Formal development of complex systems
Refinement and Proof with Event-B

Yamine AIT-AMEUR, Neeraj Kumar SINGH

IRIT/INPT-ENSEEIHT
{yamine, nsingh}@enseeiht.fr

November 2019

1. Introduction
2. Propositional logic
3. Predicate logic
4. Set theory
5. Modelling of Systems
6. The Event-B method
   - Refinement of Event-B machines
7. Proof with Event-B
   - Proof activity
   - Proofs with Event-B and the Rodin platform
8. The Rodin Platform
9. Animation of Event-B models
10. Conclusion

Plan

1. Introduction

Introduction

Many complex systems are present in engineering, finance, marketing, etc.

- Complex systems with integration of
  - software
  - hardware
  - plants
  - communications
  - humains
- Need to handle the environment in which a system evolves
- Input/output, close/open loop

Introduction

Quelques problèmes

- Expression of needs, requirement analysis
  - fonctionnal,
  - non fonctionnal
- Specification of systems
- Design of systems : composition, decomposition
- System Validation / Verification, in particular for critical systems
- SystemCertification according to certification authorities or standard requirements

Which techniques ? Which methods ?

Introduction

The development of complex systems requires the definition of modelling languages offering means for

- expressing and defining abstractions of these systems in order to
  - design and build these systems,
  - reason on these systems to check their properties,
  - predict their behaviour, if possible in any situation/context
- These languages shall
  - be rigorously/formally defined
    - non ambiguous
    - expressive
  - support the capability to express different system facets, views, etc.
    - functional
    - safety and reliability
    - real time
    - architecture
    - simulation
    - ...

Introduction

- Science of language (Jean-Piaget encyclopaedia *"Logique et Connaissance Scientifique"* or *"Scientific logics and knowledge")*

> If we refer to whom is talking, or more generally to users of the language, this investigation relates to the **pragmatics**.
> If we make abstraction of language users and analyse only language expressions and their meanings, then, we are dealing with **semantics**.
> Finally, si if we make abstraction of the meanings to analyse only the relations between expressions, then, we are dealing with **syntax**.
> These three elements are constituents of **science of language** or **semiotics**.

Introduction

Model, associated to semantics

- Interpretation of the understanding of a situation,
- Description of entities and their relations
- Definition borrowed from M. Minsky *"Société de l'esprit"*

> For an observer **A**, **M** is a **model of object O**, if **M** helps **A** to answer the questions he/she has on **O**

- The definition of system models at different abstraction levels allows designers to reason on the system to design

Models shall

- be rigorously defined
- offer reasoning mechanisms
  - interpreters,
  - proof systems,
  - simulators,
  - analysers,
  - type checkers,
  - etc.

Introduction

Objectives of the lecture

- Present a formal system development method based on
  - first order logic,
  - set theory,
  - state-transitions systems
  - refinement/composition/decomposition

Plusieurs liens avec les cours déjà effectués

- Modélisation.
- GLS
- VAS
- Spécification formelle
- ...

Plan

2 Propositional logic

Objectifs

- Recalls of basic logics concepts
- Handling proofs and proof system

Propositional logics

Propositional logics operators.

$\bot$	Constant *False*
$\top$	Constant *True*
$\neg A$	Negation
$A \wedge B$	Conjunction
$A \vee B$	Disjunction
$A \Rightarrow B$	Implication
$A \Longleftrightarrow B$	Equivalence

Propositional logics

	$A \to B = \neg A \vee B$ $A \leftrightarrow B = (A \to B) \wedge (B \to A)$
Idempotent	$A \wedge A = A$ $A \vee A = A$
	$A \wedge \neg A = \bot$ $A \vee \neg A = \top$ $A \wedge \bot = \bot$ $A \wedge \top = A$ $A \vee \bot = A$ $A \vee \top = \top$ $\neg\neg A = A$
Commutativity	$A \wedge B = B \wedge A$ $A \vee B = B \vee A$
Associativity	$(A \wedge B) \wedge C = A \wedge (B \wedge C)$ $(A \vee B) \vee C = A \vee (B \vee C)$

Propositional logics

Distributivity	$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$ $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
De Morgan	$\neg(A \wedge B) = \neg A \vee \neg B$ $\neg(A \vee B) = \neg A \wedge \neg B$ $A \vee (\neg A \wedge B) = A \vee B$ $A \wedge (\neg A \vee B) = A \wedge B$ $A \vee (A \wedge B) = A$ $A \wedge (A \vee B) = A$

Proofs and proof system

Sequent and inference rule

- Sequent

  list_of_hypotheses ⊢ *conclusion*

  The *list_of_hypotheses* may be empty (e.g. case of a theorem)

- Inference rule. Generic form $\frac{A}{C}r$ or $\frac{A_1, \ldots A_n}{C}r$
  - A is a set of sequents (may be empty) called **Antecedent**
  - C is a **Consequent** sequent

- Inference rule

$$\frac{list\_of\_sequents}{sequent}$$

  The *list_of_sequents* may be empty (e.g. case of an axiom)

Proofs and proof system

- Definition of axioms.
- Useful for definitions.

$$\frac{}{A \vdash A}$$ (Axiom for hypothesis)

$$\frac{}{\Gamma; A \vdash A}$$ (Axiom for extended hypothesis)

Proofs and proof system

- Definition of inference rules.
- Useful for inferring (deduction) of new sequents
- Implication Elimination (E) and Introduction (I)

$$\frac{\Gamma \vdash A \quad \Gamma \vdash A \to B}{\Gamma \vdash B} \quad (E_{\to})$$

$$\frac{\Gamma; A \vdash B}{\Gamma \vdash A \to B} \quad (I_{\to})$$

Proofs and proof system

- And Elimination (E) and Introduction (I)

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \quad (E^1_{\wedge})$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \quad (E^2_{\wedge})$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \quad (I_{\wedge})$$

Proofs and proof system

- Or Elimination (E) and Introduction (I)

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \quad (I^2_{\vee})$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad (I^1_{\vee})$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma; A \vdash C \quad \Gamma; B \vdash C}{\Gamma \vdash C} \quad (E_{\vee})$$

- Elimination is useful for case base reasoning

Proofs and proof system

- Handling negation

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A} \quad (E_{\bot})$$

$$\frac{}{\Gamma \vdash A \vee \neg A} \quad \text{(Tiers Exclu)}$$

$$\frac{\Gamma; (A \to \bot) \vdash \bot}{\Gamma \vdash A} \quad \text{(Pierce)}$$

Be careful, non constructive features.

Proofs and proof system. Tactics

Tactics

- Tactics are compositions of inference rules
- Useful to handle big proof steps
- "Proof programming"
  - unfolding/folding
  - choice
  - iteration

Proof systems implement

- inference rules and
- tactics

definitions

Plan

3 Predicate logic

Predicate logic, first order logic (FOL)

$\exists x.P$	Existential Quantification
$\forall x.P$	Universal Quantification

Introduction of predicates, with variables, relations and functions.

- $P(x_1, \ldots x_n)$
- $P(f(x_1), \ldots, g(x_{n-2}, x_{n-1}), x_n)$

where

- P is a predicate symbol
- f and g are function symbols

Sequents in predicate logic

$$\frac{\Gamma \vdash \forall x.A}{\Gamma \vdash [t \mid x]A} \quad (E_\forall \text{ form } 1)$$

$$\frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A} \quad (E_\forall \text{ form } 2)$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A} \quad (\ I_\forall) \quad (\text{for } x \notin FV(\Gamma))$$

The $[t/x]\Psi$ notation represents the substitution, in Ψ, of the occurrences of x by t

Sequents in predicate logic

$$\frac{\Gamma \vdash \exists x.A}{\Gamma \vdash [t \mid x]A} \quad (\mathrm{E}_\exists \text{ form 1}) \quad (\text{for } t = f(FV(\Gamma) \cup FV(A)))$$

$$\frac{\Gamma \vdash [t \mid x]A}{\Gamma \vdash \exists x.A} \quad (\mathrm{I}_\exists)$$

$$\frac{\Gamma \vdash \exists x.A \quad \Gamma; A \vdash B}{\Gamma \vdash B} \quad (\mathrm{E}_\exists \text{ form 2}) \quad (\text{for } x \notin FV(\Gamma) \cup FV(B))$$

Sequents in predicate logic

Refinement of the language. Introduction of **Equality**

- Extension of the definitions of predicates avec by the introduction of the Equality predicate

Predicate ::=	*Expression* = *Expression*
Expression ::=	...
Variable ::=	...

