Report

How to interpret failed proofs in Event-B

Author(s):
Hoang, Thai Son

Publication Date:
2010

Permanent Link:
https://doi.org/10.3929/ethz-a-006857374

Rights / License:
In Copyright - Non-Commercial Use Permitted

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information please consult the Terms of use.
How to Interpret Failed Proofs in Event-B *

Thai Son Hoang

Department of Computer Science,
Swiss Federal Institute of Technology Zurich (ETH-Zurich),
CH-8092, Zurich, Switzerland
htson@inf.ethz.ch

Abstract. In formal reasoning, modelling and proving activities are closely related. Models give rise to different proof obligations and information about failed proofs gives indications on how models should be improved. This document is an attempt to address the latter issue: to understand how to deal with unprovable obligations. We consider here proof obligations related to invariant preservation of an Event-B model: firstly, to understand the meaning of the proof obligations; secondly, to analyse various ways to fix the model accordingly. Our analysis is based on the concept of reachable states and inductive invariants.

Keywords: Event-B, invariant, inductive invariant, proof obligations, modelling and proving.

1 Introduction

Event-B [1] is a formal modelling method for developing systems via step-wise refinement, based on first-order logic. The strength of the method is enhanced by its supporting RODIN Platform for analysing and reasoning about Event-B models rigorously. Developing in Event-B using the RODIN Platform usually involves several iterations between modelling and proving. On the one hand, models give rise to proof obligations as the input for the proving process. On the other hands, information from proofs, in particular from failed proofs, acts as input for the modelling process: suggestions for improving the models. Here, what we mean by failed proofs are unprovable obligations, rather than proof attempts failed because of the incompleteness of the provers.

This document is an attempt have a closer look at the interaction between modelling and proving activities in the latter direction: How unprovable obligations influence the modelling activities. We focus in this paper on invariant preservation proof obligations for an Event-B machine. Our motivation starts from questions by developers arose in the case of failed proofs in Event-B such as “What is the meaning of this failed proof? I checked and I am sure that my model certainly does not violate this property.” or “How should I fix my model in this case? Should I add a new invariant or a new guard to my model?”. Our analysis in this paper is based on the notion of reachable states and the distinction between invariants and inductive invariants.

* Part of this research was carried out within the European Commission ICT project 214158 DEPLOY, http://www.deploy-project.eu/index.html. We thank Jean-Raymond Abrial, Andreas Fürst and Matthias Schmalz for their comments on drafts of this paper.
We first illustrate the meaning of invariant preservation proof obligations, and in passing by, we make a clear distinction between properties, i.e. some statements (predicates) constraining the state variables; invariants, i.e. properties hold for all reachable states; and inductive invariants, i.e. properties that are proved to be invariants by discharging our proof obligations. Secondly, we analyse the possible causes for failed proofs and give illustrations of different “fixes” in the model. In general, there is no definite answer to the question “How should I fix my model?”. In most cases, the answer strongly depends on the problem at hand. What we try to do with this paper is to give the developer a sequence of questions to think about in dealing with failed proofs.

The rest of the document is structured as follows. In Section 2 we give an overview of Event-B modelling method focusing on invariant preservation proof obligations. Section 3 sets our motivation by two small examples. Event-B modelling elements and changes of these elements are illustrated in Section 4. In Section 5, we look at various cases of failed proofs for invariant preservation and give suggestions how the model could be improved in order to fix the problem. We conclude in Section 6 with a step-by-step check-list for the developers.

2 The Event-B Modelling Method

A development in Event-B [2] is a set of formal models. The models are built from expressions in a mathematical language, which are stored in a repository. When presenting our models, we will do so in a pretty-printed form e.g., adding keywords and following layout conventions to aid parsing. Event-B has a semantics based on transition systems and simulation between such systems, described in [1]. We will not describe in detail the Event-B semantics here and instead just illustrate some of the proof obligations that are important for our explanation.

Event-B models are organised in terms of the two basic constructs: contexts and machines. Contexts specify the static part of a model whereas machines specify the dynamic part. Contexts may contain carrier sets, constants, axioms, and theorems. Carrier sets are similar to types [2]. Axioms constrain carrier sets and constants, whereas theorems express properties derivable from axioms. In the following, we further describe machines.

