10

Event B

Event B [ABR 10a, CAN 07a] is a modeling language which can describe state-based models and required safety properties. The main objective is to provide a technique for incremental and proof-based development of the reactive systems. It integrates set-theoretical notations and a first-order predicate calculus, models called machines; it includes the concept of refinement expressing the simulation of one machine by another machine. An Event B machine models a reactive system, i.e. a system driven by its environment and its stimuli. An important property of these machines is that its events preserve the invariant properties defining a set of reachable states. The Event B method has been developed from the classical B method [ABR 96] and it offers a general framework for developing the correct construction systems by using an incremental approach for designing the models by refinement. Refinement [BAC 79, DIJ 76, BAC 98, BAC 89] is a relationship relating two models such that one model is simulating the other model. Refinement is also called refinement and preserves properties of the AQ1
abstract model in the refined or concrete model. When an abstract model is refined by a concrete model, the concrete model simulates the abstract model and any safety property of the abstract model is also a safety property of the concrete model. In particular, the concrete model preserves the invariant properties of the abstract model. Event B aims to express models of systems characterized by invariants and by a list of safety properties. We can consider liveness properties as in UNITY [CHA 88] or TLA$^+$ [LAM 02, LAM 94] but in a restricted way.

Chapter written by Dominique MÉRY and Neeraj KUMAR SINGH.

10.1. Introduction

This chapter is organized into eight sections. Section 10.2 introduces results on the modeling and verification of systems using transition systems. The goal is to provide the basic fundamental and conceptual theories, which support Event B approach. In particular, we explain how invariant properties and safety properties are defined in the framework of a transition system, which may model a program, an algorithm or a distributed system. Section 10.3 details the Event B language and related concepts such as events, contexts, machines and refinement. We give an explanation of proof obligations (POs) generated for checking the consistency of the Event B structure. Finally, in sections 10.4 and 10.5, we develop three case studies, in order to illustrate the incremental and proof-based modeling using Event B. We emphasize the notion of proof-based patterns applied for the Event B method. Section 10.6 describes available tools for supporting the Event B modeling language and we complete this chapter with the current and future trends for this method.

10.2. Modeling and verification of a system

10.2.1. *Modeling*

A relational abstract model $\mathcal{AM}$ ($\mathcal{AM}_{\mathrm{P}}$ of a program P or $\mathcal{AM}_{\mathrm{P}}$ of a system P) is defined by a set of states Σ, a set of initial states $Init_{\mathrm{P}}$, a set of terminal states $Term_{\mathrm{P}}$ and a binary relation $\mathcal{R}$ over Σ. The set of terminal states may be empty and, in this case, the program does not terminate; this feature can be used for modeling programs or procedures of operating systems which are not terminating and cannot terminate at all. We will use system rather than program, since we can describe elements, which are more general than computer programs but also the formalism is usable for describing distributed applications.

A system is characterized by a set of traces generated from the abstract model as follows:

$s_0 \underset{R}{\longrightarrow} s_1 \underset{R}{\longrightarrow} s_2 \underset{R}{\longrightarrow} s_3 \underset{R}{\longrightarrow} \ldots \underset{R}{\longrightarrow} s_i \underset{R}{\longrightarrow} \ldots$ is a trace generated by the abstract model.

The observation of a system can be summarized by the analysis of its traces; Θ_{S} is a set of all traces of S. The expression of properties requires an assertion

language or a formulas language: $\mathcal{L}$ is an assertion language. A simple choice is to consider the language of assertions defined by $\mathcal{P}(\Sigma)$ (power set of Σ) and $\varphi(s)$ (or $s \in \varphi$), which means that φ is true in the state s. The assertion language allows us to express properties, however it may be possible that the language is not expressive enough. We assume that the language is sufficiently expressive (following Cook) and this means that the required properties for completeness can be expressed in the language.

Properties of a system S are, in particular, safety properties and liveness properties. Safety properties are, for instance, the partial correctness of a system S with respect to its specifications, the absence of runtime errors; liveness properties are, for instance, the termination of a program P with respect to its specifications or the total correctness of P with respect to its specifications. We could also consider program properties as performance but this leads to the models for expressing the non-functional properties. Properties are expressed in a language $\mathcal{L}$ and its components can be combined using logical connectives or instantiation of variables; the implication relation upto the equivalent relation defines a partial ordering over a set of formulas.

We assume that a system S is modeled by a set of states Σ_{S}, denoted by Σ, where $\Sigma \stackrel{def}{=}$ VARIABLES $\longrightarrow$ VALEURS. The expression $s \in A$ is equivalent to $s[\![\varphi(x)]\!]$, where x is a list whose elements are variables VARIABLES; this means that $s \in A$ is equivalent to $\varphi(x)$ is true in s. The meaning of a formula or a predicate can be given using an inductive process on $s[\![\varphi(x)]\!]$.

EXAMPLE 10.1.–

1) $s[\![x]\!]$ is the value of s in x, i.e. $s(x)$ or the value of x in s.

2) $s[\![\varphi(x) \wedge \psi(x)]\!] \stackrel{def}{=} s[\![\varphi(x)]\!]$ and $s[\![\psi(x)]\!]$.

3) $s[\![x = 6 \wedge y = x + 8]\!] \stackrel{def}{=} s[\![x]\!] = 6$ and $s[\![y]\!] = s[\![x]\!] + 8$.

We use the notations for simplifying the indication of a state: for instance, $s[\![x]\!]$ is the value of x in s and the name of the variable x and its value will not be distinguished; $s'[\![x]\!]$ is the value of x in s' and will be denoted by x'. Consequently, $s[\![x = 6]\!] \wedge s'[\![y = x + 8]\!]$ is simplified into $x = 6 \wedge y' = x' + 8$. The consequence is that we can write the transition between two states as a relation relating the state of variables in s and the state of variables in s'.

Let s, s' be two states of the set VARIABLES $\longrightarrow$ VALS .
$s \underset{R}{\longrightarrow} s'$ is rewritten as a relation $\mathrm{R}(x, x')$ where x and x' are values of x.

We have introduced primed variables borrowed from the Temporal Logic of Actions of Lamport [LAM 94] and x' is the value after the transition under consideration and x is the value before the transition under consideration. The expression $\exists y.\mathrm{R}(x, y)$ defines the condition for transition or the guard. We are interested in particular expressions like $cond(x) \wedge x' = f(x)$ where $cond$ is a condition over x and f is a function. We can express induction principles using relations over unprimed and primed variables. Initial conditions are defined by a predicate characterizing the initial values of variables. We propose to define more generally a relational model of a system. A set of states is Σ for a given system and we identify this set with a set of possible values of flexible variables x. We use the same notation but VALS will be a set of possible values of x.

DEFINITION 10.1.– *Relational model of a system*

A relational model $\mathcal{MS}$ for a system $\mathcal{S}$, is a structure

$$(Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\}),$$

where

– $Th(s, c)$ is a theory defining sets, constants and static properties of these elements.

– x is a list of flexible variables.

– VALS is a set of possible values for x.

– INIT(x) defines a set of initial values of x.

– $\{r_0, \ldots, r_n\}$ is a finite set of binary relations relating the prevalues x and the postvalues x'.

A relational model $\mathcal{MS} = (Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ for a system $\mathcal{S}$ is a structure for studying a system defined by a model. We assume that the r_0 is the relation $Id[\text{VALS}]$, identity over VALS .

DEFINITION 10.2.– *Let $(Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ be a relational model for a system $\mathcal{S}$. The relation* NEXT *attached to this model, is defined by the disjunction of relations r_i:* NEXT $\stackrel{def}{=} r_0 \vee \ldots \vee r_n$.

Modeling a system leads to identifying state variables x, a predicate defining the initial conditions of x and a relation NEXT expressing how the values of variables before and after are related. Induction principles are formulated in these relational models and here, we introduce the definition as follows:

Let $(Th(s,c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ be a relational model of a system $\mathcal{S}$. The theory $Th(s,c)$ is defined in an assertion language, which can express properties. An example is the set theory of the B language. When we consider a property φ, we use this set-theoretical language of B. For any variable x, we define the following values:

– x is the current value of x.

– x' is the next value of x.

10.2.2. *Safety properties*

The safety property states that *nothing bad will happen [LAM 80]*. For instance, the value of x is *always* between 0 and 67; the sum of the current values of x and y is the current value of z. The assertion language is supposed to be $(\mathcal{P}(\Sigma), \subseteq)$ and we suppose that the satisfaction relation is defined using the membership relation.

DEFINITION 10.3.– *A property φ is a safety property for a system* S*, when*

$$\forall s, s' \in \Sigma . s \in Init_{\text{S}} \wedge s \xrightarrow[R]{\star} s' \Rightarrow s' \in \varphi.$$

The expression $\xrightarrow[R]{\star}$ stands for the reflexive transitive closure of the relation $\xrightarrow[R]{}$. The safety property uses a universal quantification over states. For proving a safety property, we can either check a property for each possible state, if the set of states is finite, or, find an induction principle. In the case of an exhaustive checking, we can use an algorithm for computing the set of reachable states from the initial nodes: the *model checking* [MCM 93, HOL 97, CLA 00]. It helps to find the counter-examples and is a complementary approach to use the induction principle following in the next property.

THEOREM 10.1.– Induction principle

A property φ is a safety property for a program P if, and only if, there exists a property INV satisfying

$$\begin{cases} (1)\ Init_{\mathrm{P}} \subseteq INV \\ (2)\ INV \subseteq \varphi \\ (3)\ \forall s, s' \in \Sigma_{\mathrm{P}}.s \in INV \wedge s \underset{R}{\longrightarrow} s' \Rightarrow s' \in INV \end{cases}$$

The property INV is called a program invariant and it is a special safety property stronger than the other safety properties of a program. The justification of this principle is simple.

PROOF.–
$\langle 1 \rangle 1$. SUPPOSE THAT: There exists a property INV such that

$$\begin{cases} (1)\ Init_{\mathrm{P}} \subseteq INV \\ (2)\ INV \subseteq \varphi \\ (3)\ \forall s, s' \in \Sigma_{\mathrm{P}}.s \in INV \wedge s \underset{R}{\longrightarrow} s' \Rightarrow s' \in INV \end{cases}$$

PROVE THAT: φ is a safety property for the program P.

PROOF.– Let two states s, s' such that $s \in Init_{\mathrm{P}} \wedge s \xrightarrow[R]{\star} s'$. We can construct a sequence of states $s = s_0 \underset{R}{\longrightarrow} s_1 \underset{R}{\longrightarrow} s_2 \underset{R}{\longrightarrow} s_3 \underset{R}{\longrightarrow} \ldots \underset{R}{\longrightarrow} s_i = s'$. from assumption (1), we derive that INV holds at s. By using (3) for any state of the trace, we derive that INV holds at $s_1, s_2, \ldots s_i$. Then, we apply (2) for the state s' and we derive that s' satisfies φ. □

$\langle 1 \rangle 2$. SUPPOSE THAT: $\forall s, s' \in \Sigma.s \in Init_{\mathrm{P}} \wedge s \xrightarrow[R]{\star} s' \Rightarrow s' \in \varphi$

PROVE THAT: There exists a property INV such that

$$\begin{cases} (1)\ Init_{\mathrm{P}} \subseteq INV \\ (2)\ INV \subseteq \varphi \\ (3)\ \forall s, s' \in \Sigma_{\mathrm{P}}.s \in INV \wedge s \underset{R}{\longrightarrow} s' \Rightarrow s' \in INV \end{cases}$$

PROVE THAT: φ is a safety property for the program P

PROOF.– We define the following property $INV \stackrel{def}{=} \exists s \in \Sigma.s \in Init_{\mathrm{P}} \wedge s \xrightarrow[R]{\star} s'$. INV states that the state s' is a reachable state from some initial

state of P. $\mathcal{R}^\star$ is the reflexive transitive closure of $\mathcal{R}$. The three properties are simple to check for INV. INV is called the strongest invariant of the program P. □

$\langle 1\rangle 3$. Q.E.D.

