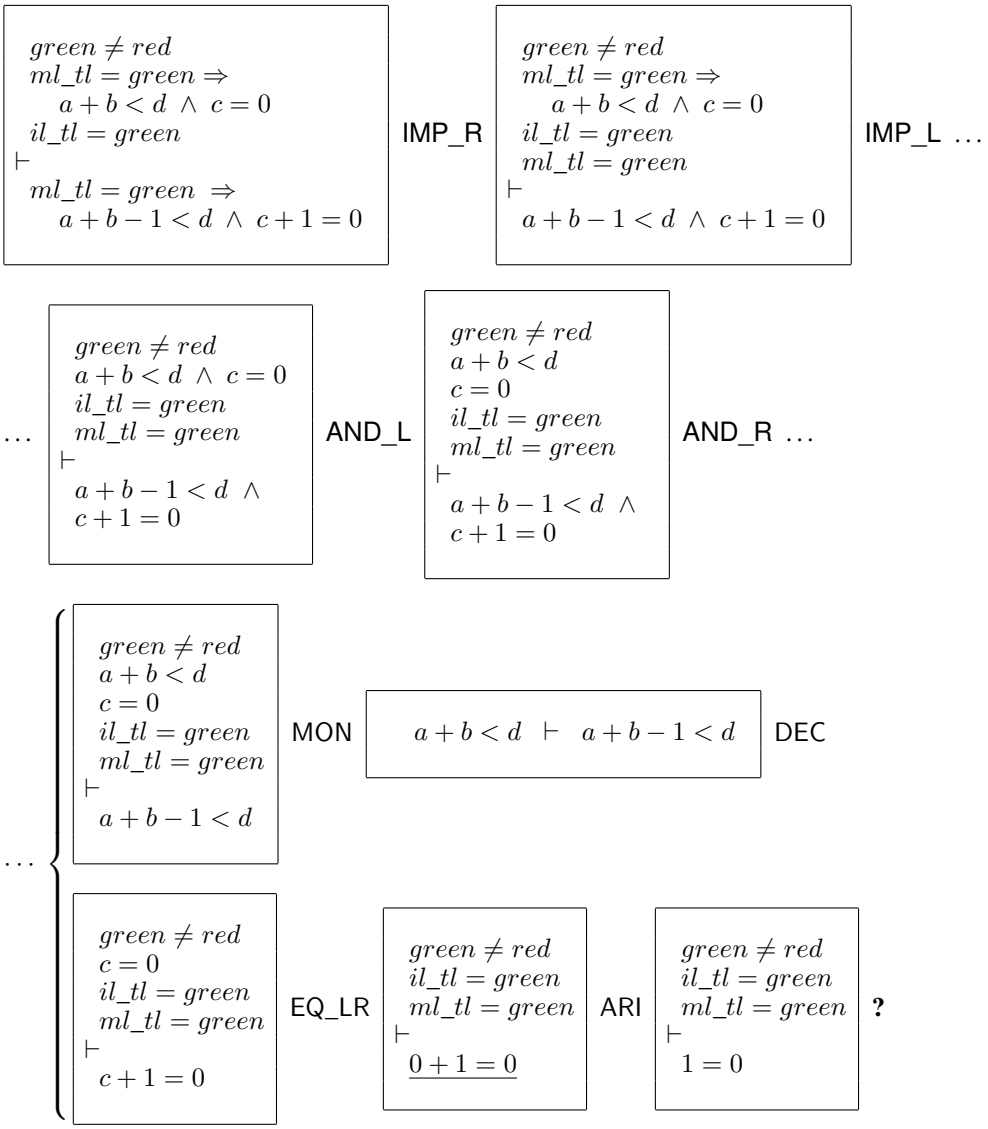


Clearly, the last sequent cannot be proved.

Proving that event IL_out preserves invariant inv2_3. Here is what we have to prove for the preservation of invariant **inv2_3** by event IL_out:

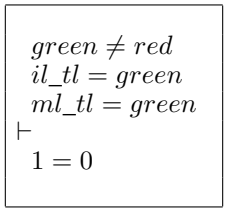


Here is the tentative proof (after applying MON):



Clearly the last sequent cannot be proved either.

