Formal specification by using Event-B

Reference

Ralph-Johan Back, Joakim von Wright, « Refinement Calculus - A Systematic Introduction », Springer, 1998
Jean-François Monin, « *Introduction aux Méthodes Formelles* », Hermès Science Publications, 2000
Jean-Raymond Abrial, « *Modeling in Event-B - System and Software Engineering* », Cambridge Press, 2010

1 History of the formal specification

As early as the 1940s, Turing's work shows that reasoning on sequential programs becomes easier if **property-related annotations** on instruction states are introduced at particular places in these programs. On the one hand, mathematicians in logic, like Hoare, propose a methodological framework to prove the coherence between the programs and these properties, and thus define the formal specification. On the other hand, the industrialists seized this technique for reliability, time saving and design cost reduction objectives: Airbus indeed developed in the 80s a computer-aided specification for first electrical flight control computers of his flagship, A320.

The formal specification is the formal language expression of a set of properties that a system must satisfy. In other words, the formal specification is a part of the standard specification that relies on a **mathematical notation** to express certain requirements as properties. Steady and proven mathematical theories, such as set theory and predicate logic, are used to model the system and describe its expected behavior. In addition, advances in computer science make it possible to provide design teams with consistent computerized tools and techniques. This is called a **formal method** when there is a framework of tools and rules for **modeling** and **reasoning** that applies to both the specification and the design of systems.

The interest of mathematics lies in the quality of its expressiveness and the processing they can allow. Contrary to an informal specification in natural language, the formal specification has the following assets of mathematics:

- **accuracy**: mathematical expression is accurate; the context is not essential for its understanding;
- **clarity**: expression allows little misinterpretation;
- **conciseness**: there are fewer mathematical symbols;
- **independence** against natural language or cultural considerations;
- **abstraction** that allows to focus on essential aspects first;
- **proof**: possibility to draw conclusions using demonstrations, the proof makes it possible to validate or not a property in a reliable way.

The drawbacks of such a specification are in its strengths:

- one first hurdle to overcome is to master a notation that uses mathematical skills. Appropriate training of future engineers is therefore required;
- modeling can reveal difficulties due to the technique used. Thus, it is likely that it increases the lead time of a project. A model is not a solution to the problem raised. The modeling, and therefore the formal specification, makes it possible to identify the properties of the system and to prove that these properties are also present in the final system to be constructed, i.e. the concept of "**correctness by construction**".

2 Event-B method

2.1 Introduction

In the mid-80s, Jean-Raymond Abrial developed a method for the production of reliable software that he called B method. B method relies on a formal language better tooled than Z language developed by Abrial a few years in the past. Method B then becomes Event-B method to take into account hardware aspects in addition to the software in a complex computer-oriented system. As far as critical systems are concerned, Event-B method relates to the specification, design, and coding of the associated software, and implements a proof framework. Thus, it makes it possible to carry out a development of correct systems by construction. To master the complexity, the specifications are represented using abstract models based on set theory and the logic of first-order predicates. Successive refinements make it possible to detail these models, which results in more concrete modeling.

Event-B method is supported by many tools:

- **B-tool** developed in England by B-core company
- **Atelier B** developed in France by ClearSy company
- **Rodin**, which is an open source platform (*http://www.event-b.org/*)

2.2 Notation

The formal Event-B language is based on notions of logic and set theory. Sets are mainly used to constrain data by associating a type. There are 3 classes of types:

- the "**abstract**" type represents a finite set of objects or individuals with common characteristics that are not explicitly defined. For example: PERSON, BUILDING;
- the "**enumerated**" type corresponds to a set of distinct elements well identified. For example: WEEK = {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};
- the "**standard**" type concerns classical sets such as natural numbers $\mathbb{N}$ or relative $\mathbb{Z}$ or intervals.

The constant features of some elements can be formally described using binary relations and functions. For example, let us describe formally the relationships that exist between people of the same family, i.e. parents, children, brother, wife, and so on. We assume a generic abstract set, called PERSON, which includes all the people involved.

