Tool Support for Qualitative Reasoning in Event-B

Master Thesis

by

EMRE YILMAZ

August, 2010

Supervisor	:	Dr. Thai Son Hoang
Professor	:	Prof. Dr. David Basin

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Chair of Information Security
Department of Computer Science
ETH-Zurich

Abstract

Event-B is a notation and method for modelling discrete transition systems by refinement. The standard reasoning in Event-B is based on non-determinism, however some system behaviours are more appropriately modelled probabilistically. Earlier work has extended Event-B with means for reasoning about qualitative probability. The extension provides proof obligations to prove almost-certain termination of systems and does not complicate the existing Event-B notation or method. However, this early work does not mention the preservation of qualitative reasoning in the case of refinement. Within our work we discuss how qualitative probabilistic reasoning can be maintained during refinement and propose some restrictions and conditions for almost-certain termination on refinement.

We continue the above investigation with the integration of qualitative probabilistic reasoning into Event-B further towards the direction of having a tool support. We extend the Rodin Platform to support proving almost-certain termination and using our new developed tool support we model some example algorithms terminating almost-certainly. In passing by, we formalise a non-trivial algorithm, namely Rabin's choice coordination. Our correctness reasoning is a combination of termination proofs in terms of probabilistic convergence and standard invariants techniques with refinement. We use the technique of splitting/merging the events to avoid having complicated proofs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In some systems, termination cannot be guaranteed for certain. Instead a slightly weaker property is mostly sufficient and appropriate: *termination with probability one*. An example having a such property is when tossing a fair coin, eventually heads will come up. In other words, the coin will turn up heads with probability one. There are many applications in distributed systems of such a “coin flip” and in particular for symmetry-breaking protocols [9, 11].

This kind of qualitative probabilistic reasoning has been integrated into Event-B [6]. Beside the standard non-deterministic actions in Event-B, a new kind of actions is added, namely, *probabilistic* actions with the probabilities for each possible alternative is neither 0 nor 1 (being “proper” [10]). Most of the time, actions of this type behave identically to the non-deterministic actions, except when reasoning about termination: they are interpreted *angelically* (as opposed to *demonically* nondeterminism). The result is a practical method for handling qualitative reasoning with the modified obligations stayed in standard first-order logic of Event-B. The integration of qualitative probabilistic reasoning into Event-B [6] gives the required proof obligations to prove almost-certain termination of events. However, it does not mention preserving qualitative reasoning in case of refinement. In this thesis, we propose some restrictions on refinement and an additional condition on variant so that probabilistic termination property is preserved.

We continue the investigation further to realise a tool support for this extension to Event-B. Our extension for Rodin Platform provides specifying probabilistic events and generates proof obligations accordingly for proving probabilistic termination of these events. Therefore, not only simple algorithms but also non-trivial algorithms can be developed with the tool support. In passing by, we formalise a non-trivial algorithm, namely, Rabin's choice coordination [11]. The reasoning on probabilistic termination of the algorithm is non-trivial, involving a lexicographic variant which need to be carefully formalised and mechanically proved to have a great assurance about the correctness of the algorithm. The case study acts as an illustration for the scalability of the approach for reasoning qualitatively in Event-B: It can be applied to more complex systems than just "coin tossing" examples.

Our development comprises of several refinements and includes reasoning about both standard and probabilistic terminations, and deadlock-freeness. Our approach is to first establish the model of the system without any termination arguments, then having several refinement layers dedicated to proving convergence properties of events according to a lexicographic variant. Essentially, with this way of development, our probabilistic termination arguments are preserved with refinement.

The thesis is structured as follows. In Chapter 2 we give a brief overview of the Event-B modelling method, focusing on proofs of convergent and qualitative reasoning. Furthermore, Chapter 3 discusses the termination in several refinement levels. Chapter 4 dedicates to the formalisation of Rabin's choice coordination algorithm. We present our tool support in Chapter 5. In Chapter 6 we give some other developments to show usage of the tool. Finally, we draw some conclusions in Chapter 7.

Chapter 2

Background and Related Work

2.1 The Event-B Modelling Method

Event-B is a notation used for developing mathematical models of discrete transition systems by refinement. It has been developed from the B-method [3]. Key features of Event-B are the use of set theory as a modelling notation, the use of refinement to represent systems at different abstraction levels and the use of mathematical proofs to verify consistency between refinement levels.

Event-B models are described in terms of two basic constructs: *contexts* and *machines*. Contexts contain the static part of a model whereas machines contain the dynamic part. Contexts and machines are presented in Section 2.1.1. In addition, machine refinement in Event-B allows us to build a model gradually by making it more and more precise. Machine refinement is presented in Section 2.1.2.

2.1.1 Contexts and Machines

Machines may contain *variables*, *invariants*, *events* and *variants*, whereas contexts may contain *carrier sets*, *constants*, and *axioms.* Machines and contexts have distinct names. Machines and contexts have various relationships: a machine can be *refined* by another one, and a context can be *extended* by another one. Moreover, a machine can *see* one or several contexts.

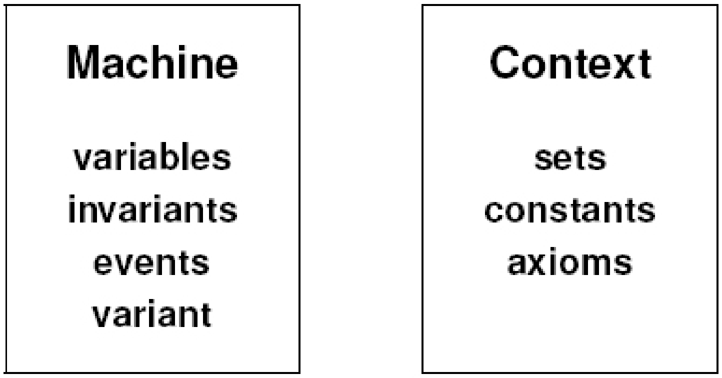


Figure 2.1: Machine and Context

In a context, *sets* list various carrier sets, which define pairwise disjoint types. *Axioms* define the main properties of the constants. Axioms can be marked as theorems denotes derived properties (to be proved) from previously declared the axioms.

In a machine, variables v[1] define the state of machine. Invariants $I(v)$ and the variant $V(v)$ state the properties of variables, where they are defined in terms of sets and constants. Events define the dynamics of the transition system. Each event is composed of a *guard* $G(t, v)$ and an *action* $S(t, v)$ where t are *parameters* the event may contain. An event can occur when

[1] From now on, variables v denotes a vector of variable

the necessary conditions stated in the guard satisfy. The action describes how the state variables evolve when the event occurs. An event E1 with parameters, an event E2 without parameters and an event E3 without guard can be represented by the following form

```
E1
  any  $t$  where
    $G(t,v)$
  then
    $S(t,v)$
  end
```

```
E2
  when
    $G(v)$
  then
    $S(v)$
  end
```

```
E3
  begin
    $S(v)$
  end
```

init is a special event that allows one to define initial situation of model. Therefore, init event has no guard, i.e. the form of E3.

The action of an event may be SKIP or composed of several *assignments* of the form

$$x := E\left(t, v\right) \tag{2.1}$$

$$x :\in E\left(t, v\right) \tag{2.2}$$

$$x :\mid Q\left(t, v, x'\right), \tag{2.3}$$

where x are some variables, $E\left(t, v\right)$ expressions, $Q\left(t, v, x'\right)$ predicate. Assigment form 2.1 is deterministic which directly assigns $E\left(t, v\right)$ to x. Assigment form 2.2 is nondeterministic which assigns an element of set $E\left(t, v\right)$ to x. Form 2.3 is a generalized form of assignment, which assigns a value x' satisfying a before-after predicate $Q\left(t, v, x'\right)$ to x. Assignments in form 2.1 and 2.2 can be written in form 2.3.

A *before-after predicate* describes the relationship between the states before and after an assignment has occured. We can represent the before values by unprimed variable names x, whereas the after values can be represented by primed variable names x'. Therefore, the before-after predicates of the assignment forms 2.1 and 2.2 are $x' = E\,(t, v)$ and $x' \in E\,(t, v)$, respectively. $Q\,(t, v, x')$ is the before-after predicate of the assignment form 2.3. Variables y, that do not appear on the left-hand side of any assignment of an action are not changed by the action. Formally, this is achieved by conjoining $Q\,(t, v, x')$ with $y' = y$. Hence, the before after predicate of the action $S(t, v)$ is $Q\,(t, v, x') \wedge y' = y$. In proof obligations, we represent the before-after predicate of an action $S(t, v)$ directly by the predicate $\mathbf{S}(t, v, v')$.

Proof obligations serve to verify certain properties of a machine. All proof obligations in this thesis are presented in the form of sequents: *hypothesis* $\vdash$ *goal*. We assume that our model have some context, hence references to constants, carrier sets and axioms are implicit, where hypothesis of all proof obligations includes axioms defined in the context.

For each non-deterministic action, *feasibility* must be proved. By proving feasibility, we achieve that $\mathbf{S}(t, v, v')$ provides an after state whenever $G(t, v)$ holds.

$I(v)$ $G(t, v)$ $\vdash$ $\exists v' \cdot \mathbf{S}(t, v, v')$	**FIS**

Invariants are supposed to hold whenever variable values change. The corresponding proof obligation is called *invariant preservation*.

$I(v)$ $G(t,v)$ $\mathbf{S}(t,v,v')$ $\vdash$ $I(v')$	**INV**

Similar proof obligations are associated with the initialisation event of a machine. The only difference is that the invariant and guard do not appear in the hypothesis of the proof obligations for *feasibility* and *invariant establishment*.

2.1.2 Machine Refinement

Refinement is a method for building models starting from the most abstract one. In each refinement level one can introduce new details and obtain more concrete model. From a given abstract machine, a new concrete machine can be built and asserted to be a refinement of the abstract machine. The state of the abstract machine is related to the state of the concrete machine by a gluing invariant $J(v,w)$, where v and w are the variables of the abstract machine and the concrete machine, respectively.

Each event ea of the abstract machine is refined by one or more concrete events ec. Let an abstract event ea and a concrete event ec be:

```
ea
  any  $t$  where
    $G(t,v)$
  then
    $S(t,v)$
  end
```

```
ec
  refines  $ea$
  any  $u$  where
    $H(u,w)$
  then
    $T(u,w)$
  end
```

Firstly, the concrete event must be feasible. Then, we can say that ec refines ea if the guard of ec is stronger than the guard of ea (*guard strengthening*), and the gluing invariants $J(v, w)$ establish a simulation of ec by ea (*simulation*).

$I(v)$ $J(v,w)$ $H(u,w)$ $\vdash$ $\exists w' \cdot \mathbf{T}(u,w,w')$	**FIS_REF**

$I(v)$ $J(v,w)$ $H(u,w)$ $\mathbf{T}(u,w,w')$ $\vdash$ $\exists t, v' \cdot G(t,v) \wedge \mathbf{S}(t,v,v') \wedge J(v',w')$	**REF**

The one-to-one correspondence between the abstract and concrete events can be relaxed. When an abstract event ea is refined by more than one concrete events ec, we say that the abstract event ea is *split* and prove that each concrete ec is a valid refinement of the abstract event. Conversely, several abstract events ea can be refined by one concrete ec. We say that these abstract events are *merged* together. A condition for merging events is that all abstract events must have an identical action and this action will become the action of the concrete event. Moreover, it is required that the guard of the concrete event is stronger than the disjunction of the abstract guards.

2.2 Termination and Variant

2.2.1 Proving Termination with Loop Variant

A loop is a way of repeating a statement a number of times until some condition no longer holds. An infinite loop may be a desired behavior in some cases. In the cases where termination of loop is desired, proving termination becomes important. A *loop variant* is used to prove termination of a loop by using *well-founded relations*. The use of well-founded relations for proving that programs terminate has been suggested by Floyd [5].

Definition 1. A *well-founded relation* is a binary relation $\prec$ on a set A such that there are no infinite descending chains $... \prec a_i \prec ... \prec a_1 \prec a_0$.

Note a well-founded relation is necessarily irreflexive i.e. for no a do we have $a \prec a$, as otherwise there would be the infinite descending chains $... \prec a \prec ... \prec a \prec a$. In addition, a set equipped with a well-founded relation is said to be a *well-founded set*.