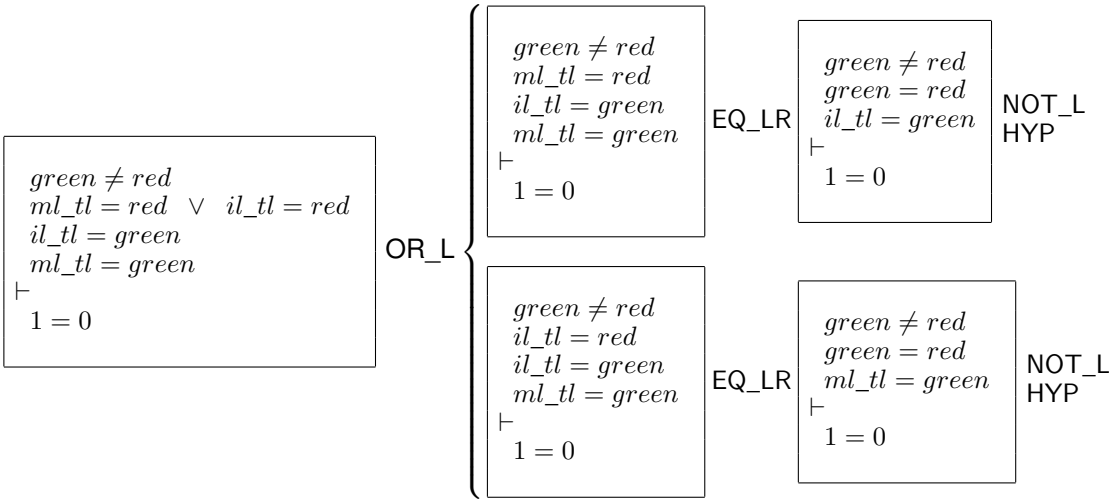
The Solution. The two previous proofs failed because we had to prove the following sequent:



What this shows is that both lights cannot be green at the same time. This is an *obvious fact*, which we have nevertheless completely forgotten to express. We thus now introduce it as an additional invariant:

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

We note that this invariant could have been a requirement, although it could have been deduced from requirement ENV-3 which says that "cars are not supposed to pass on a red traffic light, only a green one", and requirement FUN-3 which says that " the bridge is one-way or the other, *not both at the same time*". Adding this invariant will solve the problem, since then we have the following extension to our proofs:



Modifying events ML_tl_green **and** IL_tl_green**.** This new invariant **inv2_5** has to be preserved and this is clearly not the case with the new events ML_tl_green and IL_tl_green we proposed earlier in section 6.3 , unless we correct them by turning to red the other traffic light, yielding:

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

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

Proving that event ML_out **preserves invariant inv2_3.** When trying to prove the preservation of invariant **inv2_3** by event ML_out, we are again in trouble. Next is what we have to prove:

$\ldots$
inv2_3
inv2_4
Guard of ML_out
$\vdash$
Modified **inv2_3**

$\ldots$
$ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
$il\_tl = green \Rightarrow 0 < b \wedge a = 0$
$ml\_tl = green$
$\vdash$
$ml\_tl = green \Rightarrow a + 1 + b < d \wedge c = 0$

ML_out / **inv2_3** / INV

Here is the tentative proof (after applying MON):

$ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
$\vdash$
$ml\_tl = green \Rightarrow a + 1 + b < d \wedge c = 0$

IMP_R

$ml\_tl = green \Rightarrow$
$\quad a + b < d \wedge c = 0$
$ml\_tl = green$
$\vdash$
$a + 1 + b < d \wedge c = 0$

IMP_L $\ldots$

$\ldots$

$a + b < d \wedge c = 0$
$ml\_tl = green$
$\vdash$
$a + 1 + b < d \wedge c = 0$

AND_L

$a + b < d$
$c = 0$
$ml\_tl = green$
$\vdash$
$a + 1 + b < d \wedge c = 0$

AND_R $\ldots$

$\ldots$

$a + b < d$
$c = 0$
$ml\_tl = green$
$\vdash$
$a + 1 + b < d$

?

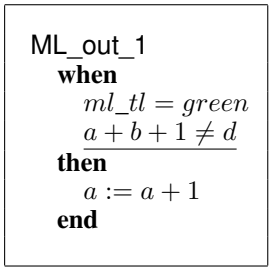
$a + b < d$
$c = 0$
$ml\_tl = green$
$\vdash$
$c = 0$

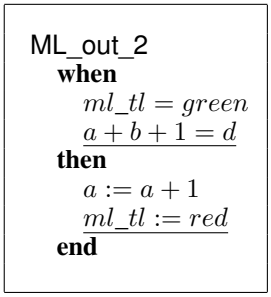
MON

$c = 0$
$\vdash$
$c = 0$

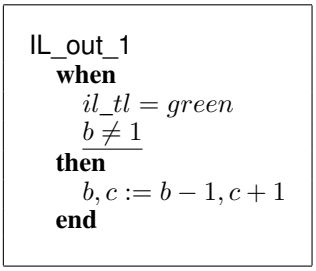
HYP

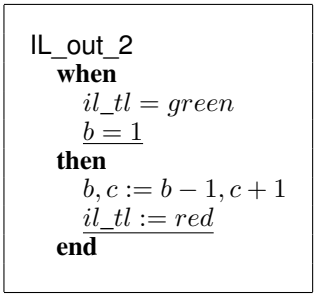
As can be seen, the first of the last two sequents cannot be proved when $a + 1 + b$ is equal to d unless ml_tl is set to red. In fact, when $a + 1 + b$ is equal to d, it means that the entering car is the last one allowed to enter at this stage because more cars would violate requirement FUN-3, which 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:





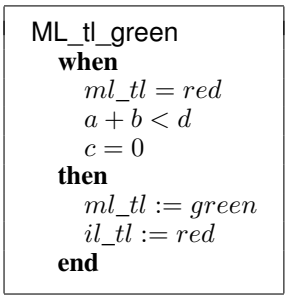
Proving that event IL_out preserves invariant inv2_4. For similar reasons, invariant **inv2_4** cannot be maintained by event IL_out when b is equal to 1. In this case, the last car is leaving the island. As a consequence, the island traffic light has to turn red. As for event ML_out in the previous section, we have to split event IL_out as follows:

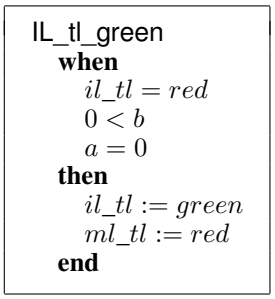




6.8 Convergence of New Events

We have now to prove that the new events cannot diverge. 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 equal to 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:





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 TRUE or FALSE, they are member of the pre-defined set BOOL made of the two distinct values TRUE and FALSE. When ml_pass is equal to TRUE, it means that one car at least has passed on the bridge going to the island since the mainland traffic light last turned green. And similarly when il_pass is equal to TRUE. These variables are formalized in the following invariants:

variables: ...
ml_pass,
il_pass

inv2_6: $ml\_pass \in \text{BOOL}$

inv2_7: $il\_pass \in \text{BOOL}$

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

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

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

IL_out_1
when
$il\_tl = green$
$b \neq 1$
then
$b := b - 1$
$c := c + 1$
$il\_pass := \text{TRUE}$
end

IL_out_2
when
$il\_tl = green$
$b = 1$
then
$b := b - 1$
$c := c + 1$
$il\_tl := red$
$il\_pass := \text{TRUE}$
end

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

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

IL_tl_green
when
$il\_tl = red$
$0 < b$
$a = 0$
$ml\_pass = \text{TRUE}$
then
$il\_tl := green$
$ml\_tl := red$
$il\_pass := \text{FALSE}$
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: $ml\_pass + il\_pass$

However, this variant is not correct as variables ml_pass and $ilpass$ are not natural number variables but boolean variables. For correcting this, we have to transform boolean expression into numeric expressions.This can be done in a straightforward way by defining the following constants b_2_n which is a function from the set $BOOL$ to the set $\{0, 1\}$:

constants: $\ldots$
b_2_n

axm2_3: $b\_2\_n \in \text{BOOL} \rightarrow \{0, 1\}$

axm2_4: $b\_2\_n(\text{TRUE}) = 1$

axm2_4: $b\_2\_n(\text{FALSE}) = 0$

The variant can be now properly defined as follows:

variant_2: $b\_2\_n(ml\_pass) + b\_2\_n(il\_pass)$

The sequents to be proved by applying proof obligation rule VAR on events ML_tl_green and IL_tl_green are the following:

$$
\begin{array}{l}
ml\_tl = red \\
a + b < d \\
c = 0 \\
il\_pass = \text{TRUE} \\
\vdash \\
b\_2\_n(il\_pass) < b\_2\_n(ml\_pass) + b\_2\_n(il\_pass)
\end{array}
$$

$$
\begin{array}{l}
il\_tl = red \\
b > 0 \\
a = 0 \\
ml\_pass = \text{TRUE} \\
\vdash \\
b\_2\_n(ml\_pass) < b\_2\_n(ml\_pass) + b\_2\_n(il\_pass)
\end{array}
$$

At this point, we figure out that it cannot be proved unless $ml\_pass = \text{TRUE}$ in the first case (so that $b\_2\_n(ml\_pass)$ is equal to 1) and $il\_pass = \text{TRUE}$ in the second one for a similar reason. This suggests adding the following invariants:

inv2_8: $ml\_tl = red \Rightarrow ml\_pass = \text{TRUE}$

inv2_9: $il\_tl = red \Rightarrow il\_pass = \text{TRUE}$

It remains now 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.

6.9 Relative Deadlock Freedom

It remains now to prove that the relative deadlock freedom proof obligation rule DLF holds. Note that the "relative" deadlock freedom becomes in our example an "absolute" deadlock freedom since we have already proved that the previous abstractions are deadlock-free. The statement to prove is then the disjunction of the various guards with some simplified assumptions (we do not need all invariants):

$$
\begin{array}{l}
d \in \mathbb{N} \\
0 < d \\
ml\_tl \in COLOR \\
il\_tl \in COLOR \\
ml\_pass \in \mathrm{BOOL} \\
il\_pass \in \mathrm{BOOL} \\
a \in \mathbb{N} \\
b \in \mathbb{N} \\
c \in \mathbb{N} \\
ml\_tl = red \Rightarrow ml\_pass = \mathrm{TRUE} \\
il\_tl = red \Rightarrow il\_pass = \mathrm{TRUE} \\
\vdash \\
(ml\_tl = red \land a + b < d \land c = 0 \land ml\_pass = \mathrm{TRUE} \land il\_pass = \mathrm{TRUE}) \lor \\
(il\_tl = red \land a = 0 \land b > 0 \land ml\_pass = \mathrm{TRUE} \land il\_pass = \mathrm{TRUE}) \lor \\
ml\_tl = green \lor \\
il\_tl = green \lor \\
a > 0 \lor \\
c > 0
\end{array}
\quad \mathsf{DLF}
$$