Introduction of **terms** with

- variables x, y, z $\cdots$
- constants a, b, , c, $\cdots$
- functions f, g, , l $\cdots$

Examples

- Terms `a, x, a+b, f(x,y,a), h(g(x),a),y)`
- Predicates `P(x), x=a P(f(x,y,a),z), l(x)=a`

Plan

4. Set theory

Objectives

Recall of basic notions in

- Set theory
- Relations
- Functions

Sets

- Introduction of the Belongs To predicate $E \in S$ where
  - E is an expression
  - S is a set
- Introduction of set constructors.
- Axiomatisation based definitions $\triangleq$

Remark.

- This lecture does not represent the whole axioms (definitions)
- Part of these axioms are given.
- They are relevant for the understanding of next steps.

Sets

Three basic constructors are considered

Let S and T be two sets, x a variable and P a predicate. The follwoing set constructions are defined.

- Cartesian Product $S \times T$
- The set of Subsets or powerset $\mathbb{P}(S)$
- Definition of sets by comprehension $\{x \mid s \in S \wedge P\}$

These constructs are used to define other set operators.

Sets. Basic operators

- Inclusion $S \subseteq T$
- Associated axioms

$S \subseteq T$	$\triangleq$	$S \in \mathbb{P}(T)$
$S = T$	$\triangleq$	$S \subseteq T \wedge T \subseteq S$

- Define an order relation on sets.

Sets. Basic operators

Union	$\cup$
Intersection	$\cap$
Difference (Subtraction)	$-$
Extension	$\{\ldots\ldots\}$
Empty Set	$\varnothing$

- A set of axioms is associated to each of these operators.

Sets. Generalised operators

Generalised Union	$\mathtt{union}(S)$
Quantified Union	$\bigcup x.(x \in S \wedge P)$
Generalised Intersection	$\mathtt{inter}(S)$
Quantified Intersection	$\bigcap x.(x \in S \wedge P)$

- A set of axioms is associated to each of these operators.

Binary Relations

Binary Relation	$S \leftrightarrow T$
Domain	$\mathtt{dom(r)}$
Co-domain (Range)	$\mathtt{ran(r)}$
Inverse	r^{-1}

- Axiomatisation. A set of axioms is associated to binary relations.

$r \in S \leftrightarrow T$	$\triangleq$	$r \subseteq S \times T$
$E \in dom(r)$	$\triangleq$	$\exists y.(E \mapsto y \in r)$
$F \in ran(r)$	$\triangleq$	$\exists x.(x \mapsto F \in r)$
$E \mapsto F \in r^{-1}$	$\triangleq$	$F \mapsto E \in r$

Binary Relations

Recall of basic notions

- Partial / Total
- Surjective / Injective / Bijective

Specific definitions of binary relations

Partial Surjective binary relation	$S \leftarrow\!\!\!\twoheadrightarrow T$
Total binary relation	$S \twoheadleftarrow\!\!\!\rightarrow T$
Total Surjective binary relation	$S \twoheadleftarrow\!\!\!\twoheadrightarrow T$

Axiomatisation

$S \leftarrow\!\!\!\twoheadrightarrow T$	If $r \in S \leftarrow\!\!\!\twoheadrightarrow T$ then $ran(r) = T$
$S \twoheadleftarrow\!\!\!\rightarrow T$	If $r \in S \twoheadleftarrow\!\!\!\rightarrow T$ then $dom(r) = S$
$S \twoheadleftarrow\!\!\!\twoheadrightarrow T$	If $r \in S \leftrightarrow T$ then $dom(r) = S \ \wedge \ ran(r) = T$

Binary Relations

Manipulation of binary relations

- Restriction and Subtraction

Domain Restriction	$S \lhd T$
Range Restriction	$S \rhd T$
Domain Subtraction	$S \mathbin{\lhd\!\!\!-} T$
Range Subtraction	$S \mathbin{\rhd\!\!\!-} T$

Axiomatisation

$S \lhd T$	$S \lhd T = \{x \mapsto y \mid x \mapsto y \in T \ \wedge \ x \in S\}$
$S \rhd T$	$S \rhd T = \{x \mapsto y \mid x \mapsto y \in T \ \wedge \ y \in T\}$
$S \mathbin{\lhd\!\!\!-} T$	$S \mathbin{\lhd\!\!\!-} T = \{x \mapsto y \mid x \mapsto y \in S \wedge x \notin S\}$
$S \mathbin{\rhd\!\!\!-} T$	$S \mathbin{\rhd\!\!\!-} T = \{x \mapsto y \mid x \mapsto y \in S \wedge x \notin T\}$

Binary Relations

Manipulation of binary relations

- Image, composition, overriding and identity

Image	$r[S]$
Composition	$p; q$
Overriding	$p \Leftarrow q$
Identity	$id(S)$

Manipulation of binary relations

- Image, composition, overriding and identity

$r[S]$	$r[S] = \{y \mid \exists .x \in S \wedge x \mapsto y\}$
$p; q$	$\forall p, q . p \in P \leftrightarrow Q \wedge q \in T \leftrightarrow U \Rightarrow$ $p; q = \{x \mapsto y \mid (\exists z . x \mapsto z \in p \wedge z \mapsto y \in q)\}$
$p \Leftarrow q$	$p \Leftarrow q = q \cup (dom(q) \lhd r)$
$id(S)$	$id(S) = \{x \mapsto x \mid x \in S\}$

Binary Relations

Manipulation of binary relations

- Products and projection

Direct Product	$p \bigotimes q$
First (Left) projection	$prj1$
Second (Right) projection	$prj2$
Parallel Product	$p \parallel q$

Axiomatisation

$p \bigotimes q$	$p \bigotimes q = \{x \mapsto (y \mapsto z) \mid x \mapsto y \in p \wedge x \mapsto z \in p\}$
$prj1$	$prj1(r) = \{x \mid x \mapsto y \in r\}$
$prj2$	$prj2(r) = \{y \mid x \mapsto y \in r\}$
$p \parallel q$	$p \parallel q = \{(x \mapsto y) \mapsto (m \mapsto n) \mid x \mapsto m \in p \wedge y \mapsto n \in q\}$

Binary Relations

Manipulation of binary relations

All these operators are associated to

- axiomatic definitions (axioms)
- properties
- definitions in predicate logic

Functions and Functions Operators

Functions

Partial Function	$S \nrightarrow T$
Total Function	$S \rightarrow T$

Axiomatisation. A Function is a Relation

$f \in S \nrightarrow T$	$\triangleq$	$f \in S \leftrightarrow T \wedge f^{-1}; f = id(ran(f))$
$f \in S \rightarrow T$	$\triangleq$	$f \in S \nrightarrow T \wedge S = dom(f)$

Functions and Functions Operators

Other Function definitions

Partial Injection	$S ⤔ T$
Total Injection	$S \rightarrowtail T$
Partial Surjection	$S ⤀ T$
Total Surjection	$S \twoheadrightarrow T$
Bijection	$S ⤖ T$

Axiomatisation

$S ⤔ T$	$S ⤔ T = \{f \cdot f \in S ⇸ T \wedge f^{-1} \in T ⇸ S\}$
$S \rightarrowtail T$	$S \rightarrowtail T = S ⤔ T \cap S \rightarrow T$
$S ⤀ T$	$S ⤀ T = \{f \cdot f \in S ⇸ T \wedge ran(f) = T\}$
$S \twoheadrightarrow T$	$S \twoheadrightarrow T = S ⤀ T \cap S \rightarrow T$
$S ⤖ T$	$S ⤖ T = S \rightarrowtail T \cap S \twoheadrightarrow T$

Definition and Function Application

Lambda expression

- Definition of a Function

$$\lambda x.(S \mid E) \quad \text{or} \quad \lambda x.(x \in S \mid E(x))$$

- Application of a Function

$$a \mapsto b \in \lambda x.(x \in S \mid E(x)) \quad \triangleq \quad E(a) = b$$

with $a \in S$

Well definedness

- Let f be a Partial Function, then

$$b = f(a) \quad \triangleq \quad a \mapsto b \in f$$

This property defines a Well- Definedness condition for a Function Definition

$$a \in dom(f)$$

Logic notations

Rewriting logic expressions

- Let us consider the predicate

$$f^{-1}; f \subseteq id$$

- It can be successfully translated to

$$\forall x, y, z \cdot \; x \mapsto y \in f \wedge x \mapsto z \in f \Longrightarrow y = z$$

Applying rewriting

$f^{-1}; f \subseteq id$
$\forall y, z \cdot \; y \mapsto z \in (f^{-1}; f) \Longrightarrow y \mapsto z \in id$
$\forall y, z \cdot \; y \mapsto z \in (f^{-1}; f) \Longrightarrow y = z$
$\forall y, z \cdot (\exists x \cdot \; y \mapsto x \in f^{-1} \wedge x \mapsto z \in f) \Longrightarrow y = z$
$\forall y, z \cdot (\exists x \cdot \; x \mapsto y \in f \wedge x \mapsto z \in f) \Longrightarrow y = z$
$\forall x, y, z \cdot \; x \mapsto y \in f \wedge x \mapsto z \in f \Longrightarrow y = z$

Recap of the whole introduced notions.