PROOF.– By steps $\langle 1\rangle 1$ and $\langle 1\rangle 2$, we infer the conclusion. □
□

The property explains Floyd's invariance proof method also known as Floyd–Hoare's methods [FLO 67, HOA 69], initially sketched by Turing in 1949 [TUR 49]. The property gives a general form for the induction and then we can rephrase it according to the required invariance properties (partial correctness and absence of deadlocks, etc.). P. and R. Cousot [COU 00, COU 79, COU 92, COU 78] give a complete synthesis on the different possible induction principles. We apply these results to the case of relational models of a system and we obtain an expression of a safety property as follows:

DEFINITION 10.4.– *Let* $(Th(s,c), x, \text{VALS}\ , \text{INIT}(x), \{r_0, \ldots, r_n\})$ *be a relational model for a system* $\mathcal{S}$*. A property* φ *is a safety property for a system* $\mathcal{S}$*, when* $\forall y, x \in \Sigma. Init(y) \wedge \text{NEXT}^\star(y,x) \Rightarrow \varphi(x)$.

From the induction principle of the previous section, we can derive the following property.

THEOREM 10.2.– (Induction principle for a relational model)

Let $(Th(s,c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ be a relational model for a system $\mathcal{S}$.

A property $\varphi(x)$ is a safety property for $\mathcal{S}$ if, and only if, there exists a property $i(x)$ such that

$$\begin{cases} (1)\ \forall x \in \text{VALS}\ . Init(x) \Rightarrow i(x) \\ (2)\ \forall x \in \text{VALS}\ . i(x) \Rightarrow \varphi(x) \\ (3)\ \forall x, x' \in \text{VALS}\ . i(x) \wedge \text{NEXT}(x,x') \Rightarrow i(x') \end{cases}$$

PROOF.– Derived from the proof of the property 10.2. □

If we transform properties, we obtain a form closer to what we will use in the next sections and closer to the concept of *abstract systems* or *abstract machines* of Event B.

THEOREM 10.3.– The two sentences are equivalent:

1) There exists a state property $\mathrm{I}(x)$ such that

$$\forall x, x' \in \textsc{Vals} : \begin{cases} (1)\ \textsc{Init}(x) \Rightarrow \mathrm{I}(x) \\ (2)\ \mathrm{I}(x) \Rightarrow \mathrm{P}(x) \\ (3)\ \mathrm{I}(x) \wedge \textsc{Next}(x, x') \Rightarrow \mathrm{I}(x') \end{cases}$$

2) There exists a state property $\mathrm{I}(x)$ such that:

$$\forall x, x' \in \textsc{Vals} : \begin{cases} (1)\ \textsc{Init}(x) \Rightarrow \mathrm{I}(x) \\ (2)\ \mathrm{I}(x) \Rightarrow \mathrm{P}(x) \\ (3)\ \forall i \in \{0, \ldots, n\} : \mathrm{I}(x) \wedge x\ r_i\ x' \Rightarrow \mathrm{I}(x') \end{cases}$$

PROOF.– The proof is obvious by applying the following rule: $\forall i \in \{0, \ldots, n\} : A \wedge x\ r_i\ x' \Rightarrow B \equiv (A \wedge (\exists i \in \{0, \ldots, n\} : x\ r_i\ x')) \Rightarrow B$ and the definition of $\textsc{Next}(x, x')$. □

We have given an explanation of the induction rule used in Floyd's method [FLO 67, TUR 49, HOA 69] which states that the invariance properties are necessary for deriving the safety properties. The invariance properties are stronger than safety properties. The invariance properties are also the special case of safety properties. There is a confusion in the literature where we claim that *always true* and *invariant* are two equivalent concepts: an invariant is an inductive property.

The Event B method uses these two kinds of properties. The clause INVARIANTS for invariants and in earlier versions of Rodin, the clause THEOREMS for safety properties. Current versions distinguish both classes by a feature stating that the predicate is either an invariant or a safety property (theorem). An invariant is obviously a safety property. Now, we summarize the Event B language and the incremental and proof-based development of event-based models.

10.3. Event B: a modeling language

Event B is both a language and a method. Its concepts are limited and allow the user to manage a simple palette of tools: axioms, theorems, theories, events, machine, context and refinement. We shortly describe these

elements, but it is clear that examples constitute the best way to learn and understand how to use tools.

The construction of an Event B model is based on concepts like sets, constants, axioms, theorems, variables, invariants and events; these syntactic constructions are organized into two kinds of structures:

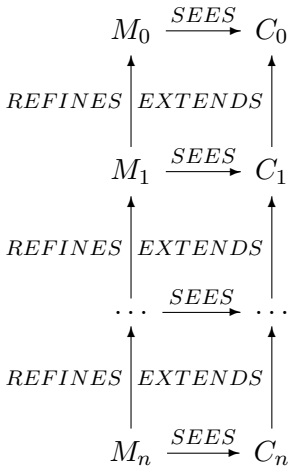
– Contexts express axiomatic static properties of the models. Contexts may contain carrier sets, constants, axioms and theorems. Axioms describe properties of carrier sets and constants. Theorems derive properties that can be proved from the axioms. POs associated with contexts are straightforward: the stated theorems must be proved, which follow from the predefined axioms and theorems. Additionally, a context may be indirectly seen by machines. Namely, a context C can be seen by a machine M indirectly if the machine M explicitly sees a context, which is an extension of the context C.

– Machines express dynamic behavioral properties of the models, which may contain variables, invariants, theorems, events and variants. Variables v represents the state of the machine. Variables, like constants, correspond to simple mathematical objects: sets, binary relations, functions, numbers, etc. They are constrained by invariants $I(v)$. Invariants are supposed to hold whenever variable values change.

When a machine is organizing events, modifying the state variables and it uses static information defined in context. These basic structure mechanisms are extended by the refinement mechanism which provides a mechanism for relating an abstract model and a concrete model by adding new events or by adding new variables. This mechanism allows us to gradually develop Event-B models and to validate each decision step using the proof tools. The refinement relationship should be expressed as follows: a model M is refined by a model P, when P simulates M. The final concrete model is close to the behavior of a real system that is executing events using real source code. We give details now regarding the definition of events, refinement and guidelines for developing complex system models.

– The consistency of a context or a machine in Event B is achieved by proving *proof obligation* generated by tools [CLE 02, ABR 10b] and sound with respect to the results of section 10.2. If these POs are discharged, then the structure (context or machine) is correct at least with respect to the typing. Indeed, the main tricky point is the statement and the proof of the invariant property of a machine, but the refinement is a technique for facilitating the

proof process and the discovery of invariants.



In the next section, we summarize each structure (context, machine) used for constructing models for a given system using relations as EXTENDS, SEES, REFINES among the structures. We summarize the general form of an Event B development in the diagram. The main advice is to use the refinement of machines and events as much as possible.

10.3.1. *Basic elements of an Event B model*

We start by defining the events that are at the heart of this method and that react to a condition called a guard. An event is characterized by a condition and an action. We define three unique possible shapes for events and this will be sufficient for modeling systems in a general sense. The first form is a normal
AQ2 form in the sense that it can be reduced to two in this form below. The second case corresponds to a guarded event and the third case corresponds to a guarded quantified event. Intuitively, the observation of an event is made in the case where the guard is true but the fact that the guard is true does not allow us to conclude that the event is or will be observed. Each event can be defined by a relationship before–after denoted by $BA(x, x')$.

An event is characterized by its guard, which is determined at the modeling phase and it can only be triggered if the guard is true. We will detail POs generated for a given event e and explain the meaning of these POs. In our
AQ3 presentation, we emphasize the role of refinement, which is defined on events. The general form of an event is as follows:

EVENT e
ANY t
WHERE
$G(c, s, t, x)$
THEN
$x :\mid (P(c, s, t, x, x'))$
END

– c and s designate constants and visible sets by an event e are defined in the context of clause SEES.

– x is the state variable or a list of state variables.

– $G(c, s, t, x)$ is the guard or the enableness condition of e.

– $P(c, s, t, x, x')$ is the predicate stating the relation between the pre value of x, denoted as x, and the post value of x, denoted as x'.

– $BA(e)(c, s, x, x')$ is the *before-after* relation for the event e defined by $\exists t.G(c, s, t, x) \ \wedge \ P(c, s, t, x, x')$.

For each event e, POs are named according to the following format: $e/inv/ < type >$ where $< type >$ is either INV, or FIS, or GRD, or SIM, or THM, or $WFIS$, or WD, etc. and correspond to generated POs for ensuring invariance, guard strengthening, simulation, safety and well-definedness, etc. We do not list the complete list of possible names and refer to the book of J.-R. Abrial [ABR 10a] for a full version, as well as to the Rodin platform [ABR 10b]. Now we analyze how the POs are generated.

10.3.2. *Invariance properties in Event B*

The invariant $I(x)$ of a model is an invariant property for all events of a system, including the initial event. If e is an event of the model, then the condition of preservation of this invariant by this event is: $I(x) \ \wedge \ BA(e)(c, s, x, x') \ \Rightarrow \ I(x') \ \ (INV)$. $I(x)$ is written as a list of predicates labeled $inv_1 \ \ldots inv_n$ and interpreted as a conjunction. The condition on the initial conditions is as follows: $Init(x, s, c) \ \Rightarrow \ I(x) \ \ (INIT)$.

When an event e defines the predicate before–after $BA(e)(c, s, x, x')$, the feasibility of this event means that under hypothesis defined by the invariant $I(x)$ and guard $\mathsf{grd}\,(e)$, of the event, there is still x' such that $BA(e)(c, s, x, x')$. In other words, it means that this event, when observed,

will not induce unwanted behaviors and we give a condition for each event: $I(x) \ \wedge \ \mathsf{grd}\,(e) \ \Rightarrow \ \exists x' \cdot BA(e)(c,s,x,x') \ \ (FIS)$.

Safety properties are derived by the proof that the system invariant implies safety property $A(x)$ and, moreover, we add the context $C(s,c)$ of this proof. The context of this proof is given by the properties $C(s,c)$, where sets s and constant c are defined in the model: $C(s,c) \ \wedge \ I(x) \ \Rightarrow \ A(s,c,x) \ \ (THM)$.

To conclude this point on POs, they are derived from the theorem 10.2 and we can therefore conclude the following property.

THEOREM 10.4.– Let $Th(s,c)$ be a theory defined by sets s, constants c and axioms $C(s,c)$ and let E be a finite list of events modifying x in the context define by the theory $Th(s,c)$. We assume the following points:

– VALS is a set of possible values for x.

