How to explain mistakes

Stefan Hallerstede and Michael Leuschel

University of Düsseldorf
Germany
{halstefa,leuschel}@cs.uni-duesseldorf.de

Abstract. Usually we teach formal methods relying for a large part on one kind of reasoning technique about a formal model. For instance, we either use formal proof or we use model-checking. It would appear that it is already hard enough to learn one technique and having to cope with two puts just another burden on the students. This is not our experience. Especially model-checking is easily used to complement formal proof. It only relies on an intuitive operational understanding of a formal model.
In this article we show how using model-checking, animation, and formal proof together can be used to improve understanding of formal models. We demonstrate how animation can help finding an explanation for a failing proof. We also demonstrate where animation or model-checking may not help and where proving may not help. For most part use of another tool pays off. Proof obligations present intentionally a static view of a system so that we focus on abstract properties of a model and not on its behaviour. By contrast model-checking provides a more dynamic view based on an operational interpretation. Both views are valuable aids to reasoning about a model.

1 Introduction

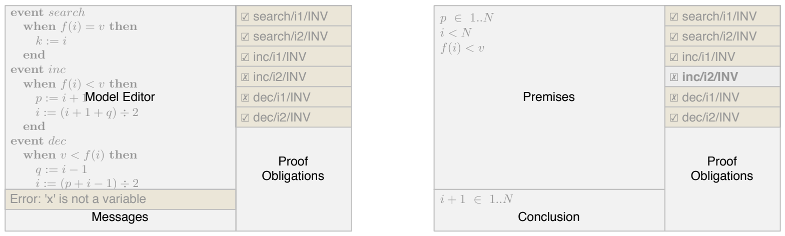
In Event-B [2] formal modelling serves primarily for reasoning: reasoning is an essential part of modelling because it is the key to understanding complex models. Reasoning about complex models should not happen accidentally but needs systematic support within the modelling method. This thinking lies at the heart of the Event-B method.

We use refinement to manage the many details of a complex model. Refinement is seen as a technique to introduce detail gradually at a rate that eases understanding. The model is completed by successive refinements until we are satisfied that the model captures all important requirements and assumptions. In this article we concern ourselves only with what is involved in coming up with an abstract model of some system. Note that refinement can also be used to produce implementations of abstract models, for instance, in terms of a sequential program [1,11]. But this is not discussed in this article.

We present a worked out example that could be used in the beginning of a course on Event-B to help students develop a realistic picture of the use of formal methods. The challenge is to state an example in such a way that it is easy to follow but provides enough opportunity to make (many) mistakes. We chose to use a sized-down variant of the access control model of [2] which we have employed for lectures at ETH Zürich

(Switzerland) and at the University of Southampton (United Kingdom). We have not used the description of a computer program because that is too rigid to exhibit possible misunderstandings. We have tried this using linear and binary search (together) but these are easy to formalise. So we have worked out this example for the next winter semester at the University of Düsseldorf. We begin by stating a problem to be solved in terms of assumptions and requirements and show how the problem can be approached using formal methods. The process of creating the model is shown in [6]. In this article we focus on how to understand mistakes made during modelling and the ensuing corrections. Whereas [6] is mostly about how proof can be used to improve a model, in this article we focus on complementary techniques based on model-checking and animation. These are only hinted at in [6]. In this respect this article can be regarded a sequel to [6]. (But it can be read independently.)

Rodin [3] is an extensible tool for formal modelling in Event-B. It is designed to support an incremental style of modelling where frequent changes are made to a model. Rodin produces proof obligations from Event-B models that subsequently are either proved automatically by an automatic theorem prover or manually by the user of the tool if the automatic theorem prover fails. Rodin is implemented on top of the rich client platform Eclipse [5]. It comes with two default screen layouts that are considered to help making the connection between formal model, proof obligations, and proof. The Eclipse platform supplies everything that is needed to make switching between the two layouts easy. Figure 1 shows a simplified sketch of the two default layouts. The



```
event search
  when f(i) = v then
    k := i
  end
event inc
  when f(i) < v then
    p := i + 1
    i := (i + 1 + q) ÷ 2
  end
event dec
  when v < f(i) then
    q := i − 1
    i := (p + i − 1) ÷ 2
```

Error: 'x' is not a variable

☑ search/i1/INV
☑ search/i2/INV
☑ inc/i1/INV
☒ inc/i2/INV
☒ dec/i1/INV
☑ dec/i2/INV

$p \in 1..N$
$i < N$
$f(i) < v$

$i + 1 \in 1..N$

☑ search/i1/INV
☑ search/i2/INV
☑ inc/i1/INV
☒ **inc/i2/INV**
☒ dec/i1/INV
☑ dec/i2/INV

(a) Modelling layout

(b) Proving layout

Fig. 1: Layouts of Rodin for Modelling and Proving

modelling layout (Figure 1a) provides an area for editing models, one for showing error messages on the bottom, and another for viewing and selecting proof obligations by name on the right. When a proof obligation is selected, the layout is changed to the one shown in Figure 1b. The proof obligation is shown in two areas arranged vertically, the hypothesis on top, the goal below it. On the right hand side the proof obligation names are displayed to permit browsing proof obligations. The tool encourages editing and experimenting with formal models. The focus is on modelling. Technicalities of proof obligation generation of concepts such as substitution are pushed aside. We believe, this is important if the tool is to be used by students having first contact with formal methods.

PROB [7,9] is an animator for B and Event-B built using constraint-solving technology. It incorporates optimisations such as symmetry reduction (see, e.g., [14]) and has been successfully applied to several industrial case studies. PROB is also used at several universities for teaching the B-Method [1,12].[1] In that context, the following features of PROB are relevant:

1. automatic animation. In particular, the tool tries to automatically find suitable values for the constants of a B-model based on constraint solving techniques,
2. consistency checking, i.e., checking whether the invariant of a B model is true in all states reachable from the initial states (called the *state space* of a model),
3. visualisation of counter examples or the full statespace of a model. Several reduction techniques have been implemented to compress large statespaces for visualisation [10].
4. trace refinement checking [8].

Figure 2 shows a layout for ProB animation where the user can inspect the current state and the history of events executed, choose the next event to be executed, and follow state changes in a graphical view of the state space.

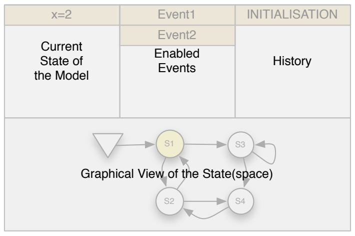

x=2
Event1
INITIALISATION
Event2
Current State of the Model
Enabled Events
History
S1
S3
S2
S4
Graphical View of the State(space)

Fig. 2: Animation Layout of ProB for Event-B Models

We believe the combined use of proof, model-checking, and animation contributes highly to a better understanding of formal models.[2] They make it also easier to approach proof obligations and proof which are particularly hard to master and relate to formal models by novices. The figures shown are edited from the output that is provided by ProB. We have done this to improve readability of this article. The output of ProB is intended for viewing on a computer screen. Still, as a side effect of this exercise we have developed some ideas for improving the output of ProB.