Definition 2. A *loop variant* is a mathematical function defined on the state space of a computer program whose value is monotonically decreased with respect to a well-founded relation by the iteration of a loop.

By the nature of the well-founded relation, the value cannot decrease indefinitely, the loop certainly terminates.

An example of a well-founded relation is the usual order relation on $\mathbb{N}$: any decreasing sequence, such as 17, 15, 12, 10, 6, ... cannot go on strictly decreasing forever, so it has to be stationary - e.g. it can go down to 0 and stay there forever.

Let v be the variables in the following loop

$$\textbf{while } G(v) \textbf{ do } S(v) \textbf{ end}$$

$G(v)$ is the the condition of the loop, where the iteration continues until $G(v)$ becomes *false*. $S(v)$ is the body of loop where variables changed. Every execution of the loop body replaces the value v by a new value v'. Therefore, this loop terminates if

1. there is a variant function V from v to a well-founded set $V(v)$,
2. the variant decreases with respect to well-founded relation on every iteration of the loop i.e. $V(v') \prec V(v)$.

2.2.2 Termination in Event-B

In Event-B, one can engage in proving convergence of any set of events. Consider a set of event CE_i, $i \in 1, .., n$. Proving that CE_i converges is the same as proving the following loop terminates.

```
while guard of some $CE_i$ hold do
Fire one such $CE_i$ event.
end
```

In Event-B, it may be proved that an event converges by proposing a variant V and prove the following:

- The variant must be finite if it is a set expression or an integer variant must be natural number. Hence, the variant values produce a well-founded order.

$I(v)$ $J(v, w)$ $\vdash$ $finite(V(w))$	**FIN**

$I(v)$ $J(v, w)$ $H_i(u, w)$ $\vdash$ $V(w) \in \mathbb{N}$	**NAT**

  FIN is proved once for a model where I and J are invariants of abstract and conrete events respectively. On the other hand, **NAT** must be proved for each convergent event CE_i, where $H_i(u, w)$ is the guard of CE_i.

- The *convergent* event CE_i must decrease the numeric variant or narrow the set variant in each iteration.

$I(v)$ $J(v, w)$ $H_i(u, w)$ $\mathbf{T}_i(u, w, w')$ $\vdash$ $V(w') < V(w)$	**VAR**

$I(v)$ $J(v, w)$ $H_i(u, w)$ $\mathbf{T}_i(u, w, w')$ $\vdash$ $V(w') \subset V(w)$	**VAR**

The convergence of an event may be proved in a later refinement. In this case the event is *anticipated* and we must prove that anticipated events do not enlarge the variant by proving: $V(w') \leq V(w)$ or $V(w') \subseteq V(w)$. In this case a *lexicographic variant* is formed which will be explained in Chapter 3.

2.3 Almost-Certain Termination

2.3.1 Proving Almost-Certain Termination with Loop Variant

In probability theory, an event happens *almost-certainly* if it happens with probability one. If an event *certainly* terminates, then every path in the execution tree leads to termination. On the other hand, if there are some infinite paths, but the collective probability of the infinite paths is zero, then we say that this event almost-certainly terminates. For instance, if a coin is flipped often enough, then almost-certainly it will turn up heads. Although there is an infinite path which has no head, the probability of this infinite path is zero.

In Section 2.2.1, loop variant is used to prove termination of a loop. If loop variant decreases at each iteration of a loop, then the loop certainly terminates. It is proved in [10] that if a variant is bounded above and on each iteration the variant decreases with at least fixed probability $\epsilon > 0$, then the loop almost-certainly terminates. Therefore, the *probabilistic variant* is allowed to increase, but not above the upper bound.

For the same loop as defined in Section 2.2.1

$$\textbf{while } G(v) \textbf{ do } S(v) \textbf{ end}$$

the loop almost-certainly terminates if

- the variant function V from v to a well-founded set $V(v)$ is bounded above,
- the variant *may* (as in contrast to *must*) decrease with respect to well-founded relation on every iteration of the loop i.e. $\exists v' \cdot V(v') \prec V(v)$,
- there is finite number of choices for v' to guarantee probability of decrease $\epsilon > 0$.

2.3.2 Almost-Certain Termination in Event-B

In [6], qualitative probabilistic reasoning was introduced into Event-B. The purpose of qualitative probabilistic reasoning is to provide the concept of almost-certain termination. According to this work, the action of an event can be either *probabilistic* or *non-deterministic* (but not both). With respect to most proof obligations, a probabilistic action is treated identically as a nondeterministic action. However, it behaves angelically with respect to the convergence proof obligations. We will say that an event is *probabilistic*, if it almost-certainly terminates. It may be proved that an event almost-certainly converges by proving:

- A constant upper bound B[2] is required that dominates the probabilistic numeric variant $V(w)$.

$I(v)$ $J(v, w)$ $\vdash$ $V(w) \leq B$	**BND**

- A *probabilistic* event *may* decrease the probabilistic variant.

$I(v)$ $J(v, w)$ $H(u, w)$ $\vdash$ $\exists w' \cdot \mathbf{T}(u, w, w') \land V(w') < V(w)$	**PRV**

- Probabilistic action $\mathbf{T}(u, w, w')$ has finite number of possible choices for w'.

$I(v)$ $J(v, w)$ $H(u, w)$ $\vdash$ $finite(\{w' \mid \mathbf{T}(u, w, w')\})$	**FINACT**

In addition, the bound must be finite if it is a set expression. It needs another proof obligation **BFN**. Moreover, the proof obligation **NAT** or **FIN** is still necessary to prove almost-certain termination.

[2] In general, this could be a non-decreasing function on the state.

$I(v)$ $J(v, w)$ $\vdash$ $finite(B)$	**BFN**

Chapter 3

Termination with Refinement in Event-B

The use of refinement to represent systems at different abstraction levels is a key feature of Event-B. In Chapter 2, necessary proof obligations to prove termination of events in a machine is given. However, the convergence of several events in different abstraction levels must be formalised carefully. In this chapter, we discuss the collective convergence of several events, where they are convergent in different refinement levels using different variants. Section 3.1 briefly explains the notion of lexicographic variant. In section 3.2, we construct a lexicographic variant for set of convergent events to prove their collective convergence. In addition, we propose some restrictions on refinement and an additional condition on variant to preserve probabilistic termination property in Section 3.3.

3.1 Lexicographic Variant

In Section 2.2.1, the definition of loop variant is given. It is used to prove termination with using well-founded relations. Sometimes standard variants are inadequate for proving termination. Therefore, a *layered* variant is needed for algorithms exhibiting several kinds of behaviour.

If $\prec_1$ is a well-founded relation on a set S_1 and $\prec_2$ a well-founded relation on a set S_2, then their lexicographic combination $\prec_{lex}$ is defined by:

$$(x', y') \prec_{lex} (x, y) \Leftrightarrow (x' \prec_1 x) \vee (x' = x \wedge y' \prec_2 y).$$

This lexicographic combination $\prec_{lex}$ is a well-founded relation on the set $S_1 \times S_2$. More generally, lexicographic ordering can be defined on n-tuple of sets. Lexicographic ordering can be used in proving termination since it is well-founded. *Lexicographic variant* defined in [10] is n-tuple of variants such that $V = (V_1, ..., V_n)$ decreases at each iteration of loop with respect to lexicographic ordering.

3.2 Certain Termination with Refinement

Let E1 and E2 are two events in an abstract machine.

```
E1
  status  convergent
  any  $t$  where
    $G_1(t, v)$
  then
    $S_1(t, v)$
  end
```

```
E2
  status  anticipated
  any  $t$  where
    $G_2(t, v)$
  then
    $S_2(t, v)$
  end
```

E1 is convergent event decreasing $V_1(v)$ with respect to a well-founded order $\prec_{V_1}$ on set $VS = \{V_1(v) \mid I\,(v) \wedge \exists t \cdot G_1\,(t, v)\}$ such that $V_1(v') \prec_{V_1} V_1(v)$, and E2 is anticipated event which does not increase $V_1(v)$.

In concrete machine we have two refined events RE1 and RE2.

```
RE1
  refines  $E1$
  status  $convergent$
  any  $u$  where
    $H_1(u, w)$
  then
    $T_1(u, w)$
  end
```

```
RE2
  refines  $E2$
  status  $convergent$
  any  $u$  where
    $H_2(u, w)$
  then
    $T_2(u, w)$
  end
```

RE2 is convergent and decreases $V_2(w)$ with respect to a well-founded order $\prec_{V_2}$ such that $V_2(w') \prec_{V_2} V_2(w)$.

We know that both events converge seperately. We want to prove both RE1 and RE2 converge collectively. Hence, we need to genereate a lexicographic variant $LV\,(w)$ and a well-founded order to prove the convergence of concrete model where both events decrease the same variant.

Since variables v changed in the concrete machine we need to restate a variant and a well-founded order for RE1. We assume the variant as identity function for simplicity.

Definition 1. $\prec_w$ is an order such that
$w' \prec_w w \Leftrightarrow (\forall v \cdot I(v) \wedge J(v, w) \Rightarrow (\exists v' \cdot I(v') \wedge J(v', w') \wedge V_1(v') \prec_{V_1} V_1(v)))$

Lemma 1. $\prec_w$ is a well-founded order on set
$WS = \{w \mid \; (\exists v \cdot I\,(v) \wedge J\,(v, w)) \wedge (\exists u \cdot H_1\,(u, w))\}$.

Proof. We will prove the lemma by contradiction. Assume $\prec_w$ is not a well-founded order on WS. Hence, there must be an infinite descending sequence of elements. Let $w_0, w_1, ... \in WS$ such that $w_{n+1} \prec_w w_n, \forall n \in \mathbb{N}$.

If we start from the first pair, $w_1 \prec_w w_0 \Rightarrow \quad (\forall v_0 \cdot I\,(v_0) \wedge J\,(v_0, w_0) \Rightarrow \exists v_1 \cdot I\,(v_1) \wedge J\,(v_1, w_1) \wedge V_1(v_1) \prec_{V_1} V_1(v_0))$. Since $w_0 \in WS$ we know $\exists v_0 \cdot I\,(v_0) \wedge J\,(v_0, w_0)$ from the definition of the set WS. Hence, for this v_0 there exists v_1 such that $I\,(v_1) \wedge J\,(v_1, w_1) \wedge V_1(v_1) \prec_{V_1} V_1(v_0)$. Similarly, for this v_1 there exists v_2 such that $V_1(v_2) \prec_{V_1} V_1(v_1)$. As a result, we obtain an infinite descending sequence of elements $v_0, v_1, ...$ such that $V_1(v_{n+1}) \prec_{V_1} V_1(v_n), \forall n \in \mathbb{N}$. Obtaining an infinite descending chain is impossible if all $V_1(v_n) \in VS$, because $\prec_{V_1}$ is a well-founded order on set VS. Therefore, we can prove that all $V_1(v_n)$ belong to VS by proving the following sequent:

$$I(v_n), J(v_n, w_n), \exists u \cdot H_1(u, w_n) \;\vdash\; I(v_n) \wedge (\exists t \cdot G_1(t, v_n))$$

$$\boxed{\begin{array}{l} \quad I(v_n) \\ \quad J(v_n, w_n) \\ \quad \exists u \cdot H_1(u, w_n) \\ \vdash \\ \quad I(v_n) \wedge (\exists t \cdot G_1(t, v_n)) \end{array}} \;\; \mathbf{EX_L} \;\; \boxed{\begin{array}{l} \quad I(v_n) \\ \quad J(v_n, w_n) \\ \quad H_1(u, w_n) \\ \vdash \\ \quad I(v_n) \wedge (\exists t \cdot G_1(t, v_n)) \end{array}} \;\; \mathbf{AND_R}$$

$$\mathbf{1.} \; \boxed{\begin{array}{l} \quad I(v_n) \\ \quad J(v_n, w_n) \\ \quad H_1(u, w_n) \\ \vdash \\ \quad I(v_n) \end{array}} \;\; \mathbf{HYP}$$

$$\mathbf{2.}\ \boxed{\begin{array}{l}\quad I(v_n)\\ \quad J(v_n, w_n)\\ \quad H_1(u, w_n)\\ \vdash\\ \quad \exists t \cdot G_1(t, v_n)\end{array}}\ \mathbf{CUT}(\exists t, v'_n \cdot G_1(t, v_n) \wedge \mathbf{S}_1(t, v_n, v'_n) \wedge J(v'_n, w'_n))$$