1. Each person is either a man or a woman;
2. no one is both man and woman;
3. only women have husbands who are men;
4. each woman has at most one husband who is not the husband of any other woman;
5. a mother is a married woman.

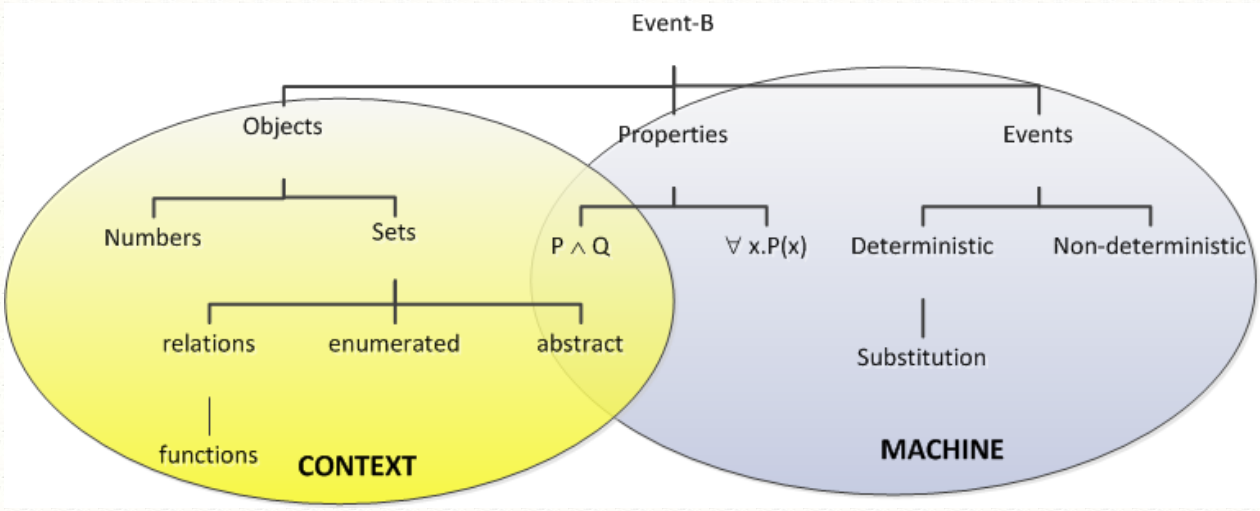
A modeling of the entities *men*, *women*, *husband* (or *has_as_a_husband*), *mother* (*has_as_a_mother*), *father* (or *has_as_a_father*) is as follows:

1. $men \subseteq PERSON$
2. $women = PERSON \setminus men$
3. $husband \in women \rightarrowtail men$ (partial injection)
4. $mother \in PERSON \nrightarrow dom(husband)$ (partial function)

Here is an example of Simpson's family:

$PERSON = \{Homer, Marge, Bart, Lisa, Maggie\}$
$man = \{Homer, Bart\}$
$woman = \{Marge, Lisa, Maggie\}$
$husband = \{Marge \mapsto Homer\}$
$mother = \{Bart \mapsto Marge, Lisa \mapsto Marge, Maggie \mapsto Marge\}$

Event-B language enables to handle the feature categories (see table hereunder).



2.3 Description of Event-B model

Modeling relies on **discrete event dynamic system** (DEDS) models such as a state machine, Petri net. In the framework on a behavioral analysis, the formal description of a DEDS highlights two components related to its state:

- a static part of the state indicating the constant features of the system and the intrinsic or specified properties associated with these features;

- a dynamic part, which represents the expected behavior, and includes the state variables that will be modified during the operation of the system. It also specifies the constraints or dynamic properties satisfied by these variables.

In Event-B, the static part of the model is called ***Context***, whereas the dynamic part is called ***Machine***. A machine consists of events (called **Event**) that specify operations that affect the value of the state variable. Likewise, it stipulates conditions which are constantly satisfied during the operation of the system. These conditions are called **Invariants**.