[1] For example, Besançon, Nantes in France; Southampton and Surrey in England; McMaster University, in Canada, Uppsala University in Sweden, and of course Düsseldorf in Germany.

[2] In a companion paper we also investigate the usefulness of graphical visualisation of formal models, which makes animation sequences even easier to comprehend.

Overview. In Section 2 we introduce Event-B. The following sections are devoted to solving a concrete problem in Event-B. In Section 3 the problem is stated. An abstract model is discussed in Section 4. In Section 5 we elaborate the model by refinement.

2 Event-B

Event-B models are described in terms of the two basic constructs: *contexts* and *machines*. Contexts contain the static part of a model whereas machines contain the dynamic part. Contexts may contain *carrier sets*, *constants*, *axioms*, where carrier sets are similar to types [4]. In this article, we simply assume that there is some context and do not mention it explicitly. Machines are presented in Section 2.1, and proof obligations in Section 2.2 and Section 2.3. All proof obligations in this article are presented in the form of sequents: "premises" $\vdash$ "conclusion".

Similarly to our course based on [2], we have reduced the Event-B formalism so that a small subset of the notation suffices and formulas are easier to comprehend. In particular, the relationship between formal model and proof obligations is much easier to exhibit.

2.1 Machines

Machines provide behavioural properties of Event-B models. Machines may contain *variables*, *invariants*, *theorems*, *events*, and *variants*. Variables $v = v_1, \ldots, v_m$ define the state of a machine. They are constrained by invariants $I(v)$.[3] Theorems are predicates that are implied by the invariants. Possible state changes are described by means of events $E(v)$. Each event is composed of a *guard* $G(t, v)$ and an *action* $x := S(t, v)$, where $t = t_1, \ldots, t_r$ are the *parameters* of the event and $x = x_1, \ldots, x_p$ are the variables it may change[4]. 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. We denote an event $E(v)$ by

```
E(v)  ≙  any  t  when                or          E(v)  ≙  begin
           G(t, v)                                          x := S(v)
         then                                             end
           x := S(t, v)
         end                                                          .
```

The short form on the right hand side is used if the event does not have parameters and the guard is true. A dedicated event of the latter form is used for *initialisation*. The action of an event is composed of several *assignments* of the form

$$x_\ell := B_\ell(t, v) \quad ,$$

[3] Given the invariant I over the variables v, $I(t_1, \ldots, t_m)$ can be seen to stand for $I[t_1/v_1, \ldots, t_m/v_m]$. E.g., $I(v) = I$. We use a similar notation for other concepts, such as events, guards and actions.

[4] Note that, as x is a list of variables, $S(t, v)$ is a corresponding list of expressions.

where x_ℓ is a variable and $B_\ell(t, v)$ is an expression. All assignments of an action $x := S(t, v)$ occur simultaneously; variables y that do not appear on the left-hand side of an assignment of an action are not changed by the action, yielding one simultaneous assignment

$$x_1, \ldots, x_p, y_1, \ldots, y_q := B_1(t, v), \ldots, B_p(t, v), y_1, \ldots, y_q \quad , \tag{1}$$

where $x_1, \ldots, x_p, y_1, \ldots, y_q$ are the variables v of the machine. The effect of an action $x := S(t, v)$ of event $E(v)$ is denoted by the formula (1), whereas in the proper model we only specify those variables x_ℓ that may change.

2.2 Machine Consistency

Invariants are supposed to hold whenever variable values change. Obviously, this does not hold a priori for any combination of events and invariants $I(v) = I_1(v) \wedge \ldots \wedge I_i(v)$ and, thus, needs to be proved. The corresponding proof obligations are called *invariant preservation* ($\ell \in 1 \mathinner{..} i$):

$$\begin{array}{ll} & I(v) \\ & G(t, v) \\ \vdash & \\ & I_\ell\,(S(t, v)) \quad , \end{array} \tag{2}$$

for every event $E(v)$. Similar proof obligations are associated with the initialisation event of a machine. The only difference is that neither an invariant nor a guard appears in the premises of proof obligation (2), that is, the only premises are axioms and theorems of the context. We say that a machine is consistent if all events preserve all invariants.

2.3 Machine Refinement

Machine refinement provides a means to introduce more details about the dynamic properties of a model [4]. A machine N can refine at most one other machine M. We call M the *abstract* machine and N a *concrete* machine. The state of the abstract machine is related to the state of the concrete machine by a *gluing invariant* $J(v, w) = J_1(v, w) \wedge \ldots \wedge J_j(v, w)$, where $v = v_1, \ldots, v_m$ are the variables of the abstract machine and $w = w_1, \ldots, w_n$ the variables of the concrete machine.

Each event $E(v)$ of the abstract machine is *refined* by a concrete event $F(w)$. Let abstract event $E(v)$ with parameters $t = t_1, \ldots, t_r$ and concrete event $F(w)$ with parameters $u = u_1, \ldots, u_s$ be

$$\begin{array}{lll} E(v) & \widehat{=} & \mathsf{any}\ \ t\ \ \mathsf{when} \\ & & \quad G(t, v) \\ & & \mathsf{then} \\ & & \quad v := S(t, v) \\ & & \mathsf{end} \end{array} \qquad \text{and} \qquad \begin{array}{lll} F(w) & \widehat{=} & \mathsf{any}\ \ u\ \ \mathsf{when} \\ & & \quad H(u, w) \\ & & \mathsf{with} \\ & & \quad t = W(u) \\ & & \mathsf{then} \\ & & \quad w := T(u, w) \\ & & \mathsf{end} \qquad . \end{array}$$

Informally, concrete event $F(w)$ refines abstract event $E(v)$ if the guard of $F(w)$ is stronger than the guard of $E(v)$, and the gluing invariant $J(v, w)$ establishes a simulation of the action of $F(w)$ by the action of $E(v)$. The term $W(u)$ denotes *witnesses* for the abstract parameters t, specified by the equation $t = W(u)$ in event $F(w)$, linking abstract parameters to concrete parameters. Witnesses describe for each event separately more specific how the refinement is achieved. The corresponding proof obligations for refinement are called *guard strengthening* ($\ell \in 1 \mathrel{..} g$):

$$\begin{array}{l} \quad I(v) \\ \quad J(v,w) \\ \quad H(u,w) \\ \vdash \\ \quad G_\ell\,(W(u),v) \quad , \end{array} \tag{3}$$

with the abstract guard $G(t, v) = G_1(t, v) \land \ldots \land G_g(t, v)$, and (again) *invariant preservation* ($\ell \in 1 \mathrel{..} j$):

$$\begin{array}{l} \quad I(v) \\ \quad J(v,w) \\ \quad H(u,w) \\ \vdash \\ \quad J_\ell\,(S(W(u),v),T(u,w)) \quad . \end{array} \tag{4}$$

Aside: Observe how the witness is used to reduce the complexity of the proof obligation compared to classical B, where a double negation appears in the refinement proof obligation [1]. Indeed, in classical B we have to prove that it is *possible* for the abstract model to make a corresponding step for every concrete step, or equivalently that it is *not* possible for the concrete model to make a step such that the abstract model can *not* imitate it and establish the gluing invariant (hence the double negation). Here, we require simply that for *all* abstract parameters having corresponding concrete parameters which make the witness predicate true, that the abstract event E can be triggered and establishes the gluing invariant. In general, Event-B contains many concepts that have been simplified compared to classical B. As such, it is inherently better suited for teaching formal methods.