Here is a sketch of the corresponding proof. In this sketch, many intermediate steps have been omitted, this is indicated by the symbol $\leadsto$ which stands for the missing intermediate steps:

$$
\begin{array}{l}
d \in \mathbb{N} \\
0 < d \\
b \in \mathbb{N} \\
ml\_tl = red \\
il\_tl = red \\
ml\_tl = red \Rightarrow ml\_pass = \mathrm{TRUE} \\
il\_tl = red \Rightarrow il\_pass = \mathrm{TRUE} \\
\vdash \\
(b < d \land ml\_pass = \mathrm{TRUE} \land \\
\quad il\_pass = 1) \lor \\
(b > 0 \land ml\_pass = \mathrm{TRUE} \land \\
\quad il\_pass = 1)
\end{array}
\quad \leadsto \quad
\begin{array}{l}
d \in \mathbb{N} \\
0 < d \\
b \in \mathbb{N} \\
ml\_tl = red \\
il\_tl = red \\
ml\_pass = \mathrm{TRUE} \\
il\_pass = \mathrm{TRUE} \\
\vdash \\
(b < d \land ml\_pass = \mathrm{TRUE} \land \\
\quad il\_pass = \mathrm{TRUE}) \lor \\
(b > 0 \land ml\_pass = \mathrm{TRUE} \land \\
\quad il\_pass = \mathrm{TRUE})
\end{array}
\quad \leadsto \cdots
$$

$$
\cdots \quad
\begin{array}{l}
0 < d \\
b \in \mathbb{N} \\
\vdash \\
b < d \lor b > 0
\end{array}
\quad \mathsf{OR\_R1} \quad
\begin{array}{l}
0 < d \\
b = 0 \\
\vdash \\
b < d
\end{array}
\quad \mathsf{EQ\_LR} \quad
\boxed{0 < d \vdash 0 < d} \quad \mathsf{HYP}
$$

6.10 Conclusion and Summary of the Second Refinement

During this refinement, we have seen again how the proofs (or rather the failed proof attempts) have helped us correcting our mistakes or improving our model. If fact, we discovered four errors, we introduced several additional invariants, we corrected four events, and we introduced two more variables. Here is the final version of this second refinement:

variables: ...
ml_tl
il_tl
ml_pass
il_pass

inv2_1: $ml\_tl \in COLOR$

inv2_2: $il\_tl \in COLOR$

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

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

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

inv2_6: $ml\_pass \in \text{BOOL}$

inv2_7: $il\_pass \in \text{BOOL}$

inv2_8: $ml\_tl = red \Rightarrow ml\_pass = \text{TRUE}$

inv2_9: $il\_tl = red \Rightarrow il\_pass = \text{TRUE}$

variant_2: $b\_2\_n(ml\_pass) + b\_2\_n(il\_pass)$

And here are the events of the second refinement:

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

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

IL_out_1
when
$il\_tl = green$
$b \neq 1$
then
$b := b - 1$
$c := c + 1$
$il\_pass := \text{TRUE}$
end

IL_out_2
when
$il\_tl = green$
$b = 1$
then
$b := b - 1$
$c := c + 1$
$il\_tl := red$
$il\_pass := \text{TRUE}$
end

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

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

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

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

7 Third Refinement: Introducing Car Sensors

7.1 Introduction

The Sensors. In this refinement, we introduce the sensors, which are devices capable of detecting the physical presence of cars entering or leaving the bridge. We remind the reader that such sensors are situated on each side of the road and at both extremities of the bridge. This is indicated on Fig. 8.

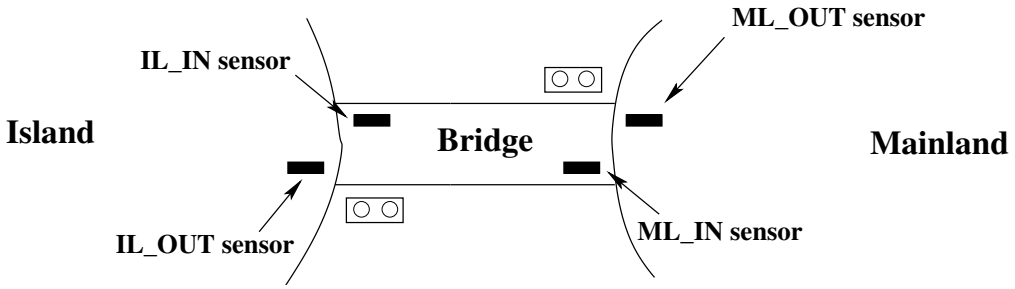


Fig. 8. The Bridge Control Equipment

Closed Model of the Controller and its Environment. The presence of the sensor must now make clearer the separation between our future *software controller* and its *physical environment*, which was mentioned at the beginning of section 2. This can be sketched as indicated in the diagram of Fig. 9

As can be seen, the software controller is equipped with two sets of *channels*: output channels connecting the controller to the traffic lights and input channels connecting the sensors to the software controller.