$$\mathbf{2.1.}\ \boxed{\begin{array}{l}\quad I(v_n)\\ \quad J(v_n, w_n)\\ \quad H_1(u, w_n)\\ \vdash\\ \quad \exists t, v'_n \cdot (G_1(t, v_n) \wedge \mathbf{S}_1(t, v_n, v'_n)\\ \quad \wedge J(v'_n, w'_n))\end{array}}\ \mathbf{CUT}(\exists w'_n \cdot \mathbf{T}_1(u, w_n, w'_n))$$

$$\mathbf{2.2.}\ \boxed{\begin{array}{l}\quad I(v_n)\\ \quad J(v_n, w_n)\\ \quad H_1(u, w_n)\\ \quad \exists t, v' \cdot G_1(t, v_n) \wedge \mathbf{S}_1(t, v_n, v'_n) \wedge J(v'_n, w'_n)\\ \vdash\\ \quad \exists t \cdot G_1(t, v_n)\end{array}}\ \mathbf{EX_L}, \mathbf{EX_R}, \mathbf{HYP}$$

$$\mathbf{2.1.1.}\ \boxed{\begin{array}{l|c}\quad I(v_n) & \\ \quad J(v_n, w_n) & \\ \quad H_1(u, w_n) & \mathbf{FIS\_REF}\\ \vdash & \\ \quad \exists w'_n \cdot \mathbf{T}_1(u, w_n, w'_n) & \end{array}}$$

$$\textbf{2.1.2.}\ \boxed{\begin{array}{l} \quad I(v_n) \\ \quad J(v_n, w_n) \\ \quad H_1(u, w_n) \\ \quad \exists w_n' \cdot \mathbf{T}_1(u, w_n, w_n') \\ \vdash \\ \quad \exists t, v_n' \cdot G_1(t, v_n) \wedge \mathbf{S}_1(t, v_n, v_n') \wedge J(v_n', w_n') \end{array}}\ \mathbf{EX_L}$$

$$\boxed{\begin{array}{l|c} \quad I(v_n) & \\ \quad J(v_n, w_n) & \\ \quad H_1(u, w_n) & \mathbf{REF} \\ \quad \mathbf{T}_1(u, w_n, w_n') & \\ \vdash & \\ \quad \exists t, v_n' \cdot G_1(t, v_n) \wedge \mathbf{S}_1(t, v_n, v_n') \wedge J(v_n', w_n') & \end{array}}$$

□

Lemma 2. Concrete event RE1 decreases the newly established variant with respect to well-founded order $\prec_w$.

Proof. We must prove the following sequent:

$$I(v), J(v, w), H_1(u, w), \mathbf{T}_1(u, w, w') \ \vdash \ w' \prec_w w$$

With using the Definition 1, we get:

$I(v)$
$J(v,w)$
$H_1(u,w)$
$\mathbf{T}_1(u,w,w')$
$\vdash$
$\forall v \cdot I\,(v) \wedge J\,(v,w) \Rightarrow$
$(\exists v' \cdot I\,(v') \wedge J\,(v',w') \wedge V_1\,(v') \prec_{V_1} V_1\,(v))$

$\mathbf{ALL_R(v := x), IMP_R}$

$...$
$H_1(u,w)$
$\mathbf{T}_1(u,w,w')$
$I(x)$
$J(x,w)$
$\vdash$
$\exists v' \cdot (I\,(v') \wedge J\,(v',w')$
$\wedge V_1\,(v') \prec_{V_1} V_1\,(x))$

$\mathbf{CUT}(\exists t, v' \cdot G_1\,(t,x) \wedge \mathbf{S}_1\,(t,x,v') \wedge J(v',w'))$

1.	$...$ $H_1(u,w)$ $\mathbf{T}_1(u,w,w')$ $I(x)$ $J(x,w)$ $\vdash$ $\exists t, v' \cdot G_1\,(t,x) \wedge \mathbf{S}_1\,(t,x,v') \wedge J(v',w')$	**REF**

2.

...

$H_1(u, w)$

$\mathbf{T}_1(u, w, w')$

$I(x)$

$J(x, w)$

$\exists t, v' \cdot G_1\,(t, x) \wedge \mathbf{S}_1\,(t, x, v') \wedge J(v', w')$

$\vdash$

$\exists v' \cdot I\,(v') \wedge J\,(v', w') \wedge V_1\,(v') \prec_{V_1} V_1\,(x)$

$\mathbf{EX_L}, \mathbf{AND_L}$

...

$I(x)$

$J(x, w)$

$G_1\,(t, x)$

$\mathbf{S}_1\,(t, x, v')$

$J(v', w')$

$\vdash$

$\exists v' \cdot I\,(v') \wedge J\,(v', w') \wedge V_1\,(v') \prec_{V_1} V_1\,(x)$

$\mathbf{CUT}(I(v'))$

2.1.

...

$I(x)$

$G_1\,(t, x)$

$\mathbf{S}_1\,(t, x, v')$

$\vdash$

$I(v')$

INV

2.2.

$\ldots$
$I(x)$
$J(x,w)$
$G_1(t,x)$
$\mathbf{S}_1(t,x,v')$
$J(v',w')$
$I(v')$
$\vdash$
$\exists v' \cdot (I(v') \wedge J(v',w')$
$\wedge V_1(v') \prec_{V_1} V_1(x))$

CUT$(V_1(v') \prec_{V_1} V_1(x))$

2.2.1.

$\ldots$
$I(x)$
$G_1(t,x)$
$\mathbf{S}_1(t,x,v')$
$\vdash$
$V_1(v') \prec_{V_1} V_1(x)$

VAR

2.2.2.

$\ldots$
$J(v',w')$
$I(v')$
$V_1(v') \prec_{V_1} V_1(x)$
$\vdash$
$\exists v' \cdot I(v') \wedge J(v',w') \wedge V_1(v') \prec_{V_1} V_1(x)$

$\mathbf{EX_R}$

$$\boxed{\begin{array}{ll} & \ldots \\ & J(v', w') \\ & I(v') \\ & V_1\,(v') \prec_{V_1} V_1\,(x) \\ \vdash & \\ & I\,(v') \land J\,(v', w') \land V_1\,(v') \prec_{V_1} V_1\,(x) \end{array}} \quad \mathbf{AND_R}, \mathbf{HYP}$$

□

We proved that concrete event RE1 decreases the newly established variant (identity function). Similarly we can prove RE2 does not increase the newly established variant. In addition, we know RE2 decreases $V_2(w)$. Therefore, we can construct our lexicographic variant $LV(w) = (id, V_2(w))$, where both events decrease it. As a result, RE1 and RE2 converge collectively which guarantees the termination of iteration.

3.3 Almost-Certain Termination with Refinement

In Section 3.2, we proved the collective convergence of two convergent events in the case of refinement. However, one of the events may be probabilistic and we must consider the refinement of probabilistic event. To prove probabilistic convergence of an event, one must prove that this event may decrease the variant (**PRV**). Therefore, there exists a "good" choice for probabilistic action decreasing the variant. However, probabilistic convergence argument does not preserved by standard refinement since a *good* choice for termination can be removed accidentally. Hence, we require that after proving probabilistic convergence the probabilistic event must keep the *good* choice in later refinements. For this problem, we propose *superposition refinement* to preserve consistency of probabilistic events. Superposition refinement is a special case of refinement when the event and variable system is kept in the refinement.

For the same events of Section 3.2

```
E1
  any  $t$  where
    $G_1(t, v)$
  then
    $S_1(t, v)$
  end
```

```
E2
  any  $t$  where
    $G_2(t, v)$
  then
    $S_2(t, v)$
  end
```

```
RE1
  refines  $E1$
  any  $u$  where
    $H_1(u, v)$
  then
    $T_1(u, v)$
  end
```

```
RE2
  refines  $E2$
  any  $u$  where
    $H_2(u, v)$
  then
    $T_2(u, v)$
  end
```

we do not need to restate a variant for the first event since variables v are kept in the superposition refinement. Therefore, $V = (V_1, V_2)$ is the lexicographic variant of the model.

We consider the case of collective convergence of two events where at least one of them is probabilistic. Both events almost-certainly converges if the lexicographic variant is bounded above and events decrease the variant with at least fixed probability $\epsilon > 0$. We know that both convergent and probabilistic events decrease the variant with at least fixed probability $\epsilon > 0$. Hence, the only remaining condition is the upper bound for lexicographic variant. Since the variants for probabilistic events are bounded above, it must be guaranteed that all other variants are bounded above. Otherwise, a probabilistic event may take the control and increase the lexicographic variant forever.

As a result, we require that not only the variant concerning with the probabilistic events, but *all other variants need to be bounded above* as well. Furthermore, event splitting by having additional guards and event merging preserve the probabilistic convergence proofs, because they must have an identical action as mentioned in Section 2.1.2.

Chapter 4

Case Study: Rabin's Choice Coordination

Rabin's Choice Coordination algorithm as explained in [11] is an example of the use of probability for symmetry breaking. The Choice Coordinaton is a problem that processes $P_1, ..., P_n$ must reach a common choice out of k alternatives $A_1, ..., A_k$. It does not matter which alternative will be chosen at the end. The protocol uses k shared variables $v_1, ..., v_k$, one for each alternative. A process P_j arriving at A_i can access and modify v_i in one step without any interruption from other processes. The algorithm proposed by Rabin terminates with probability 1.

4.1 Description of the Problem and Algorithm

We will look at a simplified version of the problem and the corresponding algorithm as described by Morgan et. al. [10]. Instead of n processes and k alternatives we have n tourists and 2 destinations (which we call $LEFT$ and

RIGHT accordingly). We also distinguish the inside and outside for each place.

ENV 1	Each tourist can be in one of the following locations: *inside-left*, *inside-right*, *outside-left*, *outside-right*.

The possible movements of the tourists can be seen in Figure 4.1.

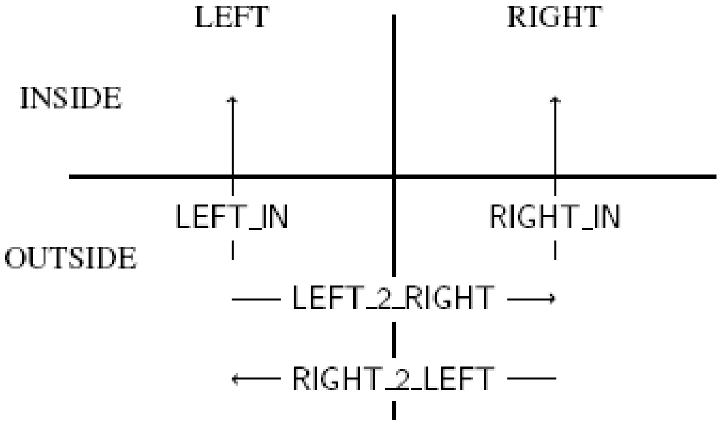


Figure 4.1: Possible places and movements of tourists

Each tourist can move between the two outside locations, i.e. from outside-left to outside-right and vice versa. Furthermore, a tourist can move from the outside to the inside of the same place, e.g. from outside-left to inside-left.

ENV 2	A tourist can move between the two outside locations.

ENV 3	A tourist can move from the outside to the inside of the same place.

Other movements of the tourists are forbidden, in particular if a tourist enters an inside place, he cannot change his place anymore.

ENV 4	A tourist in an inside place cannot change his location.

The purpose of the algorithm is to have all tourists to reach a common decision of entering the same place, without communicating directly with each other.

FUN 5	Eventually, all tourists enter the same place.

Rabin's Choice Coordination algorithm as described by Morgan et. al. in [10] is as follows. Each tourist carries a notepad and he can write a number on it. Moreover, there are two noticeboards on the outside-left and outside-right.

ALG 6	Each tourist has a notepad which he can write a number on it.

ALG 7	There are noticeboards at the outside-left and outside-right.

In the beginning, number 0 is written on all tourist notepads and on the two noticeboards. Initially, each tourist independently chooses LEFT or RIGHT place and go to the outside location of that place (i.e. outside-left or outside-right). Afterwards, a tourist at an outside location can alternate between different locations according the following algorithm.