2.4 Operational Interpretation

For the purpose of linking between Event-B to animation and model-checking it is convenient to give an operational interpretation to Event-B models [4]. We can observe events occurring and the resulting state changes. No two events may occur simultaneously. For the progress of "execution" resulting from event occurrences there are two possibilities:
(i) Some event guards are true: one of those events must occur.
(ii) All event guards are false: "execution" stops.
Following the informal description we can build a labelled transition system which represents our operational interpretation. We treat events as relations on a state space. The state space of a model is defined as the Cartesian product of the types of each of

the model's variables. For convenience, we assume that every possible value can also be written as a constant expression.[5] A state in the state space is thus a vector of constant expressions describing the values for the variables.

Let $E(v)$ be an event with guard $G(t, v)$ and action $x := S(t, v)$, as defined above in Section 2.1. The events induce a labelled transition relation on states in the state space: state s is related to state s' by event $E(v)$ with parameter values a, denoted by $s \rightarrow^{M}_{E.a} s'$, when $G(a, s)$ holds and s' corresponds to the left-hand side of formula (1), that is to the successor state, with $t = a$ and $v = s$.

The syntactic constraints on the initialisation event in Event-B are such that the outcome of the initialisation will be independent of the initial values of the variables. This means the initialisation has the form

```
begin                                                    (5)
    $v := E$
end
```

where E is a constant expression. The expression E is used to define the initial state for a machine.

Graphically, the state space of an Event-B model looks like in Figure 3. (In general, we would have a set of initial states in a non-deterministic initialisation represented by a predicate P. For convenience we therefore add a special state $root$, where we define $root \rightarrow^{M}_{initialise} s$ if s satisfies the initialisation predicate P. The root is shown here because it is used in the general form of initialisation.)

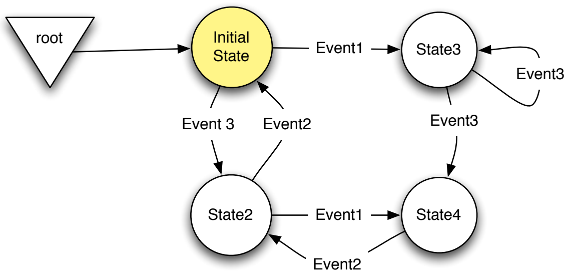


Fig. 3: A simple state space with four states

[5] This is true for booleans, integers, enumerated sets and combinations thereof. It is generally not true for carrier sets; but in that case we can assume that a carrier set is instantiated by an enumerated set just for the purpose of animation and model checking.

3 Problem Statement

In the following sections we develop a simple model of a secure building equipped with access control. The problem statement is inspired by a similar problem used by Abrial [2]. In [6] we have presented how the model could have been produced, especially, making mistakes that could have been made and subsequent improvements to the model. In this article we focus on how to comprehend mistakes made using different views on the model, easing the formal character of modelling. In order to do this we rely heavily on the ProB tool.

The model to be developed is to satisfy the following properties:

P1 : The system consists of persons and one building.
P2 : The building consists of rooms and doors.
P3 : Each person can be at most in one room.
P4 : Each person is authorised to be in certain rooms (but not others).
P5 : Each person is authorised to use certain doors (but not others).
P6 : Each person can only be in a room where the person is authorised to be.
P7 : Each person must be able to leave the building from any room where the person is authorised to be.
P8 : Each person can pass from one room to another if there is a door connecting the two rooms and the person has the proper authorisation.
P9 : Authorisations can be granted and revoked.

Properties P1, P2, P8, and P9 describe environment assumptions whereas properties P3, P4, P5, P6, and P7 describe genuine requirements. It is natural to mix them in the description of the system. Once we start modelling, the distinction becomes important. We have to prove that our model satisfies P3, P4, P5, P6, and P7 assuming we have P1, P2, P8, and P9.

4 Abstract Model

Our aim is to produce a faithful formal model of the system described by the properties P1 to P9 of Section 3. We choose to proceed in two modelling steps:
(i) the *abstract machine* (this section) models room authorisations;
(ii) the *concrete machine* (Section 5) models room and door authorisations.

To create an abstract model we need an abstract way of representing persons and rooms. Using these two concepts we can model property P4 as a relation between persons and rooms and property P3 as a function from persons to rooms. We declare two carrier sets for persons and rooms, $Person$ and $Room$, and a constant $\boldsymbol{O}$, where $\boldsymbol{O} \in Room$. Constant $\boldsymbol{O}$ models the *outside* of the building. We choose to describe the state by two variables for authorised rooms and locations of persons, arm and loc, with invariants

$$inv1 \ : \ arm \in Person \leftrightarrow Room \qquad \text{Property P4}$$
$$inv2 \ : \ Person \times \{\boldsymbol{O}\} \subseteq arm$$
$$inv3 \ : \ loc \in Person \rightarrow Room \qquad \text{Property P3}$$

$inv4 \; : \; loc \subseteq arm$ Property P6

The invariant $inv2$ states that a person is always allowed to be outside, and as such partly formalises P7. These four invariants form the foundation of our abstract model. Next we present the events that model the dynamic aspects of the model.

In this and the next section we encounter mistakes in the model similar to those in [6]. However, we are mostly interested in motivating and explaining mistakes and relating them to the formal model. Novices often have problems when presented with more complicated formulas or proof obligations. A difficulty with formal methods, in general, is that formulas get complicated rather quickly. By choosing different views at the same mistakes we can make them more approachable. For students to get the most out of this exercise, the same software tools that we use to demonstrate formal reasoning should be available to them. This encourages use of the tools and by experimenting with formal models the students can gain a deeper understanding of formal models they create.