2.3.1 MACHINE component

MACHINE	*name*
SEES	*context*
VARIABLES	*variables*
INVARIANTS	*invariants*
THEOREMS	*theorems*
VARIANTS	*variants*
EVENTS	*events*

Let consider a control system composed of sensors and actuators. At a given time, the state of this system is defined by the value of a sensor or an actuator. Let us call the state variable *value*. This state is a total function of observed quantity (set *E_OBSERVATION*) in relation with a real number (set *VALUE*).

```
MACHINE
  EQUIP_0
SEES
  UAV0_C0
VARIABLES
  value
INVARIANTS
  inv1   :   value ∈ E_OBSERVATION ⟶ VALUE
EVENTS
```

```
s_send   ≙
STATUS
  ordinary
ANY
  sen
  v
WHERE
  grd1   :   sen ∈ SENSOR
  grd2   :   v ∈ VALUE
THEN
  act1   :   value(eqt_obs(sen)) ≔ v
END
```

Parametric guard

Action

A particular event in Event-B is the initialization event, which assigns initial value to state variable.

$value := E\_OBSERVATION \times \{v_{init}\}$ (deterministic initialization event)
$value :\in E\_OBSERVATION \nrightarrow VALUE$ (non deterministic initialization event)

2.3.2 CONTEXT component

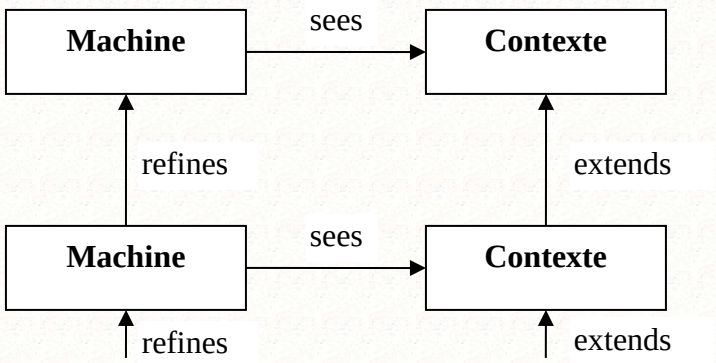
Context component enables to explain the constraints and the parameters of the machine component by defining sets and constants, and expressions of properties in terms of axioms.

CONTEXT	*name*
SETS	*sets*
CONSTANTS	*constants*
AXIOMS	*axioms*
THEOREMS	*theorems*

In our aforementioned example, the context is as follows.

```
CONTEXT
  UAV0_C0

SETS
  EQUIPMENT
  VALUE
  RESOURCE

CONSTANTS
  ACTUATOR
  SENSOR
  E_OBSERVATION
  eqt_obs
  vinit

AXIOMS
  axm1   :   SENSOR ⊆ EQUIPMENT
  axm2   :   ACTUATOR ⊆ EQUIPMENT
  axm3   :   SENSOR ∩ ACTUATOR = ∅
  axm4   :   EQUIPMENT = SENSOR ∪ ACTUATOR
  axm5   :   E_OBSERVATION ∈ ℙ1 (RESOURCE)
  axm6   :   eqt_obs ∈ EQUIPMENT ⟶ E_OBSERVATION
  axm7   :   vinit ∈ VALUE
  axm8   :   E_OBSERVATION ≠ ∅

END
```

The relationship linking a Machine to a Context is a relationship with a visual connotation, noted "***see***". Machines are meant to see Contexts. Between them, Contexts are *extended*, while the Machines are *refined*.



In the Function folder, select the top function and then double-click on the eFFBD tab at the bottom-right:

2.4 Proof obligation rules

2.4.1 Inference rules

A sequent is a mathematical concept, useful to express our proof obligation rules. By convention, the form of a sequent is:

$$\boldsymbol{H} \vdash \boldsymbol{G}$$

meaning the **goal/conclusion** G is provable under the set of **hypotheses/assumptions** H
The symbol ⊢ stands for "implies", "entails", "yields".

A few examples of well-known axioms are given hereinafter.