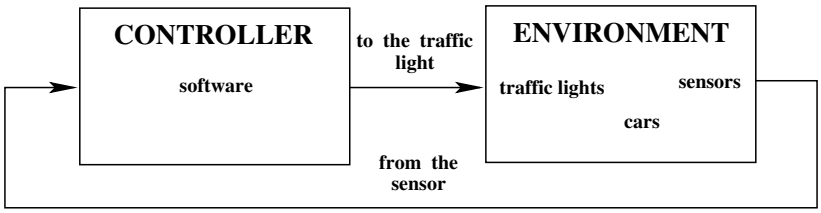


Fig. 9. Controller and Environment

Our intention is to build now a *closed model* corresponding to a complete mathematical simulation of the pair formed by the software controller and its environment. The reason for building such a model is that we want to be sure that the controller works in perfect harmony with the environment, provided, of course, the latter obeys a number of assumptions which have to be made completely clear (this will be made precise in section 7.2). We would like to review now the variables which help us constructing the models of the various constituents of our closed model.

Controller Variables. The model of the software controller has a number of variables which we have already encountered: variables a, b, and c denoting the number of cars on the bridge (a and c) and on the island (b) and two boolean variables il_pass and ml_pass which were introduced in the previous section. What is important to understand here is that variables a, b, and c do not correspond exactly to the physical numbers of cars on the bridge and on the island, which we shall introduce in the next sub-section. In fact, the controller is always working with an approximate picture of the environment, but we want to *prove* that, despite of that, it is able to control the environment in a correct fashion: this is the heart of the modelling process.

Environment Variables. The environment is formalized by means of four variables corresponding to the state of the sensors. More precisely, 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. Notice that we use upper case letters to name these variables, this is to remember that they are *physical variables* denoting objects of the real world. We shall also introduce three variables A, B, and C denoting the *physical* number of cars on the bridge going to the island (A), on the island (B), and on the bridge going to the mainland (C).

Output Channels. Now we have to explain how the controller and the environment communicate. We have already introduced variables ml_tl and il_tl: they corresponds to the output channels from the controller to the environment. To simplify matters and as an abstraction, we suppose that the physical traffic lights are also directly represented by these variables. It is an abstraction as we can imagine that there is a (very) slight delay between the controller changing one of these channels color and the real change occurring on the physical traffic light, but we consider the corresponding delay so small that we can take it to be equal to zero.

Input Channels. It remains now for us to explain how the sensors communicate with the controller. We are not interested in the precise technology used in the sensors, only in their external behavior. As said above, a sensor can be in two different states: either "on" or "off". When the state of a sensor moves from "off" to"on", it means that a car has just been detected as arriving on it: nothing has to be sent to the controller in this case. Note that the state of a sensor can remain "on" for a certain time when the car has

to wait because the associated traffic light is red. When the state of a sensor moves from "on" to"off", it means that a car, which was on it, has just left it. In that case, a message has to be sent to the controller. All this is illustrated in the diagram of Fig. 10

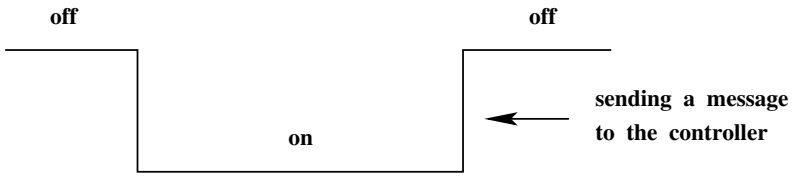


Fig. 10. Controller and Environment

We thus introduce four input channel variables corresponding to the different sensors: ml_out_10, ml_in_10, il_in_10, and il_out_10.