In our abstract model to satisfy $inv2$, $inv3$ and $inv4$ we let

```
initialisation
  begin
    $act1 : arm := Person \times \{\boldsymbol{O}\}$
    $act2 : loc := Person \times \{\boldsymbol{O}\}$
  end .
```

We model passage from one room to another by event $pass$,

```
$pass$
  any $p, r$ when
    $grd1 : p \mapsto r \in arm$      $p$ is authorised to be in $r$
    $grd2 : p \mapsto r \notin loc$      but not already in $r$
  then
    $act1 : loc := loc \mathbin{<\!\!+} \{p \mapsto r\}$
  end .
```

Event $pass$ partially models property P8 ignoring doors for the moment. Granting and revoking authorisations for rooms is modelled by the two events

```
$grant$
  any $p, r$ when
    $grd1 : p \in Person$
    $grd2 : r \in Room$
  then
    $act1 : arm := arm \cup \{p \mapsto r\}$
  end

$revoke$
  any $p, r$ when
    $grd1 : p \in Person$
    $grd2 : p \mapsto r \notin loc$
  then
    $act1 : arm := arm \setminus \{p \mapsto r\}$
  end .
```

The two events do not yet model all of P9 which refers to authorisations in general, including authorisations for doors. Events $grant$ and $revoke$ appear easy enough to get right. However, a simple oversight can lead to a mistake. Event $revoke$ violates invariant $inv2$,

$$\begin{array}{lll} & Person \times \{\boldsymbol{O}\} \subseteq arm & \textit{Invariant } inv2 \\ & p \in Person & \textit{Guard } grd1 \\ & p \mapsto r \notin loc & \textit{Guard } grd2 \\ \vdash & & \\ & Person \times \{\boldsymbol{O}\} \subseteq arm \setminus \{p \mapsto r\} & \textit{Modified invariant } inv2 \end{array}$$

We could find out what is wrong only by inspecting the proof obligation. This would require carrying out some proof steps and understanding where it fails. Alternatively we can model-check our abstract model based on the operational interpretation. In an instance of the model with two different rooms $\boldsymbol{I}$ and $\boldsymbol{O}$ and one person P the model-checker yields the counter example in the form of a state trace shown in Figure 4. The

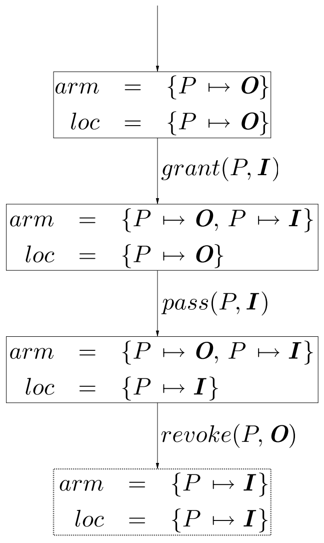

$arm = \{P \mapsto \boldsymbol{O}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$
$grant(P, \boldsymbol{I})$
$arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{I}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$
$pass(P, \boldsymbol{I})$
$arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{I}\}$
$loc = \{P \mapsto \boldsymbol{I}\}$
$revoke(P, \boldsymbol{O})$
$arm = \{P \mapsto \boldsymbol{I}\}$
$loc = \{P \mapsto \boldsymbol{I}\}$

Fig. 4: A state trace leading to an inconsistent state

state

$$arm = \{P \mapsto \boldsymbol{I}, P \mapsto \boldsymbol{O}\}, \quad loc = \{P \mapsto \boldsymbol{I}\}$$

is reachable in three steps. Letting parameter $r = \boldsymbol{O}$ in $revoke$, a state violating $inv2$ is reached. We see that we must not remove $\boldsymbol{O}$ from the set of authorised rooms of any person. To achieve this, we add a third guard to event $revoke$:

$$grd3 \ : \ r \neq \boldsymbol{O} \quad .$$

The counter example provides valuable information based on what the model "does". We can look at event $revoke$ to see what needs to be changed, or better, feed the finding into the proof obligation. With the new information at hand we can see clearly that the conclusion $Person \times \{\boldsymbol{O}\} \subseteq arm \setminus \{p \mapsto r\}$ does not hold if $r = \boldsymbol{O}$.

The model we have obtained thus far is easy to understand. Ignoring the doors in the building, it is quite simple but already incorporates properties P3, P4, and P6. Its simplicity permits us to judge more readily whether the model is reasonable. We can inspect it or animate it and can expect to get a fairly complete picture of its behaviour. We may ask: Is it possible to achieve a state where some person can move around in the building? A simplified state-transition graph can summarise such information comprehensively.

Figure 5 shows a slice of the model constructed by ProB for two persons and two rooms $\boldsymbol{I}$ and $\boldsymbol{O}$ where only the ranges of loc and of arm are considered. The states have

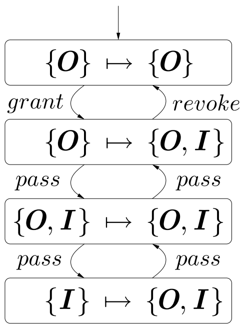


Fig. 5: Transitions on a simpler state space, showing $ran(loc) \mapsto ran(arm)$

been aggregated in sets according to the expression $ran(loc) \mapsto ran(arm)$. We can see how the two persons can pass between the rooms. It is not possible to tell which person is in which room but we can see that event $pass$ can occur and that locations change.

5 Concrete Model

We are satisfied with the abstract model of the secure building for now and turn to the refinement where doors are introduced into the model. A door will be represented as a pair consisting of two rooms. In the refined model we employ two variables adr for authorised doors and loc for the locations of persons in the building (as before). Variable adr is a function which indicates for every person the set of doors he is allowed to use. The intention is to keep the information contained in the abstract variable arm implicitly in the concrete variable adr. That is, in the refined model variable arm would be redundant. We specify

$inv5 \;:\; adr \in Person \rightarrow (Room \leftrightarrow Room)$ Property P5

$inv6 \;:\; \forall q \cdot \mathsf{ran}(adr(q)) \subseteq arm[\{q\}]$ Property P4

Any problem we can have with initialisation we can have with any other event, too; but in an initialisation they look less interesting because nothing can happen if an initialisation is wrong. (This is not an argument for ignoring initialisation when teaching modelling but for presenting interesting problems. And those usually do not appear in initialisations.) However, we need to know what the initialisation is in order to analyse other events. We reason: In the abstract model all persons can only be outside initially. This corresponds to them not being authorised to use any doors,

```
initialisation
  begin
    $act1 : adr := Person \times \{\varnothing\}$
    $act2 : loc := Person \times \{\boldsymbol{O}\}$
  end .
```

5.1 Moving between rooms

Let us first look at event $pass$. Only a few changes are necessary to model property P8,

```
$pass$
  any $p, r$ when
    $grd1 : loc(p) \mapsto r \in adr(p)$    person $p$ is authorised to enter room $r$ from current location
  then
    $act1 : loc := loc \Leftarrow \{p \mapsto r\}$
  end .
```

We only have to show guard strengthening, because loc does not occur in $inv5$ and $inv6$. The abstract guard $grd1$ is strengthened by the concrete guards because $r \in \mathsf{ran}(adr(p))$ and by $inv6$, $\mathsf{ran}(adr(p)) \subseteq arm[\{p\}]$. The second guard strengthening proof obligation of event $pass$ is:

	$loc \in Person \to Room$	*Invariant* $inv3$
	$loc(p) \mapsto r \in adr(p)$	*Concrete guard* $grd1$
$\vdash$		
	$p \mapsto r \notin loc$	*Abstract guard* $grd2$

Using $inv3$ we can rephrase the goal,

$$\begin{array}{lll} & p \mapsto r \notin loc & \{\, inv3 \,\} \\ \Leftrightarrow & loc(p) \neq r & \end{array}$$

Neither concrete guard $grd1$ nor the invariants $inv1$ to $inv6$ imply this. If we animate the abstract and the concrete machine simultaneously, we find that in the concrete machine a person can pass from some room into the same room. In the abstract machine this is not possible as can be seen in Figure 6. We could add a guard $loc(p) \neq r$ to

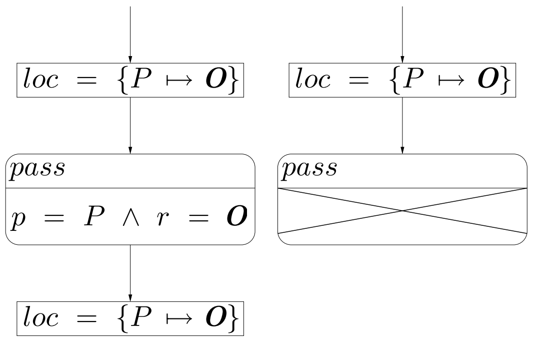

$loc = \{P \mapsto \mathbf{O}\}$
$loc = \{P \mapsto \mathbf{O}\}$
$pass$
$p = P \wedge r = \mathbf{O}$
$pass$
$loc = \{P \mapsto \mathbf{O}\}$

Fig. 6: Simultaneous animation of concrete (left) and abstract (right) machine

the concrete model but this would make event $pass$ express *a person can pass through a door if it connects two different rooms*. However, the model should not contain doors that connect rooms to themselves in the first place. The invariant is too weak. We do not specify that doors connect *different* rooms. In fact, our model of the building is rather weak. We decide to model the building by the doors that connect the rooms in it. They are modelled by a constant $Door$. We make the following two assumptions about doors:

$axm1$	$: Door \in Room \leftrightarrow Room$	*Each door connects two rooms.*
$axm2$	$: Door \cap \mathsf{id}_{Room} = \varnothing$	*No door connects a room to itself.*

A new invariant $inv7$ prevents doors connecting rooms to themselves. We realise that it captures much better property P5 than invariant $inv5$,

$$inv7 \quad : \forall q \cdot adr(q) \subseteq Door \quad . \qquad\qquad \text{Property P5}$$

Using $inv7$ and $axm2$, we can prove $loc(p) \mapsto r \in adr(p) \Rightarrow loc(p) \neq r$ allowing to discharge the guard strengthening proof obligation above.

5.2 Leaving the building

It may be necessary to pass though various rooms in order to leave the building. Hence, we need to specify a property about the transitive relationship of the doors. Property P7 is more involved.

A relation x is called *transitive* if $x\,;x \subseteq x$. In other words, any composition of elements of x is in x. The transitive closure of a relation x is the least relation that contains x and is transitive. We define the *transitive closure* x^+ of a relation x by

$$\forall x \cdot x \subseteq x^+ \tag{6}$$
$$\forall x \cdot x^+ ; x \subseteq x^+ \tag{7}$$
$$\forall x, z \cdot x \subseteq z \wedge z\,;x \subseteq z \Rightarrow x^+ \subseteq z \quad . \tag{8}$$

That is, x^+ is the least relation z satisfying $x \cup z\,;x \subseteq z$.

Using the transitive closure of authorised rooms we can express that every person can at least reach the authorised rooms from the outside,

$$inv8 \;:\; \forall q \cdot arm[\{q\}] \subseteq adr(q)^+[\{\boldsymbol{O}\}] \cup \{\boldsymbol{O}\} \quad .$$

This invariant does not quite correspond to property P7. However, by the end of Section 5 we will be able to prove that all invariants jointly imply property P7 which we formalise as a theorem,

$$thm1 \;:\; \forall q \cdot (arm[\{q\}] \setminus \{\boldsymbol{O}\}) \times \{\boldsymbol{O}\} \subseteq adr(q)^+ \quad . \qquad \text{Property P7}$$

We proceed like this because we expect that proving $inv8$ to be preserved would be much easier than doing the same with $thm1$. Note, that being able to leave the building has little to do with moving between rooms but with granting and revoking authorisations. We do not formalise leaving the building only ability to do so. And this can appropriately done by means of an invariant or a theorem such as the above.

5.3 Granting door authorisations

A new door authorisation can be granted to a person if (a) it has not been granted yet and (b) authorisation for one of the connected rooms has been granted to the person. We introduce constraint (a) to focus on the interesting case and constraint (b) to satisfy invariant $inv8$. Thus,

```
$grant$
  any  $p, s, r$  when
    $grd1 : \{s \mapsto r, r \mapsto s\} \subseteq Door \setminus adr(p)$
    $grd2 : s \in \mathsf{dom}(adr(p)) \cup \{\boldsymbol{O}\}$
  then
    $act1 : adr := adr \Lleftarrow \{p \mapsto adr(p) \cup \{s \mapsto r, r \mapsto s\}\}$[6]
  end
```

In our model a door connecting a room r to another room s is modelled by the set of pairs $\{s \mapsto r, r \mapsto s\}$. Doors are modelled by their property of connecting two rooms in both directions. Each door D is a *symmetric* relation, that is, $D \subseteq D^{-1}$.

Unfortunately, we have introduced a deadlock. Figure 7 shows an example of a state trace leading to a deadlock. The guard of event $grant$ seems to be too strong. The problem is caused by the set of doors. It satisfies $Door \cap Door^{-1} = \varnothing$. We have not specified symmetry as a property of the set $Door$. Hence, there may not be any door $\{s \mapsto r, r \mapsto s\}$ contained in that set. Symmetry of the set $Door$ needs to be specified, too:

$$axm3 \;:\; Door \subseteq Door^{-1} \qquad \textit{Each door can be used in both directions}$$

[6] Event-B has the shorter (and more legible) notation $adr(p) := adr(p) \cup \{s \mapsto r, r \mapsto s\}$ for this. We do not use it because we can use the formula above directly in proof obligations. We also try as much as possible to avoid introducing more notation than necessary.

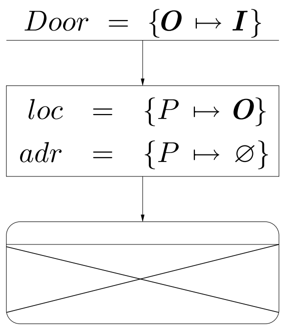

$Door = \{\boldsymbol{O} \mapsto \boldsymbol{I}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$
$adr = \{P \mapsto \varnothing\}$

Fig. 7: State trace leading to a deadlock

It is another assumption we have taken into account when modelling the building. It was not the guard of event $grant$ that was too strong, rather the assumptions about the building were too weak.