2.1 Machines

Machines specify behavioural properties of Event-B models. Machines may contain variables, invariants, events. Variables v define the state of a machine. They are constrained by invariants \( I(v) \). Possible state changes are described by events. Each event is composed of a guard \( G(t, v) \) (the conjunction of one or more predicates) and an action \( S(t, v) \), where \( t \) are the parameters of the event.\(^1\) The guard states the necessary condition under which an event may occur, and the action describes how the state variables evolve when the event occurs. An event can be represented by the term

\[ v_{new} = S(t, v_{old}) \]

\(^1\) When referring to variables \( v \) and parameters \( t \), we usually allow for multiple variables and parameters, i.e., they may be “vectors”. 

“any $t$ where $G(t, v)$ then $S(t, v)$ end”. A dedicated event of the last form “begin $S(v)$ end” is used for the initialisation init.

The action of an event is composed of one or more assignments of the form

\[
\begin{align*}
x & := E(t, v) \quad \text{(1)} \\
x & \in E(t, v) \quad \text{(2)} \\
x & | Q(t, v, x') \quad \text{(3)}
\end{align*}
\]

where $x$ is a variable contained in $v$, $E(t, v)$ is an expression, and $Q(t, v, x')$ is a predicate. Assignments of the form (1) are deterministic, whereas the other two forms are nondeterministic. In (2), $x$ (which must be a single variable) is assigned an element of a set. In (3), $Q$ is a “before-after predicate”, which relates the values $x$ (before the action) and $x'$ (afterwards). (3) is the most general form of assignment and nondeterministically selects an after-state $x'$ satisfying $Q$ and assigns it to $x$. Variables other than $x$ are unchanged by the above assignments. There is also a side condition on the action of an event: the variables on the left-hand side of the assignments contained in the action must be disjoint.

Proof obligations serve to verify certain properties of machines. We only describe the proof obligation for invariant preservation. Formal definitions of all proof obligations are given in [1]. We focus in this document on the following proof obligations: “invariant preservation” and “invariant establishment”.

Invariant preservation states that invariants hold whenever variables change their values. Obviously, this does not hold a priori for any combination of events and invariants and therefore must be proved. For each event and each machine invariant, we must prove that the invariant is re-established after the event is carried out. More precisely, under the assumption of the invariants and the event’s guard, we must prove that the invariant still holds in any possible state after the event’s execution.

Similar proof obligations are associated with a machine’s initialisation event init called invariant establishment. The only difference is that there is no assumption that the invariant holds.

3 Motivation

In this section, we illustrate our motivation with two examples of failed proofs.

3.1 The First Example

We consider the following Event-B model.

\[
\text{variables: } n \quad \text{invariants: } \begin{cases} \text{inv1: } n \leq 2 \end{cases}
\]
In the above model, the value of variable \( n \) is initially 0 and could be decreased infinitely by event \text{dec}. However, \( n \) can be increased by 1 only when its value is not 1. As a result, its value will never be greater than 1. Intuitively, for all the reachable states of the model, invariant \text{inv1}, i.e. \( n \leq 2 \) holds.

We consider here the proof obligation \text{inc/inv1/INV} for proving that event \text{inv} preserves invariant \text{inv1}.

\[
\begin{array}{c|c}
\text{n} & \text{inc/inv1/INV} \\
\hline
\leq 2 & \\
\neq 1 & \\
\hline
\implies & n + 1 \leq 2
\end{array}
\]

The obligation is not provable with a counter example where \( n = 2 \). There are several ways to “fix” the model, amongst them are invariant strengthening and guard strengthening.

**Invariant strengthening** We can strengthen the invariant, for example add a new invariant \text{inv2} to \( n \neq 2 \).

**Guard strengthening** We can add a guard to the event \text{inc}, for example \( n \neq 2 \).

Which of the above two alternatives is the “right” choice to take in this situation? The quick answer is invariant strengthening and we will justify our answer in later sections.