Recap document from Ken. Robinson (Uni. South Wales - Sydney - Australia

- See the document providing a recap of the set of constructions introduced previously.
- The correspondences between mathematical notations and ASCII code available in this document are useful for the users of the Rodin Platform.

This document is ditributed to Students

Plan

5. Modelling of Systems

General notions on Systems

- A system is observed through its evolution during life time
- Observation of the system elements/components changing over time
- A system is characterised by state
- A state is made of
  - /contextual / fixed / non-modifiable information defined is a dans la theory containing all the required definitions and resources allowing a system designer to describe a state
  - modifiable / flexible information that record the changes of the system state during time

  associated to the system to design, to analyse, to simulate, etc.

System Constants and variables

- **Constants** define contextual / fixed / non-modifiable information
- **Variables** define modifiable/ flexible information

When the systems are described, using the mathematical constructs presented in previous chapters **Constants** and **Variables** are defined

Remark. Note that other system modelling languages are available : type based, synchronous/asynchronous, simulation, semi-formal modelling languages, programming languages, etc.

General notions on Systems

States as a set of variables

- A state S is defined as a set of state variables $\{x, \cdots\}$
- State variables are valued. They are associated to variable values.

States evolution

- A state S may evolve after the occurrence of an event
- Notation $\mathbf{x} \xrightarrow{\mathbf{ev}} \mathbf{x}'$ where
  - ev is an event
  - x et x' represent respectively a state variable x before and after the occurrence of event ev.

General notions on Systems

Before-After predicates as a relation on states

- Event ev defines a relation on states.
  - $\mathbf{BAA(x, x')}$ is a Before-After Predicate characterising the event ev.

    Example.

    If $\mathbf{ev}$ is $\mathbf{x := x + 1}$ then $\mathbf{BAA(x, x')}$ is $\mathbf{x' = x + 1}$ or

$$BAA(x, x') \equiv x' = x + 1$$

  This definition is the assignment definition of Hoare Logic $\{[x/E]\Psi\}x := E\{\Psi\}$

First order logic for BAA

- Again, in the course, we rely on first order logics to describre Before-After Predicates
- The logic notions presented in the previous chapters will be used.

Remark. Note that other logics could have been used to describe such a relation : temporal logics, dynamic logics or type systems

General notions on Systems

- **BAA** describes a single state variable change only.
- We need to describe evolution of states along time

Traces as sequences of state evolutions

- A trace is a sequence of events occurrences

$$x_0 \xrightarrow{e_1} x_1 \xrightarrow{e_2} x_2 \xrightarrow{e_3} x_3 \xrightarrow{e_3} x_4 \xrightarrow{e_4} x_5 \xrightarrow{e_5} x_6 \xrightarrow{e_6} x_7 \ldots x_n \xrightarrow{e_n} x_{n+1} \ldots$$

- A trace with events which do not modify state variables can be described as well (presence of τ- transitions)

$$x_0 \xrightarrow{e_1} x_1 \xrightarrow{e_2} x_2 \xrightarrow{\tau} x_3 \xrightarrow{e_3} x_4 \xrightarrow{e_4} x_5 \xrightarrow{\tau} x_6 \xrightarrow{e_6} x_7 \ldots x_n \xrightarrow{e_n} x_{n+1} \ldots$$

  The τ events describe stuttering steps.
- The set of all traces allows a designer to observe the behaviour of a system

General notions on Systems

- A safety property S on a state x asserts that nothing bad happens in state x
  Notation **S(x)**
- An invariant is a safety property in all the states of all the observed traces
- The property S shall be observable in all the states of the system

$$S(x_0) \xrightarrow{e_1} S(x_1) \xrightarrow{e_2} S(x_2) \xrightarrow{\tau} S(x_3) \xrightarrow{e_3} S(x_4) \xrightarrow{e_4} S(x_5) \xrightarrow{\tau} S(x_6)$$

$$\xrightarrow{e_6} S(x_7) \ldots S(x_n) \xrightarrow{e_n} S(x_{n+1}) \ldots$$

- We write

$$\forall i \in \mathbb{N}. \quad S(x_i)$$

General notions on Systems

$$\forall i \in \mathbb{N}. \quad S(x_i)$$

- To prove this kind of properties we rely on induction on the length of the traces
  - The safety property holds at initial state, at initialisation
  - If this property holds in any state x_i (recurrence hypothesis) and it still holds after the occurrence of any triggered event, then this property holds for all states of traces of the system
- The proof of this property may be complex when it is realised on the whole set of events of a system
  - Use **refinement/abstraction** to reason on **"less complex"** or on **abstract** traces which hide some events (using τ events) of the **concrete** trace
  - **Refinement/abstraction** shall preserve the link between abstract and concrete traces
    Need to define a refinement/**simulation** relationship

General notions on Systems

- A liveness property **P leads_to Q** for a state x asserts that there exist a path in the traces that lead from a state x where P holds leading to a state x' where Q holds
  Notation $P \leadsto Q$
- A liveness property $P \leadsto Q$ asserts that a state x' where $Q(x')$ holds is reachable from a state x where $P(x)$ holds
- The property $P \leadsto Q$ is defined on a trace such that when $P(x_i)$ holds, there exists a future state $x_j, j > i$ where $Q(x_j)$ holds.

$$x_i \xrightarrow{e_1} x_{i+1} \xrightarrow{e_2} x_{i+2} \xrightarrow{\tau} x_{i+3} \xrightarrow{e_3} x_{i+4} \xrightarrow{e_4} x_{i+5} \xrightarrow{\tau} x_{i+6}$$

$$\xrightarrow{e_6} x_{i+7} \ldots x_j \xrightarrow{e_j} x_{j+1} \ldots$$

- We write

$$\text{For a state } x_i.\ i \in \mathbb{N}. \quad \exists j \in \mathbb{N}.\ j > i. \quad P(x_i) \Longrightarrow Q(x_j)$$

General notions on Systems

For a state $\mathbf{x_i},\ \mathbf{i} \in \mathbb{N}.\quad \exists \mathbf{j} \in \mathbb{N}.\ \mathbf{j} > \mathbf{i}.\quad \mathbf{P(x_i)} \Longrightarrow \mathbf{Q(x_j)}$

- To prove this kind of properties we rely on the definition of a variant i.e. a sequence of decreasing natural numbers
  - We know that each sequence of decreasing natural numbers is finite and converges to 0
  - Let x_k be a state in the trace. Initially $x_k = x_i$ (i.e. $k = i$.
  - Then, k increases to reach the suited state
  - When state x_j is reached, then $x_k = x_j$
  - Here, the sequence $j - k$ is a decreasing sequence
- This reasoning holds for any liveness property
- Again this proof is an induction. We shall show that
  - $j - k$ is a natural number
  - $j - k$ is a decreasing sequence

Requirements for a system modelling language

1. The values of state variables x belong to a set of licit *VALUES*
   $\Longrightarrow$ **Require to define this of these sets**
2. The events are relations on the set of states $\{ev_1, \ldots ev_n\}$
   $\Longrightarrow$ **Require to define events as transitions from a state to another one**
3. Invariants express, on traces, properties of the system
   $\Longrightarrow$ **Require of a language to define such properties**
4. Invariants are proven on traces
   $\Longrightarrow$ **Require a proof system (in particular induction)**
5. The definition of less abstract traces allows to express properties *"simpler"* to prove
   $\Longrightarrow$ **Require a refinement/abstraction operation which links abstract traces to concrete traces of two systems ie.e simulation relationship**

Plan

6. The Event-B method
   - Refinement of Event-B machines

The Event-B method (J.R. Abrial). Overview

Event-B is a formal method for system development

- It is based on
  - Set theory and First order Logic
- An Event-B model defines
  - a state of the system to model
  - an initialisation event and a set of events characterising state evolutions and changes
  - an invariant formalising the safety properties of a the system
  - Other properties of the system e.g. liveness, deadlock freeness, determinism, etc.
  - A refinement operation allowing to describe a concrete system which refines an abstract one.
    - It allows adding design decision and precise information on the behaviour of the system to design.
    - It preserves the properties of the abstract system in the concrete system thanks to a gluing invariant.