There is yet another problem. The guard of concrete event $grant$ is to weak to prove preservation of invariant $inv6$. (We could also look for problem with the action but this does not appear promising given all it does is to add the door $\{s \mapsto r, r \mapsto s\}$ to the authorisations of person p.) In fact, this cannot be spotted by animation or model-checking. The state of the system always satisfies the invariant and cannot reach an inconsistent state. However, the invariant does not provide enough information to prove this. We have a look at the corresponding proof obligation. For invariant $inv6$ we have to prove:

$$
\begin{array}{lll}
 & \forall q \cdot \mathsf{ran}(adr(q)) \subseteq arm[\{q\}] & \textit{Invariant } inv6 \\
 & \{s \mapsto r, r \mapsto s\} \subseteq Door \setminus adr(p) & \textit{Concrete guard } grd1 \\
 & s \in \mathsf{dom}(adr(p)) & \textit{Concrete guard } grd2 \\
\vdash & & \\
 & \mathsf{ran}((adr \mathbin{\lhd\!\!\!-} \{p \mapsto adr(p) \cup \{s \mapsto r, r \mapsto s\}\})(q)) & \\
 & \quad \subseteq (arm \cup \{p \mapsto r\})[\{q\}] & \textit{Modified invariant } inv6
\end{array}
$$

for all q. For $q \neq p$ the proof is easy. For the other case $q = p$ we prove,

$$
\begin{array}{ll}
 & \mathsf{ran}(adr(p) \cup \{s \mapsto r, r \mapsto s\}) \subseteq (arm \cup \{p \mapsto r\})[\{p\}] \\
\Leftarrow & \dots \\
\Leftarrow & s \in \mathsf{ran}(adr(p))
\end{array}
$$

We would expect $s \in \mathsf{ran}(adr(p))$ to hold because doors are symmetric and because by concrete guard $grd2$ we have $s \in \mathsf{dom}(adr(p))$. Although only symmetric relations $\{s \mapsto r, r \mapsto s\}$ are added to $adr(p)$ it is recorded nowhere that $adr(p)$ itself is therefore a symmetric relation. We have to specify it explicitly,

$$inv9 \ : \ \forall q \cdot adr(q) \subseteq adr(q)^{-1} \quad . \qquad \text{(see axiom } axm3\text{)}$$

We can continue the proof where we left off

$$\begin{array}{lll} & s \in \mathsf{ran}(adr(p)) & \{\ inv9 \text{ with “}q := p\text{” }\} \\ \Leftarrow & s \in \mathsf{dom}(adr(p)) & \end{array}$$

By adding invariant $inv9$ we have stated a property of the model that was already true. It just was not mentioned explicitly in the model. This property could only be discovered by proof [7].

5.4 Revoking door authorisations

We model revoking of door authorisations symmetrically to granting door authorisations. A door authorisation can be revoked if (a) there is an authorisation for the door, (b) the corresponding person is not in the room that could be removed, (c) the room is not the outside, and (d) all rooms except for the room to be removed must still be reachable from the outside after revoking the authorisation for a door leading to that room. Condition (a) is just chosen symmetrically to $grd1$ of refined event $revoke$ (for the same reason). The other two conditions (b) and (c) are already present in the abstraction. The refined events $grant$ and $revoke$ together model property P9.

```
$revoke$
  any  $p, s, r$  when
    $grd1 : s \mapsto r \in adr(p)$
    $grd2 : p \mapsto r \notin loc$
    $grd3 : r \neq \boldsymbol{O}$
    $grd4 : \mathsf{ran}(adr(p)) \setminus \{r\} \subseteq (adr(p) \setminus \{s \mapsto r, r \mapsto s\})^{+}[\{\boldsymbol{O}\}] \cup \{\boldsymbol{O}\}$
  then
    $act1 : adr := adr \Leftarrow \{p \mapsto adr(p) \setminus \{s \mapsto r, r \mapsto s\}\}$
  end
```

We succeed proving guard strengthening of the abstract guards $grd1$ to $grd3$ and preservation of $inv5$, $inv6$, $inv7$, and $inv9$. But preservation of $inv6$ cannot be proved:

$$\begin{array}{lll} & \forall q \cdot \mathsf{ran}(adr(q)) \subseteq arm[\{q\}] & \textit{Invariant } inv6 \\ & s \mapsto r \in adr(p) & \textit{Concrete guard } grd1 \\ & p \mapsto r \notin loc & \textit{Concrete guard } grd2 \\ & r \neq \boldsymbol{O} & \textit{Concrete guard } grd3 \\ \vdash & & \\ & \mathsf{ran}((adr \Leftarrow \{p \mapsto adr(p) \setminus \{s \mapsto r, r \mapsto s\}\})(q)) & \\ & \quad \subseteq (arm \setminus \{p \mapsto r\})[\{q\}] & \textit{Modified invariant } inv6 \end{array}$$

for all q. For $q = p$ we have to prove $\mathsf{ran}(adr(p) \setminus \{s \mapsto r, r \mapsto s\}) \subseteq arm[\{p\}] \setminus \{r\}$, thus, $r \notin \mathsf{ran}(adr(p) \setminus \{s \mapsto r, r \mapsto s\})$. This does not look right. Model-checking yields a counter example in form of a state trace; see Figure 8. (ProB permits to search directly for a violation of invariant $inv6$.) We find a counter example with one