### 3.2 The Second Example

Our next example is a continuation of the previous example with the invariant 1 “corrected” as \( n \leq 1 \). We want to add a new invariant \text{inv2} stating that \( 0 \leq n \). The model is as follows.

\[
\begin{array}{c}
\text{variables: } n \\
\hline
\text{invariants: } \text{inv1} \quad n \leq 1 \\
& \text{inv2} \quad 0 \leq n
\end{array}
\]
We consider the proof obligation $\text{dec/inv2}/\text{INV}$ for proving that event dec preserves invariant $\text{inv2}$.

$$
\begin{align*}
\quad & n \leq 1 \\
\quad & 0 \leq n \\
\vdash & 0 \leq n - 1
\end{align*}
$$

The above obligation is unprovable with a counter example $n = 0$. Again, there are several ways to “fix” the above model.

**Invariant weakening** Weaken the invariant, for example, dropping the invariant $\text{inv2}$.

**Guard strengthening** Strengthen the guard of the event dec, for example by adding a guard $n \neq 0$.

**Action changing** Changing the action of the event dec, for example by replacing the action by $n := n \div 2$.

Once again, the question here is which of the above alternatives are the “right” choice to take in this situation. We will address this question in the later sections.

## 4 Illustration of Modelling Elements and Proof Obligations

In this section, we will give the illustration of Event-B modelling elements and the proof obligations for invariant preservation using various diagrams. We also give the illustration of several changes to different modelling elements. We consider a general Event-B model with variable $v$, invariant $I(v)$, the initialisation init and events of the form evt.

### 4.1 Invariant

The invariant $I(v)$ corresponds to a set of states satisfying the invariant. Intuitively, when the invariant is weakened, this set is enlarged and conversely, when the invariant is strengthened, the set is contracted. This is illustrated in Figure 1.
4.2 Events

For each event \( \text{evt} \), the guard \( G(x, v) \) specifies a set of state the event is enable and the action \( v : S_i(x, v, v') \) specifies the relationship between the before-after states. This is illustrated in Fig. 2a. Non-deterministic corresponding to different values of the parameter \( x \) and/or different values of the after value \( v' \). This is represented in Fig. 2a by having more than one arrows going out from the same state.

We are going to consider the following possible modifications of events:
Guard strengthening The guard \( G \) of the event is strengthen, hence it contracted. As a result, some of the original transition is removed, since the starting states is no longer satisfy the guard. This is illustrated in Figure 2b.

Action changing As a result of changing the event action, some transitions can be changed. This is illustrated in Figure 2c.

Non-determinism reducing Reducing the non-determinism of the event removes some transitions corresponding to the event. This is illustrated in Figure 2d.

The initialisation is a special event with no “before-state”, it specifies a set of initial states corresponding to the after predicate \( K \).

\[
\begin{array}{c}
\text{iK holds}
\end{array}
\]

Fig. 3. Initialisation

4.3 Reachable states

Each Event-B model describes a set of reachable states. This is recursively defined as follows.

– Any initial state is a reachable state.
– If \( s \) is a reachable state, and there exists a transition from state \( s \) to state \( t \) by one of the events \( \text{evt} \), then \( t \) is also a reachable state.

The reachable states are illustrated in Figure 4. A run of an Event-B system corresponds to a path within the set of reachable states starting from an initial state. Note that by definition the set of initial states must be a sub-set of the set of reachable states.

Fig. 4. Initial states and reachable states
4.4 Invariants and Inductive Invariants

$I$ is an invariant of a system if the invariant $I$ hold for all the reachable states of the system. Hence illustratively, the set of reachable states is included inside the set of states where the invariant $I$ holds. This is illustrated in Figure 5. Ideally, we want these two sets as close together as possible.

![Invariant](Diagram1.png)

**Fig. 5. Invariant**

![Invariant preservation](Diagram2.png)

**Fig. 6. Inductive invariant**

An *inductive invariant* for an Event-B system satisfies the following conditions.

1. The invariant $I$ holds in any initial state. We call this condition *invariant establishment*. This is illustrated in Figure 6a.
2. For each event $\text{evt}_i$, if $I$ holds in the before state then $I$ holds again in the after state. We call this condition \textit{invariant preservation}. This is illustrated in Figure 6b.