ALG 8	• If there is any tourist inside, he enters this place. • Otherwise, he compares the number n on his notepad with the number N on the noticeboard. – If $N < n$, the tourist goes inside. – If $N > n$, the tourist replaces n with N on his notepad and goes to the outside of other place. – If $N = n$, the tourist tosses a coin. If the coin comes up head, the tourist sets N' to $N+2$. Otherwise, he sets N' to conjugate[1] of $N+2$. Then, he writes N' on the noticeboard and his notepad and goes to the outside of the other place.

We formalise Morgan et. al.'s proof [10] accordingly in Event-B in the next section.

[1] conjugate of a number n (denoted by $\overline{n}$) is defined to be $n+1$ if n is even and $n-1$ if n is odd.

4.2 Formal Development

In this section, we present the formal development of Rabin's choice coordination algorithm in Event-B[2]. Our proof method for reasoning about probabilistic termination has several important features as follows.

- To prove that eventually all tourists will end up in the same place, we follow the approach in [8] for reasoning about liveness properties, with the correctness argument combining appropriate proofs of event convergent (both standard and probabilistic) and deadlock-freeness.
- We first establish the full algorithm with several *anticipated* events, before converting them to *convergent*, either *standard* or *probabilistic*. The use of anticipated events is first hinted in [6].
- Finally, we rely on the fact that probabilistic convergence is preserved during the refinement. This holds for the type of refinement that we use after proving probabilistic convergence: We keep the variable and event system the same, only splitting and merging of events are allowed, with additional invariants added accordingly to prove the convergent properties of events. By separating the proof of convergence into several refinement, we indeed produce a lexicographic variant for all convergent events.

The details of refinement strategy for our development is as follows.

Initial Model Introduce the tourists inside the left *lin* and inside the right *rin* (Requirements **ENV 3** and **ENV 4**).

Refinement 1 Introduce the tourists outside the left *lout* and outside the right *rout* (Requirements **ENV 1** and **ENV 2**).

[2] The archive of the development can be found on-line [12].

Refinement 2 Introduce the noticeboard on the left L and on the right R and notepads for all tourists np (Requirements **ALG 6**, **ALG 7**). Rabin's algorithm is introduced (**ALG 8**).

Refinements 3–6 Complete the proofs of (probabilistic) convergence properties for events.

Refinement 7 Proving deadlock-freeness theorem.

The main property **FUN 5**, i.e. all tourists eventually end up in the same place, is a consequence of our convergence and deadlock-freeness theorems.

4.2.1 Initial Model. The Sets of Inside Tourists

We assume that there is a context with a *finite* carrier set T representing the set of tourists. In this initial model, we have two sets of tourists, namely *lin* and *rin*, representing those at the inside-left and inside-right accordingly. Note that invariant **inv0_3** states that at least one of the two locations is always empty. Initially, both variables are empty sets, since all tourists are outside.

variables: *lin*, *rin*

invariants:
inv0_1: $lin \subseteq T$
inv0_2: $rin \subseteq T$
inv0_3: $lin = \varnothing \lor rin = \varnothing$

```
init
  begin
    $lin := \varnothing$
    $rin := \varnothing$
  end
```

We have two events L_IN and R_IN to model the situation when a tourist enters the inside-left or inside-right accordingly (**ENV 3**). Moreover, there are no leaving events: a tourist once inside cannot change his location (**ENV 4**).

```
L_IN
  status  convergent
  any  $t$  where
    $rin = \varnothing$
    $t \notin lin$
  then
    $lin := lin \cup \{t\}$
  end
```

```
R_IN
  status  convergent
  any  $t$  where
    $lin = \varnothing$
    $t \notin rin$
  then
    $rin := rin \cup \{t\}$
  end
```

variant: $T \setminus (lin \cup rin)$

The two events are *convergent*, with the variant V_0 representing the set of tourists not inside the two places. Note that the variant V_0 here is bounded above by the set of tourists T which is finite.

The movement of tourists at this level can be seen in Figure 4.2, as at the moment, we do not distinguish the two outside locations.

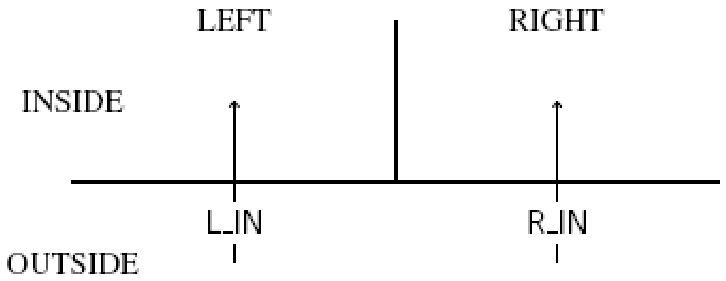


Figure 4.2: Movements of the tourists in the initial model

Finally, we have one *ordinary* event, namely final. This is an *observer* event (similar to what has been defined in [8]) in that sense that this does not change the state of model, but to *observe certain condition* about the state of the model. The observing condition is encoded as the guard of the events: If the event is enabled, the condition is satisfied. Here we are interested in the fact that all tourists will end up in the same place. Note that according to invariant **inv0_3**, if all tourists are in one place, the other place must be empty.

```
final
  when
    $rin = T \lor lin = T$
  then
    SKIP
  end
```

Further refinements keep event final unchanged and our goal is to prove that eventually event final is enabled. At the end of the development, beside event final we have a number of events $e_1, \ldots, e_n$. We will prove that all events $e_1, \ldots, e_n$ are convergent (standard or probabilistically). We must prove that the event system containing $e_1, \ldots, e_n$ and final are deadlock-free. According to the convergent argument, all events $e_1, \ldots, e_n$ will eventually converge, i.e. these events will be disable. Together with the deadlock-freeness argument, the only event that does not deadlock is final whose guard is $\top$.

4.2.2 Refinement 1. The Sets of Outside Tourists

There are two new variables *lout* and *rout* representing the tourists outside the two places. Invariant **inv1_1** states that a tourist cannot be at two locations at the same time, and each tourist must be in one of the location[3]. This corresponds to requirement **ENV 1**. Initially, some tourists decide to go to the outside-left and some tourists to the outside-right.

variables: $\ldots, lout, rout$

invariants:
inv1_1: $partition(T, lin, rin, lout, rout)$

[3] $partion(S, s_1, \ldots, s_n)$ means that subsets $s_1, \ldots, s_n$ together form a partition of S, i.e. they are pairwise disjoint and having their union is S.

```
init
  begin
    ...
    $lout, rout :\mid lout' = T \setminus rout'$
  end
```

There are two new events namely L_2_R and R_2_L to model the movement of a tourist between the two outside locations: from outside-left to outside-right and vice versa. This corresponds to the requirement **ENV 2**.

```
L_2_R
  status  anticipated
  any  $t$  where
    $t \in lout$
    $lin = \varnothing$
  then
    $rout := rout \cup \{t\}$
    $lout := lout \setminus \{t\}$
  end
```

```
R_2_L
  status  anticipated
  any  $t$  where
    $t \in rout$
    $rin = \varnothing$
  then
    $lout := lout \cup \{t\}$
    $rout := rout \setminus \{t\}$
  end
```

The guards $lin = \varnothing$ and $rin = \varnothing$ state that the tourists can only alternate between the outside locations if there is no one inside. This is a part of the algorithm described by requirement **ALG 8**. The two new events only modify new variables *rout*, *lout* hence clearly refine SKIP. Moreover, invariant **inv1_1** is preserved since the events only change the location for one particular tourist from outside-left to outside-right and vice versa. These events are *anticipated* at the moment, we will consider their convergence property in subsequent refinements.

Events L_IN and R_IN are refined accordingly to take into account the new variables. Since the events corresponding to the $LEFT$-place and the $RIGHT$-place are symmetric, from now on, we present only events corresponding to the $LEFT$-place. The refinement of event L_IN is as follows.

```
(abstract_)L_IN
  any  $t$  where
    $rin = \varnothing$
    $\underline{t \notin lin}$
  then
    $lin := lin \cup \{t\}$
  end
```

```
(concrete_)L_IN
  any  $t$  where
    $rin = \varnothing$
    $\underline{t \in lout}$
  then
    $lin, lout := lin \cup \{t\}, lout \setminus \{t\}$
  end
```

Note that the guard strengthening proof obligation **GRD** follows from the fact that a tourist can only be in one location at a time (invariant **inv1_1**). The assigned expressions to old variable *lin* are the same in both abstract and concrete events. Moreover, invariant **inv1_1** is maintained since the event merely moves a tourist from one location (outside-left) to another (inside-left). The movement of the tourists is now complete as in Figure 4.1.

4.2.3 Refinement 2. Rabin's Algorithm

We introduce the two noticeboards outside the places and the tourists' notepads where they can write some number on it. Initially, number 0 is written on the noticeboards and all the notepads. This corresponds to the requirements **ALG 6** and **ALG 7**.

variables: $\ldots, L, R, np$

```
invariants:
  inv2_1:  $L \in \mathbb{N}$
  inv2_2:  $R \in \mathbb{N}$
  inv2_3:  $np \in T \rightarrow \mathbb{N}$
  inv2_4:  $\forall x \cdot x \in lout \Rightarrow np(x) \leq R$
  inv2_5:  $\forall x \cdot x \in lin \Rightarrow np(x) \leq R$
  inv2_6:  $\forall x \cdot x \in rout \Rightarrow np(x) \leq L$
  inv2_7:  $\forall x \cdot x \in rin \Rightarrow np(x) \leq L$
  inv2_8:  $lin \neq \varnothing \Rightarrow (\exists x \cdot x \in lin \wedge np(x) > L)$
  inv2_9:  $rin \neq \varnothing \Rightarrow (\exists x \cdot x \in rin \wedge np(x) > R)$
```

```
init
  begin
    $\ldots$
    $L := 0$
    $R := 0$
    $np := T \times \{0\}$
  end
```

Invariants **inv2_4**, **inv2_5**, **inv2_6** and **inv2_7** states that every tourist at the $LEFT$-place (respectively $RIGHT$-place) carrying a number smaller than or equal to the right-noticeboard (respectively left-noticeboard). Invariants **inv2_8** and **inv2_9** states that if there is somebody at the inside-left (respectively inside-right), there exist a tourist at the inside-left (respectively inside-right) whose notepad is greater than L (respectively R). We will prove these invariants are maintained by all events later on.

We can now specify under which condition a tourist can move from one location to another. Events modelling the movement of a tourist from an outside location to an inside location are guard-strengthened. The refinement of event L_IN is as follows.

```
(abstract_)L_IN
  any  $t$  where
    $\underline{rin = \varnothing}$
    $t \in lout$
  then
    $lin, lout := lin \cup \{t\}, lout \setminus \{t\}$
  end
```

```
(concrete_)L_IN
  any  $t$  where
    $t \in lout$
    $\underline{L < np(t) \vee lin \neq \varnothing}$
  then
    $lin, lout := lin \cup \{t\}, lout \setminus \{t\}$
  end
```

The guard $L < np(t) \vee lin \neq \varnothing$ states that a tourist t can move inside the left place only if the number on his notepad is greater than the number on the left-noticeboard or if there is already someone at inside-left. To prove guard strengthening proof obligation (**GRD**), we must prove that the concrete guard implies the abstract one. If $lin \neq \varnothing$, then from **inv0_3** rin must be empty. If $L < np(t)$, from **inv2_4** we obtain $np(t) \leq R$. Hence, $L < R$. If $L < R$, then rin must be empty. Otherwise using **inv2_7** and **inv2_9** we obtain $L > R$ which is a contradiction.

For events modelling the movement of a tourist between two outside locations, there are two different cases. The events corresponding to the movement of a tourist from the $LEFT$ to $RIGHT$ are modelled by two events L_2_R_EQ and L_2_R_NEQ depending on if the number on the tourist notepad is equal or strictly smaller than the number on the noticeboard. Using $\overline{n}$ for the conjugate number of n, the two events are as follows.

```
L_2_R_NEQ
  refines L_2_R
  status anticipated
  any $t$ where
    ...
    $np(t) < L$
  then
    ...
    $np(t) := L$
  end
```

```
L_2_R_EQ
  refines L_2_R
  status anticipated
  any $t$ where
    ...
    $np(t) = L$
  then
    ...
    $L, np :\mid L' \in \{L + 2, \overline{L + 2}\} \wedge np' = np \mathbin{<\!\!+} \{t \mapsto L'\}$
  end
```

The actions of the above events update the tourist notepad and the noticeboard accordingly. Note that both events are refinements of the original event L_2_R, i.e. the original event is *split* into two cases. Similar events, used to model the movement of a tourist from $RIGHT$ to $LEFT$, are omitted here. Note that these events model the movement of a tourist according to requirement **ALG 8**, with the exception that we use nondeterministic choice at the moment in L_2_R_EQ. This is an abstraction of the actual probabilistic implementation (i.e. coin tossing), which we will introduce when necessary.