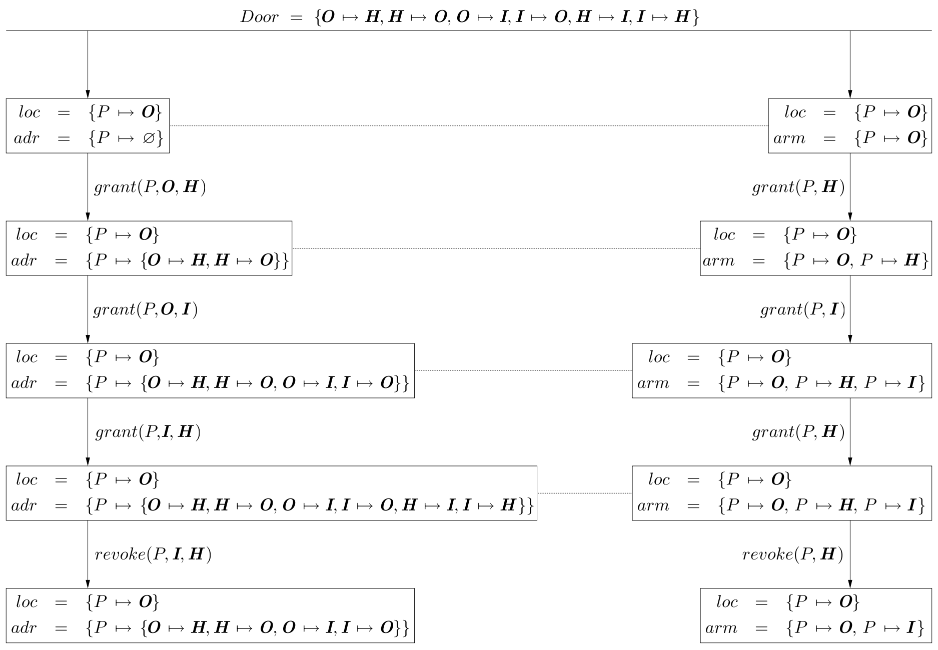

$Door = \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}, \boldsymbol{H} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{H}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \varnothing\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}\}$
$grant(P,\boldsymbol{O},\boldsymbol{H})$
$grant(P,\boldsymbol{H})$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{H}\}$
$grant(P,\boldsymbol{O},\boldsymbol{I})$
$grant(P,\boldsymbol{I})$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{H}, P \mapsto \boldsymbol{I}\}$
$grant(P,\boldsymbol{I},\boldsymbol{H})$
$grant(P,\boldsymbol{H})$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}, \boldsymbol{H} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{H}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{H}, P \mapsto \boldsymbol{I}\}$
$revoke(P,\boldsymbol{I},\boldsymbol{H})$
$revoke(P,\boldsymbol{H})$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{I}\}$

Fig. 8: Simultaneous state trace leading to an invariant violation

person P and three different rooms $\boldsymbol{H}, \boldsymbol{I}, \boldsymbol{O}$. We can reach the state:

$$\begin{aligned} adr &= \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}, \boldsymbol{I} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{I}\}\} \\ arm &= \{P \mapsto \boldsymbol{H}, P \mapsto \boldsymbol{I}, P \mapsto \boldsymbol{O}\} \\ loc &= \{P \mapsto \boldsymbol{O}\} \quad . \end{aligned}$$

Revoking a door authorisation with parameters

$$p = P \quad s = \boldsymbol{I} \quad r = \boldsymbol{H}$$

leads to a state violating invariant $inv6$. In order to resolve this problem we could remove all doors connecting to r. But this seems not acceptable: we grant door authorisations one by one and we should revoke them one by one. We have to look at the problem from another angle. Figure 9 shows a continuation of the simultaneous trace of the abstract and the concrete machine. We cannot say anymore what happens in the abstract machine because the gluing invariant $inv6$ is violated. But we can still see what the concrete machine could "do" next. Concrete revoke may occur again removing a door to $\boldsymbol{H}$. We could strengthen the guard of the concrete event requiring, say, $adr(p)[\{r\}] = \{s\}$. But then we would not be able to revoke authorisations once there are two or more doors for the same room. The problem is in the abstraction! We should allow abstract event $revoke$ to occur more often. It should not always remove r when it occurs. We weaken the guard of the abstract event using a set R of at most one room

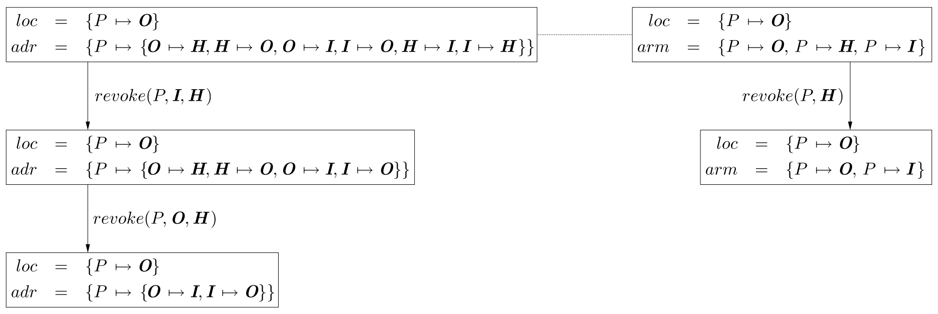

$loc = \{P \mapsto \boldsymbol{O}\}$
$adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}, \boldsymbol{H} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{H}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$
$arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{H}, P \mapsto \boldsymbol{I}\}$
$revoke(P, \boldsymbol{I}, \boldsymbol{H})$
$revoke(P, \boldsymbol{H})$
$loc = \{P \mapsto \boldsymbol{O}\}$
$adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$
$arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{I}\}$
$revoke(P, \boldsymbol{O}, \boldsymbol{H})$
$loc = \{P \mapsto \boldsymbol{O}\}$
$adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}\}\}$

Fig. 9: The concrete trace continued

instead of r. If $R = \varnothing$, then $\{p\} \times R = \varnothing$. So, for $R = \varnothing$ event $revoke$ does not change arm and for $R = \{r\}$ the effect of the event corresponds to the first attempt at abstract event $revoke$:

```
$revoke$
  any  $p, R$  when
    $grd1 : p \in Person$
    $grd2 : loc(p) \notin R$
    $grd3 : R \in \mathbb{S}(Room \setminus \{\boldsymbol{O}\})$
  then
    $act1 : arm := arm \setminus (\{p\} \times R)$
  end ,
```

where for a set X by $\mathbb{S}(X)$ we denote all subsets of X with at most one element:

$$Y \in \mathbb{S}(X) \quad \widehat{=} \quad Y \subseteq X \land (\forall x, y \cdot x \in Y \land y \in Y \Rightarrow x = y) \quad .$$

With this the proof obligation for invariant preservation of $inv6$ becomes:

$\forall q \cdot \mathsf{ran}(adr(q)) \subseteq arm[\{q\}]$	*Invariant* $inv6$
$s \mapsto r \in adr(p)$	*Concrete guard* $grd1$
$p \mapsto r \notin loc$	*Concrete guard* $grd2$
$r \neq \boldsymbol{O}$	*Concrete guard* $grd3$
$\vdash$	
$\mathsf{ran}((adr \lhd\!\!- \{p \mapsto adr(p) \setminus \{s \mapsto r, r \mapsto s\}\})(q))$	
$\quad \subseteq (arm \setminus (\{p\} \times R))[\{q\}]$	*Modified invariant* $inv6$