Note that \textit{inductive invariant} is an invariant, i.e. it holds for all reachable states of the system, but not all invariants are \textit{inductive invariant}. As an example, the original invariant $\text{inv1}$ in the first example in Section 3.1, i.e. $n \leq 2$, is an invariant but it is not inductive, since we cannot prove that event inc preserves this invariant.

5 \hspace{0.5em} \textbf{Failed Proofs}

In this section, we are going to look at different failed proofs for invariant preservation and establishment. We first consider invariant establishment proof obligations (Section 5.1), follow by invariant preservation proof obligations (Section 5.2).

5.1 \hspace{0.5em} \textbf{Invariant Establishment}

A failed invariant establishment proof corresponding to the case where there exists an initial state where the invariant $I$ does not hold. This means that the set of initialising states is not entirely contained in the set of states satisfying the invariant $I$ (Figure 7).

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig7.png}
\caption{Failed invariants establishment}
\end{figure}

There are two reasons for this failed proof: the invariant is too strong, or the initialisation is incorrect. The judgement for which is the real reason depend on the actual modelling intention of the developer. Consider the following example in which the model has only one variable $n$.

\begin{verbatim}
init
begin
  n :\in \{0, 1\}
end
\end{verbatim}

\begin{verbatim}
invariants:
  inv1  n < 1
\end{verbatim}
In this case, the initialisation does not establish the invariant. One interpretation could be that the invariant is too strong and what the intention of the developer is that \( n \leq 1 \) rather than \( n < 1 \) (in general, the invariant could be totally wrong and need to be corrected). Another interpretation is that the developer intends to have a system where \( n \) is not positive and give the possible initial value 1 for the variable \( n \) by mistake. In this situation, the initialisation can be corrected to \( n := 0 \). Here the decision on why the proof fails depends on the problem at hand. We are going to illustrate about the possible fix depending on the reason of failures (which should be decided by the developers).

![Diagram](image)

(a) Invariant weakened  

(b) \( \text{init corrected} \)

**Fig. 8. Fixing a failed invariant establishment obligation**

**The invariant is too strong** In this situation, the invariant need to be weaken until the set of states satisfying the invariant contains the set of initial states. This is illustrated in Figure 8a. A more general case is that the invariant is incorrect and need to be corrected.

**The initialisation is incorrect** In this case, the initialisation need to be corrected so that it is included in the set of states satisfying the invariant. This is illustrated in Figure 8b.

5.2 Invariant Preservation

A failed invariant preservation proof corresponding to the case where there exists an event transition starting from a state satisfying both the invariant \( I \) and the guard \( G \) but ending in an after state not satisfying the invariant \( I \). This situation is illustrated in Figure 9.
In this situation, the $I$ cannot be an inductive invariant of system, and we consider two different cases: $I$ is an invariant (even though not inductive) and $I$ is not even an invariant of the system. The first case occurs if the before state of erroneous transition does not belong to the set of reachable state, and the second case occurs this belongs to the set of reachable state.

**Unreachable State** This situation is illustrated in Figure 10. In this case $I$ is an invariant of the system, i.e. all reachable states satisfy $I$, but $I$ is not inductive.

We show an example of this situation earlier in our first example in Section 3.1. Amongst the possible fixes to the system, i.e. invariant strengthening or guard strengthening, we choose the former solution. We need to strengthen the invariant enough so
that the before-state of the erroneous transition no longer satisfies the invariant. This is illustrated in Figure 11. Our arguments to favour this solution is as follows.

- The proof obligation is now provable. This is because when the invariant is strengthen, the before-state of the erroneous transition could be no longer satisfied the invariant, hence we do not need to consider these case for invariant preservation proof obligation.
- We have a stronger (hence better) invariant properties about our system. Intuitively, by strengthening the invariant, we make the set of reachable states and set of states satisfying the invariant closer to each other.
- We do not change the “execution” behaviour of our system, e.g. the events of the system. In particular, on the one hand, since invariants are properties that are “proved” for the system, they do not add any extra amount of work to implementation. On the other hand, guards are “implementation details” and it is the responsibility of the implementer of the system to ensure that when the assignments of an event are carried out, the guards of that event are satisfied.