Event based system modelling with Event-B : Modelling Principles —— Event-B

- Event based systems modelling
- Concurrents systems
- Software, hardware (or both) systems
- Refinement and proof
- A system is seen as a **state-transitions system**
- Refinement offers a decomposition mechanism of state-transitions systems
- A **simulation relationship** links an abstract model and its refinement
- Simulation is a requirement for refinement correctness
- **"Correct by construction"** approach i.e. the system is explicitly correctly built

Model definition with Event-B

- Models are defined incrementally
  - A first/initial abstract **machine** is designed
  - A sequence of **refinement** of an existing machine is designed incrementally moving from an abstract level to a concrete level
- Models rely on **sets** and **constants** defined in a **context** Event-B component. The definitions are given by axioms and theorems may be introduced.
- Three relations define links between Event-B components
  - The **sees** relation expresses the use, by a machine, of constants and sets defined through axioms and fulfilling theorems of a context
  - The **extends** relation expresses the extension (enrichment of a context) by adding new sets, constants, axioms and theorems
  - The **refines** relation states that an Event-B model (machine) resp. event is refined by another Event-B model or event reps.

Machines and contexts

Machine Defines the system model as a state-transitions (state variables and events)

- `REFINES` an other machine
- `SEES` a context
- `VARIABLES` of the model
- `INVARIANTS` satisfied by the variables (state)
- `THEOREMS` satisfied by the variables (state) and deduced from invariants and seen contexts
- `VARIANT` decreasing
- `EVENTS` modifying state variables

Context It contains the definitions of the domain concepts needed to model the system. It also defines the proof context.

- `EXTENDS` an other context
- `SETS` declares news sets
- `CONSTANTS` défines a list of constants
- `AXIOMS` defining properties of sets and constants
- `THEOREMS` a list of theorems deduced from axioms

Event B contexts

CONTEXT
  ctx
EXTENDS
  actx
SETS
  s
CONSTANTS
  c
AXIOMS
  ax_i : ....
THEOREMS
  Tc_i : ....
END

Context

- `ac` extends the context `c` and adds new concepts
- `s` sets defined by comprehension or intention
- `k` definition of constants
- `ax1` axioms defining sets and constants
- `T(x)` set of theorems deduced from axioms and theorems.

Event B Machines

MACHINE m **REFINES** am **SEES** ctx **VARIABLES** x **INVARIANTS** I(x) **THEOREMS** T(x) **VARIANT** v **EVENTS** ev1 = ... ev2 = ... **END**

Machine

- `m` abstract machine corresponding to the system model
- `am` machine refined by `m`
- c visible contexts of machine `m`. They define the context $\Gamma(m)$
- `x` variables defining machine machine `m` state
- `I(x)` Invariants de la machine `m`
- `T(x)` Theorems deduced from the context and invariant
- `v` expression defining a decreasing variant (either a natural number or a set)
- `ev1, ...` list of machine events describing state changes with at least an `INITIALISATION` event

Modification of state variables

- State variables **modified** by **actions** or **substitutions** in events
- Different types of substitutions (variables modifications) are available
- Substitutions are characterised by **Before-After Predicates BAA**

Skip	Null/Empty Action
$x := E$	Becomes expression E (Simple Assignement)
$x :\in S$	Becomes element of S (Arbitrary choice in a set S)
$x :\mid P$	Becomes such that P (Arbitrary choice such that P
$f(x) := E$	Equivalent to $f := f \Leftarrow \{x \mapsto E\}$

- Substitution $x :\mid P$ encodes all the other substitutions. Its BAA is $P(x, x')$
- The previous substitutions can be extend to multiple variables modifications

$$x_1, \ldots x_n :\mid P$$

- $x :\mid P$ and $x :\in S$ are non-deterministic actions

Event-B : Events

- Initialisation
  - Definition initial values of state variables. $x : |P$
- Modification of state variables with Before-After Predicates BAA
  - $BAA(x, x')$. Example $x' = x + 1$ pour $x := x + 1$ or for $x :\mid (x' = x + 1)$
- Three types of events

Non guarded event	Guarded Event	Parameterised Event
event_1 = BEGIN $x : \mid BAA(x, x')$ END	event_2 = WHEN $G(x)$ THEN $x : \mid BAA(x, x')$ END	event_3 = ANY l WHERE G $G(l, x)$ THEN $x : \mid BAA(x, x', l)$ END

where

- x is a (set of) variables
- l is a list of parameters
- $G(x)$ is a boolean expression on state variables expressing a guard
- $BAA(x, x')$ and $BAA(x, x', l)$ are before-after predicate recording a state change

Events. Definition of associated BAA

Event : E	Before-After Predicate (BAA)
begin $x : \mid P(x, x')$ end	$P(x, x')$
when $G(x)$ then $x : \mid P(x, x')$ end	$G(x) \wedge P(x, x')$
any t where $G(t, x)$ then $x : \mid P(x, x', t)$ end	$\exists t. (G(t, x) \wedge P(x, x', t))$

Events. Definition of associated guards

Event : E	Guard : grd(E)
begin S **end**	$TRUE$
when $G(x)$ **then** T **end**	$G(x)$
any t **where** $G(t,x)$ **then** T **end**	$\exists t.\, G(t,x)$

Machine Proof Obligations

	Proof obligation
(INV1)	Invariant preservation at initialisation
(INV2)	Invariant preservation by each event
(DEAD)	Deadlock freeness
(SAFE)	Theorems shall be prove,
(FIS)	Events shall be feasible

Machine Proof Obligations

	Proof obligation
(INV1)	$\Gamma(s,c) \vdash Init(x) \Rightarrow I(x)$
(INV2)	$\Gamma(s,c) \vdash I(x) \wedge BAA(e)(x,x') \Rightarrow I(x')$
(DEAD)	$\Gamma(s,c) \vdash I(x) \Rightarrow (\mathrm{grd}(e_1) \vee \ldots \mathrm{grd}(e_n))$
(SAFE)	$\Gamma(s,c) \vdash I(x) \Rightarrow T(x)$
(FIS)	$\Gamma(s,c) \vdash I(x) \wedge \mathrm{grd}(E) \Rightarrow \exists x'.\, P(x,x')$

Event based system modelling with Event-B : Modelling Principles —– Event-B

Components of an Event-B model

- Contexts
- Machines

Event-B models. Handling contexts

Contexts define theories associated to models.

- Definition Of the theories required by system models
- Contexts are imported by machines using the Sees Clause

```
Context C0
Sets s
Constants c
Axioms Ax(s, c)
Theorems Tc(s, c)
End
```

Contexts

- constants (c)
- sets (s)
- Axioms Ax(s,c)
- Theorems Tc(s,c)

Event-B models. Machine definition

Machines define a state-transitions system.

- **Machines.** Initial state + events (transitions between states).

- **Machines** : Variables (état), Events (transitions),
- Invariants $I(s, c, x_A)$, Theorems $T(s, c, x_A)$
- Proof Obligations
- Non determinism
- Interleaving semantics with stuttering
- Traces correspnd to sequences of event triggerings

```
Machine Spec
Sees C0
Variables x_A
Invariants Inv(s, c, x_A)
Theorems T(s, c, x_A)
Events
  Event Initialisation ≙
          begin
             x_A :| Init(s, c, x'_A)
          end ;
  Event An_event ≙ Any l
          Where
             G_A(l,s,c,x_A)
          Then
             x_A :| AC1(l,s,c,x_A,x'_A)
          End
  Event Another_event ≙
          When
             GG_A(s,c,x_A)
          Then
             x_A :| AC2(s,c,x_A,x'_A)
          End
    ...
        End
```


Event-B models. Machine definition

- We recall the three types of Events

Non guarded event	Guarded event	Parameterised event
événement_1 = BEGIN $x :\| BAA(x, x')$ END	événement_2 = WHEN $G(x)$ THEN $x :\| BAA(x, x')$ END	Evenement_3 = ANY l WHERE G $G(l, x)$ THEN $x :\| BAA(x, x', l)$ END

- where
  - x is a (set of) variables
  - l is a list of parameters
  - $G(x)$ is a boolean expression on state variables expressing a guard
  - $BAA(x, x')$ and $BAA(x, x', l)$ are before-after predicate recording a state change
- Correspondence between an Event-B **event** and a TLA **action** (TLA-L. Lamport)
- Events describe a **state-transitions system** with **interleaving semantics** for events

Event-B proof obligations - Core POs (recall)