Summary. Here is a summary of the different kinds of variables we have just presented:

Input Channels	$ml\_out\_10, ml\_in\_10, il\_in\_10, il\_out\_10$
Controller	$a, b, c, ml\_pass, il\_pass$
Output Channels	ml_tl, il_tl
Environment	$A, B, C, ML\_OUT\_SR, ML\_IN\_SR, IL\_OUT\_SR, IL\_IN\_SR$

The diagram of Fig. 11 shows the various categories of variables of our closed system. In principle, the input channel variables are set by the environment and tested by the controller. Likewise, the output channel variables are set by the controller and tested by the environment. But we shall explain at the end of section 7.3 that in this example there will be an exception to these rules. On the other hand, the controller variables are set and tested by the controller only while the environment variables are set and tested by the environment only. There is no exception to these rules

7.2 Refining the state

We first introduce an additional carrier set defining the various states ("on" and "off") of a sensor. This is done as follows:

sets:	$\ldots, SENSOR$
constants:	$\ldots, on, off$

axm3_1:	$SENSOR = \{on, off\}$
axm3_2:	$on \neq off$

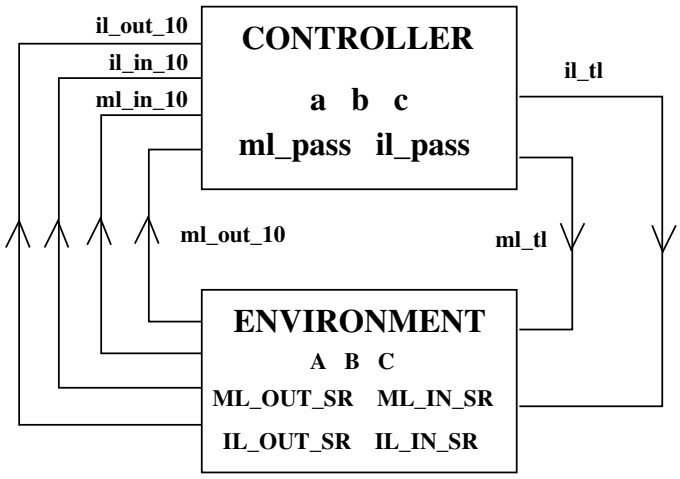


Fig. 11. Controller, Environment, and their Variables

Here are the new variables together with their basic typing invariants **inv3_1** to **inv3_11**:

$$\begin{array}{ll} \textbf{inv3\_1}: & ML\_OUT\_SR \in SENSOR \\ \textbf{inv3\_2}: & ML\_IN\_SR \in SENSOR \\ \textbf{inv3\_3}: & IL\_OUT\_SR \in SENSOR \\ \textbf{inv3\_4}: & IL\_IN\_SR \in SENSOR \end{array}$$

$$\begin{array}{ll} \textbf{inv3\_5}: & A \in \mathbb{N} \\ \textbf{inv3\_6}: & B \in \mathbb{N} \\ \textbf{inv3\_7}: & C \in \mathbb{N} \\ \textbf{inv3\_8}: & ml\_out\_10 \in \text{BOOL} \\ \textbf{inv3\_9}: & ml\_in\_10 \in \text{BOOL} \\ \textbf{inv3\_10}: & il\_out\_10 \in \text{BOOL} \\ \textbf{inv3\_11}: & il\_in\_10 \in \text{BOOL} \end{array}$$

We are now going to state the more interesting invariants concerned with these variables. First, we have an invariant stating that when the sensor IL_IN_SR is on, then A is positive. In other words, there is at least one physical 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:

$$\begin{array}{lll} \textbf{inv3\_12}: & IL\_IN\_SR = on & \Rightarrow \quad A > 0 \\ \textbf{inv3\_13}: & IL\_OUT\_SR = on & \Rightarrow \quad B > 0 \\ \textbf{inv3\_14}: & ML\_IN\_SR = on & \Rightarrow \quad C > 0 \end{array}$$

Second, when input channel ml_out_10 is TRUE, it means that a car has just left the sensor ML_OUT_SR. For this to be possible the mainland traffic light must be green. This invariant formalizes the fact that car

drivers obey the traffic light indications. We have a similar case with input channel il_out_10. This is formalized by means of the following two invariants:

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

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

Our next group of invariants is dealing with the relationship between the sensor status and the messages sent to the controller. They say that no message is on an input channel when a car is on the corresponding sensor. Here are these invariants:

$$\textbf{inv3\_17}: \quad IL\_IN\_SR = on \;\Rightarrow\; il\_in\_10 = \text{FALSE}$$

$$\textbf{inv3\_18}: \quad IL\_OUT\_SR = on \;\Rightarrow\; il\_out\_10 = \text{FALSE}$$

$$\textbf{inv3\_19}: \quad ML\_IN\_SR = on \;\Rightarrow\; ml\_in\_10 = \text{FALSE}$$

$$\textbf{inv3\_20}: \quad ML\_OUT\_SR = on \;\Rightarrow\; ml\_out\_10 = \text{FALSE}$$

These invariants state that when a car is on a sensor then the previous message coming from that sensor has been treated by the controller. There are two possible interpretations for these invariants:

(A) Cars must wait before touching a sensor until the controller is ready,

(B) The controller is fast enough so that it is always ready for the next car.

