How to make mistakes*

Corrected Version

Stefan Hallerstede

University of Düsseldorf
Germany
halstefa@cs.uni-duesseldorf.de

Abstract. When teaching Event-B to beginners, we usually start with models that are already good enough, demonstrating occasionally some standard techniques like "invariant strengthening". We show that we got it essentially right but need to make improvements here and there. However, this is not how we really create formal models. To a beginner, getting shown only nearly perfect models is overwhelming. So we should start earlier and show how we usually get models wrong initially. This provides ample opportunity to demonstrate the strengths of formal reasoning (and the weaknesses). The principal strength of formal reasoning lies in its capacity to locate mistakes in a model and to suggest corrections. A beginner should learn how to profit from his mistakes by improving his understanding of the model. A weakness of formal reasoning is that we only find mistakes that we expect, for example, invariant violation or non-termination. Mistakes that do not fall into one of these categories may slip through.
In this article we present how a formal model is created by refinement and alteration. The approach employs mathematical methodology for problem solving and a software tool. Both aspects are important. Mathematical methodology provides ways to turn mistakes into improvements. The software tool is necessary to ease the impact of changes on a model and to obtain rapid feed back. We begin with a set of assumptions and requirements, the problem, and set out to solve it, giving a more vivid picture of how formal methods work.

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

* 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/.

that refinement can also be used to produce implementations of abstract models, for instance, in terms of a sequential program [1,8]. 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 to 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 it does not leave enough room for misunderstanding. 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. It is not possible to show everything that happens during an actual development of a formal model. But it is possible to point at the main difficulties encountered and show how to approach them. Students should be encouraged to make mistakes and experiment with formal models. On the way they will learn about strengths and weaknesses of formal methods.

We approach the model of the access control in small increments, learning at each increment something about the problem and the model. We understand this incremental approach in two ways. The first way is by formal refinement. An existing model is proved to be refined by another: all properties of the existing model are preserved in the refined model. The second way is by alteration of an existing model: properties of the existing model may be broken. When a model is shown to be not consistent, it needs to be modified in order to make it consistent. This reflects a learning process supported by various forms of reasoning about a model, for instance, proof, animation, or model-checking. This way of thinking about a model is common in mathematical methodology [6,9]. The first way is commonly used and taught in formal methods, whereas the second is at least not acknowledged. In order to apply formal methods successfully both ways need to be mastered. This is only feasible in the presence of software tools that make reasoning easy and modifications to a model painless. We have relied on the Rodin modelling tool [3] for Event-B for proof obligation generation and proof support and on the ProB tool [7] for animation and model-checking. Both tools are integrated in the Rodin platform and can be used seamlessly. In later sections we do not further specify the tools used, though, as this should be clear from the context. Also note that we present proof in an equational style [5,10] whereas the Rodin tool uses sequents as in [2].

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. Section 4 provides a more detailed overview of Sections 5 to 9. It is intended to help the reader keeping track of how the solution of the problem advances in the ensuing sections. A first model is produced and discussed in Section 5. In Sections 6 and 8 we elaborate the model by refinement. Section 7 contains a small theory of transitive closures that is needed in the refinement. In Section 9 some further improvements of the model are made and limitations of formal modelling discussed.

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 and based on [2], we have reduced the Event-B notation used so that only a little notation suffices and formulas are easier to comprehend, in particular, concerning the relationship between formal model and proof obligations. We have also reduced the number of proof obligations associated with a model. We have done this for two reasons: firstly, it is easier to keep track of what is to be proved; secondly, it permits us to make a point about a limitation of formal methods later on.

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 $I(v)$. 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 *parameters* the event may contain and $x = x_1, \ldots, x_p$ are the variables it may change[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. We denote an event $E(v)$ by

$$
\begin{array}{lcl}
E(v) & \widehat{=} & \mathsf{any}\ \ t\ \ \mathsf{when} \\
 & & \quad G(t,v) \\
 & & \mathsf{then} \\
 & & \quad x := S(t,v) \\
 & & \mathsf{end}
\end{array}
\qquad \text{or} \qquad
\begin{array}{lcl}
E(v) & \widehat{=} & \mathsf{begin} \\
 & & \quad x := S(v) \\
 & & \mathsf{end}
\end{array}
\quad .
$$

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

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

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

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

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

where $x_1, \ldots, x_p, y_1, \ldots, y_q$ are the variables v of the machine. The action $x := S(t, v)$ of event $E(v)$ denotes 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* ($\ell \in 1 \,..\, i$):