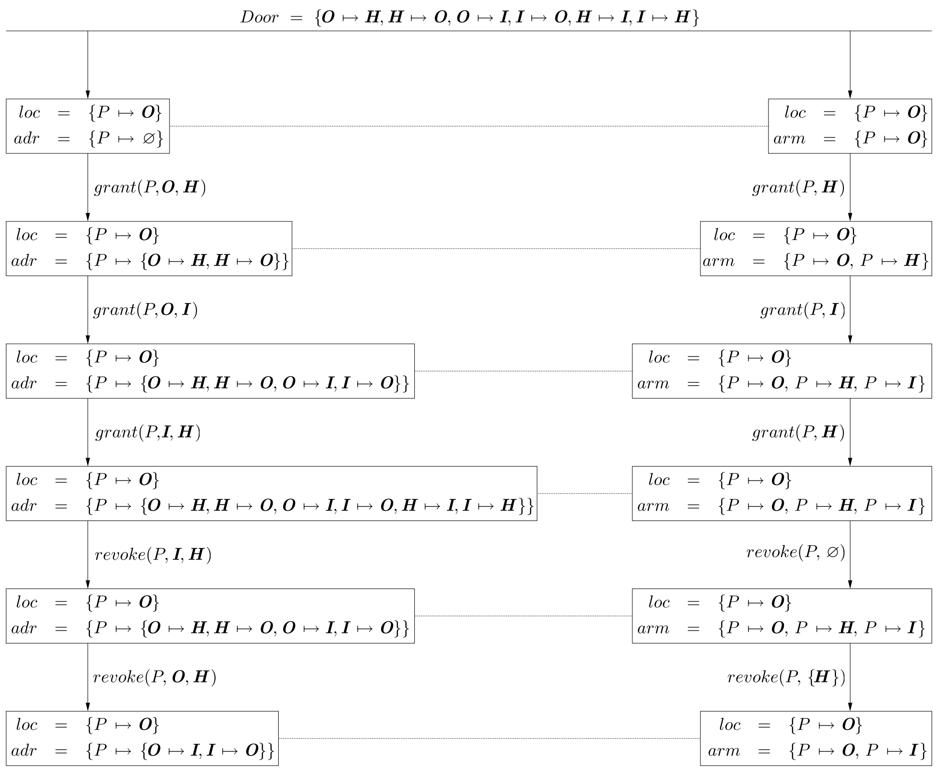
for all q. For $q = p$ we have to prove,

$$\mathsf{ran}(adr(p) \setminus \{s \mapsto r, r \mapsto s\}) \subseteq arm[\{p\}] \setminus R \quad . \tag{9}$$

We need to make a connection between r and R. We need a witness for R. After some reflection we decide for

$$R = \{r\} \setminus \mathsf{ran}(adr(p) \setminus \{s \mapsto r, r \mapsto s\}) \quad . \tag{10}$$

Witness (10) explains how the concrete and the abstract event are related. If there is only one door s connecting to room r, then $R = \{r\}$ and the authorisation for room r is revoked. Otherwise, $R = \varnothing$ and the authorisation for room r is kept. The simultaneous trace (Figure 10) now confirms the correct behaviour and the proof succeeds too.



$Door = \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}, \boldsymbol{H} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{H}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \varnothing\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}\}$
$grant(P, \boldsymbol{O}, \boldsymbol{H})$
$grant(P, \boldsymbol{H})$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{H}\}$
$grant(P, \boldsymbol{O}, \boldsymbol{I})$
$grant(P, \boldsymbol{I})$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{H}, P \mapsto \boldsymbol{I}\}$
$grant(P, \boldsymbol{I}, \boldsymbol{H})$
$grant(P, \boldsymbol{H})$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}, \boldsymbol{H} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{H}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{H}, P \mapsto \boldsymbol{I}\}$
$revoke(P, \boldsymbol{I}, \boldsymbol{H})$
$revoke(P, \varnothing)$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{H}, P \mapsto \boldsymbol{I}\}$
$revoke(P, \boldsymbol{O}, \boldsymbol{H})$
$revoke(P, \{\boldsymbol{H}\})$
$loc = \{P \mapsto \boldsymbol{O}\}$, $adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}\}\}$
$loc = \{P \mapsto \boldsymbol{O}\}$, $arm = \{P \mapsto \boldsymbol{O}, P \mapsto \boldsymbol{I}\}$

Fig. 10: Simultaneous trace of corrected model

6 Conclusion

We have shown how different techniques of formal reasoning can be used jointly to understand and improve a formal model. In this article we have included formal proof, model-checking, and animation. This intended to be an open list. Sometimes we have used the result of model-checking or animation of a model to understand better problems that appeared in proof obligations. We have also seen cases where only model-checking showed that there was a problem. In the fragment of Event-B defined in Section 2 there is no mention of deadlock freedom, but the model-checker of ProB checks for it based on the operational interpretation. In all cases we have used the information gained as evidence from where to investigate and explain errors. We also saw that not all problems are found by model-checking or animation. Neither formal proof nor model-checking are complete in this sense.

A danger of using the operational interpretation is that students *only* think in terms of it, making it difficult to convey at the same time the usefulness of abstract reasoning by formal proof. But we think this is a very limited risk and the benefits outweigh it by far. In particular, comparing refinements by simultaneous traces helps greatly understanding particular refinements. An improved implementation of simultaneous model-checking and animation in ProB, using ideas of Brama [13], is under way.

Acknowledgement. This research was carried out as part of the EU research project DEPLOY (Industrial deployment of system engineering methods providing high dependability and productivity) http://www.deploy-project.eu/.

References

1. Jean-Raymond Abrial. *The B-Book: Assigning Programs to Meanings*. CUP, 1996.
2. Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2008. To appear.
3. Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, and Laurent Voisin. An open extensible tool environment for Event-B. In Z. Liu and J. He, editors, *ICFEM 2006*, volume 4260, pages 588–605. Springer, 2006.
4. Jean-Raymond Abrial and Stefan Hallerstede. Refinement, Decomposition and Instantiation of Discrete Models: Application to Event-B. *Fundamentae Informatica*, 77(1-2), 2007.
5. *Eclipse* platform homepage. http://www.eclipse.org/.
6. Stefan Hallerstede. How to make mistakes. In *TFM B*, 2009. To appear.
7. Michael Leuschel and Michael Butler. ProB: A model checker for B. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, LNCS 2805, pages 855–874. Springer-Verlag, 2003.
8. Michael Leuschel and Michael Butler. Automatic refinement checking for B. In Kung-Kiu Lau and Richard Banach, editors, *Proceedings ICFEM'05*, LNCS 3785, pages 345–359. Springer-Verlag, 2005.
9. Michael Leuschel and Michael J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.
10. Michael Leuschel and Edward Turner. Visualizing larger states spaces in ProB. In Helen Treharne, Steve King, Martin Henson, and Steve Schneider, editors, *Proceedings ZB'2005*, LNCS 3455, pages 6–23. Springer-Verlag, April 2005.
11. Carroll C. Morgan. *Programming from Specifications: Second Edition*. Prentice Hall, 1994.
12. Steve Schneider. *The B-Method: An Introduction*. Palgrave, 2002.
13. Thierry Servat. BRAMA: A new graphic animation tool for B models. In J. Julliand and O. Kouchnarenko, editors, *B 2007*, volume 4355 of *LNCS*, pages 274–276. Springer, 2007.
14. Corinna Spermann and Michael Leuschel. ProB gets nauty: Effective symmetry reduction for B and Z models. In *Proceedings Symposium TASE 2008*, pages 15–22, Nanjing, China, June 2008. IEEE.