Now, we can prove the correctness of the declared invariants.

invariants:
inv2_4: $\forall x \cdot x \in \mathit{lout} \Rightarrow \mathit{np}(x) \leq R$
inv2_5: $\forall x \cdot x \in \mathit{lin} \Rightarrow \mathit{np}(x) \leq R$
inv2_6: $\forall x \cdot x \in \mathit{rout} \Rightarrow \mathit{np}(x) \leq L$
inv2_7: $\forall x \cdot x \in \mathit{rin} \Rightarrow \mathit{np}(x) \leq L$

Initially, all invariants are correct, because the value on all notepads and noticeboards is 0. L and R never decrease. Notepads can only increase in L_2_R and R_2_L events. If a tourist moves from LEFT to RIGHT, he writes L on his notepad which does not falsify the invariants. Similarly, a movement form RIGHT to LEFT does not falsify the invariants. Therefore, invariants hold forever.

invariants:
inv2_8: $\mathit{lin} \neq \varnothing \Rightarrow (\exists x \cdot x \in \mathit{lin} \wedge \mathit{np}(x) > L)$
inv2_9: $\mathit{rin} \neq \varnothing \Rightarrow (\exists x \cdot x \in \mathit{rin} \wedge \mathit{np}(x) > R)$

A tourist can enter a place if the value on his notepad is greater than noticeboard or if there is anybody inside. Hence, when the first tourist enters inside, it means that the value on his notepad is greater than noticeboard. If we assume a tourist entered LEFT place, then L will never increase anymore, because it increases only in L_2_R_EQ event and this event has a guard $\mathit{plin} = \varnothing$ which is not correct anymore.

Up to this refinement model we have modelled all the requirements except for **FUN 5**. In other words, we have established the model of the problem and the algorithm. Subsequent refinements are dedicated to prove the main property of the algorithm, i.e. eventually all tourists end up in the same place.

4.2.4 Refinements 3–6. Proving Convergence

Recall in the previous model, we have an *ordinary* event final, two *convergent* events, namely L_IN and R_IN, and *anticipated* events L_2_R_NEQ, L_2_R_EQ, R_2_L_NEQ and R_2_L_EQ. In this section, we describe our proof of (probabilistic) convergence of the anticipated events. We formalise the variant that has been proposed in [10]. The variant is a lexicographic one, with two layers: the outer layer (with higher priority) deals with the changes of L and R, the inner layer (with lower priority) deals with the tourists' movements. The components of the lexicographic variant will be spread over several refinement steps, thus help us to simplify the proof of convergence.

Outer Layer

We compare the values of L and R and notice how they can be varied. In order to understand the variant at this layer, we look at the definition of conjugate numbers. We separate the set of natural numbers into pairs:

$$(0, 1) \mid (2, 3) \mid (4, 5) \mid (6, 7) \mid \ldots$$

For each pair, a number is the conjugate of the other number in the pair and vice versa (e.g. $\overline{4} = 5$, $\overline{5} = 4$). The even number of each pair is also the minimum of the two (e.g. $\widetilde{4} = \widetilde{5} = 4$). We will refer to this splitting of natural numbers later in our reasoning. We reason about the outer variant in two refinement steps.

Refinement 3. We have the following invariants about the relationship between L and R. Below, we use the notation $\widetilde{n}$ to denote the minimum of n and its conjugate $\overline{n}$.

invariants:

inv3_1: $\widetilde{L} - \widetilde{R} \in \{-2, 0, 2\}$

inv3_2: $\overline{L} \notin np[rout]$

inv3_3: $\overline{R} \notin np[lout]$

Invariant **inv3_2** (respectively **inv3_3**) states that there is no tourist at the outside-right (respectively outside-left) carrying the number which is the conjugate of the number on the left-noticeboard (respectively right-noticeboard). Invariant **inv3_2** initially holds. It can be falsified only when a tourist goes from left to right or when L increases. When a tourist t goes from left to right, he writes L on his notepad. Since $np(t) = L \neq \overline{L}$, invariant still holds. Then invariant can only be falsified when L increases. When L increases, it will be equal to $L + 2$ or $\overline{L + 2}$. It is obvious that both values are greater than $\overline{L}$. From **inv2_6**, $\forall x \cdot x \in rout \Rightarrow np(x) \leq L$. Since all tourists at outside-right cannot have a greater notepad value than L, we can say that $\overline{L} \notin np[rout]$ when L increases. Proof of **inv3_3** is analogous.

Invariant **inv3_1** states that the values of the two noticeboards cannot be “too far apart”. Referring to the splitting of natural numbers into pairs, this invariant states that L and R must be in a same pair or in two adjacent pairs. It holds initially and can be falsified when noticeboard values change (i.e. L_2_R_EQ and R_2_L_EQ increase the noticeboard values). For L_2_R_EQ, $np(t) = L$. From the invariant **inv2_4**, we obtain $L \leq R$. Hence, $\widetilde{L} \leq \widetilde{R}$. It means that L is in same pair with R or L is in lower adjacent pair. When we increment L, $\widetilde{L} - \widetilde{R}$ can be at most 2. Therefore, invariant still holds. The proof is similar for R_2_L_EQ.

Note that when $\widetilde{L} = \widetilde{R}$, i.e. they are in a same pair, there can be two cases, either $L = R$ or $L = \overline{R}$ (equivalently $R = \overline{L}$). We can distinguish the relationship between L and R in three different cases: either $\widetilde{L} - \widetilde{R} \in \{-2, 2\}$ or $L = \overline{R}$ or $L = R$. Our variant is based on this relationship.

Refinement 4. For the outer variant, we first define the following constant function rE as follows

axioms:

rE_1: $rE \in \mathbb{N} \times \mathbb{N} \mapsto \{0, 1, 2\}$

rE_2: $\forall l, r \cdot l \mapsto r \in \mathrm{dom}(rE) \Leftrightarrow \widetilde{l} - \widetilde{r} \in \{-2, 0, 2\}$

rE_3: $\forall l, r \cdot l \in \mathbb{N} \land l = r \Rightarrow rE(l \mapsto r) = 2$

rE_4: $\forall l, r \cdot l \in \mathbb{N} \land l = \overline{r} \Rightarrow rE(l \mapsto r) = 0$

rE_5: $\forall l, r \cdot l \in \mathbb{N} \land \widetilde{l} - \widetilde{r} \in \{-2, 2\} \Rightarrow rE(l \mapsto r) = 1$

and define the variant V_1 as $rE(L \mapsto R)$ with an upper bound of 2. We reason about the variant V_1 as follows. We split event L_2_R_EQ into three different cases, depending on the current value of $rE(L \mapsto R)$.

L_2_R_EQ_0
  refines L_2_R_EQ
  status *convergent*
  any t **where**
    $t \in lout$
    $lin = \varnothing$
    $np(t) = L$
    $\underline{rE(L \mapsto R) = 0}$
  then
    $\ldots$
  end

L_2_R_EQ_1
  refines L_2_R_EQ
  status *probabilistic*
  any t **where**
    $t \in lout$
    $lin = \varnothing$
    $np(t) = L$
    $\underline{rE(L \mapsto R) = 1}$
  then
    $\ldots$
  end

L_2_R_EQ_2
  refines L_2_R_EQ
  status *convergent*
  any t **where**
    $t \in lout$
    $lin = \varnothing$
    $np(t) = L$
    $\underline{rE(L \mapsto R) = 2}$
  then
    $\ldots$
  end

We prove that L_2_R_EQ_0, L_2_R_EQ_2 are *convergent*, and L_2_R_EQ_1 is *probabilistically convergent* whereas L_2_R_NEQ is *anticipated* (which will be convergent with using the inner variant). The convergence attribute for the events corresponding to the $RIGHT$ are symmetric.

- First of all, we need to prove that the variant is bounded above (**BND**) by the declared upper bound. It is trivial since by definition, $rE(L \mapsto R) \leq 2$.

- For L_2_R_EQ_0, this corresponds to the case that never happens, since we have $rE(L \mapsto R) = 0$, i.e. $L = \overline{R}$, hence $np(t) = \overline{R}$. However, since $t \in lout$ and according to invariant **inv3_3**, we have $\overline{R} \notin np[lout]$ which is a contradiction. In other words, the guard of L_2_R_EQ_0 can be used to derive $\perp$. Hence, anything can be proved under the assumption of the guard of this events, including convergence proof.

- For L_2_R_EQ_2, we have $rE(L \mapsto R) = 2$, i.e. $L = R$. The action will change L to either $L + 2$ or $\overline{L+2}$, and keep R the same, hence the new value L' will be different from R', hence $rE(L' \mapsto R') \neq 2$ which is less than $rE(L \mapsto R)$. As a result, the variant V_1 is decreased, hence the event satisfies **VAR**.

- For L_2_R_NEQ, it does not change the value of L or R, hence the value of V_1 stays the same, i.e. non-increasing.

- For L_2_R_EQ_1, firstly we have that the possible alternatives of the after states are finite (2 in this case) hence the event satisfies **FINACT**. Secondly, we prove that the event *may* decrease the variant V_1, i.e. satisfies **PRV**. The actual proof obligation (with some simplifications by removing unnecessary hypotheses) is as follows.

$$
\begin{array}{l}
\quad rE(L \mapsto R) = 1 \\
\quad \forall x \cdot x \in lout \Rightarrow np(x) \leq R \\
\quad t \in lout \\
\quad np(t) = L \\
\vdash \\
\quad \exists L', np' \cdot L' \in \{L + 2, \overline{L+2}\} \land np' = np \mathbin{<\!\!\!+} \{t \mapsto L'\} \land \\
\quad rE(L' \mapsto R) < rE(L \mapsto R)
\end{array}
$$

We know that $\widetilde{L} - \widetilde{R} \in \{-2, 2\}$ from $rE(L \mapsto R) = 1$. In particular, from invariant **inv2_4**, i.e. $\forall x \cdot x \in lout \Rightarrow np(x) \leq R$, and from event's guards $t \in lout$ and $np(t) = L$, we have that $L \leq R$ hence $\widetilde{L} - \widetilde{R}$ must be -2. Referring to the splitting of natural numbers into pairs, when we have $\widetilde{L} - \widetilde{R} = -2$, it means that L is in one pair and

R is in the next higher adjacent pair, for example, if L is either 2 or 3 then R is either 4 or 5. The meaning of the action assigning L' to either $L + 2$ or $\overline{L + 2}$ is to have L' to be in the same pair as R, hence one of the alternative will satisfy condition $L' = \overline{R}$. For this case, $rE(L' \mapsto R) = 0 < 1 = rE(L \mapsto R)$. As a result, we have proved that L_2_R_EQ_1 *may* decrease the variant V_1.

Inner Layer

The variant for the inner layer is used to prove the convergence property of events L_2_R_NEQ and R_2_L_NEQ. This is done in two refinement steps. The variants that we used are simpler than the variants that has been proposed in [10], because Event-B allows to use set expressions as variant.

Refinement 5. We prove that L_2_R_NEQ converges and R_2_L_NEQ is anticipated with the variant V_2 defined to be $\{t \mid np(t) < L\}$, i.e. the set of tourists carrying a number on strictly smaller than the left-noticeboard. Event L_2_R_NEQ changes the value of a tourist notepad from *strictly less* to *equal to* L hence it decreases V_2. Event R_2_L_NEQ increase the value of a tourist notepad, hence it *cannot increase* V_2.

Refinement 6. In the second step, we prove that R_2_L_NEQ converges with a symmetric variant V_3 to be $\{t \mid np(t) < R\}$ following similar reasoning as above.

Note that both variant V_2 and V_3 are bounded above by the finite set of tourists T.