First Peano (P1) axiom is: $\vdash \ \mathbf{0} \in \mathbb{N}$

Second Peano (P2) axiom is: $\boldsymbol{n} \in \mathbb{N} \vdash \boldsymbol{n} + \mathbf{1} \in \mathbb{N}$

and a derived second Peano axiom (P2') is $\boldsymbol{n} \in \mathbb{N}, \quad 0 < \boldsymbol{n} \vdash \boldsymbol{n} - \mathbf{1} \in \mathbb{N}$

Third Peano (P3) axiom is: $\boldsymbol{n} \in \mathbb{N} \vdash \mathbf{0} \leq \boldsymbol{n}$

INC axiom is: $\boldsymbol{n} \in \mathbb{N}, \boldsymbol{m} \in \mathbb{N}, \quad \boldsymbol{n} < \boldsymbol{m} \vdash \boldsymbol{n} + \mathbf{1} \leq \boldsymbol{m}$

DEC axiom is: $\boldsymbol{n} \in \mathbb{N}, \boldsymbol{m} \in \mathbb{N}, \quad \boldsymbol{n} \leq \boldsymbol{m} \vdash \boldsymbol{n} - \mathbf{1} \leq \boldsymbol{m}$

Consequently, the inference rules are mathematical rules that enable to justify formally the transformation of sequent. For instance the below rule called monotonicity of hypotheses is stated as follows:

$$\frac{\boldsymbol{H1} \vdash \boldsymbol{G}}{\boldsymbol{H1}, \boldsymbol{H2} \vdash \boldsymbol{G}} \ \boldsymbol{MON}$$

where the upper sequent is called **antecedent** of the rule, whereas the lower is the **consequent**. In the framework of the transformation of sequent, as soon as the rule of inference is valid, it states that if the antecedent is proved then the consequent is proved consequently. Therefore, in order to prove the consequent, we only need to prove the antecedent.

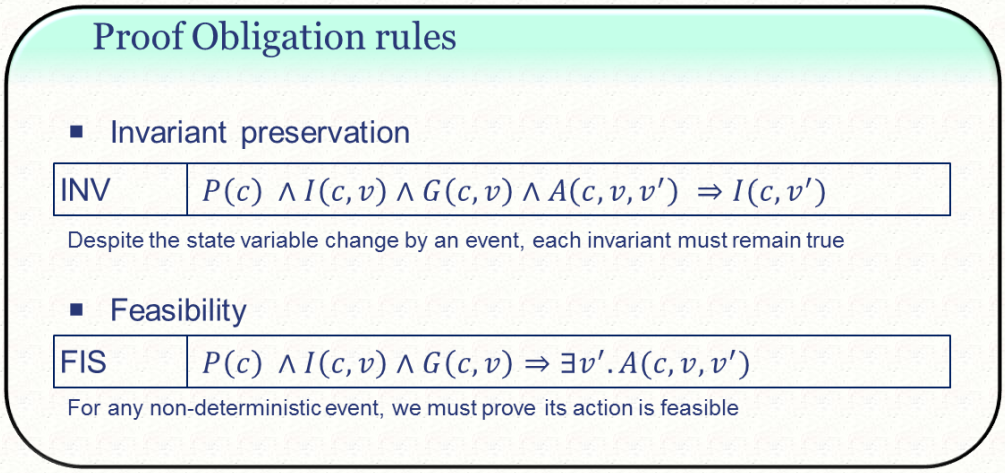
In the present case, it says that in order to have a proof of goal **G** under the two sets of assumptions **H1** and **H2**, it is sufficient to have a proof of **G** under **H1** only.