– $\{r_0,\ldots,r_n\}$ is a set of relations $BA(e)(s,c,x,x')$ defined for event e of E and one of the events is the event $skip$.

– INIT(x) is the predicate defining the initial conditions of x.

If the POs (INIT) and (INV) are valid, then the relational model $(Th(s,c),x,\text{VALS}\ ,\text{INIT}(x),\{r_0,\ldots,r_n\})$ satisifies the invariant $I(x)$ and the safety properties $A(s,c,x)$.

We now give the various POs generated from the general form given above. We assume that the context of theory is $C(s,c)$ and we use the notation $C(s,c) \vdash P$ to express the proof obligation P in $C(s,c)$ context. So, we have the following reformulation:

– ***INIT/I/INV***: $C(s,c), INIT(c,s,x) \vdash I(c,s,x)$

– ***e/I/INV***: $C(s,c), I(c,s,x), G(c,s,t,x), P(c,s,t,x,x') \vdash I(c,s,x')$

– ***e/act/FIS***: $C(s,c), I(c,s,x), G(c,s,t,x) \vdash \exists x'.P(c,s,t,x,x')$

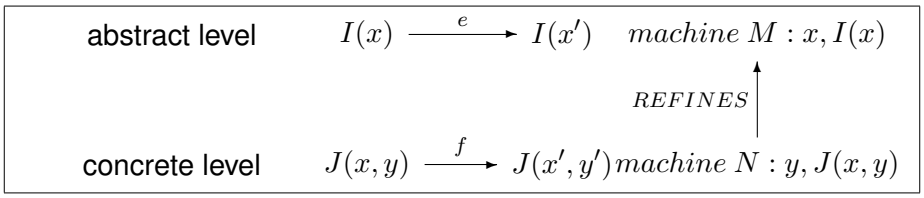
We have instantiated the induction principle to ensure invariance of I. The POs generator performs also important simplifications for facilitating the proof checking process by the provers.

10.3.3. *Refinement of events*

EVENT e
ANY t
WHERE
$G(c, s, t, x)$
THEN
$x :\mid (P(c, s, t, x, x'))$
END

EVENT f
REFINES e
ANY u
WHERE
$H(c, s, u, y)$
WITNESSES
$t : W1(c, s, t, u, y)$
$x' : W2(c, s, t, x', y)$
THEN
$y :\mid (Q(c, s, t, y, y'))$
END

In the above schema, $t : W1(c, s, t, u, y)$ is a proof witness to connect the current parameter u and the parameter t of the event e, $x' : W2(c, s, t, x', y)$ is a proof witness to connect the variable y and the next value of x. The event f refines the event e, when the observation of f at the concrete level, implies that the event e at the abstract level also appears. More formally, the refinement of e by f is defined by the formula: I $I(c, s, x) \wedge J(c, s, x, y) \wedge BA(f)(c, s, y, y') \Rightarrow \exists x'.(BA(e)(c, s, x, x') \wedge J(c, s, x', y'))$ where $J(c, s, x, y)$ is the invariant of the concrete level ensuring the relationship between concrete and abstract variables. We schematize refinement as follows:



Note that the role of predicates $W1$ and $W2$ is to provide values to prove existential properties induced by parameters but also by reference to the abstract level. Without these hints, the user should propose possible values while using the interactive proof tools. The general form of proof obligation for the refinement of f by e also takes into account the case where f is a new event at the concrete level and in this case f refines $skip$ that does not change the variable x. We will give the above formulation as generated POs. AQ4

– *e/act/SIM*: $\begin{pmatrix} C(s,c), \\ I(c,s,x), J(c,s,x,y), H(c,s,t,y), \\ Q(c,s,t,y,y'), \\ W1(c,s,t,u,y), W2(c,s,t,x',y) \end{pmatrix} \vdash P(c,s,t,x,x')$

– *e/grd/FIS*: $\begin{pmatrix} C(s,c), \\ I(c,s,x), J(c,s,x,y), \\ H(c,s,t,y), W1(c,s,t,u,y) \end{pmatrix} \vdash G(c,s,t,x)$

We have given the clear definitions of generated POs to verify the refinement of an event by others. It remains to define the structures of machines and contexts.

10.3.4. *Structures for Event B models*

The Event B modeling language provides a framework for supporting our methodology as applied to the development of sequential programs. Abrial [ABR 03b] has demonstrated the possibility of developing sequential programs using Event B. The modeling process deals with various languages, as seen by considering the triptych of Bjoerner [BJO 06a, BJO 06b, BJO 06c, BJØ 07]: $\mathcal{D}, \mathcal{S} \longrightarrow \mathcal{R}$. Here, the domain $\mathcal{D}$ deals with properties, axioms, sets, constants, functions, relations and theories. The system model $\mathcal{S}$ expresses a model or a refinement-based chain of models of the system. Finally, $\mathcal{R}$ expresses requirements for the system to be designed. Considering the Event B modeling language, we notice that the language can express *safety* properties, which are either *invariants* or *theorems* in a machine corresponding to the system. Recall that two main structures are available in Event B.

– Contexts express static information about the model.

– Machines express dynamic information about the model, invariants, safety properties and events.

10.3.4.1. *Contexts*

The first structure is called a context (see Figure 10.1), and it provides the definition of the sets, constants, axioms for sets and constants, and theorems that can be derived from the axioms of the context $\mathcal{D}$. The context $\mathcal{AD}$ is a previous context that has already been defined, and it extends the current context. A context is validated when sets $S_1, \ldots, S_n$, constants $C_1, \ldots, C_m$

and axioms $ax_1, \ldots, ax_p$ are well-formed and when all theorems $th_1, \ldots, th_q$ are proved.

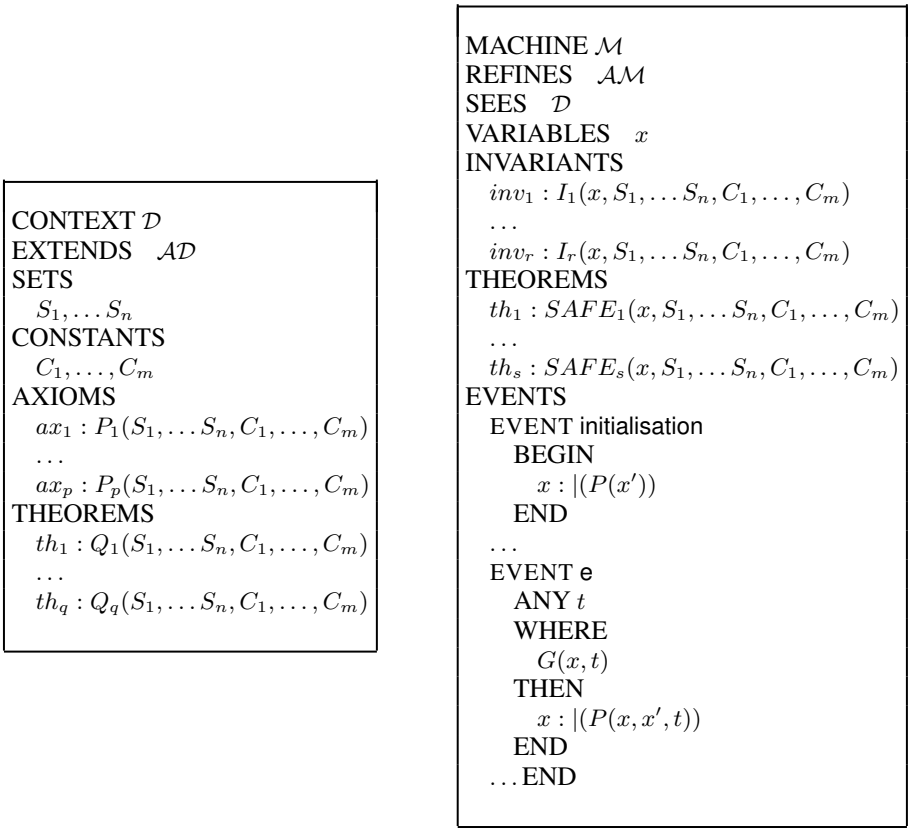


Figure 10.1. *Context and Machine*

A context clearly states the static properties of the (system) model under construction. The *extends* construct enables reuse by extending a previously defined context.

The proof process is based on the management of sequents, with an associated environment for proof called $\Gamma(\mathcal{D})$. The proof environment includes axioms, properties and theorems already proved. An environment is initially provided, but the intention is to add new theorems. This means that we intend to prove the following properties in the sequent calculus style:

$$\text{for any } j \text{ in } \{1..q\}, \Gamma(\mathcal{D}) \vdash th_j : Q_j(S_1, \ldots S_n, C_1, \ldots, C_m).$$

Theorems for the context are proved using the RODIN tool, but it is clear that the process for constructing the domain $\mathcal{D}$ is crucial to modeling the system, from consideration of the triptych of Bjoerner [BJO 06a, BJO 06b, BJO 06c, BJØ 07] and variations of this methodology.

The possibility of reusing former definitions is crucial, but we do not consider this point in this paper. Instead, we *simulate* the reuse of theories by manipulating the contexts directly. Among the requirements, we can list the theorems of the context, and we can, in fact, interpret the triptych as follows: for any

$$j \text{ in } \{1..q\}, \mathcal{D} \longrightarrow th_j : Q_j(S_1, \ldots S_n, C_1, \ldots, C_m).$$

Here, it appears that the system is not mentioned, and this is the case for static properties. Therefore, we have an interpretation of the triptych for the static information, which can be reused later for any system.

10.3.4.2. *Machines*

The dynamic part of a model is expressed using the notion of the *machine* (see Figure 10.1). A machine is either a basic machine or a refinement of an abstract machine. A machine models a state via a list of variables x that are assumed to be modifiable by events listed in the machine. The view is assumed to be closed with respect to events. Each event maintains an assertion called an *invariant*, which is a conjunction of logical statements called inv_j. Each reached state satisfies properties of the theorem part called safety properties. POs are given in section 10.6, and they are generated and checkable by the RODIN framework. The validation of the machine M leads to the validation of the safety and invariance properties.

We can obtain a variation of the triptych ($\Gamma(\mathcal{D}, M)$ is an associated environment for proof) as follows:

– For any j in $\{1..r\}$, $\Gamma(\mathcal{D}, M) \vdash INITIALISATION(x') \Rightarrow I_j(x', S_1, \ldots S_n, C_1, \ldots, C_m)$

– For any j in $\{1..r\}$, for any event e of M, $\Gamma(\mathcal{D}, M) \vdash (\bigwedge_{j \in \{1..r\}} I_j(x, S_1, \ldots S_n, C_1, \ldots, C_m)) \wedge BA(e)(x, x') \Rightarrow I_j(x', S_1, \ldots, S_n, C_1, \ldots, C_m)$

– For any k in $\{1..s\}$, $\Gamma(\mathcal{D}, M) \vdash (\bigwedge_{j\in\{1..r\}} I_j(x, S_1, \ldots, S_n, C_1, \ldots, C_m)) \Rightarrow SAFE_k(x, S_1, \ldots S_n, C_1, \ldots, C_m)$

We use temporal operators for expressing the safety and invariant properties.