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

for every event $E(v)$. Similar proof obligations are associated with the initialisation event of a machine. The only difference is that for an initialisation event 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

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

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 specifically how the refinement is achieved. The corresponding proof obligations for refinement are called *guard strengthening* ($\ell \in 1 \mathinner{..} g$):

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

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

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

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]. Instead of presenting a fully developed model, we illustrate the process of how we arrive at the model. We can not follow the exact path that we took when working on the model: we made changes to the model as a whole several times. So we would soon run out of space. We comment on some of the changes without going too much into detail in the hope to convey some of the dynamic character of the modelling process.

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 Storyboard

In this section we give a brief overview of the development as it will unfold in Sections 5 to 9. The reader is encouraged to return to this section while reading through those sections. We have tried to keep a natural flow in the presentation to make it more credible. It also requires some skill to stay on track when solving a complex problem.

It lies in the nature of the presented material that we can easily lose sight of our aim. By the way, what is our aim? This will be stated in the beginning of Section 5 before starting any work:

> *Our aim is to produce a faithful formal model of the system described by the properties P1 to P9 of Section 3.*

Once we know what we want to achieve we can get started. In Section 5 we begin by making a rough plan of how to proceed and create a first abstract machine introducing four invariants $inv1$ to $inv4$ and abstract events $pass$, $grant$, and $revoke$. The reason for the briefness of Section 5 is simply that we needed space to make some more interesting points later on. It is not to suggest that inventing a first model would be trivial.

In Section 6 we commence with the refinement of the abstract machine, introducing invariants $inv5$ to $inv7$, and event $pass$. We uncover that the properties P1 to P9 do not say enough about the nature of the doors used in the building. Incompleteness of assumptions is a common difficulty dealing with which carries the risk of introducing undocumented assumptions into the model. At this point we have incorporated properties P1 to P6 and property P8 into our model.

In order to express P7 formally, a little bit of theory about transitive closures is needed which is introduced in Section 7. (Nothing new is "invented". We simply use an existing theory.) We use this in Section 8 to have a first go at property P7 by means of invariant $inv8$. (Property P7 is formalised as a more intricate theorem $thm1$ implied by the invariants. This is a common strategy to ease the effort of verifying invariant preservation.) However, when analysing $inv8$ we find that it is too strong and weaken it to $inv8'$.

In the remainder of Section 8 we refine events $grant$ and $revoke$. (Proving refinement of event $grant$ we find the need for invariant $inv9$, which would also permit us to prove $thm1$.) When trying to refine event $revoke$ we discover that event $revoke$ of the abstract machine is wrong. We have to alter the abstract machine in order to get a model that is consistent and represents P9 adequately. By the end of Section 8 we feel that properties P1 to P9 have all been incorporated into the model. To ensure this we have made some effort to trace them into the formal model and argued that each one has been adequately captured.

In Section 9 we have another look at the model, animating it. We immediately see that the concrete machine cannot "do" anything. It deadlocks right after the initialisation of the machine. We rectify the problem and use it to illustrate limitations of a formal method. This is also a good place to advertise the usefulness of complementary techniques in analysing formal models.

5 Getting started with a fresh model

Our aim is to produce a faithful formal model of the system described by the properties P1 to P9 of Section 3. The first decision we need to make concerns the use of refinement. We have decided to introduce the properties of the system in two steps. In the first step we deal only with persons and rooms, in the second also with doors. This approach appears reasonable. At first we let persons move directly between rooms. Later we state how they do it, that is, by passing through doors. In order to specify doors we need to know about rooms they connect. It is a good idea, though, to reconsider the strategy chosen for refinement when it turns out to be difficult to tackle the elements of the model in the planned order. For now, we intend to produce a model with one refinement:

(i) the *abstract machine* (this section) models room authorisations;
(ii) the *concrete machine* (sections 6 and 8) models room and door authorisations.