Summary

The summary of convergence property for events according to each refinement is in Table 4.1 (we skip the refinement level where nothing change to the convergence property). In the initial model we have two convergent events L_IN and R_IN. In refinement 1, we introduced two anticipated events L_2_R and R_2_L. As a result of splitting these two anticipated events, we obtained four anticipated events (i.e. L_2_R_EQ, L_2_R_NEQ, R_2_L_EQ and R_2_L_NEQ) in second refinement. In third refinement, only some additional invariants are introduced. In refinement 4, we proved the (probabilistic) convergence of L_2_R_EQ and R_2_L_EQ events by splitting events. In the fifth and sixth refinements, convergence of L_2_R_NEQ and R_2_L_NEQ events are proved respectively.

	L_IN	R_IN	L_2_R				R_2_L			
I.M.	*conv.*	*conv.*	–				–			
R.1	*conv.*	*conv.*	*anticipated*				*anticipated*			
	L_IN	R_IN	EQ_0	EQ_1	EQ_2	NEQ	EQ_0	EQ_1	EQ_2	NEQ
R.4	*conv.*	*conv.*	*conv.*	*prob.*	*conv.*	*anti.*	*conv.*	*prob.*	*conv.*	*anti.*
R.5	*conv.*	*conv.*	*conv.*	*prob.*	*conv.*	*conv.*	*conv.*	*prob.*	*conv.*	*anti.*
R.6	*conv.*	*conv.*	*conv.*	*prob.*	*conv.*	*conv.*	*conv.*	*prob.*	*conv.*	*conv.*

Table 4.1: Summary of event convergence

At the end, $V = (V_0, V_1, V_2, V_3)$ is the lexicographic variant of the model with an upper bound $U = (T, 2, T, T)$. All events except final decrease the lexicographic variant V with at least fixed probability $\epsilon > 0$. Therefore, the set of convergent events (standard and probabilistic) will almost-certainly terminate.

4.2.5 Refinement 7. Deadlock-freeness

In this final refinement, we merge the events that have been split earlier together, i.e. L_2_R_EQ and R_2_L_EQ. Combining the convergent attribute of the sub-events, we have now that these two events are probabilistic convergent. We add a theorem to prove that our system at this point is deadlock-free, i.e. the disjunction of all guards always holds.

$$\begin{array}{ll}
 & (rin = T \vee lin = T) \\
\vee & (\exists t \cdot t \in lout \wedge (L < np(t) \vee lin \neq \varnothing)) \\
\vee & (\exists t \cdot t \in rout \wedge (R < np(t) \vee rin \neq \varnothing)) \\
\vee & (\exists t \cdot t \in lout \wedge lin = \varnothing \wedge np(t) = L) \\
\vee & (\exists t \cdot t \in lout \wedge lin = \varnothing \wedge np(t) < L) \\
\vee & (\exists t \cdot t \in rout \wedge rin = \varnothing \wedge np(t) = R) \\
\vee & (\exists t \cdot t \in rout \wedge rin = \varnothing \wedge np(t) < R)
\end{array}$$

The proof of deadlock-freeness can be completed by contradiction. If the first guard is false, then there must be a tourist outside. Let $\exists t \cdot t \in lout$ (symmetric with *rout*), then one of the guards of L_IN, L_2_R_NEQ and L_2_R_EQ must be true. If the the guard of L_IN is false, then we obtain $L \geq np(t) \wedge lin = \varnothing$. Therefore, one of the guards of L_2_R_NEQ and L_2_R_EQ must be true.

Together with the proof of convergence earlier, we can now ensure that our system satisfies requirement **FUN 5**. Our reasoning is based on the approach in [8] and is as follows. At the last model, we have the following events: Event final which is *ordinary*, events L_IN, R_IN, L_2_R_NEQ, R_2_L_NEQ which are *convergent* and events L_2_R_EQ and R_2_L_EQ which are *probabilistically convergent*. Because of the convergence proof, we ensure that together the set of convergent events (standard and probabilistic) will terminate (being disabled) with probability 1. Moreover, because of the deadlock-freeness proof, when the convergent events are disabled, event final is the only one left, and must be enabled, i.e. all tourists are in the same place.

4.2.6 Proof Statistics

The statistics for our proofs is in Table 4.2. A large number of manual proofs is in models for proving the outer variant and in the second refinement, since we need several additional supporting invariants. In particular, in order to prove obligations related to the outer variant, we decided to split the events L_2_R_EQ and R_2_L_EQ into different cases. As a result, we have more proof obligations, which are simpler to prove. As an alternative, we can do the split while proving, i.e. to do *proof by cases*, without splitting the events. This will reduce the number of proof obligations, however, it hides the termination argument inside the proofs and they become more complicated. Our development is more intuitive, with the correctness being easier to observe by splitting the events accordingly.

Model	Total	Auto. (%)	Man. (%)
Initial model	6	6 (100%)	0 (N/A)
1st Refinement	8	7 (88%)	1 (12%)
2nd Refinement	63	49 (78%)	14 (22%)
Outer variant	54	29 (54%)	25 (46%)
Inner variant	11	8 (73%)	3 (27%)
Deadlock-freeness	4	0 (N/A)	4 (100%)
Total	146	99 (68%)	47 (32%)

Table 4.2: Proof statistics

Chapter 5

Tool Support

The Rodin tool [4] is an industrial-strength tool for creating and analyzing Event-B models. It is an Eclipse-based IDE and extensible with plugins. Therefore, it can be adapted easily to different application domains and development methods.

The Rodin tool chain consists of three major components: the static checker (SC), the proof obligation generator (POG), and the proof obligation manager (POM). The static checker analyses Event-B contexts and machines and provides feedback to users about syntactical and typing errors in them. The proof obligation generator generates proof obligations many of which have been outlined in Chapter 2. The proof obligation manager maintains the proof status and the proofs associated with the obligations.

For tool support, we introduced the notion of probabilistic event in Section 5.1. In addition, we created bound element mentioned in Section 5.2 to check if the variant is bounded above for probabilistic events. Section 5.3 and Section 5.4 dedicate to extention of the SC and the POG, respectively.[1] Section 5.5 explains activating SC and POG extensions.

[1] Detailed information for extending Rodin Platform can be found in [1]

5.1 Introducing Probabilistic Events

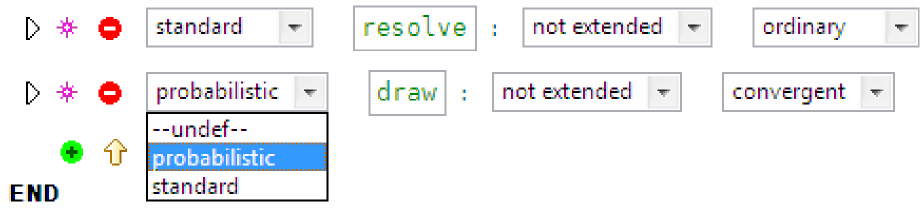
The convergence of an event in Event-B is denoted by the keyword *status* with three possible values: *convergent*, *anticipated*, and *ordinary*. To integrate qualitative probabilistic reasoning into Event-B, we introduced a new type of event called *probabilistic* in Section 2.3.2. A probabilistic event is only treated differently from standard event when it comes to convergence proof obligation.

There are different ways to extend Rodin Database. Rodin Database includes elements (e.g. event, variant) and attributes related to elements (e.g. convergence attribute of events). To introduce probabilistic event we added an attribute called *probabilistic*. `org.rodinp.core.attributeTypes` is the extension point to declare a new attribute. The value of this attribute is either *probabilistic* or *standard*. According to its convergence value and its probabilistic value the status of an event can have four possible values: convergent, probabilistic, anticipated and ordinary. Table 5.1 shows the status of an event with all possible convergence and probabilistic values.

Prob. Attr. / Conv. Attr.	Ordinary	Anticipated	Convergent
Standard	*Ordinary*	*Anticipated*	*Convergent*
Probabilistic	`Warning`	`Warning`	*Probabilistic*

Table 5.1: Types of events

The structure editor is also needed to be extended for displaying and editing the probabilistic attribute by users. `org.eventb.ui.editorItems` is the extension point for adding a new *choice attribute* for probabilistic attribute. In addition, our new probabilistic attribute must be related with an element. Hence, an *attribute relation* created between the *Event* element (`org.eventb.core.event`) and probabilistic attribute. As a result, each event has probabilistic attribute which can be chosen probabilistic or standard from the editor (Figure 5.1).



standard resolve : not extended ordinary

probabilistic draw : not extended convergent

Figure 5.1: Choosing probabilistic attribute

5.2 Bound Element

As mentioned earlier, the variant must be bounded above to prove almost-certain termination. Therefore, we need to add a new internal element called *bound* from the extension point `org.rodinp.core.internalElementTypes`.

Editing and displaying the bound element requires an extension of structure editor, where `org.eventb.ui.editorItems` is the related extension point. Firstly, we added an editor *element* for bound connected with newly defined internal element. The bound can be a natural number or set to be consistent with the variant. Hence, we have not created a new attribute for bound. We used a *text attribute* called *expression* (`org.eventb.ui.expression`) which is a predefined attribute for variant. An *attribute relation* created between the bound element and expression attribute. Finally, the bound element must be connected with machines. Therefore, a *child relation* created between the machine file (`org.eventb.core.machineFile`) and the bound element, where bound is a child of machine. The relationhip diagram for created child relations and attribute relations can be seen in Figure 5.2. Therefore, the bound element can be specified using structure editor (Figure 5.3).

5.3 Extending The Static Checker

As mentioned before, the SC analyses Event-B contexts and machines and provides feedback to users about syntactical and typing errors in them. Our

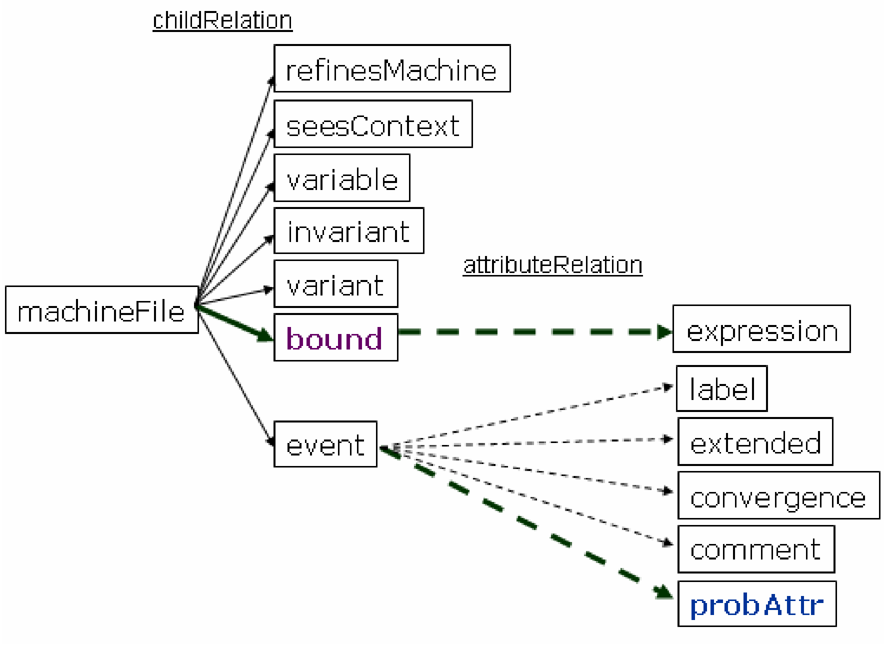


Figure 5.2: Relationship diagram of editor items

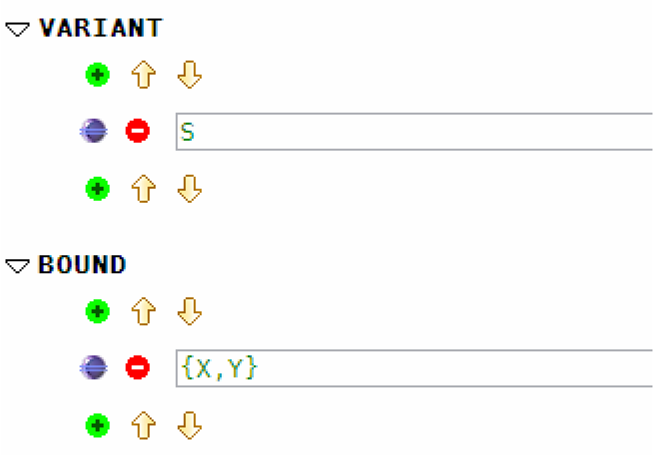


Figure 5.3: Specifying a bound

additions to the Rodin Database are probabilistic attribute and bound element. Hence, they must be checked statically and syntactical and typing errors must be reported to user.