- *POs for theorems*
  $$A(s,c) \Rightarrow Tc(s,c)$$
  $$A(s,c) \wedge I(s,c,x) \Rightarrow T(s,c,x)$$
- *Invariant preservation PO*
  $$A(s,c) \wedge I(s,c,x) \wedge G(s,c,l,x) \wedge BAA(s,c,l,x,x') \Rightarrow I(s,c,x')$$
- *Event feasibility PO*
  $$A(s,c) \wedge I(s,c,v) \wedge G(s,c,l,x) \Rightarrow \exists v'.BAA(s,c,l,x,x')$$
- *Variant PO*
  $$A(s,c) \wedge I(s,c,x) \wedge G(s,c,l,x) \Rightarrow V(s,c,x) \in \mathbb{N}$$
- *Variant PO*
  $$A(s,c) \wedge I(s,c,x) \wedge G(s,c,l,x) \wedge BAA(s,c,l,x,x') \Rightarrow V(s,c,x') < V(s,c,x)$$

Event-B proof obligations - Other POs

Other PO are added to the previous ones

- Deadlock freeness (DEAD) : disjunction of guards
  - A single guard is true at each event triggering (Deterministic system)
  - At least one guard is true at each event triggering (Non determinism)
  - No gaurd may be true at event triggering (The developed system may deadlock)
- Liveness and reachability (LIV) *Leads_to* or $P \leadsto Q$ or

  ```
  when
      P
  then
      Q
  ```

  Similar to Liveness in temporal logic. For example, in LTL with *leads_to* noted $\leadsto$ operator or $\diamond$
- Refinement
  - Preservation of the invariant thanks to the introduction of a gluing invariant
  - Do not allow an event of refined machine to be triggered infinitely many times (use of a variant). This a livelock
  - The refined system does not deadlock more than the abstract one
- ....

An example

```
contexts
  data
sets
  MESSAGES
  AGENTS
  DATA
constants
  n
  infile
axioms
  axm1 : n ∈ ℕ
  axm2 : n ≠ 0
  axm3 : infile ∈ 1 .. n → DATA
end
```


An example (cont.)

```
MACHINE agents
SEES data
VARIABLES
  sent
  got
  lost
INVARIANTS
  inv1 : sent ⊆ AGENTS × AGENTS
  inv2 : got ⊆ AGENTS × AGENTS
  inv4 : (got ∪ lost) ⊆ sent
  inv6 : lost ⊆ AGENTS × AGENTS
  inv7 : got ∩ lost = ∅
```

```
INITIALISATION
BEGIN
  act1 : sent := ∅
  act2 : got := ∅
  act4 : lost := ∅
END
```


An example (cont.)

```
sending a message
ANY
  a
  b
WHERE
  grd11 : a ∈ AGENTS
  grd12 : b ∈ AGENTS
  grd1 : a ↦ b ∉ sent
THEN
  act11 : sent := sent ∪ {a ↦ b}
END
```

```
getting a message
ANY
  a
  b
WHERE
  grd11 : a ∈ AGENTS
  grd12 : b ∈ AGENTS
  grd13 : a ↦ b ∈ sent \ (got ∪ lost)
THEN
  act11 : got := got ∪ {a ↦ b}
END
```

```
loosing a messge
ANY
  a
  b
WHERE  grd1 : a ∈ AGENTS
  grd2 : b ∈ AGENTS
  grd3 : a ↦ b ∈ sent \ (got ∪ lost)
THEN
  act1 : lost := lost ∪ {a ↦ b}
END
```

Another example

- An example of specification.
- A single event `selection`

```
Context C0
Sets
PRODUCTS, SITES
...
End
```

- Many refinements are possible

```
Machine M1
Variables P, carts, selection_done
Invariants
  P ⊆ PRODUCTS
  carts ⊆ SITES × PRODUCTS
  selection_done ∈ BOOL
  selection_done ⇒ ran(carts) = P
  ∀p, p ∈ ran(carts) ⇒ card(carts⁻¹[{p}]) = 1
Events
  Event initialisation =
      P :∈ ℙ(PRODUCTS)
      carts := ∅
      selection_done := FALSE

  Event selection =
    Any someCarts
    Where
      someCarts ⊆ SITES × PRODUCTS
      ran(someCarts) = P
      ∀p, p ∈ ran(carts) ⇒ card(carts⁻¹[{p}]) = 1
    Then
      carts := someCarts
      selection_done := TRUE
    End
End
```


Refinement in Event-B

- **New events** may appear.
  - They refine the *Skip* event
  - Definition of a simulation (weak) relation
- The **concrete events (refined events)** shall not introduce more deadlock than available in the abstraction
- The set of new events **may lead to liveness (due to stuttering)**
  - Need to use a decreasing variant to allow **triggering of the abstract events**
- The abstract model use variables x while the concrete ones use variables y, then
  - a **gluing invariant** $J(x, y)$ shall link abstract and concrete variables x and y
- Each **abstract event** is refined by a **concrete event**

Refinement in Event-B. Proof obligations

Guarded events

Let us consider an **abstract** event and the corresponding refining **concrete** event such that

```
EVENT =            EVENT =
  when               when
    G(x)               H(y)
  then               then
    x := E(x)          y := F(y)
  end                end
```

Invariant preservation proof obligation

Let $I(x)$ and $J(x, y)$ be the **invariants** , then we need to **prove** the **refinement invariant preservation** as

$$I(x) \wedge J(x,y) \wedge H(y) \implies G(x) \wedge J(E(x), F(y))$$

Refinement in Event-B. Proof obligations

Parameterised events

Let us consider an **abstract** event and the corresponding refining **concrete** event such that

```
EVENT =              EVENT =
  any v where          any w where
    G(x,v)               H(y,w)
  then                 then
    x := E(x,v)          y := F(y,w)
  end                  end
```

Invariant preservation proof obligation

Let $I(x)$ and $J(x, y)$ be the **invariants** , then we need to **prove** the **refinement invariant preservation** as

$$I(x) \wedge J(x,y) \wedge H(y,w) \implies \exists v . (G(x,v) \wedge J(E(x,v), F(y,w)))$$

Refinement in Event-B. Proof obligations

New events

Let us consider a **new** event refining the **skip** event as follows

```
EVENT =            EVENT =
  SKIP               when
 end                   H(y)
                     then
                       y := F(y)
                     end
```

Invariant preservation proof obligation

Let $I(x)$ and $J(x,y)$ be the **invariants** , then we need to **prove** the **refinement invariant preservation** as

$$I(x) \wedge J(x,y) \wedge H(y) \implies J(x,F(y))$$

Remark. In Rodin, no need to declare the event `Skip` of the abstraction. By default, any new event refines the `Skip` event.

Refinement in Event-B

```
Context C1
Extends C0
Sets s_r
Constants c_r
Axioms A(s, c, s_r, c_r)
Theorems Tc(s, c, s_r, c_r)
End
```

- Extension of `Contexts`.
- `Machines` are refined.
- New variables.
- New events.
- *Gluing* Invariants.
- Refinement Proof Obligations.

```
Machine Spec_Ref
Refines Spec
Sees C1
Variables y
Invariants Inv_r(s, c, s_r, c_r, x, y)
Theorems T_r(s, c, s_r, c_r, x, y)
Events
  Event Initialisation =
            begin
              y :| Init(s, c, y')
            end ;
  Event An_event_ref
      Refines An_event =
            Any e
            Where
              G1_r(e,s,c,s_r,c_r, y)
            Then
              y :| AC1_r(e,s,c,s_r,c_r, y,y')
            End
  Event Another_event_ref
      Refines Another_event =
            When
              G2_r(s,c,s_r,c_r, y)
            Then
              y :| AC2(s,c,s_r,c_r, y,y')
            End
  Event New_event
            When
              G3_r(s,c,s_r,c_r, y)
            Then
              y :| AC3(s,c,s_r,c_r, y,y')
            End
      ...
        End
```


Refinement in Event-B. An example

Introduction

1. The objective is to design an information system to manage orders and invoices
2. To issue an invoice, the state of an order shall be changed (moving from state *"pending"* to *"invoiced"*).
3. On an order, only one reference to an ordered product is available together with a quantity. Quantity may be different from an order to another.
4. A given product reference may appear on several orders.
5. The state of an order moves to *"invoiced"* if the ordered quantity is less or equal to the quantity of available products in the stock.

Refinement in Event-B. An example

The following cases shall be considered.