In Event-B we usually begin modelling by stating invariants that a machine should preserve. When events are introduced subsequently, we think more about how they preserve invariants than about what they would do. The focus is on the properties that have to be satisfied. 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

$inv1 : arm \in Person \leftrightarrow Room$	Property P4
$inv2 : Person \times \{\boldsymbol{O}\} \subseteq arm$	
$inv3 : loc \in Person \rightarrow Room$	Property P3
$inv4 : loc \subseteq arm$	Property P6

Invariant $inv2$, that *each person is authorised to be outside*, is necessary because we decided to model location by a total function making the outside a special room. For instance, person p is outside is written formally $loc(p) = \boldsymbol{O}$. In a first attempt, we made loc a partial function from $Person$ to $Room$ expressing that a person not in the domain of loc is outside. However, this turned out to complicate the gluing invariant when introducing doors into the model later on. (Because of property P7 we need an explicit representation of the outside in the model.) As a consequence of our decision we had to introduce invariant $inv2$. It corresponds to a new requirement that is missing from the list in Section 3 but that we have uncovered while reasoning formally about the system. In the following we focus on how formal reasoning is used to improve the model of the system.

In order 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
    $act1 : loc := loc \lhd\!\!\!- \{p \mapsto r\}$
  end .

Event $pass$ preserves the invariants. 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 them right. But it is as easy to make a mistake. This is why we have specified invariants: to safeguard us against mistakes. If the proof of an invariant fails, we have the opportunity to learn something about the model and improve it. The two events preserve all invariants except for $revoke$ which violates invariant $inv2$,

$Person \times \{\boldsymbol{O}\} \subseteq arm$	*Invariant* $inv2$
$p \in Person$	*Guard* $grd1$
$p \mapsto r \notin loc$	*Guard* $grd2$
$\vdash$	
$Person \times \{\boldsymbol{O}\} \subseteq arm \setminus \{p \mapsto r\}$	*Modified invariant* $inv2$

In an instance of the model with two different rooms $\boldsymbol{I}$ and $\boldsymbol{O}$ and one person P we find a counter example:

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

In fact, 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 $revoke$:

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

A counter example provides valuable information, pointing to a condition that it does not satisfy. It may not always be as simple to generalise but at least one can obtain an indication where to look closer.

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? We have only partially modelled the assumptions P1, P2, P8, and P9. We could split them into smaller statements that would be fully modelled but have decided not to do so. Instead, we are going to document how they are incorporated in the refinement that is to follow.

6 Elaboration of more details

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. In the refined model we employ two variables adr for authorised doors and loc for the locations of persons in the building (as before). 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

$$inv5 \;:\; adr \in Person \to (Room \leftrightarrow Room) \qquad \text{Property P5}$$
$$inv6 \;:\; \forall q \cdot \mathsf{ran}(adr(q)) \subseteq arm[\{q\}] \qquad \text{Property P4}$$

6.1 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
    $grd1 \;:\; loc(p) \mapsto r \in adr(p)$
  then
    $act1 \;:\; loc := loc \lhd\!\!+ \{p \mapsto r\}$
  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 $r \in \mathsf{ran}(adr(p))$. The second guard strengthening proof obligation of event $pass$ is:

$$\begin{array}{ll} & loc \in Person \to Room & \textit{Invariant } inv3 \\ & loc(p) \mapsto r \in adr(p) & \textit{Concrete guard } grd1 \\ \vdash & & \\ & p \mapsto r \notin loc & \textit{Abstract guard } grd2 \end{array}$$

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. 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 three assumptions about doors:

$$axm1 \ : \ Door \in Room \leftrightarrow Room \qquad \textit{Each door connects two rooms.}$$
$$axm2 \ : \ Door \cap \mathsf{id}_{Room} = \varnothing \qquad \textit{No door connects a room to itself.}$$
$$axm3 \ : \ Door \subseteq Door^{-1} \qquad \textit{Each door can be used in both directions.}$$[2]

These assumptions are based on our domain knowledge about properties of typical doors. They were omitted from the problem description because they seemed obvious. However, the validity of our model will depend on them. As such they ought to be included. We began to think about properties of doors because we did not succeed in proving guard strengthening. If $adr(p)$, for $p \in Person$, would share the property of the set $Door$ given by $axm2$, we should succeed. Hence, we add a new invariant $inv7$. We realise that it captures much better property P5 than invariant $inv5$,

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

Using $inv7$ and $axm2$, we can prove $\forall x, y \cdot x \mapsto y \in adr(p) \Rightarrow x \neq y$, and with "$x, y := loc(p), r$" we are able to show the guard strengthening proof obligation above.

7 Intermezzo on transitive closures

Property P7 is more involved. It may be necessary to pass though various rooms in order to leave the building. We need to specify a property about the transitive relationship of the doors. We can rely on the well-known mathematical theory of the transitive closure of a relation.