Firstly, new internal element called *SC Bound* is added using the extension `org.rodinp.core.internalElementTypes`. SC bound is a bound that has been statically checked and ready to use in proof obligations.

Secondly, *Bound Information* is added as a SC *state*. The static check of the elements in a file are independent of each other (different and independent modules). Since the element checks are independent, the solution is to *share data* through states implemented using the extension point `org.eventb.core.scStateTypes`. Therefore, we can use the added state in the SC modules.

At this point, we can check the probabilistic attribute and bound element using SC *modules*. There are three types of modules: *filter*, *processor*, and *root*. `org.eventb.core.scModuleTypes` is the extension point for adding a module. We used a filter and a processor in our tool support. Filters are used to validate inserted elements. After the successful validation of all elements, they can be processed and stored in the statically-checked file. Processors literally process the elements, storing them in the static checked file, running sub-processors and adding states to the repository if required.

In the filter, the following conditions are checked for probabilistic events:

- Ordinary and anticipated events cannot be probabilistic. SC gives a warning and assumes ordinary or anticipated, respectively.

- There must be a bound if there is a probabilistic event and a variant in machine. Otherwise, SC gives an error.

- If abstract event is probabilistic, the concrete one must be probabilistic. Otherwise, SC gives an error.

- Probabilistic event can refine a probabilistic event or an anticipated event. Otherwise, SC gives an error.
- The result of merging a probabilistic event and a convergent event is a probabilistic event. Otherwise, SC gives an error.
- Each event must have a probabilistic attribute value. Otherwise, SC gives a warning and assumes standard.

In the processor, the following conditions are checked for bound:

- The bound must be of the same typed (i.e. integer or set) as the declared variant. In addition, it must be constant if it is an integer. Otherwise, SC gives an error.
- The bound is unnecessary if there is no probabilistic event or no variant. Otherwise, SC gives a warning.
- There cannot be more than one bound. Otherwise, SC gives an error.
- The bound must be well-formed (e.g. it cannot include an undeclared identifier). Well-formedness is defined in terms of the syntax of the mathematical language, dependencies between modelling elements, and type-correctness of all formulas and declared identifiers. Otherwise, SC gives an error.

5.4 Extending The Proof Obligation Generator

The function of the POG in the Rodin tool chain is to generate proof obligations with using the output of the SC. The POG does not check the output of the SC which are well-formed elements. In Section 2.3.2, necessary proof obligations to integrate qualitative probabilistic reasoning into Event-B are given. We extended the POG in accordance with required proof obligations.

Firstly, *Machine Bound Information* is added as a POG *state*. The purpose of using POG state is similar with SC state mentioned in Section 5.3. `org.eventb.core.pogStateTypes` is the related extension point.

Secondly, we added 3 POG *modules* to generate necessary proof obligations. `org.eventb.core.pogModuleTypes` is the extension point for adding a POG module. Following proof obligations are generated by newly created POG modules:

- First module generates the **BWD** and **BFN** proof obligations. **BWD** is the proof obligation for well-definedness of the bound. **BFN** is generated in a machine to prove finiteness of the bound if it is a set.

$$\boxed{\begin{array}{ll|c} & I(v) & \\ & J(v,w) & \textbf{BFN} \\ \vdash & & \\ & finite(B) & \end{array}}$$

- Second module generates the **BND** proof obligation if there is a bound, variant and a probabilistic event. **BND** is the proof obligation to prove that the bound dominates the variant.

$$\boxed{\begin{array}{ll|c} & I(v) & \\ & J(v,w) & \textbf{BND} \\ \vdash & & \\ & V(w) \leq B & \end{array}}$$

- Third module generates the **VAR** if the event is convergent or anticipated. If the event is probabilistic, the module generates **PRV**. In the **VAR**, event *must* decrease (or not increase if the event is anticipated) the variant, whereas it *may* decrease the variant in the **PRV**.

$$\boxed{\begin{array}{ll|c} & I(v) & \\ & J(v,w) & \\ & H(u,w) & \mathbf{VAR} \\ & \mathbf{T}(u,w,w') & \\ \vdash & & \\ & V(w') < V(w) & \end{array}}$$

$$\boxed{\begin{array}{ll|c} & I(v) & \\ & J(v,w) & \\ & H(u,w) & \mathbf{PRV} \\ \vdash & & \\ & \exists w' \cdot \mathbf{T}(u,w,w') \land V(w') < V(w) & \end{array}}$$

5.5 Configuration

The configuration is used to define which modules are used by the SC and POG. Each Event-B component has a configuration and the standard configuration is `org.eventb.core.fwd`. In our configuration we added all SC and POG modules in the standard configuration except the POG module producing **VAR**, because our third POG module produces this proof obligation. In addition, we added the SC and POG modules which we created. The extension point is `org.eventb.core.configurations` for adding a module into configuration. As a result, to use our tool, one must change the standard configuration (`org.eventb.core.fwd`) into our probabilistic configuration namely `ch.ethz.eventb.quantprob.probconf`.

Chapter 6

Tool Usage with Examples

In this chapter, we consider two example algorithms which terminate almost-certainly. We developed these algorithms in Rodin platform and proved their almost-certain convergence with our tool. The first example is duelling cowboys explained in section 6.1. We give the development of simple contention resolution algorithm for Firewire protocol in section 6.2.

6.1 Duelling Cowboys

The description of the problem previously defined in [7, Chapter 6] is as follows. There are two cowboys X and Y fighting a duel. They take turns to shoot at each other. In each shoot, the probability for hitting the opponent is neither 0 nor 1. Therefore, almost-certainly one cowboy will die and the duel will be completed.

6.1.1 Formal Development in Event-B

Firstly, to use our tool we changed the standard configuration into probabilistic configuration in context and machines with using a text editor. The context of the model includes the definition of cowboys as follows.

carrier sets: $COWBOY$

constants: X, Y

axioms:
$partition(COWBOY, \{X\}, \{Y\})$

In the initial model we have the set of alive cowboys S. Initially both of them are alive.

variables: S

invariants:
inv1 $S \subseteq COWBOY$
inv2 $S \neq \varnothing$
DLF $S = \{X, Y\} \vee (S = \{X\} \vee S = \{Y\})$

```
init
  begin
    $S := \{X, Y\}$
  end
```

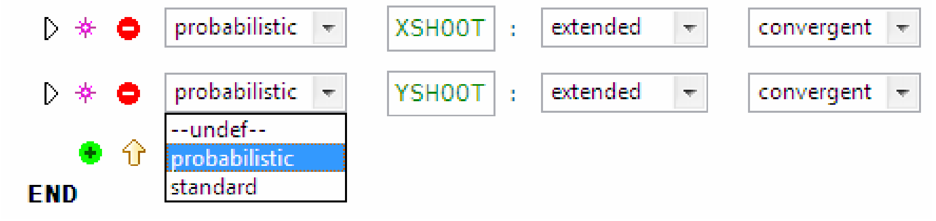
We have two events XSHOOT and YSHOOT for cowboys to shoot at each other. Each cowboy has two possibilities: *hit* or *miss*. Furthermore, we have an *observer* event FINAL (similar event to final in Section 4.2.1) to observe certain condition about the state of the model, where we are interested in the fact that only one cowboy survives.

```
XSHOOT
  status  probabilistic
  when
    $S = \{X, Y\}$
  then
    $S :| S' = \{X, Y\} \lor S' = \{X\}$
  end
```

```
YSHOOT
  status  probabilistic
  when
    $S = \{X, Y\}$
  then
    $S :| S' = \{X, Y\} \lor S' = \{Y\}$
  end
```

```
FINAL
  when
    $S = \{X\} \lor S = \{Y\}$
  then
    SKIP
  end
```

The proof of **inv2** and deadlock-freeness **DLF** are straight-forward. We want to prove that shooting events almost-certainly terminate. Therefore, we select them as probabilistic with using tool (Figure 6.1). In addition we must specify a variant and a bound (Figure 6.2).



probabilistic ▾ XSHOOT : extended ▾ convergent ▾

probabilistic ▾ YSHOOT : extended ▾ convergent ▾

--undef--
probabilistic
standard

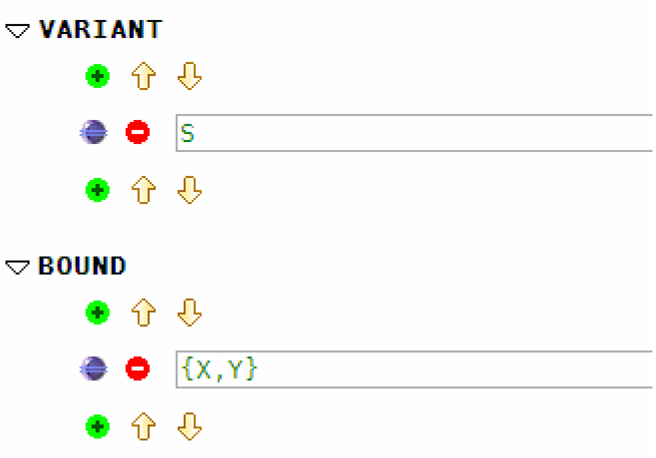
END

Figure 6.1: Selecting probabilistic events

variant: S

bound: $\{X, Y\}$

As a result, necessary proof obligations are generated by POG (Figure 6.3). With **BFN** (Figure 6.4) we prove the finiteness of the bound which is obvious.



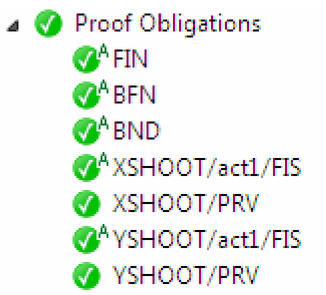
VARIANT

S

BOUND

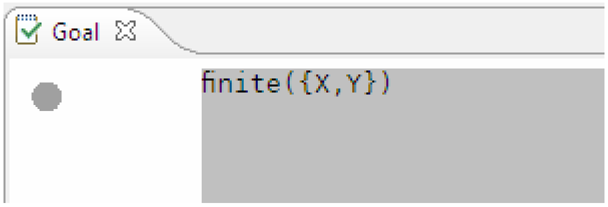
{X,Y}

Figure 6.2: Specifying a variant and a bound



- Proof Obligations
  - FIN
  - BFN
  - BND
  - XSHOOT/act1/FIS
  - XSHOOT/PRV
  - YSHOOT/act1/FIS
  - YSHOOT/PRV

Figure 6.3: Generated proof obligations

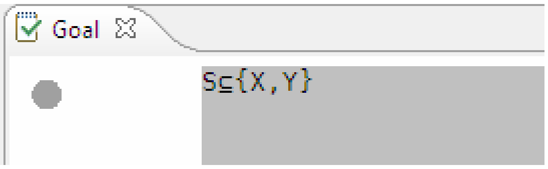


Goal

```
finite({X,Y})
```

Figure 6.4: **BFN** proof obligation

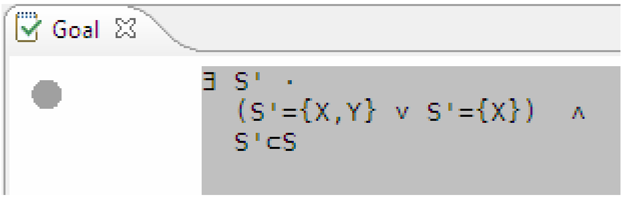
BND proof obligation (Figure 6.5) states that the bound dominates the variant. It is correct since no event enlarges the set S.



```
S⊆{X,Y}
```

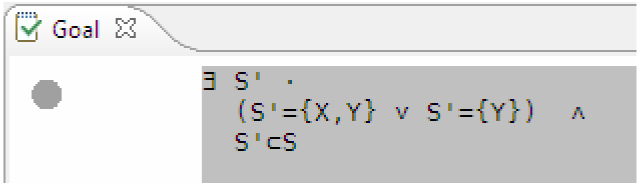
Figure 6.5: **BND** proof obligation

Finally, with **PRV** proof obligations (Figure 6.6 and 6.7) we prove that probabilistic events *may* decrease the variant. For **XSHOOT**/**PRV**, if we select $S' = \{X\}$, then $S' = \{X\} \subset \{X, Y\} = S$. Similarly, for **YSHOOT**/**PRV** we can instantiate S' with $\{Y\}$. Therefore, one of the two alternatives (i.e. hitting the opponent) decreases the variant and events almost-certainly terminate.



```
∃ S' ·
  (S'={X,Y} ∨ S'={X})  ∧
  S'⊂S
```

Figure 6.6: **XSHOOT**/**PRV** proof obligation



```
∃ S' ·
  (S'={X,Y} ∨ S'={Y})  ∧
  S'⊂S
```

Figure 6.7: **YSHOOT**/**PRV** proof obligation

As a result, since the model is deadlock-free and shooting events almost-certainly converge, FINAL event will be enabled with probability 1.