Reachable State  We are now going to consider the case where the before-state of the erroneous transition belong to the set of reachable states. In this case, \(I\) is not an invariant: it does not hold for some reachable state, namely, the after-state of the erroneous transition. This is illustrated in Figure 12.

There could be several reasons for this failure and the interpretation depends on the developers of the system with the problem at hand. We present below possible fixes to the model for this problem.

Invariant weakening  In this case, the proposed \(I\) is too strong hence it needs to be weaken in order to cover all the reachable states (see Figure 13a). We showed an
example of this case in Section 3.2 on page 4. In this situation, the developer decided that the proposed invariant \( \text{inv}_2 \), i.e. \( 0 \leq n \) is too strong, hence could be dropped. In more general situations, the invariant might need to be corrected (not necessarily only weakening).

**Guard strengthening** In this case, the guard of the event is too weak for preventing the invariant to be violated. In the example in Section 3.2, one interpretation is that the event \( \text{dec} \) should prevent the value of variable \( n \) to be decreased bellow 0 (as constraint by \( 0 \leq n \)). Note that by strengthen the guard, we remove the possible erroneous transition for the event and also reduce the set of reachable states to be contained within the set of states satisfying \( I \). This is illustrated in Figure 13b.

**Action changing** Another interpretation of this failed proof obligation is that there is a mistake in the action of the event. As showed in the example in Section 3.2, the action of the event can be changed so that the proof obligation can be proved. In this case, as a consequence of changing the action, the set of reachable states is contracted to be included within the set of states satisfying \( I \). This is illustrated in Figure 13c.

**Event non-determinism reducing** To fix the problem of the failed proof, we can reduce the non-determinism of the event to remove the erroneous transition, and a consequence to contract the set of reachable states to satisfy the invariant \( I \). This is illustrated in Figure 13d.

## 6 Conclusions

In summary, we presented in this document an analysis about failures in proving invariant preservation obligations, with illustration about the proof obligations, the failed proofs and various fixes to the models. We consider here those failed proof attempts because the corresponding obligations is unprovable, in contrast with those failed proof attempts because of the weakness of the theorem provers or because of the background theory is too weak (these problems need to have different treatments).
In passing by, we use the notion of reachable states and make a clear distinction between invariants, i.e. properties hold for all reachable state, and inductive invariants, i.e. hold initially and preserved by all transitions. An important relationship between the two types of properties is that inductive invariants are always invariants, whereas not all invariants are inductive. This is also the distinction between approach such as model checker and theorem proving (by induction). In general, model checking, by exploring all the reachable states and checking for invariant violations, does not distinguish if an invariant is inductive or not. As showed in this document, theorem proving does distinguish between the two type of properties: an invariant successfully proved by our proof obligations is an inductive invariant.

The reason why this distinction is important for us is that during the development many proposed properties are invariants but seldom inductive. This is also the answer...
for the first question mentioned earlier in Section 1: The property might hold for all reachable state (hence invariant) but its proofs failed because the property is not inductive. In this case, the invariant need to be strengthen to be inductive. Moreover, note that the set of reachable states corresponds to an inductive invariant by definition, hence for every property that is invariant, there exists an inductive invariant which can be used to prove the this property (however finding such inductive invariant might be difficult).

Our analysis strongly based on the notion of reachable states and the ability to know if a state is reachable in our system. In general, this is a difficult question to answer, but using animation/model checking techniques such as AnimB [4], Brama [3], ProB [5] could help to decide this.

In many cases, the fixes to the model for failed proofs depends on the problem at hand. We summarise here the questions that developers need to think about for fixing the model.

**Invariant establishment obligation** This proof obligation ensures that the initialisation establishes the invariant.

- **Reason for failure?**
  - invariant is too strong
  - initialisation is incorrect

- **Weaken the invariant**
- **Correct the initialisation**

**Invariant preservation obligation** This proof obligation ensures that the invariant is preserved by any event transition. We start our analysis with the counter example of the failed proof.

- **Is the counter-example reachable?**
  - no
  - yes
   - Strengthen the invariant
   - Reason for failure?
     - invariant too strong
     - guard too weak
     - action is incorrect
     - too non-deterministic
     - Weaken the invariant
     - Change the action
     - Reduce non-determinism
     - Strengthen the guard

**References**