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

$$\forall x \cdot x \subseteq x^+ \tag{5}$$
$$\forall x \cdot x^+ ; x \subseteq x^+ \tag{6}$$
$$\forall x, z \cdot x \subseteq z \wedge z ; x \subseteq z \Rightarrow x^+ \subseteq z \quad . \tag{7}$$

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

$$\forall x \cdot x \cup x^+ ; x = x^+ \tag{8}$$

The transitive closure is monotonic and maps the empty relation to itself,

$$\forall x, y \cdot x \subseteq y \Rightarrow x^+ \subseteq y^+ \tag{9}$$
$$\varnothing^+ = \varnothing \quad . \tag{10}$$

[2] We say $Door$ is a *symmetric* relation.

8 Towards a full model of the building

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

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

This invariant does not quite correspond to property P7. However, given the discussion about properties of doors in Section 6 we should 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 $thm1$. Let us continue working with $inv8$ for now and return to $thm1$ later.

8.1 Initialisation

In the abstract model all persons can only be outside initially. This corresponds to their not being authorised to use any doors,

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

The invariant preservation proof obligations for $inv5$ and $inv6$ hold, as can easily be seen letting "$arm,\ adr := Person \times \{\boldsymbol{O}\},\ Person \times \{\varnothing\}$" in $inv5$, $inv6$, and $inv7$. For invariant $inv8$ there is more work to do. We have to show:

$$\vdash \quad \forall q \cdot (Person \times \{\boldsymbol{O}\})[\{q\}] \subseteq (Person \times \{\varnothing\})(q)^+[\{\boldsymbol{O}\}]$$

Using law (10), $(Person \times \{\varnothing\})(q)^+[\{\boldsymbol{O}\}] = \varnothing \not\supseteq \{\boldsymbol{O}\} = (Person \times \{\boldsymbol{O}\})[\{q\}]$. Invariant $inv8$ is too strong! Because of invariant $inv7$ we cannot initialise adr to $Person \times \{\{\boldsymbol{O} \mapsto \boldsymbol{O}\}\}$ and because of $inv6$ we cannot use any other door. Thus, we must weaken invariant $inv8$. We replace it by:

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

8.2 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
    $grd1 : s \mapsto r \in Door \setminus adr(p)$
    $grd2 : s \in \mathsf{dom}(adr(p))$
  then
    $act1 : adr := adr \Leftarrow \{p \mapsto adr(p) \cup \{s \mapsto r, r \mapsto s\}\}$ [3]
  end

Invariant $inv5$ is preserved by event $grant$ by definition of relational overwriting $\Leftarrow$. For invariant $inv6$ we have to prove:

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

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

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

We would expect $s \in \mathsf{ran}(adr(p))$ to hold because doors are symmetric and because of concrete guard $grd2$, that is, $s \in \mathsf{dom}(adr(p))$. We specified symmetry in axiom $axm3$ but this property is not covered by invariant $inv7$. We have to specify it explicitly,

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

We can continue the proof where we left off

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

It is easy to show that invariants $inv7$ and $inv9$ are preserved by event $grant$. Preservation of $inv8'$ can be proved using law (8) and law (9).

Having specified invariant $inv9$ we would now succeed proving theorem $thm1$ postulated in the beginning of this section. This shows that our model satisfies property P7. We do not carry out the proof but turn to the last event not yet refined.

[3] 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.

8.3 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, and (c) the room is not the outside. Condition (a) is just chosen symmetrically to $grd1$ of refined event $grant$ (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
    $grd1 : s \mapsto r \in adr(p)$
    $grd2 : p \mapsto r \notin loc$
    $grd3 : r \neq \boldsymbol{O}$
  then
    $act1 : adr := adr \Leftarrow \{p \mapsto adr(p) \setminus \{s \mapsto r, r \mapsto s\}\}$
  end

We expect that the guard of event $revoke$ will be too weak to preserve invariant $inv8'$. We are going to search for it in the corresponding proof. But we can get started without it, in particular, proving guard strengthening of the abstract guards $grd1$ to $grd3$ and preservation of $inv5$, $inv6$, $inv7$, and $inv9$. For instance, preservation of $inv6$:

$\forall q \cdot \mathsf{ran}(adr(q)) \subseteq arm[\{q\}]$	*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* $grd3$
$\vdash$	
$\mathsf{ran}((adr \Leftarrow \{p \mapsto adr(p) \setminus \{s \mapsto r, r \mapsto s\}\})(q))$	
$\subseteq (arm \setminus \{p \mapsto r\})[\{q\}]$	*Modified invariant* $inv6$

for all q. For $q = p$ we have to prove $\mathsf{ran}(adr(p) \setminus \{s \mapsto r, r \mapsto s\}) \subseteq arm[\{p\}] \setminus \{r\}$, thus, $r \notin \mathsf{ran}(adr(p) \setminus \{s \mapsto r, r \mapsto s\})$. This does not look right. Indeed, we find a counter example with one person P and three different rooms $\boldsymbol{H}$, $\boldsymbol{I}$, $\boldsymbol{O}$:

$$adr = \{P \mapsto \{\boldsymbol{O} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{O}, \boldsymbol{O} \mapsto \boldsymbol{I}, \boldsymbol{I} \mapsto \boldsymbol{O}, \boldsymbol{I} \mapsto \boldsymbol{H}, \boldsymbol{H} \mapsto \boldsymbol{I}\}\}$$
$$arm = \{P \mapsto \boldsymbol{H}, P \mapsto \boldsymbol{I}, P \mapsto \boldsymbol{O}\}$$
$$loc = \{P \mapsto \boldsymbol{O}\} \quad p = P \quad s = \boldsymbol{I} \quad r = \boldsymbol{H}$$

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 could also 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! The abstract event $revoke$ should not always remove r. We weaken the guard of the abstract event using a set R of at most one room instead of r. If $R = \varnothing$, then $\{p\} \times R = \varnothing$.

So, for $R = \varnothing$ event $revoke$ does not change arm and for $R = \{r\}$ it 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:

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

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

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

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

$$\mathsf{ran}(adr(p) \setminus \{s \mapsto r, r \mapsto s\}) \subseteq arm[\{p\}] \setminus R \quad . \tag{11}$$

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

$$R = \{r\} \setminus \mathsf{ran}(adr(p) \setminus \{s \mapsto r, r \mapsto s\}) \quad . \tag{12}$$

Witness (12) explains how the concrete and the abstract event are related. If there is only one authorised 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. Now we can prove (11) using $inv6$ and (12). We note without showing the proofs that guard strengthening of the abstract guards $grd1$ to $grd3$ and preservation of $inv5$, $inv7$, and $inv9$ all hold. Only preservation of invariant $inv8'$ remains:

$$\begin{array}{ll}
 & \forall q \cdot arm[\{q\}] \subseteq adr(q)^+[\{\boldsymbol{O}\}] \cup \{\boldsymbol{O}\} & \textit{Invariant } inv8' \\
 & s \mapsto r \in adr(p) & \textit{Concrete guard } grd1 \\
 & p \mapsto r \notin loc & \textit{Concrete guard } grd2 \\
 & r \neq \boldsymbol{O} & \textit{Concrete guard } grd3 \\
\vdash & (arm \setminus (\{p\} \times R))[\{q\}] & \textit{Modified invariant } inv8' \\
 & \quad \subseteq (adr \lhd \{p \mapsto adr(p) \setminus \{s \mapsto r, r \mapsto s\}\})(q)^+[\{\boldsymbol{O}\}] \cup \{\boldsymbol{O}\} &
\end{array}$$

for all q. Let $D = \{s \mapsto r, r \mapsto s\}$. For $q = p$ we have to show

$$(arm \setminus (\{p\} \times R))[\{p\}] \subseteq (adr(p) \setminus D)^+[\{\boldsymbol{O}\}] \cup \{\boldsymbol{O}\} \quad . \tag{13}$$

We have seen above that the term on term on the left hand side is either $arm[\{p\}]$ or $arm[\{p\}] \setminus \{r\}$. So we won't succeed proving (13) unless we add a guard to event $revoke$. We cannot use $arm[\{p\}]$ in the guard because the refined machine does not contain variable arm. If $inv6$ was an equality, we could use $\mathsf{ran}(adr(p))$ instead of $arm[\{p\}]$, obtaining the guard

$$grd4 \ : \ \mathsf{ran}(adr(p)) \setminus \{r\} \subseteq (adr(p) \setminus D)^+[\{\boldsymbol{O}\}] \cup \{\boldsymbol{O}\} \quad .$$