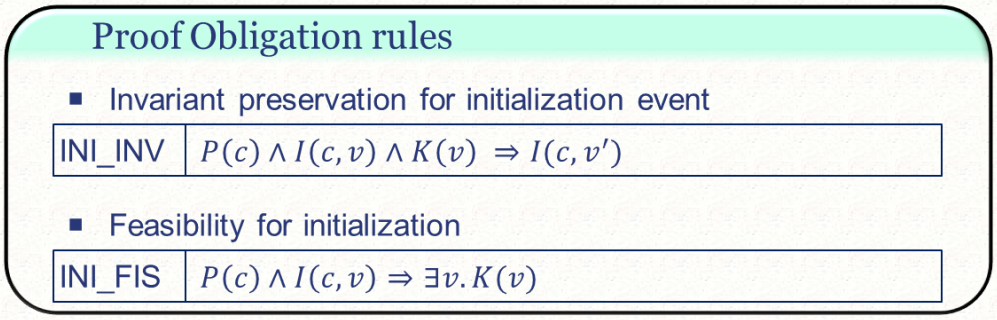
$\dfrac{H \vdash P}{H \vdash P \vee Q}$ *OR_R*	$\dfrac{H, P \vdash R \quad H, Q \vdash R}{H, P \vee Q \vdash R}$ *OR_L*
$\dfrac{}{P \vdash P}$ *HYP*	$\dfrac{}{\bot \vdash P}$ *CNTR*
$\dfrac{H(F),\ E = F \vdash P(F)}{H(E),\ E = F \vdash P(E)}$ *EQ_LR* where P(E) is a predicate depending on an expression E (idem for H(E) and H(F))	$\dfrac{}{\vdash E = E}$ *EQL*

$\dfrac{H, \neg P \vdash Q}{H \vdash P \vee Q}$ *NEG*	
$\dfrac{H, P, Q \vdash R}{H, P \wedge Q \vdash R}$ *AND_L*	$\dfrac{H \vdash P \quad H \vdash Q}{H \vdash P \wedge Q}$ *AND_R*
$\dfrac{}{H, P, \neg P \vdash Q}$ *NOT_L*	$\dfrac{H, P \vdash Q \quad H, P \vdash \neg Q}{H \vdash \neg P}$ *NOT_R*
$\dfrac{H, P, Q \vdash R}{H, P, P \Rightarrow Q \vdash R}$ *IMP_L*	$\dfrac{H, P \vdash Q}{H \vdash P \Rightarrow Q}$ *IMP_R*

2.4.2 Invariant preservation PO rule

A collection of proof obligation rules is associated with the Event-B formalism through the system modeling. These rules have to be discharged in order to guarantee the correctness by construction and the relevance of the model. Among these rules, the invariant preservation rule states that any event must hold the invariants after state variables modification, as well as the feasibility rule applied to the events with non-deterministic actions.





2.4.3 Dead-lock freedom rule

Another non-mandatory rule is the dead-lock freedom property of the system that stems from the full availability of certain systems. This rule is not valid for all models, but is often a requirement of embedded systems that should never stop once started. A risk of dead-lock occurs when the guards of all events are continually false. The dead-lock freedom rule (DLF set for Deadlock Freedom) is then set up as follows:

Axioms $\wedge$ **Invariants** $\Rightarrow$ $\mathbf{G_1} \vee \ldots \vee \mathbf{G_m}$

which means that under the conditions where the axioms and invariants of the model are fulfilled, one of the guards is obligatorily true.

2.5 Refinement

2.5.1 Principle

In general, refinement is an incremental technique to transform an abstract model into a more concrete model with more details. These added details may relate to variables, invariants or events.

One machine refines another machine, if it contains at least one refined event. An event *re* (resp., machine *M_1*) is called a **refinement** of *ae* (respectively *M_0*), and *ae* (respectively *M_0*) is called an **abstraction** of *re* (resp. *M_1*). During a refinement, a machine *M_0* is replaced by

another machine *M_1* having events of the same name and with the same signature (number of parameters, and domain of values of the input and output parameters are identical) as the events of *M_1*. An event *ae* (resp. machine *M_0*) is refined by another event *re* (resp. *M_1*), if the use of *re* (resp. *M_1*) in place of *ae* (resp. *M_0*) is imperceptible for the user.