– For any j in $\{1..r\}$, $\mathcal{D}, M \longrightarrow \Box I_j(x, S_1, \ldots S_n, C_1, \ldots, C_m)$.

– For any k in $\{1..s\}$, $\mathcal{D}, M \longrightarrow \Box SAFE_k(x, S_1, \ldots S_n, C_1, \ldots, C_m)$.

We summarize the requirements expressed by the machine M as follows:

$$\mathcal{D}, M \longrightarrow \Box \left(\begin{pmatrix} \bigwedge_{j\in\{1..r\}} I_j(x, S_1, \ldots S_n, C_1, \ldots, C_m) \end{pmatrix} \begin{pmatrix} \bigwedge_{k\in\{1..s\}} SAFE_k(x, S_1, \ldots S_n, C_1, \ldots, C_m) \end{pmatrix} \right)$$

We will use the notation $\mathcal{I}(M)$ to stand for the invariant of the machine M and $\mathcal{SAFE}(M)$ to stand for the safety properties of the machine M. We have shown that requirements $\mathcal{R}$ are first expressed using the *always* temporal operator. To specify total correctness properties, we should extend the scope of the requirements language by adding *eventuality* properties. Eventuality properties will be defined in section 10.4, which will be specific to our methodology.

10.4. Formal development of a sequential algorithm

In this section, we discuss two simple case studies to illustrate how we can develop sequential algorithms using the Event B method following two development techniques. From previous works, we quote case studies developed by J.-R. Abrial [ABR 03b] and his transformation rules used from Event B models to obtain sequential algorithms; however, we have also proposed a method [MÉR 09b, MÉR 09a] providing a framework to guide refinement steps, relying on an interpretation of an event as a procedure call. The second approach allows us to produce recursive algorithms from an Event B model and to express an invariant in a simple way. Transformations techniques can be applied on the resulting recursive algorithms are used to

produce iterative algorithms implemented in a real programming language like Spec# [MÉR 13]. We illustrate these two techniques by two very simple examples: the problem *computing the sum of a vector of integer values* $v_1, \ldots, v_n$ and the problem *searching for an item* x *in a table* t.

10.4.1. *Derivation of an algorithm for computing the sum of a sequence of values by refinement and transformation of the model into an algorithm*

10.4.1.1. *Description of the problem*

At first, we state the sum s of the sequence v in the Event B language; the mathematical expression is easy: $s = \sum_{k=1}^{k=n} v(k)$. As the notation for the summation of a finite sequence is not available in Event B, we have to *define* this notion in a context *summation0*, which will contain inputs and specific notations of the problem.

Inputs of the problem n and v are defined as a non zero natural number ($axm1$ and $axm2$) and a total function defined on $1..n$ and ranging over $\mathbb{N}$ ($axm3$). We have to define the underlying theory of the problem.

Second, we introduce a sequence u of values defining partial summations: $\sum_{k=1}^{k=i} v(k)$, which is inductively defined:

– u is a total function from $\mathbb{N}$ into $\mathbb{N}$ (axiom $axm4$).

- Initially, the summation starts by 0 and $u(0) = 0$ (axiom $axm5$).

– When i is smaller than n, the value $u(i)$ is defined from $u(i-1)$ and $v(i)$ (axiom $axm6$).

– When i is greater than n, the value of $u(i)$ is $u(n)$ (axiom $axm7$).

Axioms are given in the context *summation0* and constitutes a theory which will be used for proving properties of models.

CONTEXT $summation0$
CONSTANTS
n, v, u
AXIOMS
$axm1 : n \in \mathbb{N}$
$axm2 : n \neq 0$
$axm3 : v \in 1 \mathinner{..} n \rightarrow \mathbb{N}$
$axm4 : u \in \mathbb{N} \rightarrow \mathbb{N}$
$axm5 : u(0) = 0$
$axm6 : \forall i \cdot i \in \mathbb{N} \wedge i > 0 \wedge i \leq n \Rightarrow u(i) = u(i-1) + v(i)$
$axm7 : \forall i \cdot i \in \mathbb{N} \wedge i > n \Rightarrow u(i) = u(n)$
THEOREMS
$thm1 : \forall i \cdot i \in \mathbb{N} \Rightarrow u(i) \geq 0$
END

In the above context, it is noted that the clause THEOREMS is used and its use allows us to derive properties for mathematical data defined by their axioms. In the current tool Rodin, the authors merge axioms and theorems in the clause AXIOMS. However, among the list of statements, the tool identifies the two different sets of statements for axioms and theorems. We use a notation that allows a better expression of these theories. Finally, each axiom is validated by a set of generated POs to ensure consistency of definitions. It is the same for theorems that must be proved from an environment defined by the axioms with the rules of proof assistant. So we have defined the mathematical framework of the problem and we will now define the problem of summation of the sequence v.

10.4.1.2. *Specification of the problem to solve*

The problem is to calculate the value of sum of elements of the sequence v. We define a machine *summation1,* which is a model expressing through the event summation, the expression of the *postcondition* $sum = u(n)$. In fact, new value of the variable sum amount is $u(n)$, when the event summation1 has been observed. The initial value of sum is any initialization. Finally, the variable sum must satisfy the simple invariant $inv1 : sum \in \mathbb{N}$. The event summation1 is simply an assignment of value $u(n)$ to sum.

MACHINE $summation1$
SEES $summation0$
VARIABLES
sum
INVARIANTS
$inv1 : sum \in \mathbb{N}$
EVENT INITIALISATION
BEGIN
$act1 : sum :\in \mathbb{N}$
END
EVENT summation1
BEGIN
$act1 : sum := u(n)$
END
END

We can state an expression as a HOARE triple HOARE: $\{n > 0 \wedge v \in 1 \,..\, n \rightarrow \mathbb{N}\} SUMMATION \{sum = u(n)\}$ where $SOMMATION$ is the algorithmic solution. The visible data or inputs are in the context *summation0*. The problem is then to find an algorithm $SUMMATION$ computing the value $u(n)$ and storing it in the variable sum. C. Morgan [MOR 90] uses the same method and we are only simulating his refinement calculus, with the objective to construct an algorithmic solution from a pre and post specification.

We have described the domain of the problem and we have formulated what we want to calculate. The next step is the development of *calculation method*, which requires an *idea of solution* using refinement.

10.4.1.3. *Refining for computing*

We have defined the specification of the problem *calculating the sum of elements of the sequence* v and now we must find an algorithmic method for computing the value $u(n)$. In the previous machine, we state that *what to compute* and now we define *how to compute*. The assignment $sum := u(n)$ is an expression for combining a variable sum and a constant $u(n)$. A well-known trivial and inefficient solution is to store the values of sequence u in a table t and to translate the assignment as $sum := t(n)$ where t verifies the property $\forall k.k \in dom(t) \Rightarrow t(k) = u(k)$ and this property forms an invariant $inv8$. The idea is to use the variable t ($t \in 0 \; nupto \nrightarrow \mathbb{N}$) to control the calculation and its progression. The progression is ensured by the event step2 that decreases the value $n - i$ and thus ensures the convergence of the process.

MACHINE $summation2$
REFINES $summation1$
SEES $summation0$
VARIABLES
sum, t, i
INVARIANTS
$inv1 : i \in \mathbb{N}$
$inv2 : i \geq 0$
$inv3 : i \leq n$
$inv4 : t \in 0 \mathinner{..} n \pfun \mathbb{N}$
$inv5 : dom(t) = 0 \mathinner{..} i$
$inv6 : n \notin dom(t) \Rightarrow i < n$
$inv7 : dom(t) \subseteq dom(u)$
$inv8 : \forall k \cdot \begin{pmatrix} k \in dom(t) \\ \Rightarrow \\ t(k) = u(k) \end{pmatrix}$
$inv9 : dom(u) = \mathbb{N}$

EVENT INITIALISATION
BEGIN
$act1 : sum :\in \mathbb{N}$
$act2 : t := \{0 \mapsto 0\}$
$act3 : i := 0$
END
EVENT summation2
REFINES summation1
WHEN
$grd1 : n \in dom(t)$
THEN
$act1 : sum := t(n)$
END
EVENT step2
WHEN
$grd11 : n \notin dom(t)$
THEN
$act11 : t(i+1) := t(i) + v(i+1)$
$act12 : i := i + 1$
END
END

The model $summation2$ describes a process that gradually fills t and therefore retains all intermediate results. POs are fairly easy at some extent to prove through proof assistant. We summarize a proof statistics table at the end of development. It is quite clear that the variable t is in fact a witness or a track of intermediate values and this variable can be hidden in this model, when it will be refined. Before to hide this variable, we will put aside the value to maintain $t(i)$.

10.4.1.4. *Focus on a value to keep*

The next refinement $summation3$ leads to introduce a new variable $psum$ that will hold the value $t(i)$. It thus makes a *superposition* [CHA 88] on the model. The idea is that this model refines or simulates the previous model $summation2$; it also means that the properties of the refined model are verified by the new model $summation3$ as long as all the POs are discharged.

MACHINE $summation3$
REFINES $summation2$
SEES $summation0$
VARIABLES
$sum, i, t, psum$
INVARIANTS
$inv1 : psum \in \mathbb{N}$
$inv2 : psum = u(i)$
EVENT INITIALISATION
BEGIN
$act1 : sum :\in \mathbb{N}$
$act2 : i := 0$
$act3 : t := \{0 \mapsto 0\}$
$act4 : psum := 0$
END

EVENT summation3
REFINES summation2
WHEN
$grd1 : n \in dom(t)$
$grd2 : i = n$
THEN
$act1 : sum := psum$
END
EVENT step3 REFINES step2
WHEN
$grd1 : n \notin dom(t)$
$grd2 : i < n$
THEN
$act1 : t(i+1) := t(i) + v(i+1)$
$act2 : i := i + 1$
$act3 : psum := psum + v(i+1)$
END
END

This model is very expressive and provides extensive information to ensure that the model is correct with respect to the specification expressed in the model $summation1$. It is even clear that this model $summation3$ is expensive in terms of use of variables. Refinement allows us to select only useful variables for calculation. In the following, we will make the more algorithmic model and keep the model sufficient concrete for calculating variables.

10.4.1.5. *Obtaining an algorithmic model*

In this last step, we refine the model $summation3$ by a model $summation4$ and we hide the variable t in the abstract model $summation3$. Thus, the model $summation4$ includes variables sum, $psum$ and i and it should also be noted that it satisfies safety properties called theorems in the model $summation4$. The properties are proved from the properties of the previous refined models. Here, we have got a model with an initialization and two events:

– The event summation4 is observed, when the value of i is n and, in this case, the variable $psum$ contains the value $u(n)$. The invariant ensures that the value of $psum$ is $u(n)$.

– The event step4 is observed, when the value of i is smaller than n. It means that, while this value is smaller than n, the event can be observed and the traces generated from these events correspond to an iterative construct.

MACHINE $summation4$
REFINES $summation3$
SEES $summation0$
VARIABLES
$sum, i, psum$
THEOREMS
$inv1 : psum = u(i)$
$inv2 : i \leq n$
EVENT INITIALISATION
BEGIN
$act1 : sum :\in \mathbb{N}$
$act2 : i := 0$
$act3 : psum := 0$
END

EVENT summation4 REFINES summation3
WHEN
$grd1 : i = n$
THEN
$act1 : sum := psum$
END
EVENT step4 REFINES step3
WHEN
$grd1 : i < n$
THEN
$act1 : i := i + 1$
$act2 : psum := psum + v(i + 1)$
END
END

J.-R. Abrial [ABR 10a] proposes rules for progressively transforming models into algorithm. These rules are simple and we are considering them in our example.