Obviously, (A) is not acceptable. So, we postulate choice (B). Should it not be the case, then the controller could miss some cars entering or leaving the system. In other words, *this assumption has to be checked when installing the system*. In fact, it corresponds to a requirement which is obviously missing in our requirement document:

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

Our next series of invariants is dealing with the relationship that exists between the physical number of cars (A, B, and C) and the corresponding numbers dealt with by the controller (a, b, and c) :

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

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

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

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

These invariants are easy to understand. When, say, $il\_in\_10 = \text{TRUE}$, it means that a car has left the bridge to enter the island, but the controller does not know it yet: thus A is incremented and B in decremented while a and b are left unchanged. Likewise, when $ml\_out\_10 = \text{TRUE}$, it means that a new car has entered the bridge coming from the mainland, but the controller does not know it yet: thus A is incremented while a is left unchanged. We have similar invariants dealing with B and b and with C and c as shown below:

$$\begin{array}{ll}
\textbf{inv3_25}: & il_in_10 = \text{TRUE} \ \wedge \ il_out_10 = \text{TRUE} \ \Rightarrow \ B = b \\
\textbf{inv3_26}: & il_in_10 = \text{TRUE} \ \wedge \ il_out_10 = \text{FALSE} \ \Rightarrow \ B = b + 1 \\
\textbf{inv3_27}: & il_in_10 = \text{FALSE} \ \wedge \ il_out_10 = \text{TRUE} \ \Rightarrow \ B = b - 1 \\
\textbf{inv3_28}: & il_in_10 = \text{FALSE} \ \wedge \ il_out_10 = \text{FALSE} \ \Rightarrow \ B = b
\end{array}$$

$$\begin{array}{ll}
\textbf{inv3_29}: & il_out_10 = \text{TRUE} \ \wedge \ ml_in_10 = \text{TRUE} \ \Rightarrow \ C = c \\
\textbf{inv3_30}: & il_out_10 = \text{TRUE} \ \wedge \ ml_in_10 = \text{FALSE} \ \Rightarrow \ C = c + 1 \\
\textbf{inv3_31}: & il_out_10 = \text{FALSE} \ \wedge \ ml_in_10 = \text{TRUE} \ \Rightarrow \ C = c - 1 \\
\textbf{inv3_32}: & il_out_10 = \text{FALSE} \ \wedge \ ml_in_10 = \text{FALSE} \ \Rightarrow \ C = c
\end{array}$$

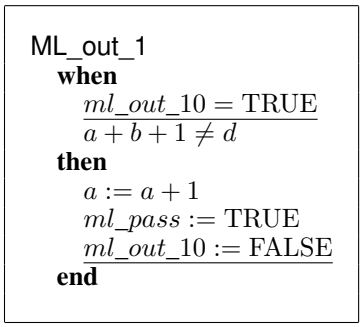
The last two, and probably most important, invariants in this refinement are the ones which say that the two main properties (one way bridge and limited number of cars) *hold for the physical number of cars*.

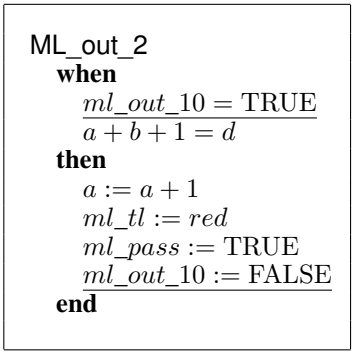
$$\begin{array}{ll}
\textbf{inv3_33}: & A = 0 \ \vee \ C = 0 \\
\textbf{inv3_34}: & A + B + C \leq d
\end{array}$$

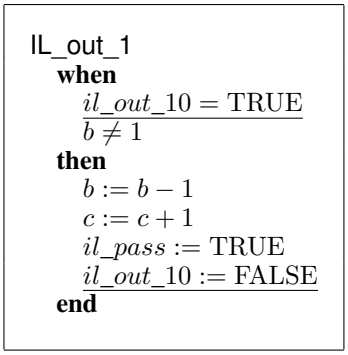
In other words, the controller although working with slightly time-shifted information concerning A, B and C (the controller bases its decision on a, b, and c), nevertheless maintain the basic properties on the physical numbers of cars A, B, and C.

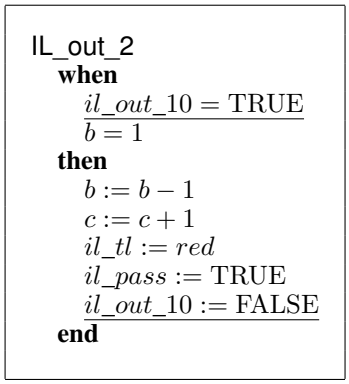
7.3 Refining Abstract Events in the Controller

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

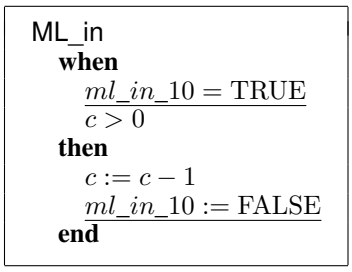


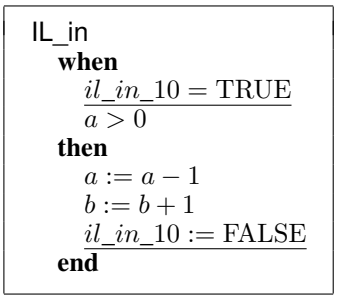






Notice that in their abstract versions these events were testing the green status of the corresponding traffic lights. In these refined version, it is not necessary any more since these events are now triggered by the input channels ml_out_10 or il_out_10 which ensure through invariants **inv3_15** and **inv3_16** that the corresponding lights are green.