6.2 Contention Resolution

The contention problem in the Firewire tree identify protocol [2] is one example of a use of probability to break the symmetry. We do not deal with the full model but focus only on the contention problem that is explained in [6]. We developed the same model as described in [6].

6.2.1 Event-B Model of the Contention Problem

We define a carrier set *WAIT* containing the two constants: *short* and *long*.

carrier sets: $WAIT$

constants: $short, long$

axioms:
$partition(WAIT, \{short\}, \{long\})$

In initial model, two variables *x* and *y* represent the state of the two nodes in the contention: either sending the message in a *short* or *long* delay.

variables: x, y

invariants:
$x \in WAIT$
$y \in WAIT$

```
init
  begin
    $x :\in WAIT$
    $y :\in WAIT$
  end
```

There is only one event which resolves the contention (in one-shot) by assigning different values to x and y. This only specifies that the problem is to be resolved but not how.

```
(abstract_)resolve
  when
    $x = y$
  then
    $x, y :| x' \in WAIT \land y' \in WAIT \land x' \neq y'$
  end
```

We refine the abstract model, introducing two new variables, namely u and v. They represent the intermediate states of the two nodes during contention resolution.

```
variables:  $x, y, u, v$
```

```
invariants:
    $u \in WAIT$
    $v \in WAIT$
```

```
init
  begin
    $u :\in WAIT$
    $v :\in WAIT$
  end
```

A new event draw models (probabilistically) the effect of randomly choosing for both the two nodes either sending messages after a *long* or a *short* delay. The new event is enabled when the values of u and v are the same.

Event resolve has an additional guard $u \neq v$ indicating that two different delay times u and v have been successfully drawn. In this case, x and y will be assigned to u and v, respectively, and the contention is resolved.

draw
  status *probabilistic*
  when
    $u = v$
  then
    $u :\in WAIT$
    $v :\in WAIT$
  end

(concrete_)resolve
  when
    $u \neq v$
    $x = y$
  then
    $x := u$
    $y := v$
  end

We want to prove that the new event draw does not take the control of the system forever. To prove almost-certain termination of the event, we select the event as probabilistic (Figure 6.8). We define a constant r to constitute a variant as follows:

axioms:
  axm_1: $r \in WAIT \times WAIT \longrightarrow \{0, 1\}$
  axm_2: $\forall x, y \cdot x \in WAIT \land y \in WAIT \land x = y \Rightarrow r(x \mapsto y) = 1$
  axm_3: $\forall x, y \cdot x \in WAIT \land y \in WAIT \land x \neq y \Rightarrow r(x \mapsto y) = 0$

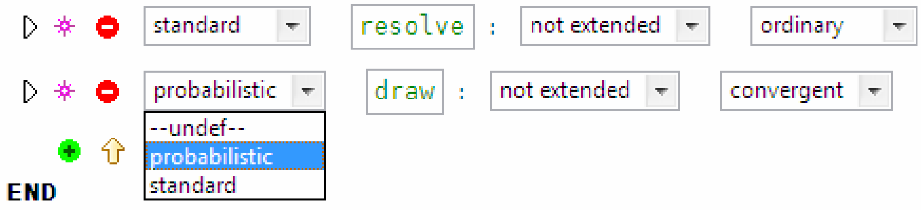
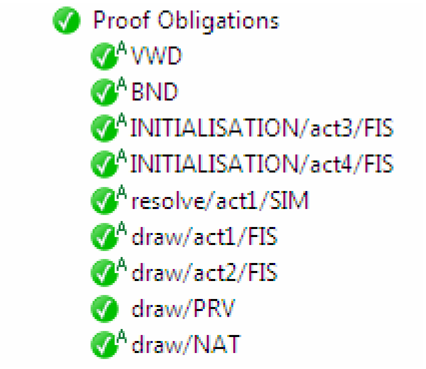

standard resolve : not extended ordinary
probabilistic draw : not extended convergent
--undef--
probabilistic
standard
END

Figure 6.8: Selecting probabilistic event

$r(x \mapsto y)$ is defined to have value 1 if $x = y$ and 0 otherwise. After specifying the variant and the bound, generated proof obligations can be seen in Figure 6.9.

variant: $r(u \mapsto v)$

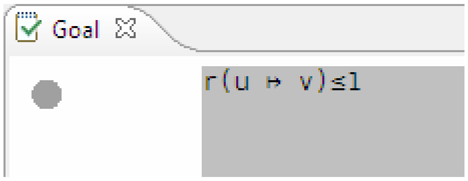
bound: 1



✓ Proof Obligations
- ✓ᴬ VWD
- ✓ᴬ BND
- ✓ᴬ INITIALISATION/act3/FIS
- ✓ᴬ INITIALISATION/act4/FIS
- ✓ᴬ resolve/act1/SIM
- ✓ᴬ draw/act1/FIS
- ✓ᴬ draw/act2/FIS
- ✓ draw/PRV
- ✓ᴬ draw/NAT

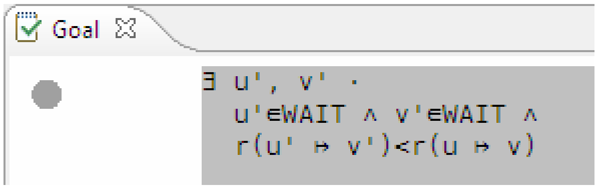
Figure 6.9: Generated proof obligations

Proof obligation **BND** (Figure 6.10) is obvious from the definition of r. Moreover, **PRV** (Figure 6.11) is obvious since two of the four alternatives (i.e. distinct u' and v' values) decrease the variant (e.g. $u' = short$ and $v' = long \Rightarrow r(u' \mapsto v') = 0 < 1 = r(u \mapsto v)$). Therefore, the new event draw will terminate with probability 1 and the event resolve will be enabled.



```
r(u ↦ v)≤1
```

Figure 6.10: **BND** proof obligation



```
∃ u', v' ·
  u'∈WAIT ∧ v'∈WAIT ∧
  r(u' ↦ v')<r(u ↦ v)
```

Figure 6.11: **PRV** proof obligation

Chapter 7

Conclusions and Future Work

7.1 Conclusion

In conclusion, integrating qualitative probabilistic reasoning into Event-B [6] provides to prove almost-certain termination with very little cost of extra proof effort. The method preserves the simplicity of Event-B proof obligations only requiring a modest extension to existing proof obligations. Therefore, we have extended the Rodin Platform to support almost-certain termination with proposing some restrictions and conditions based on refinement.

Secondly, we have presented a development of Rabin's choice coordination algorithm [11] in Event-B with extensions for reasoning about termination with probability one [6]. In particular, we have formalised the lexicographical variant as presented in [10]. The example of Rabin's choice coordination is also used in [7, Chapter 3] as an illustrative example for reasoning about almost-certain termination using classical B. The main difference between the two developments is that in classical B, one ends up with a sequential program which is a model of the algorithm. Our development in Event-B gives us a model of a fully distributed system. Furthermore, the formalisation of lexicographic variants suited better for Event-B since in classical B, one can only have a single natural number variant. As a result, the lexicographic variant has to be encoded (unnaturally) into a natural number variant, which leads to more complicated proofs.

Moreover, we used the technique of splitting/merging the events to avoid having complicated proofs. We have presented the rule for establishing convergence value of merged events. The rule relies on the fact that convergence arguments (both standard and probabilistic) are preserved by refinement. However, a necessary condition for event merging is that all sub-events must have the same actions, which will be the action of the merged event. This type of refinement will preserve the probabilistic reasoning earlier since there are all possible alternatives from the abstraction retained by the refinement, including the choice for termination.

7.2 Future Work

7.2.1 Theoretical Work

When we developed our tool and Rabin's choice coordination algorithm, we only use superposition refinement, in particular, when dealing with convergence proofs, we merely keep the models the same, and the various refinements are there to accommodate the lexicographic variant. For this reason, i.e. having the same model through out, our reasoning about probabilistic termination is preserved. However, in general, standard refinement does not preserve this type of reasoning: A valid standard refinement can accidentally remove the choice that lead to possible termination. The argument becomes more complicated with data refinement, i.e. when one replaces some abstract variables by some new concrete variables. Additional proof obligation(s) will be needed to guarantee that our reasoning at the abstract level about probabilistic convergence remains valid at the concrete level. We regard this as possible future work.

7.2.2 Tool Support

We extended the Rodin Platform for supporting the generation of appropriate proof obligations concerning with qualitative reasoning. However, the generation of **FINACT** proof obligation is not possible in current tool. In the near future, we will add this proof obligation into our tool with creating another POG module.

When using the tool, new Event-B components (i.e. machines and contexts) are created with the standard configuration (`org.eventb.core.fwd`). One has to change configuration manually with using text editor to use our static checker and proof obligation generator extensions. With a future work, it may be possible to choose configuration when creating new project. Furthermore, in Pretty Print view of machines the status of an event can be seen. However, the status of probabilistic events are convergent in current tool support. It may be changed to probabilistic with a future work. In addition bound may also be added into Pretty Print view.

7.2.3 Case Studies

Rabin's Choice Coordination

We developed a simplified version of the algorithm described by Morgan et. al. [10] which has n tourists and 2 alternatives. The original version of the algorithm with n tourists and k alternatives can be developed in the future. However, proving convergence of the original version is more complicated, because it necessitates to build generalized invariants and variants.

Other examples

Using our new developed tool support, we have modelled other examples for proving termination including *contention resolution* [6] and *duelling cowboys* [7, Chapter 6]. In the near future, we will try to integrate the reasoning about contention resolution with the development of Firewire protocol [2].

Bibliography

[1] Rodin Developer Support. http://wiki.event-b.org/index.php/Rodin_Developer_Support.

[2] J-R. Abrial, D. Cansell, and D. Méry. A mechanically proved and incremental development of ieee 1394 tree identify protocol. *Formal Asp. Comput.*, 14(3):215–227, 2003.

[3] Jean-Raymond Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.

[4] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. RODIN: An open toolset for modelling and reasoning in Event-B. *Internation Journal on Software Tools for Technology Transfer (STTT)*, apr 2010.

[5] Robert W. Floyd. Assigning meanings to programs. In *Proceedings of the Symposium on Applied Math.*, volume 19, pages 19–32. American Mathematical Society, 1967.

[6] S. Hallerstede and T.S. Hoang. Qualitative Probabilistic Modelling in Event-B. In Jim David and Jeremy Gibbons, editors, *IFM 2007: Integrated Formal Methods, Proceedings of the 6th International Conference*, volume 4591 of *LNCS*, pages 293–312, Oxford, U.K., jul 2007. Springer Verlag.

[7] Thai Son Hoang. *The Development of a Probabilistic B-Method and a Supporting Toolkit.* PhD thesis, The University of New South Wales, 2005.

[8] T.S. Hoang, H. Kuruma, D. Basin, and J-R. Abrial. Developing topology discovery in event-b. *Sci. Comput. Program.*, 74(11-12):879–899, 2009.

[9] IEEE. *IEEE Standard for a High Performance Serial Bus (supplement). Std 1394a-2000*, 2000.

[10] Carroll Morgan and Annabelle McIver. *Abstraction, Refinement and Proof for Probabilistic Systems.* Springer Verlag, 2005.

[11] Michael Rabin. The choice coordination problem. *Acta Informatica*, 17:121–134, 1982.

[12] E. Yilmaz. Rabin's choice coordination development. `http://deploy-eprints.ecs.soton.ac.uk/232/`, June 2010.