Let be two events *ae* and *re*, a formal definition of refinement ***ae ⊆ re*** (meaning that "*ae* is refined by *re*") is as follows:

$$\forall s \,.\, (s \subseteq V \Rightarrow str(ae)(s) \subseteq str(re)(s))$$

Thus, it is stated that *ae* is refined by *re* when for every subset *s* of the space of the state variables *V*, the valuation of the set transformation *str (ae)* applied to *s* is included in the valuation of *str (re)* for the same subset *s*.This gives rise to the following property concerning the activation conditions (denoted "*grd*") of the events considered and the set of binary relations between the state variables before the event is triggered and after the event has been triggered (noted "*rel*"):

$$ae \subseteq re \Leftrightarrow grd(ae) \subseteq grd(re) \wedge rel(re) \subseteq rel(ae)$$

It is deduced that refinement aims at **reducing the non-determinism** of an event or at **reinforcing the guard** of an event.

Example of a refinement allowing a reduction of the non-determinism of an event:

$E = [0..9]$ ae $\hat{=}$ $x :\in E$

$x \in E$ re $\hat{=}$ $x := 5$

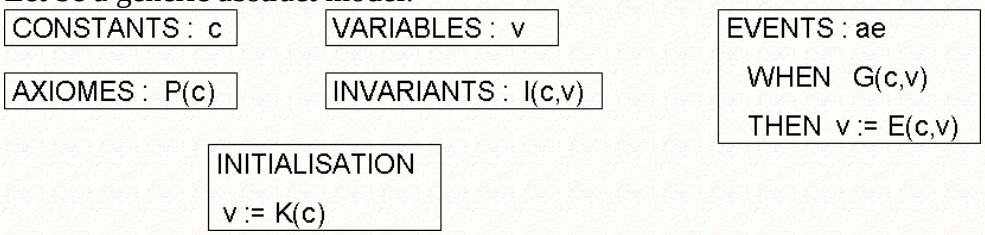
Example of a refinement reinforcing the guard of an event:

$E = [0..9]$ ae $\hat{=}$ $x=0 \mid x := 5$

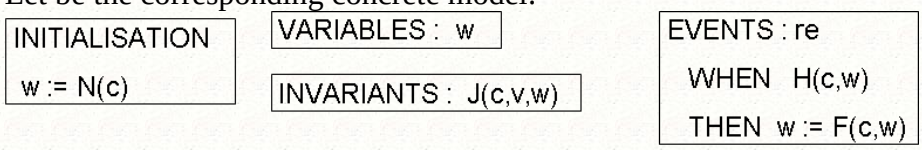
$x,y \in E$ re $\hat{=}$ $x=0 \wedge y=3 \mid x := 5$

2.5.2 Properties

Let be a generic abstract model:



Let be the corresponding concrete model:



The property of reinforcing the guard is expressed as follows:

P(c) I(c,v) J(c,v,w) H(c,w) ⊢ G(c,v)	GRD_REF

An additional property allows to correctly define a refined event i.e., **correct refined substitution** property.

P(c) I(c,v) J(c,v,w) H(c,w) ⊢ $J_i(c,E(c,v), F(c,w))$	INV_REF

Idem for initialization:

P(c) ⊢ $J_i(c,K(c), N(c))$	INI_INV_REF

The relative deadlock freedom property for refinement is:

P(c) I(c,v) J(c,v,w) $G_1(c,v) \vee \ldots \vee G_m(c,v)$ ⊢ $H_1(c,w) \vee \ldots \vee H_n(c,w)$	DLF_REF

We define a new event "***skip***" which corresponds to empty action:

- No guard;
- v' := v

In a refined model, adding a new event refines the "*skip*" event in the abstract model. For this new event, the invariant preservation rule is still valid to justify this event is correct.
The rule of relative deadlock freedom involves that if the abstract model is not blocked then the deadlock freedom of the refined model implies that the new events cannot be activated indefinitely (which corresponds to infinite "*skip*" in the abstract model; inconsistent with the deadlock freedom of the abstract model). To avoid the divergence of the new events, we set a ***variant*** for all the new events and we make sure of the **decrease** of this variant for each activation of a new event.

P(c) I(c,v) J(c,v,w) H(c,w) ⊢ V(c,w) ∈ N	VAR_REF1

P(c) I(c,v) J(c,v,w) H(c,w) ⊢ V(c,F(c,w)) < V(c,w)	VAR_REF2