Fusion of two events for deriving an iteration

Consider the two events which can be merged to obtain an algorithmic expression:

– If P is an invariant for S, then the two events can be merged into one event:

WHEN
P
Q
THEN
S
END

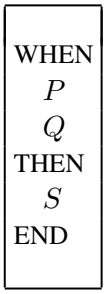
WHEN
P
$\neg Q$
THEN
T
END

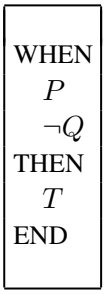
WHEN
P
THEN
WHILE Q DO
S
OD;
T;
END

– If P is not in the events, then there is no guard.

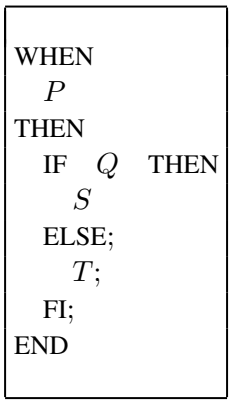
Merging two events for deriving a conditional statement.

Consider the two events which can be merged to obtain an algorithmic expression:

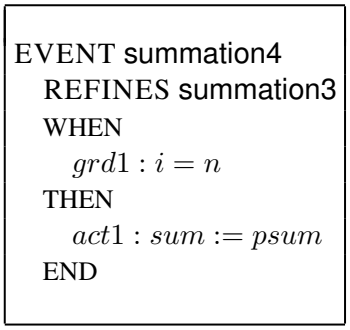


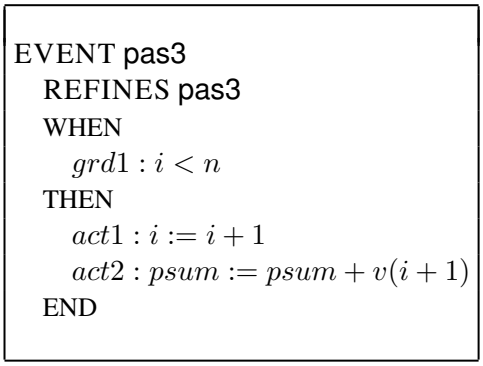


– If the condition on P is weaker then we can introduce a conditional statement:



These two transformations are correct, since they preserve the generated traces. In our case, we can apply the first transformation on the model $summation4$. Let the three events of $summation4$:





EVENT INITIALISATION
BEGIN
$act1 : sum :\in \mathbb{N}$
$act2 : i := 0$
$act3 : psum := 0$
END

BEGIN
$act1 : sum :\in \mathbb{N}$;
$act2 : i := 0$;
$act3 : psum := 0$;
WHILE $grd1 : i < n$ DO
$act2 : psum := psum + v(i+1)$
$act1 : i := i + 1$
OD;
$act1 : sum := psum$
END

The algorithm is obtained by merging the two events summation4 and step4 in an iteration and by sequential composition of the initialization.

$$\left\{\begin{pmatrix} n > 0 \\ \wedge v \in 1\,..\,n \to \mathbb{N} \end{pmatrix}\right\}$$

BEGIN
$sum :\in \mathbb{N}$;
$i := 0$;
$psum := 0$;
WHILE $i < n$ DO
$psum := psum + v(i+1)$
$i := i + 1$
OD;
$sum := psum$
END

$$\{sum = u(n)\}$$

Examples which have been treated with this method, can be found on the website dedicated to the Rodin project. In the publications [ABR 10a, ABR 03b], J.-R. Abrial has addressed both this technique and examples in more or less complicated way. Before concluding this study, it is important to give statistics on the number of POs and the difficulties of proofs, which are proved manually with the help of proof assistant. Table 10.1 indicates that 78.2% of POs are proved automatic but 21.7%, made by interaction with proof assistant are not complicated, as long as we use the progressive refinement.

Model	Total	Auto	Manual	% Auto	% Manual
summation0	5	0	5	0%	100%
summation1	4	3	1	75%	25%
summation2	23	21	2	87%	13%
summation3	7	5	2	71%	29%
summation4	7	7	0	100%	0%
Total	46	36	10	78,2%	21,7%

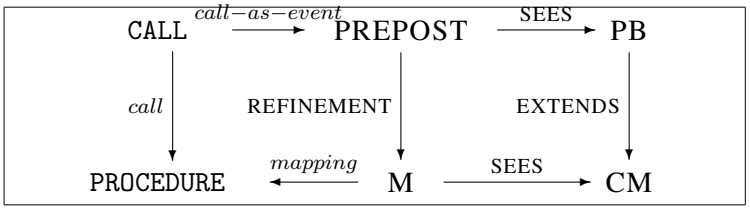
Table 10.1. *Table for statistics for the development of summation*

10.4.2. *Development of a sequential algorithm using the proof-based pattern call-as-event*

If we consider the problem to solve, we recall that we try to write a PROCEDURE correctly with respect to the pre/post specification:

PROCEDURE PROCEDURE(x; VAR y)
PRECONDITION $P(x)$
POSTCONDITION $Q(x, y)$

For the second development of a sequential algorithm, we use the proof-based pattern:



The schema is explained as follows:

– CALL is the call of PROCEDURE.

– PREPOST is the machine containing the events stating the pre- and post-conditions of CALL and PROCEDURE, and M is the refinement machine of PREPOST, with events including control points defined in CM.

– The *call-as-event* transformation produces a model PREPOST and a context PB from `CALL`.

– The *mapping* transformation allows us to derive an algorithmic procedure that can be mechanized.

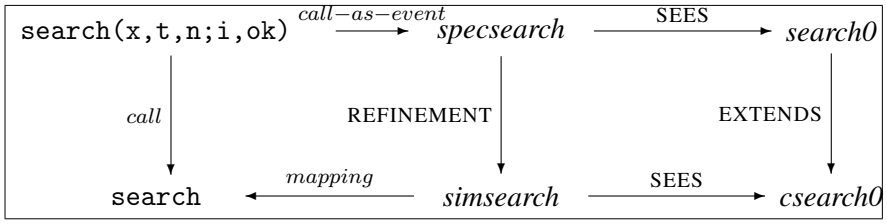
– `PROCEDURE` is a node corresponding to a procedure derived from the refinement model M. `CALL` is an instantiation of `PROCEDURE` using parameters x and y.

– M is a refinement model of PREPOST, which is transformed into `PROCEDURE` by applying structuring rules. It may contain events corresponding to the calls of other procedures.

We consider the problem *searching a value v in a table t*. The specification is stated as follows:

PROCEDURE `search`$(x, t, n;$ VAR $i, ok)$
PRECONDITION $x \in A \land n > 0 \land t \in 1 \,..\, n \rightarrow A$
POSTCONDITION $\begin{pmatrix} (\forall k \cdot k \in 1 \,..\, n \Rightarrow t(k) \neq x) \Rightarrow ok = no \\ (\exists k \cdot k \in 1 \,..\, n \land t(k) = x) \Rightarrow ok = yes \end{pmatrix}$

We try to identify the following elements of the pattern for our problem:



CONTEXT $search0$
SETS
A
$REPLIES$
CONSTANTS
x, t, n, yes, no
AXIOMS
$axm1 : n \in \mathbb{N}_1$
$axm2 : x \in A$
$axm3 : t \in 1 \mathinner{..} n \rightarrow A$
$axm4 : REPLIES = \{yes, no\}$
$axm5 : yes \neq no$
END

Describing the context of the problem PB is given by a context $search0$ that easily defines the structure of search t. We also need to define a set of possible results. This context is actually used to correctly describe the pre-condition of the search $search0$. We can now define the specification itself by writing the machine $specsearch$ which will include events simulating the call `search`.

The machine $specsearch$ includes an initialization event and two events: find that models the procedure search, when it finds an element x in the table t, and unfind which models the procedure `search`, when it finds no element x in the table t. In fact, the two events give a definition of *what* but not of *how*, and these events are a simple way to describe the expected behavior. To define in a more operational way, we will *refine*. Thus, these two events are only two instances of calling this procedure.

MACHINE $specsearch$
SEES $search0$
VARIABLES
i, ok
INVARIANTS
$inv1 : ok \in REPLIES$
$inv2 : i \in 1 \mathinner{..} n$

EVENT INITIALISATION
BEGIN
$act1 : i :\in 1 \mathinner{..} n$
$act2 : ok := no$
END

EVENT find
ANY
j
WHERE
$grd1 : j \in 1 \mathinner{..} n$
$grd2 : t(j) = x$
THEN
$act1 : ok := yes$
$act2 : i := j$
END
EVENT unfind
WHEN
$grd1 : \forall k \cdot k \in 1 \mathinner{..} n \Rightarrow t(k) \neq x$
THEN
$skip$
END
END

To solve our problem from an operational point of view, we have to analyze the problem by considering several cases and we introduce a new variable c, which models the control of the search process. We use a new context $csearch0$ that extends $search0$ defining possible control points:

CONTEXT $csearch0$
EXTENDS $search0$
SETS
  $LOCS$
CONSTANTS
  $start, end, call1$
AXIOMS
  $axm1 : partition(LOCS, \{start\}, \{end\}, \{call1\})$
END

MACHINE $simsearch$
  REFINES $specsearch$
SEES $csearch0$
VARIABLES
  i, ok, c
INVARIANTS
  $inv1 : c \in LOCS$
  $inv2 : c = call1 \Rightarrow n \neq 1 \wedge ok = no$
  $inv3 : c = call1 \Rightarrow t(n) \neq x$
  $inv4 : c = end \wedge ok = yes \Rightarrow t(i) = x$
  $inv5 : c = end \wedge ok = no \Rightarrow (\forall g \cdot g \in 1 \,..\, n \Rightarrow t(g) \neq x)$
  $inv6 : c = start \Rightarrow ok = no$
...

The invariant describes what is happening during the computation:

– When the control point is at the *end* and when the variable *ok* is yes, then $t(i) = x$.

– When the control point is at the *end* and when the variable *ok* is no, then i contains any value and x does not occur in t.

To perform this calculation, two cases are introduced either n is equal to 1 or n is not equal to 1. Consider the case where n is 1. In this case, we refine find by findone, to explain how the value of x can be found in an array with only one value, if it is indeed in this table, and we refine unfind by notfoundone in case the value of x is not in t (i.e. $t(1) \neq x$). We consider the two sub-cases and the translation into an algorithmic notation of these two events is immediate. Each event (findone and notfindone) is translated in the form of a conditional statement. We can also use EB2ALL [MÉR 11b] tool to translate the full model and get a C, C++, C# or Java program.