It says that all rooms except for r must still be reachable from the outside after revoking the authorisation for door D leading to room r. This sounds reasonable. We find that it is not possible to turn the set inclusion into an equality in invariant $inv6$. However, we can still prove the weaker theorem

$$thm2 \ : \ \forall q \cdot \mathsf{ran}(adr(q)) \cup \{\boldsymbol{O}\} = arm[\{q\}] \quad ,$$

using $inv2$, $inv6$, $inv8'$, and property (8) of the transitive closure. The authorised rooms are maintained precisely by means of the authorised doors. As a matter of fact, initially we used $thm2$ as invariant instead of $inv6$ but then weakened the invariant to $inv6$ and proved $thm2$ as a theorem. This is a useful strategy for reducing the amount of proof necessary while keeping powerful properties such as $thm2$.

9 Towards a better model

After all the serious thinking about the model we are confident that the model captures the properties P1 to P9. Assuming we have one person P and three different rooms $\boldsymbol{H}$, $\boldsymbol{I}$, and $\boldsymbol{O}$ we can inspect how the modelled system would behave.

Initially variables adr and loc have the values

$$adr = Person \times \{\varnothing\}$$
$$loc = Person \times \{\boldsymbol{O}\} \quad .$$

Event $pass$ is disabled as expected; $grd1$, that is, $loc(p) \mapsto r \in adr(p)$ cannot be satisfied for any p and r. Similarly, event $revoke$ is disabled, but also event $grant$: guard $grd2$, $s \in \mathsf{dom}(adr(p))$, cannot be satisfied for any s. Deadlock! We have not proved all properties we would expect from our model. This property seems to be implicitly contained in properties P8 and P9, but we have missed it. We have to weaken $grd2$,

$$grd2' \ : \ s \in \mathsf{dom}(adr(p)) \cup \{\boldsymbol{O}\}$$

As a consequence, we have to check again that concrete event $grant$ preserves all invariants. Fortunately, all proofs succeed.

Note, that just adding another proof obligation will not suffice to solve the problem in general. We can easily imagine a lift, say, that does not have a deadlock because some

button could always be pressed and the lift could always move; but the doors of the lift would remain always shut. If we do not specify that we expect the doors to open on some occasions, a model of the lift may not have this property. Because such properties are common sense they are often not mentioned but then they are also easily forgot. We have to analyse the model using complementary techniques such as proving, model-checking, and animation in order to find such mistakes. In the end, the best we can hope for is a model of good quality that captures the required properties well. This problem holds for formal modelling in general. However, it is very visible in the incremental approach described in this article. The proof obligations shown in Section 2 have been restricted not to take into account deadlock-freedom to emphasise the problem that we only verify properties where we expect difficulties but not more. So we can see better the benefits of using jointly the three techniques of proof, model-checking, and animation.

10 Conclusion

What we have learned: We have used proof to verify that the model is consistent and to get indications for improvements of the model. We have used model-checking before attempting a proof. If a counter example was found, the effort of proving could be saved, and the counter example could be analysed. (We could also have started a proof knowing that it would fail.) Finally we have used animation to "try out" the model, to see whether it behaves reasonably. When we animated the model, we found a problem that was not discovered by the other two techniques. Whereas trying out (or systematic testing) does not show absence of errors as proof does, proof only verifies properties where we expect problems. In this sense proof is incomplete too. We have developed a more realistic impression of what a formal method can achieve.

References

1. J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. CUP, 1996.
2. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2008. To appear.
3. J.-R. Abrial, M. Butler, S. Hallerstede, and L. 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. J.-R. Abrial and S. Hallerstede. Refinement, Decomposition and Instantiation of Discrete Models: Application to Event-B. *Fundamentae Informatica*, 77(1-2), 2007.
5. D. Gries and F. B. Schneider. *A Logical Approach to Discrete Math*. Springer, 1994.
6. I. Lakatos. *Proofs and Refutations*. Cambridge University Press, 1976.
7. M. Leuschel and M. Butler. ProB : an automated analysis toolset for the B method. *International Journal on Software Tools for Technology Transfer*, 10(2):185–203, 2008.
8. C. C. Morgan. *Programming from Specifications: Second Edition*. Prentice Hall, 1994.
9. G. Pólya. *How to Solve It: A New Aspect of Mathematical Method*. Princeton Science Library. Princeton University Press, second edition, 1957.
10. A. J. M. van Gasteren. *On the Shape of Mathematical Arguments*, volume 445 of *LNCS*. Springer, 1990.