Case 1

All the order references are available in the stock. The stock and the orders may change and evolve due to

- arrival of new orders or cancellations of orders
- the supplying of products with new quantities added to the stock

But, we do not have to take these entries into account. This means that you will not receive two entry flows (orders, entries in stock). The stock and the set of orders are always given to you in a up-to-date state

Case 2

We shall take into account

- arrivals of new orders
- cancellations of orders
- arrivals of new quantities added to the stock

End of case study

Refinement in Event-B. An example

```
model
  Case1
sets
  ALL_ORDERS; PRODUCTS
properties
  ALL_ORDERS ≠ ∅
variables
  orders, stock, invoiced_orders, reference, quantity
invariant
  orders ⊆ ALL_ORDERS ∧
  stock ∈ PRODUCTS ⟶ ℕ ∧
  invoiced_orders ⊆ orders ∧
  quantity ∈ orders ⟶ ℕ* ∧
  reference ∈ orders ⟶ PRODUCTS
initialisation
  stock, invoiced_orders, orders, quantity, reference := PRODUCTS × {0}, ∅, ∅, ∅, ∅
events
  ...
END
```


Refinement in Event-B. An example

```
invoice_order =
  ANY
    o
  WHERE
    o ∈ orders − invoiced_orders ∧
    quantity(o) ≤ stock(reference(o))
  THEN
    invoiced_orders := invoiced_orders ∪ {o}
    stock(reference(o)) := stock(reference(o)) − quantity(o)
  END;
```

```
delivery_to_stock =
  BEGIN
    stock :| (stock′ ∈ PRODUCTS ⟶ ℕ)
  END
```


Refinement in Event-B. An example

```
cancel_orders =
  BEGIN
    orders, quantity, reference :| (orders′ ⊆ ALL_ORDERS ∧
    invoiced_orders′ ⊆ orders′ ∧
    quantity′ ∈ orders′ ⟶ ℕ* ∧
    reference′ ∈ orders′ ⟶ PRODUCTS)
  END;
```

```
new_orders =
  BEGIN
    orders, quantity, reference :| (orders′ ⊆ ALL_ORDERS ∧
      invoiced_orders′ ⊆ orders ∧
      quantity′ ∈ orders′ ⟶ ℕ* ∧
      reference′ ∈ orders′ ⟶ PRODUCTS)
  END;
```


Refinement in Event-B. An example

```
model
  Case2
refines
  Case1
variables
  orders, stock, invoiced_orders, reference, quantity
initialisation
  stock, invoiced_orders, orders, quantity, reference := PRODUCTS × {0}, ∅, ∅, ∅, ∅
events
  ...
```

Refinement in Event-B. An example

```
cancel_orders
  Refines   cancel_orders =
  ANY
    o
  WHERE
    o ∈ orders − invoiced_orders
  THEN
    orders := orders − {o}
    quantity := {o} ⩤ quantity
    reference := {o} ⩤ reference
  END ;
```

```
delivery_to_stock
  Refines   delivery_to_stock =
  ANY
    p. n
  WHERE
    p ∈ PRODUCTS
    n ∈ ℕ
  THEN
    stock(p) := stock(p) + n
  END
END
```

```
new_orders
  Refines   new_orders =
  ANY
    o. q. p
  WHERE
    o ∈ ALL_ORDERS − orders
    q ∈ ℕ*
    p ∈ PRODUCTS
  THEN
    orders := orders ∪ {o}
    quantity(o) := q
    reference(o) := p
  END ;
```


Refinement in Event-B. Methodology

Some methodological principles

- Find the right abstraction at the right abstraction level
- Define a refinement strategy
  - What are the refinement steps for a development ?
- Write the right invariants and this, the right properties
  - Model animation can help to identify the invariant

Caution

- Introduce properties at different refinement levels, when their expression becomes possible
- Take advantage from refinement in order to ease the proof process

Structure of an Event-B development

- A machine models
  - the static part of a system i.e. state with state variables
  - the dynamic of a system i.e. set of events
- The properties, formalising requirements, are described in the *INVARIANT*, *THEOREM*, *VARIANT*
  - safety
  - deadlock freeness
  - function of the system
  - reachability,
  - etc.

Structure of an Event-B development

- A set of machines linked by a refinement relationship (simulation)
- Expressed requirements are handled incrementally during the refinement process
- Contexts are extended each it is necessary to introduce new definitions and axiomatisations of needed concepts
- The *SEES* clause makes contexts in a machine
- Development activities : modelling, refinement, proof, animation, exhaustive verification, code generation, close loop modelling, etc.
- Event-B method handles the development of complex systems

Plan

7. Proof with Event-B
   - Proof activity
   - Proofs with Event-B and the Rodin platform

Proof in logic

Proof strategy for a sequent S

- Let
  - A collection $\mathcal{T}$ of inference rules of the form $\frac{A}{C}$ **r**
  - A sequent container K, containing S at initialisation

While K is not empty
- **CHOOSE** an inference rule $\frac{A}{C}$ **r** in $\mathcal{T}$ such that its conclusion C is in K
- **SUBSTITUTE** C in K by the hypotheses A (if there are)

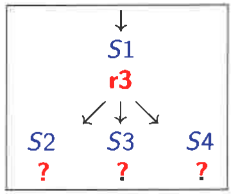
End

- The proof succeeds when K becomes empty
- The proof is said to be *Goal Oriented*

The result is a Proof Tree

The Proof Tree

Let us consider $\frac{S_2, S_3, S_4}{S_1}$ **r3** inference rule



- Inference rule **r3** is applied to the sequent S1
- This rule produces the sequents S2, S3, and S4

The Proof Tree. An example

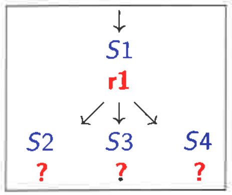
Let us consider the following inference rules

$\frac{S_2, S_3, S_4}{S_1}$ **r1** $\quad \frac{S_7}{S_4}$ **r2** $\quad \frac{}{S_2}$ **r3**

$\frac{}{S_5}$ **r4** $\quad \frac{S_5, S_6}{S_3}$ **r5** $\quad \frac{}{S_6}$ **r6**

$\frac{}{S_7}$ **r7**

Our objective is to prove the sequent S_1



The Proof Tree. An example

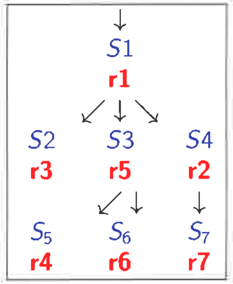
Let us consider the following inference rules

$$\frac{S_2, S_3, S_4}{S_1}\,\mathbf{r1} \qquad \frac{S_7}{S_4}\,\mathbf{r2} \qquad \frac{}{S_2}\,\mathbf{r3}$$

$$\frac{}{S_5}\,\mathbf{r4} \qquad \frac{S_5, S_6}{S_3}\,\mathbf{r5} \qquad \frac{}{S_6}\,\mathbf{r6}$$

$$\frac{}{S_7}\,\mathbf{r7}$$

Our objective is to prove the sequent S_1



Plan

7. Proof with Event-B
   - Proof activity
   - Proofs with Event-B and the Rodin platform

Proof in logics

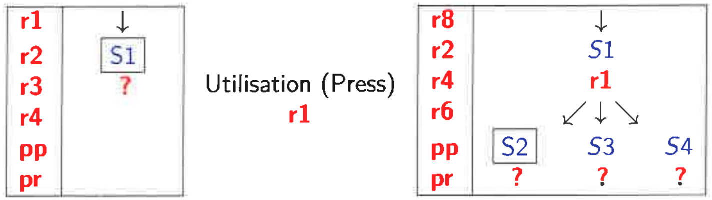
A proof in sequent calculus is a tree.

- L'application of inference rules ⇝ a proof tree.
- Two types of reasoning
  - Forward reasoning.

    Top-Down Application of inference rules $\frac{A}{C}\downarrow$
  - Backward reasoning.

    Bottom-Up Application of inference rules $\frac{A}{C}\uparrow$
- Building the proof tree represents the proof activity
  - Automatic building the proof tree using automatic provers
    - Examples : Predicate provers, reasoners, SAT or SMR Solvers, static analysis, etc.
  - Interactive application of inference rules or deduction rules available in proof assistants
    - Examples : CoQ, Atelier B, Rodin, Isabelle/HOL, etc.
  - Mixed building of the proof tree combing both automatic and interactive proofs
  - Use of proof tactics with CoQ, Atelier B, Rodin, Isabelle/HOL, TLAPS etc.

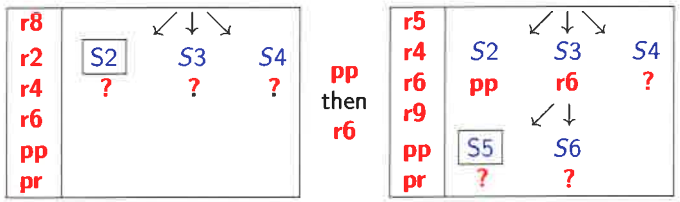
Prover Interface : A Tree and a Palette of buttons the inference rules



We may use

- Inference Rules (r_i)
- Automatic provers like *pr*, *pp* or SMT, etc.