EVENT INITIALISATION
BEGIN
$act1 : i :\in 1 \,..\, n$
$act2 : ok := no$
$act3 : c := start$
END

EVENT findone
REFINES find
ANY
j
WHERE
$grd1 : j \in 1 \,..\, n$
$grd2 : t(j) = x$
$grd3 : c = start$
$grd4 : n = 1$
$grd5 : t(n) = x$
THEN
$act1 : ok := yes$
$act2 : i := 1$
$act4 : c := end$
END

EVENT nofindone
REFINES unfind
WHEN
$grd1 : \forall k \cdot k \in 1 \,..\, n \Rightarrow t(k) \neq x$
$grd2 : n = 1$
$grd3 : t(n) \neq x$
$grd4 : c = start$
THEN
$act1 : c := end$
END

For the second case, we assume that n is not equal to 1 and we will refine both events find and unfind, for simulating a recursive search. We have events in parts related to the recursive analysis and these events are controlled using c:

– foundlastone *finds* the x in the last cell of the table t and sets the variable ok to yes. The control variable c gets the value end and the search is completed.

– notfoundlastone does find the value of x in the last cell of the table t and the *searching* process should continue on the remaining unvisited cells of the table t between 1 and $n-1$. The control is switching to $call1$ by updating c.

– The two next events are observed when $c = call1$, depending on whether there is a value. Each of these events simulates the procedure `search` between 1 and $n-1$. Obviously, it does not say how the searching process is done and we translate these two events by recursive calls. This point simplifies invariants and proofs; reference may be refer to the document [MÉR 09b, MÉR 09a] introducing this technique to refer the calculation of the shortest path and thus to find that the invariant is fairly easy to find even if it is complex in its final form.

EVENT foundlastone
REFINES find
WHEN
$grd1 : n \neq 1$
$grd2 : t(n) = x$
$grd3 : c = start$
WITNESSES
$j : j = n$
THEN
$act1 : c := end$
$act2 : i := n$
$act3 : ok := yes$
END
EVENT notfoundlastone
WHEN
$grd1 : c = start$
$grd2 : n \neq 1$
$grd3 : t(n) \neq x$
THEN
$act1 : c := call1$
END

EVENT foundrec
REFINES find
ANY
k
WHERE
$grd1 : k \in 1 \,..\, n-1$
$grd2 : c = call1$
$grd3 : t(k) = x$
WITNESSES
$j : j = k$
THEN
$act1 : c := end$
$act2 : i := k$
$act3 : ok := yes$
END
EVENT notfounrec
REFINES unfind
WHEN
$grd1 : \forall l \cdot l \in 1 \,..\, n-1 \Rightarrow t(l) \neq x$
$grd2 : c = call1$
THEN
$act1 : c := end$
END
END

We have to derive an algorithm from the list of events in the last model.

```
Procedure search(x, t, n; i, ok)
BEGIN
  i :∈ 1 .. n; ok := no;
  IF  n = 1 ∧ t(n) = x  THEN
    ok := yes;  i := 1
  ELSE IF  n = 1 ∧ t(n) ≠ x  THEN
    skip
  ELSE IF  n ≠ 1 ∧ t(n) = x  THEN
    ok := yes;  i := n;
  ELSE  search(x, t, n − 1, i, ok);
  FI
END
```

The procedure (or algorithm) is generated by transformations of events into fragments of codes and these fragments are organized according to the variable c.

As we have already pointed out, this technique simplifies the construction of the invariant and also simplifies its proof. We [MÉR 09b, MÉR 09a] have made a quite as classic examples calculation of the binomial coefficients, calculating the shortest path, the primitive recursive functions, the CYK algorithm analysis syntax. The tool EB2ALL [MÉR 11b] could be used to translate these models into C, C++, C# or Java. These two techniques of development simulate the method of C. Morgan [MOR 90] but the difference lies in the systematization of the refinement as simple as possible. Do not develop the model too quickly but introduce machines or intermediate models that will simplify the work of proof. Finally, note that this model has used the clause WITNESSES in the event foundrec in the form of $j : j = k$, this clause allows the prover to help for instantiating an existential quantifier that expresses in the abstract event refined by foundrec, it must be given a value j to observe the abstract event. This device allows to retain information in the proof of the abstract model. At the balance sheet of POs, Table 10.2 describes automatic and interactive proofs, where three interactive proofs require some simple interactions.

AQ5

10.5. Development of a distributed algorithm

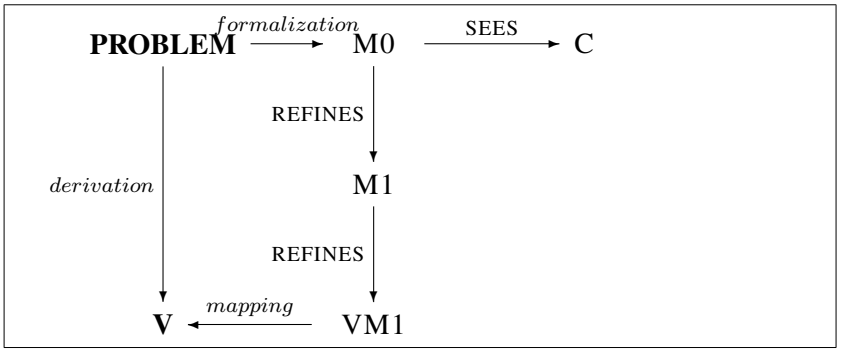
10.5.1. *Modeling distributed algorithms*

We will illustrate a technique for developing distributed algorithms using the Event B method. This technique relies on a model of distributed

computing called Visidia [MOS] and the objective is to produce an algorithm. We consider the problem of spanning tree of a graph and we consider a proof-based pattern integrating the refinement and allowing to develop Visidia algorithms, which can be simulated on the platform Visidia [MOS]. The pattern of development is characterized by the following diagram:

Model	Total	Auto	Manual	% Auto	% Manual
search0	0	0	0	0%	0%
csearch0	0	0	0	0%	0%
specsearch	5	5	0	100%	0%
simsearch	52	49	3	94,2%	5,8%
Total	57	54	3	94,7%	5,3%

Table 10.2. *Table with statistics for proof effort in the development of the* `search` *procedure*



– The context *C* details the required properties of graphs, as distributed algorithms often use properties of graphs.

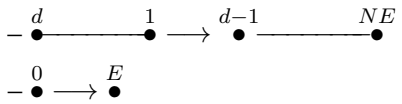
– The machine *M0* describe the problem to solve by giving an abstract event-based expression, for instance, the leader election in a network is expressed by the emergence of a node, which knows that it is the leader and the other nodes know that they are not leader but a leader can be elected. The existence of a solution obviously depends on the properties of the supporting graph of the distributed computing.

– Refinement of *M0* by *M1* expresses how a Visidia model performs a computation; a model in Visidia is a list of relabeling rules for graph, that simulates the execution of a distributed computing by localizing the computations at node or even between two neighbors. The model is very simple and relatively abstract, but is supported by a simulation tool.

– The next refinement simplifies the model *M1*, a model where no relabeling rule appears.

– *V* is a Visidia model derived from *VM1*; *mapping* ensures the translation of *VM1* into VISIDIA [MOS].

The leader election is simply defined by rules applied on two neighbors nodes:



Each node is labeled by the number of neighbors and the application of rules is non-deterministic. The leader is a unique node, where all neighboring nodes request this node to be a leader. We have developed the leader election protocol for IEEE 1394 [ABR 03a] on this principle, but up to a more concrete level, without resolving the issue of probabilities inherent in this type of algorithm. In Figure 10.2, we give a leader example in graph labeled with execution rules. Note that these rules calculate the leader in a graph without cycle.

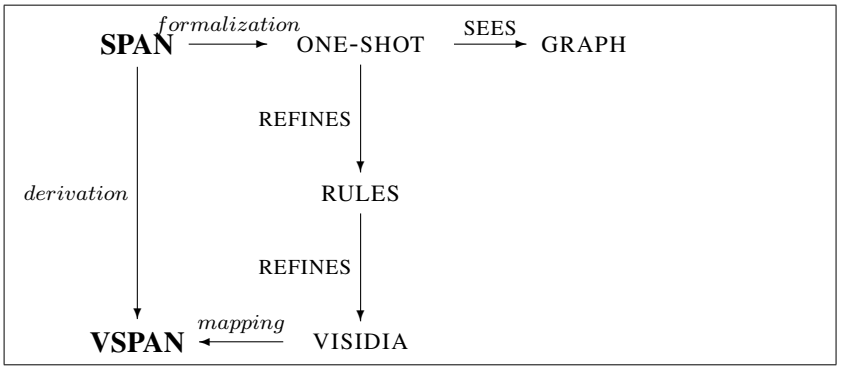
Figure 10.2. *Graph for the leader election*

AQ6

We introduce rules for computing the spanning tree in the computing model. We present the development of the computation of spanning tree in this model (see Figure 10.3).

Figure 10.3. *Computing the spanning tree of graph with a model Visidia*

We apply the proof-based pattern for solving the problem of computing a spanning tree of a connected graph. The problem is called SPAN.



10.5.2. *Elements of a proof-based pattern*

The modeling of graphs is the starting point of this development. The context $graph$ defines a graph g as a subset of the set $N \times N$ and adds axioms characterizing that it is symmetrical ($axm2$) and connected ($axm3$).

CONTEXT $graph$
SETS
  N
CONSTANTS
  g, r
AXIOMS
  $axm0 : r \in N$
  $axm1 : g \subseteq N \times N$
  $axm2 : g = g^{-1}$
  $axm3 : \forall s \cdot s \subseteq N \wedge r \in s \wedge g[s] \subseteq s \Rightarrow N \subseteq s$
END

Then, we give a predicated expression in an event for computing in one shot of a spanning tree. The machine $one - shot$ has a single-event span that sets the variable $span$ a spanning tree of graph g. The invariant is simply the expression $span$ is a subset of g, but it is a spanning tree as indicated by the expression of the value ar. The important point is to demonstrate that the value at exists. This is proved by showing that this event is feasible and derived from the existence of a spanning tree in the mathematical world.

MACHINE $one - shot$
SEES $graph$
VARIABLES
$span$
INVARIANTS
$inv2 : span \subseteq g$
EVENT INITIALISATION
BEGIN
$act2 : span := \varnothing$
END
EVENT span
ANY
at
WHERE
$grd1 : at \subseteq g$
$grd2 : at \in N \setminus \{r\} \rightarrow N$
$grd3 : \forall s \cdot s \subseteq N \wedge r \in s \wedge at^{-1}[s] \subseteq s \Rightarrow N \subseteq s$
THEN
$act1 : span := at$
END
END

Then, we refine this machine by another machine *simulating* the computation of this tree using two variables a and r. The variable a is used to contain nodes already selected during the calculation for the family tree and tr contains the spanning tree in construction. The invariant expresses that r is a forest that is to say that tr is a subset of g without cycle ($inv7$). The invariant $inv6$ expresses that tr is a total function with a domain a without r and tr plays the role of root of this tree.

MACHINE $rules$ REFINES $one-shot$
SEES $graph$
VARIABLES
$span, tr, a$
INVARIANTS
$inv4 : a \subseteq N$
$inv2 : tr \subseteq g$
$inv5 : r \in a$
$inv6 : tr \in a \setminus \{r\} \rightarrow a$
$$inv7 : \forall s \cdot \begin{pmatrix} s \subseteq a \\ \wedge r \in s \\ \wedge tr^{-1}[s] \subseteq s \end{pmatrix} \Rightarrow a \subseteq s$$

Two events span and rule1 model the possible modifications of tr and a. We note that it is important to choose a special node starting the process and a is initialized to the singleton containing r an arbitrary node. The event span detects the end of the process by testing whether a contains all the elements of N and sets $span$ to the value of tr. The role of rule1 is different and it chooses a node y not yet in a but that is reachable from a node of a by the graph g. This condition aims to avoid creating a cycle.