In the 6 previous events, which are triggered by the input channels, you can see that the channels in question are all reset by the events. This is to indicate that the corresponding controller operation has finished. Events xxx_arr in the next section will test these resetting in their guards so as to "allow" another car to occupy the relevant sensor. This interplay is a formal way to express the rapid reaction of the controller, running faster than cars may arrive!

ML_tl_green
when
$ml\_tl = red$
$a + b < d$
$c = 0$
$il\_pass = \text{TRUE}$
$il\_out\_10 = \text{FALSE}$
then
$ml\_tl := green$
$il\_tl := red$
$ml\_pass := \text{FALSE}$
end

IL_tl_green
when
$il\_tl = red$
$a = 0$
$ml\_pass = \text{TRUE}$
$ml\_out\_10 = \text{FALSE}$
then
$il\_tl := green$
$ml\_tl := red$
$il\_pass := \text{FALSE}$
end

The new guard $il\_out\_10 = \text{FALSE}$ in event ML_tl_green is indispensable to maintain invariant **inv3_16**, that is

inv3_16 : $il\_out\_10 = \text{TRUE} \Rightarrow il\_tl = green$

This is so because il_tl is set to red in event ML_tl_green. We have a similar guard ($ml\_out\_10 = \text{FALSE}$) in event IL_tl_green: it is necessary in order to maintain invariant **inv3_15**.

It would be possible to add others guards in the two previous event. The idea would be to turn a light to green only if there is a car willing to pass. In order to do so, one would have to make the two sensors, which are situated close to the traffic lights, sending an additional information when a car is coming onto them. We leave it to the reader to make this extension of the system.

7.4 Adding New Events in the Environment

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

ML_out_arr
when
$ML\_OUT\_SR = off$
$ml\_out\_10 = \text{FALSE}$
then
$ML\_OUT\_SR := on$
end

ML_in_arr
when
$ML\_IN\_SR = off$
$ml\_in\_10 = \text{FALSE}$
$C > 0$
then
$ML\_IN\_SR := on$
end

IL_in_arr
when
$IL\_IN\_SR = off$
$il\_in\_10 = \text{FALSE}$
$A > 0$
then
$IL\_IN\_SR := on$
end

IL_out_arr
when
$IL\_OUT\_SR = off$
$il\_out\_10 = \text{FALSE}$
$B > 0$
then
$IL\_OUT\_SR := on$
end

In each case, we suppose that the previous message has been treated: the input channels are all tested for FALSE. Moreover, the physical number of cars is tested as expected. It expresses the fact that the setting to "on" of a sensor is due to the presence of cars. This is compatible with our requirement ENV-5 saying that the sensors are used to detect the presence of a car entering or leaving the bridge. We finally have four events corresponding to a car leaving a sensor.:

ML_out_dep
when
$ML\_OUT\_SR = on$
$ml\_tl = green$
then
$ML\_OUT\_SR := off$
$ml\_out\_10 := \text{TRUE}$
end

ML_in_dep
when
$ML\_IN\_SR = on$
then
$ML\_IN\_SR := off$
$ml\_in\_10 := \text{TRUE}$
$C = C - 1$
end

IL_in_dep
when
$IL\_IN\_SR = on$
then
$IL\_IN\_SR := off$
$il\_in\_10 := \text{TRUE}$
$A = A - 1$
$B = B + 1$
end

IL_out_dep
when
$IL\_OUT\_SR = on$
$il\_tl = green$
then
$IL\_OUT\_SR := off$
$il\_out\_10 := \text{TRUE}$
$B = B - 1$
$C = C + 1$
end

It is important to notice that a car leaving the mainland out-sensor can do so provided the corresponding traffic light is green. Likewise, a car leaving the island out-sensor can do so provided the corresponding traffic light is green. Here we take into account requirement ENV-3 saying that "cars are not supposed to pass on a red traffic light, only on a green one". It is also possible to see that in each case a message is sent to the controller. Finally the physical number of cars are modified as expected: we simulate what happens in the environment.

7.5 Convergence of the New Events

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

variant_3: $12 - (ML\_OUT\_SR + ML\_IN\_SR + IL\_OUT\_SR + IL\_IN\_SR + 2 * (ml\_out\_10 + ml\_in\_10 + il\_out\_10 + il\_in\_10))$

Notice that, as for **variant_2**, the previous variant is not correct. On has to convert the boolean or sensor expressions to numerical expressions. We leave this to the reader.

7.6 No Deadlock

We leave it to the reader to prove that this third refinement does not deadlock.