A Difficulty : Size of the window



The previous *interface* is not well adapted

- Les Sequents are usually of big size (many hypotheses)
- The Proof Tree may be very deep

Current form of a sequent

In general, sequents issued from Proof Obligations are of the following form

$$H \vdash L \Longrightarrow G$$

- Conclusion is usually an implication
- G is Goal is a predicate, usually a non-conjunctive
- L is the set of so-called local hypotheses (may be an empty set)

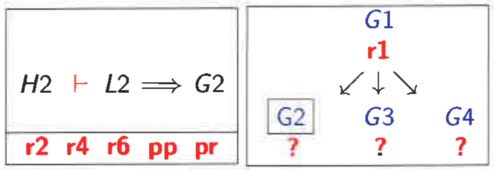
General form of a sequent for the proof of invariant

$$H \vdash I(x) \land G(x) \land P(x,x') \Longrightarrow I(x')$$

A possible solution : Use of two windows

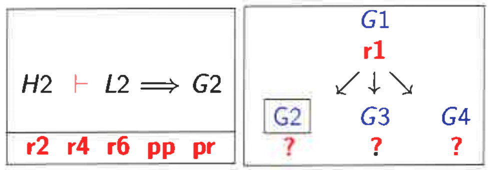
- A Tree Window with simplified sequents (goals only)

- Sequent Window containing the **complete sequent of interest**

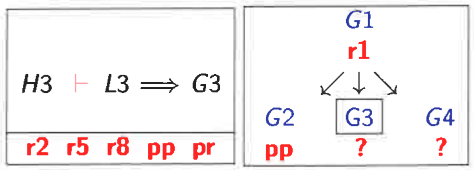


Fenêtre Séquent Fenêtre Arbre

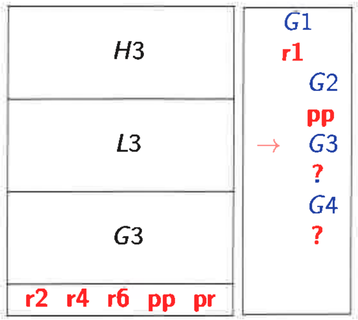
A proof step



Use of (Press) pp



More realistic windows



More elaborated sequents

$$\underbrace{hidden \; ; \; searched \; ; \; cached}_{list\_of\_hypotheses} \vdash \underbrace{local \Longrightarrow goal}_{conclusion}$$

hidden	Non visible in the sequent window
searched	Visible after search in *hidden*
cached	Visible but is no more part of the conclusion
local	Visible and part of the conclusion

Plan

8 The Rodin Platform

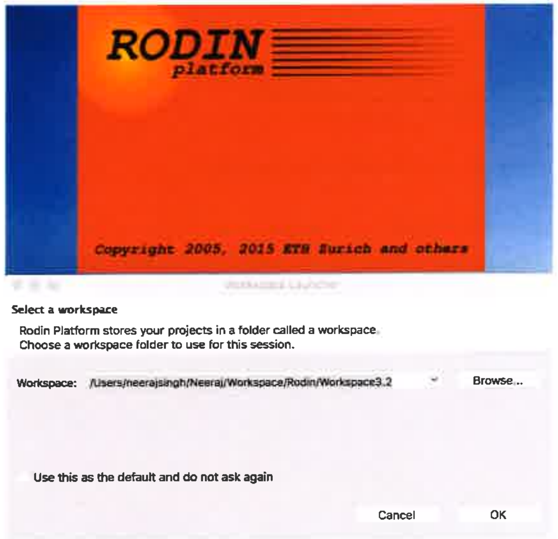
The Rodin Platform

It is an application developed on Eclipse for the management and development of system models supporting verification of system model correctness.
Download `http://www.event-b.org`

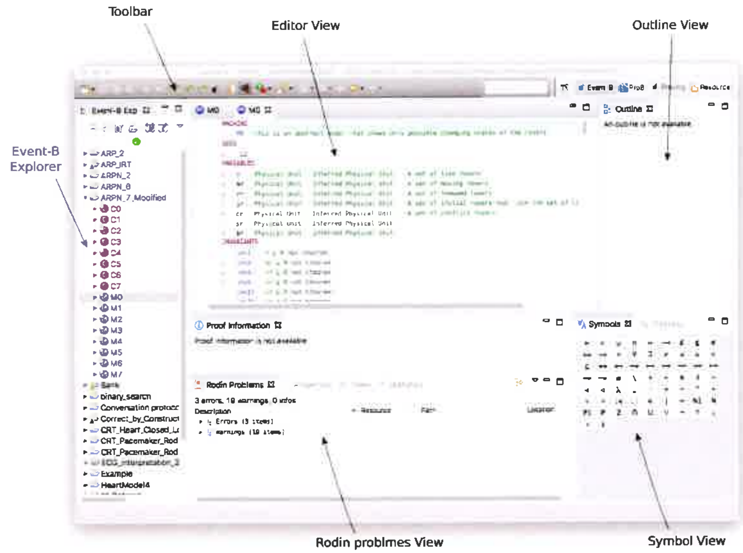
Rodin Platform and Plug-in Installation

Name	Installation
Rodin platform	• Requires Java 1.6 • Download the Core: Rodin Platform file for your platform. To install, just unpack the archive anywhere on your hard-disk and launch the "rodin" executable in it. • Start Rodin • Information on the latest release.
Plug-ins	• Plug-ins are installed from within Rodin by selecting Help/Install New Software. Then select the appropriate update site from the list of download sites. • Details on plug-ins. • Install the Atelier B Provers plugin from the Atelier B Provers Update site to take full advantage of Rodin proof capabilities • Install the ProB plugin from the ProB Update site for powerful model checking and animation
User manual and Tutorial	• Rodin Handbook

The Rodin Platform. Launching Rodin & definition of Workspace

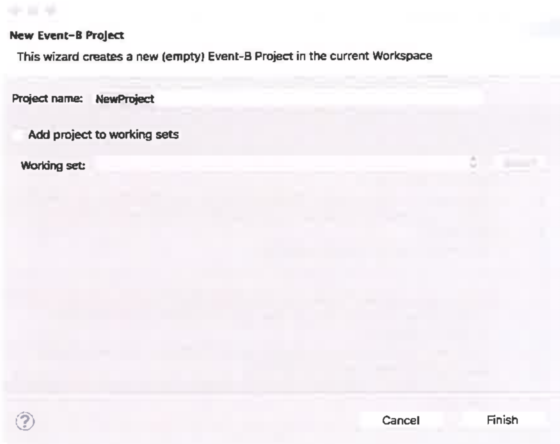


The Rodin Platform. The Rodin interface (GUI)



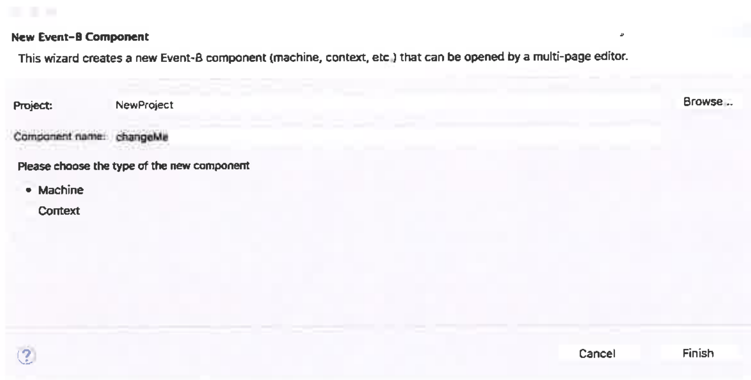
The Rodin Platform. Creating a new Project)

File > New > Event-B Project

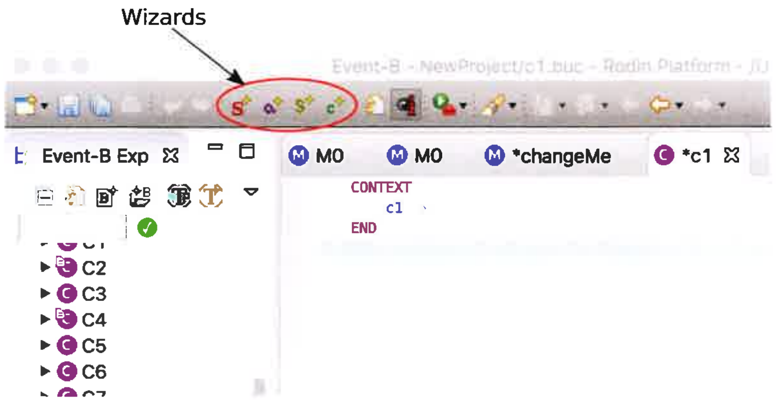


The Rodin Platform. Creating Event-B Components

File > New > Event-B Components



The Rodin Platform. Construction of contexts (Context component)