EVENT INITIALISATION
BEGIN
$act4 : span := \varnothing$
$act2 : tr := \varnothing$
$act3 : a := \{r\}$
END
EVENT span
REFINES span
WHEN
$grd1 : a = N$
WITNESSES
$at : at = tr$
THEN
$act1 : span := tr$
END

EVENT rule1
ANY
x, y
WHERE
$grd1 : x \in N$
$grd2 : y \in N$
$grd3 : x \mapsto y \in g$
$grd4 : x \in a$
$grd5 : y \notin a$
THEN
$act2 : a := a \cup \{y\}$
$act1 : tr := tr \cup \{y \mapsto x\}$
ENDEND

The machine $visidia$ refines the machine $rules$ by making it closer to Visidia model. In fact, it is a refinement to make symmetric the tree and to transform events in the rules of model $visidia$. For representing the membership of a, we use the black color. The new variable lb is used to localize this information for each node a and the invariant $inv2$ expresses this relationship property between nodes a and black nodes. Colors of marking are expressed by the set $MARKING$. At the initialization, all nodes are in white color except r.

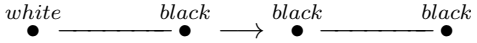
MACHINE $visidia$ REFINES $rules$
SEES $cvisidia$
VARIABLES
tr, lb
INVARIANTS
$inv1 : lb \in N \rightarrow MARKING$
$inv2 : \forall i \cdot i \in a \Leftrightarrow lb(i) = BLACK$

The two events span and rule1 refine events with the same name in the model $rules$ and express local conditions in the variable lb.

EVENT INITIALISATION
BEGIN
$act2 : tr := \varnothing$
$act4 : lb := lb0$
END
EVENT span
REFINES span
WHEN
$grd1 : \forall i \cdot i \in N \Rightarrow lb(i) = BLACK$
THEN
$skip$
END

EVENT rule1
REFINES rule1
ANY
x, y
WHERE
$grd3 : x \mapsto y \in g$
$grd4 : lb(x) = BLACK$
$grd5 : lb(y) = WHITE$
THEN
$act2 : lb(y) := BLACK$
$act1 : tr := tr \cup \{y \mapsto x\}$
END
END

The last step is the generation of rules in the distributed programming model Visidia and from the event rule1, we derive only one rule when one of the nodes is black at the initial state.



We extract the rule defining the Visidia program from the events of the machine $Visidia$, which contains in its events only localizable information. We can therefore deduce the distributed program that builds a spanning tree. The issue of convergence of this system is inferred from the analysis of decreasing the set $N\ a$ by the event rule1.

10.6. Tools

The Event B method is supported by tools like Atelier B [CLE 02] tool and Rodin [ABR 10b] platform.

10.6.1. *Atelier B*

The Atelier B tool [CLE 02] is freely distributed by the company ClearSy, which is proposed for the four platforms Windows, Linux, MacOS and Solaris; distribution under the license and provides access to the documentations and case studies. This platform proposes features in a single frame for the Classical B method and Event B, where Event B syntax is slightly different. The offered features include the generation of POs to support the interactive proof, automatic refinement with Bart [CLE 10] tool and translation tools to C or ADA . The same company continues the free distribution of a platform called B4Free [CLE 04] based on the joint work of J.-R. Abrial and D. Cansell on the balbulette [ABR 03c]. The idea of balbulette provides an interface with the components of Atelier B as proof obligation generator (POG), the prover or translators, to facilitate the developer's task in the approach of interactive proof and project management. One of the difficulties in the use of tools such as Atelier B lies in the interactive use of the proof assistant to discharge the POs that could not be handled by the automatic procedures. B4Free offers support during the process of proof and applying rules. This tool was a great success with the academic partners and its features are integrated into the Rodin platform.

10.6.2. *The Rodin platform*

The Rodin platform is supporting the Event B method in the Eclipse environment, and follows the work in the framework tools like Click'n'Prove [CAN]. It is dedicated to Event B but only provides

functionality as plugins (translation into programming languages from Event B models or integration methodologies like UML). The Rodin platform was used to develop case studies illustrating this text and we [MÉR 09c, MÉR 10d, MÉR 10b, MÉR 10c] have used complementary tools like ProB [Hei], which provides the functionality, such as animation and model checking.

10.7. Conclusion and perspectives

10.7.1. *Applications in case studies*

The applications of this technique are numerous and the development of tools has facilitated these case studies. In our presentation, we have mainly used the Rodin platform but the Atelier B platform can be substituted. The proof assistant is partly provided by the platform and provers have been developed for Rodin. We will mention some applications developed on this platform, in order to show both the Rodin platform and the Atelier B
AQ7 platform, which may also be modeled and the strength of tools.

Distributed algorithms [ABR 03a] constitute a class of interesting complex algorithmic problems; the development of the leader election in the case of a acyclic undirected network has opened avenues of research for exploring issues of time integration [CAN 07b, REH 09] in development and management inherent and often implicit. Among distributed algorithms, the cryptographic algorithms constituent also an interesting class to measure the impact of refinement in their derivation but measure the expressive power of language Event B face model the Dolev−Yao attacker [BEN 08]. This has led to the development of algorithms for authentication distribution key [BEN 09c, BEN 09b, BEN 09a, BEN 10b, BEN 10a] highlighting basic mechanisms constituting these algorithms. To some extent, the difficulties lie in the understanding of property being modeled. These studies have also led
AQ8 the proposed development patterns facilitating introducing time [CAN 07b, REH 09] and patterns of development in the programming model distributed Visidia [MÉR 11a, MÉR 10a]. More recently, the issue of dynamic networks like networks graphs that evolve over time was studied in Event B for the discovery of topology [HOA 09b] or routing dynamic [MÉR 11c].

Regarding sequential algorithms, J.-R. Abrial [ABR 03b] has proposed rules for translation the Event B models into an algorithmic notation. The approach has been outlined in this chapter and actually allows us to (re)develop sequential algorithms. The approach based on the relation *call - event* [MÉR 09b, MÉR 09a] allows a relatively simple development of sequential algorithms facilitating the expression of invariant and highlighting a recursive analysis of the problem.

More conventionally, the Event B method is used to develop systems integrating software components and requiring objective arguments to certify their operation. J.-R. Abrial has designed a model of a mechanical press [ABR 10a] for ensuring to maintain the security properties. Event B is an effective engineering framework with a formal system and a set of proof-based patterns development [ABR 10a, HOA 09a] and structures, refinement charts [MÉR 11f]. Among the important studies, there are several case studies like pacemaker modeling [MÉR 10b, MÉR 11e, SIN 11] electric heart model [MÉR 11g, SIN 13, SIN 11] and medical protocols [MÉR 11g, SIN 13, SIN 11]. Modeling related to the security issues as the access control [BEN 07, BEN 10a] have also showed that the Event B language is very flexible to integrate access control models as RBAC or ORBAC. Finally, the development of Event B models [MÉR 11h, MÉR 11d] can produce the code using integrated tools in the Rodin platform, which can be further used for assembling the system.

10.7.2. *Conclusion and perspectives*

The Event B method is based on a powerful language based on set theory and first order predicate calculus; it provides simple structures *machines* to describe reactive systems. To some extent, it can be described in other languages for reactive systems but refinement is a key concept that allows to develop incrementally and safely complex models of relatively large size systems like a mechanical press or a pacemaker. Furthermore, the tools have matured in both the interface and the proof tools; they require some practice, but with the proof assistant or the ProB animator, each sheds light on developed models and contributes to the validation of models. At the end of outlook, we believe that the treatment of time, probabilistic aspects of systems integration languages less formal, proof-based patterns of

development and case studies are points to explore, while now a development
AQ9 tool [**?**] is freely available.

10.8. Bibliography

[ABR 96] ABRIAL J.-R., *The B book - Assigning Programs to Meanings*, Cambridge University Press, 1996.

[ABR 03a] ABRIAL J.-R., CANSELL D., MÉRY D., "A mechanically proved and incremental development of IEEE 1394 tree identify protocol.", *Formal Aspects of Computing*, vol. 14, no. 3, pp. 215–227, 2003.

[ABR 03b] ABRIAL J.-R., "Event based sequential program development: application to constructing a pointer program", ARAKI K., GNESI S., MANDRIOLI D. (eds.), *FME*, *Lecture Notes in Computer Science*, Springer, vol. 2805, pp. 51–74, 2003.

[ABR 03c] ABRIAL J.-R., CANSELL D., "Click'n prove: interactive proofs within set theory", BASIN D.A., WOLFF B. (eds.), *TPHOLs*, *Lecture Notes in Computer Science*, Springer, vol. 2758, pp. 1–24, 2003.

[ABR 10a] ABRIAL J.-R., *Modeling in Event-B: System and Software Engineering*, Cambridge University Press, 2010.

[ABR 10b] ABRIAL J.-R., BUTLER M. J., HALLERSTEDE S., *et al.,* "Rodin: an open toolset for modelling and reasoning in Event-B", *STTT*, vol. 12, no. 6, pp. 447–466, 2010.

[BAC 79] BACK R.J.R., "On correct refinement of programs", *Journal of Computer and System Sciences*, vol. 23, no. 1, pp. 49–68, 1979.

[BAC 89] BACK R.-J., KURKI-SUONIO R., "Decentralization of process nets with centralized control", *Distributed Computing*, vol. 3, no. 2, pp. 73–87, 1989.

[BAC 98] BACK R.-J., VON WRIGHT J., *Refinement Calculus A Systematic Introduction*, Graduate Texts in Computer Science, Springer-Verlag, 1998.

[BEN 07] BENAISSA N., CANSELL D., MERY D., "Integration of security policy into system modeling", *The 7th International B Conference – B2007*, Besançon, France, January 2007.

[BEN 08] BENAISSA N., "Modelling attacker's knowledge for cascade cryptographic protocols", BÖRGER E., BUTLER M., BOWEN J.P., *et al.* (eds.), *First International Conference on Abstract State Machines, B and Z – ABZ 2008*, *Lecture Notes in Computer Science*, Springer, London, United Kingdom, vol. 5238, pp. 251–264, 2008.

[BEN 09a] BENAISSA N., MÉRY D., "Cryptographic protocols analysis in Event B", *Seventh International Andrei Ershov Memorial Conference "PERSPECTIVES OF SYSTEM INFORMATICS" – PSI 2009*, Lectures Notes in Computer Science, Springer-Verlag, Novosibisrk, Russie, Fédération De, November 2009.

[BEN 09b] BENAISSA N., MÉRY D., "Cryptologic protocols analysis using proof-based patterns", *Seventh International Andrei Ershov Memorial Conference "PERSPECTIVES OF SYSTEM INFORMATICS" – PSI 2009*, Lecture Notes in Computer Science, Springer-Verlag, Novosibirsk, Russie, Fédération De, June 2009.

[BEN 09c] BENAISSA N., MÉRY D., "Développement combiné et prouvé de systèmes transactionnels cryptologiques", *Approches Formelles dans l'Assistance au Développement de Logiciels – AFADL 2009*, Toulouse, France, January 2009.

[BEN 10a] BENAISSA N., La composition des protocoles de sécurité avec la méthode B événementielle, PhD Thesis, Henri Poincaré University, Nancy I, May 2010.

[BEN 10b] BENAISSA N., MÉRY D., "Proof-based design of security protocols", MAYR E.W. (ed.), *5th International Computer Science Symposium in Russia, CSR 2010*, *Lecture Notes in Computer Science*, KAZAN, Russie, Fédération De, Farid Ablayev, Springer, vol. 6072, pp. 25–36, June 2010.

[BJO 06a] BJORNER D., *Software Engineering 1 Abstraction and Modelling*, Texts in Theoretical Computer Science. An EATCS Series, ISBN: 978-3-540-21149-5, Springer-Verlag, 2006.

[BJO 06b] BJORNER D., *Software Engineering 2 Specification of Systems and Languages*, Texts in Theoretical Computer Science. An EATCS Series, ISBN: 978-3-540-21150-1, Springer-Verlag, 2006.

[BJO 06c] BJORNER D., *Software Engineering 3 Domains, Requirements, and Software Design*, Texts in Theoretical Computer Science. An EATCS Series, ISBN: 978-3-540-21151-8, Springer-Verlag, 2006.

[BJØ 07] BJØRNER D., HENSON M.C. (eds.), *Logics of Specification Languages*, EATCS Textbook in Computer Science, Springer, 2007.

[CAN] CANSELL D., "Click'N'Prove". Available at http://plateforme-qsl.loria.fr/click%20n%20prove.php.

[CAN 07a] CANSELL D., MÉRY D., "*The Event-B Modelling Method: Concepts and Case Studies*", pp. 33–140, Springer, 2007. (see [BJØ 07])

[CAN 07b] CANSELL D., MÉRY D., REHM J., "Time constraint patterns for Event B development", JULLIAND J., KOUCHNARENKO O. (eds.), *7th International Conference of B Users, January 17–19, 2007*, of *Lecture Notes in Computer Science*, Besançon, France, Springer-Verlag, vol. 4355, pp. 140–154, 2007.

[CHA 88] CHANDY K.M., MISRA J., *Parallel Program Design A Foundation*, ISBN 0-201-05866-9, Addison-Wesley Publishing Company, 1988.

[CLA 00] CLARKE E.M., GRUNBERG O., PELED D.A., *Model Checking*, The MIT Press, 2000.

[CLE 02] CLEARSY, AIX-EN-PROVENCE (F), ATELIER B., 2002. Available at http://www.atelierb.eu.

[CLE 04] CLEARSY, AIX-EN-PROVENCE (F), B4FREE, 2004. Available at http://www.b4free.com.

[CLE 10] CLEARSY, AIX-EN-PROVENCE (F), BART, 2010. Available at http://www.atelierb.eu.

AQ10

[COU 78] COUSOT P., Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes, PhD Thesis, Scientific and Medical University of Grenoble, National Polytechnic Institute of Grenoble, AQ11 21 March 1978.

[COU 79] COUSOT P., COUSOT R., "Systematic design of program analysis frameworks", *Proceedings Records of Sixth Proceedings of the Symposium on Principles of Programming Languages*, San Antonio, Texas, pp. 269–282, 1979.

[COU 92] COUSOT P., COUSOT R., "Abstract interpretation frameworks", *Journal of Logic and Computation*, vol. 2, no. 4, pp. 511–547, 1992.

[COU 00] COUSOT P., "Interprétation abstraite", *Technique et science informatique*, vol. 19, no. 1–2–3, pp. 155–164, January 2000.

[DIJ 76] DIJKSTRA E.W., *A Discipline of Programming*, Prentice-Hall, 1976.

[FLO 67] FLOYD R.W., "Assigning meanings to programs", SCHWARTZ J.T. (ed.), *Proc. AQ12 Symp. Appl. Math. 19, Mathematical Aspects of Computer Science*,pp. 19–32, 1967.

[Hei] HEINRICH-HEINE-UNIVERSITÄT DÜSSELDORF, "The ProB animator and model checker". Available at http://www.stups.uni-duesseldorf.de/ProB.

[HOA 69] HOARE C.A.R., "An axiomatic basis for computer programming", *Communications of the Association for Computing Machinery*, vol. 12, pp. 576–580, 1969.

[HOA 09a] HOANG T.S., FURST A., ABRIAL J.-R., "Event-B patterns and their tool support", HUNG D.V., KRISHNAN P. (eds.), *SEFM*, IEEE Computer Society, pp. 210–219, 2009.

[HOA 09b] HOANG T.S., KURUMA H., BASIN D.A., *et al.,* "Developing topology discovery in Event-B", *Sci. Comput. Program.*, vol. 74, no. 11–12, pp. 879–899, 2009.

[HOL 97] HOLZMANN G., "The spin model checker", *IEEE Trans. on software engineering*, vol. 16, no. 5, pp. 1512–1542, May 1997.

[LAM 80] LAMPORT L., "Sometime is sometimes not never: a tutorial on the temporal logic of programs", *Proceedings of the Seventh Annual Symposium on Principles of Programming Languages*, pp. 174–185, 1980.

[LAM 94] LAMPORT L., "A temporal logic of actions", *Transactions On Programming Languages and Systems*, vol. 16, no. 3, pp. 872–923, May 1994.

[LAM 02] LAMPORT L., *Specifying Systems: The TLA$^+$+ Language and Tools for Hardware and Software Engineers*, Addison-Wesley, 2002.

[MCM 93] MCMILLAN K.L., *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

[MÉR 09a] MÉRY D., "A simple refinement-based method for constructing algorithms", *ACM SIGCSE Bulletin*, vol. 41, no. 2, pp. 51–59, June 2009.

[MÉR 09b] MÉRY D., "Refinement-Bbsed guidelines for algorithmic systems", *International Journal of Software and Informatics*, vol. 3, nos. 2–3, pp. 197–239, September 2009.

[MÉR 09c] MÉRY D., SINGH N.K., Pacemaker's functional behaviors in Event-B, Research report, University of Lorraine, 2009.

[MÉR 10a] MÉRY D., MOSBAH M., TOUNSI M., "Proving distributed algorithms by combining refinement and local computations", BENDISPOSTO J., LEUSCHEL M., ROGGENBACH M. (eds.), *AVOCS 2010 10th International Workshop on Automated Verification of Critical Systems*, Dusseldorf, Allemagne, Germany, September 2010.

[MÉR 10b] MÉRY D., SINGH N.K., "Functional behavior of a cardiac pacing system", *International Journal of Discrete Event Control Systems (IJDECS)*, December 2010. AQ13

[MÉR 10c] MÉRY D., SINGH N.K., Technical report on formal development of two-electrode cardiac pacing system, Research report, University of Lorraine, February 2010.

[MÉR 10d] MÉRY D., SINGH N.K., "Trustable formal specification for software certification", MARGARIA T., STE B. (eds.), *4th International Symposium On Leveraging Applications of Formal Methods – ISOLA 2010*, of Lecture Notes in Computer Science, Heraklion, Crete, Greece, Springer, vol. 6416, pp. 312–326, October 2010.

[MÉR 11a] MÉRY D., MOSBAH M., TOUNSI M., "Refinement-based verification of local synchronization algorithms", *17th International Symposium on Formal Methods*, Lecture Notes in Computer Science, Limerick, Irlande, Springer, June 2011.

[MÉR 11b] MÉRY D., SINGH N.K., "EB2C: a tool for Event-B to C conversion support", 2011. Available at http://eb2all.loria.fr.

[MÉR 11c] MÉRY D., SINGH N.K., "Analysis of DSR protocol in Event-B", *13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2011)*, 2011.

[MÉR 11d] MÉRY D., SINGH N.K., "Automatic code generation from Event-B models", *Proceedings of the 2011 Symposium on Information and Communication Technology (SoICT)*, Hanoi, Vietnam, 2011.

[MÉR 11e] MÉRY D., SINGH N.K., "Formal development and automatic code generation: cardiac pacemaker", *International Conference on Computers and Advanced Technology in Education (ICCATE 2011)*, 2011.

[MÉR 11f] MÉRY D., SINGH N.K., "Formal specification of medical systems by proof-based refinement," *ACM Transaction Embedded Computing Systems*, 2011. AQ14

[MÉR 11g] MÉRY D., SINGH N.K., "Formalisation of the heart based on conduction of electrical impulses and cellular-automata", *International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2011)*, 2011.

[MÉR 11h] MÉRY D., SINGH N.K., "A generic framework: from modeling to code", *Fourth IEEE International workshop UML and Formal Methods (UML&FM'2011), (to be appeared in special issue of ISSE NASA Journal, Innovations in Systems and Software Engineering)*, 2011.

[MÉR 11g] MÉRY D., SINGH N.K., "Medical protocol diagnosis using formal methods", *International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2011)*, 2011.

[MÉR 13] MÉRY D., MONAHAN R., "Transforming Event B models into verified C# implementations", LISITSA A., NEMYTYKH A.P. (eds.), *VPT@CAV*, of *EPiC Series*, EasyChair, vol. 16, pp. 57–73, 2013.

[MOR 90] MORGAN C., *Programming from Specifications*, Prentice Hall International Series in Computer Science, Prentice Hall, 1990.

[MOS] MOSBAH M., "VISIDIA". Available at http://visidia.labri.fr.

[REH 09] REHM J., Gestion du temps par le raffinement, PhD Thesis, Henri Poincaré University, Nancy I, December 2009.

[SIN 11] SINGH N.K., Fiabilité et sûreté des systèmes informatiques critiques, University Thesis, UHP, October 2011.

[SIN 13] SINGH N.K., *Using Event-B for Critical Device Software Systems*, Springer, 2013.

[TUR 49] TURING A., "On checking a large routine", *Conference on High-Speed Automatic Calculating Machines*, University Mathematical Laboratory, Cambridge, 1949.

AQ1: Should this be a different word? "Refinement is also called refinement" doesn't make sense.

AQ2: Please check if the edit in the sentence “The first form is a normal form in the sense” conveys the intended meaning of the sentence.

AQ3: Please check the sentence `` In our presentation, we emphasize” and correct if necessary.

AQ4: Please check the sentence for the edits made in it `` The general form of proof obligation” and correct if necessary.

AQ5: Please check the sentence ``We have made a quite as classic” and correct if necessary.

AQ: Please provide the figures 10.2 and 103.

AQ7: Please check if the edit in the sentence “We will mention some applications developed” conveys the intended meaning of the sentence.

AQ8: Please check the sentence ``These studies have” for clarity.

AQ9: Please check the “[]” given in the last sentence of the chapter and provided the reference citation if it is meant for reference citation.

AQ10: Please provide the publication year of the references [CAN], [Hei] and [MOS], for correctness.

AQ11: Please check the translation of the university in the reference [COU 78] for correctness.

AQ12: Please provide the full form of the conference, etc. in references [FLO 67] and [HOA 09b].

AQ13: Please provide updation for ``in press” reference [MER 11d], [MER 11f] and [MER 11a]

AQ14: Provide the volume number and page range for [MER 10b].

AQ15: Please provide the complete name of the university for clarity.