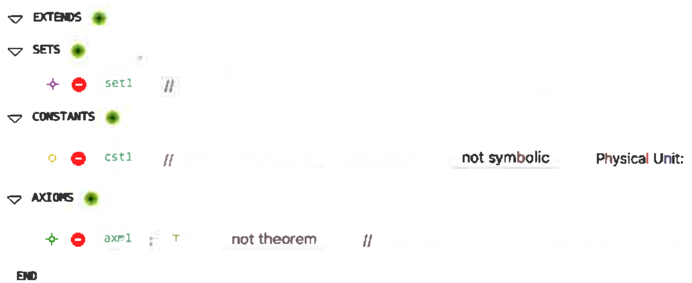


The Rodin Platform. Construction of contexts (Context component) with Wizards

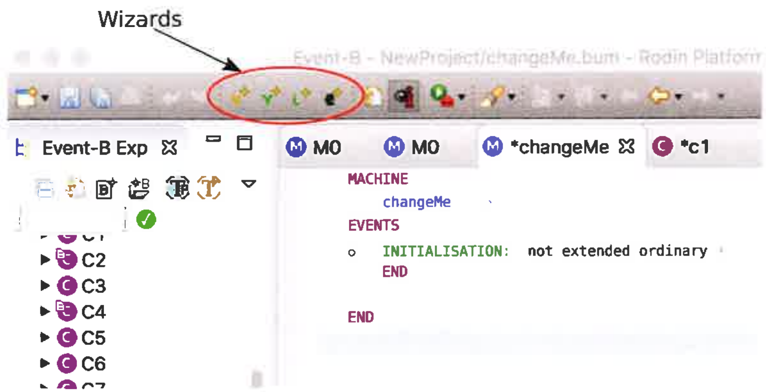
FIGURE – New sets (Enumerated Set) and new axioms (Axioms)

FIGURE – New sets (New Carrier Sets) and constants definitions (Constants)

The Rodin Platform. Rodin Editor for contexts components (Context)



The Rodin Platform. Building Machines



The Rodin Platform. Building Machines with Wizards

FIGURE – New Variables, Invariants and Variants

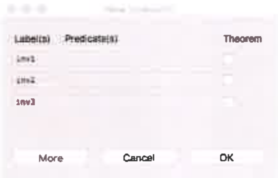
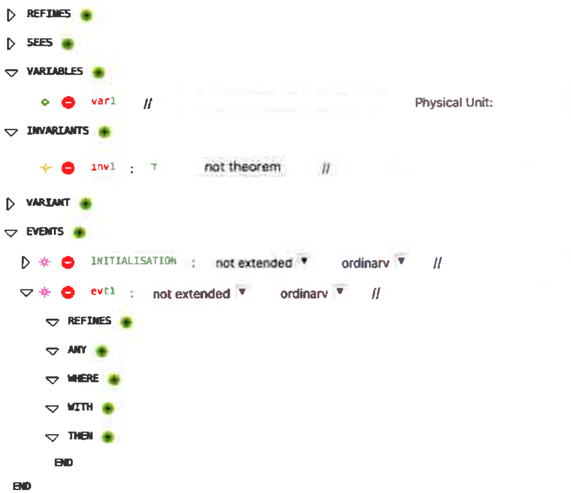


FIGURE – Adding Invariants

The Rodin Platform. New events (Events)

FIGURE – New Events)

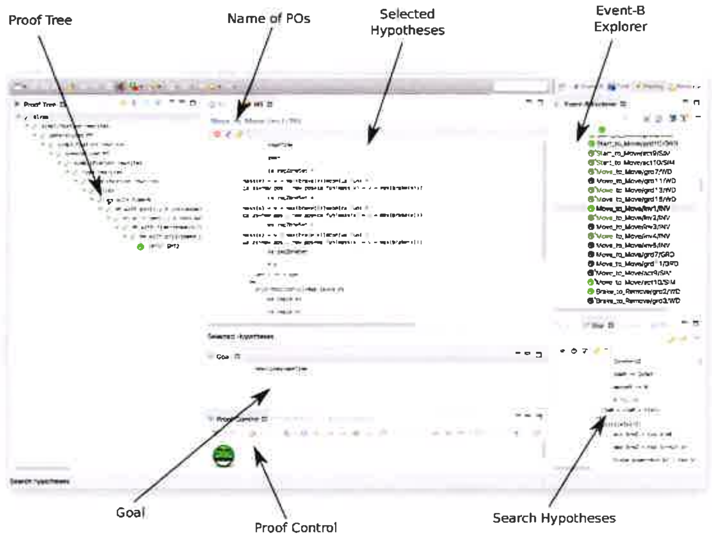
The Rodin Platform. Rodin Editor for the Machine component



The Rodin Platform. Proof support

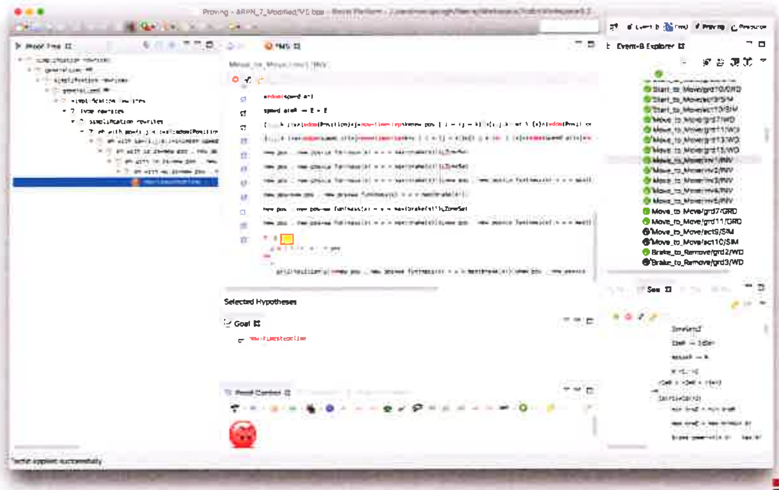
The RODIN Prover

The Rodin Platform. Interface (GUI) of the Rodin Prover

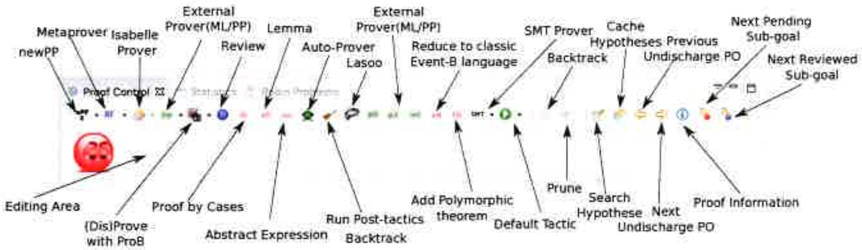


The Rodin Platform. Interface (GUI) of the Rodin Prover

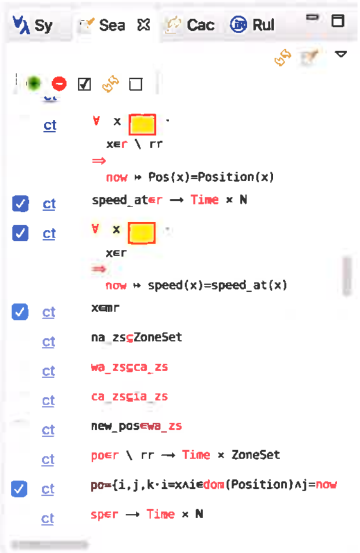
Interface (GUI) for an unsuccessful proof



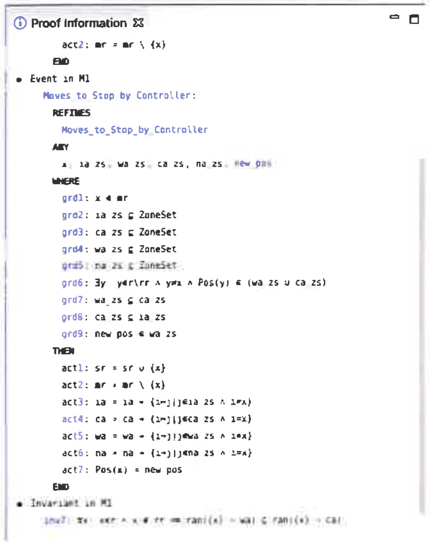
The Rodin Platform. Interface (GUI) of the Rodin Prover View of the Proof Control



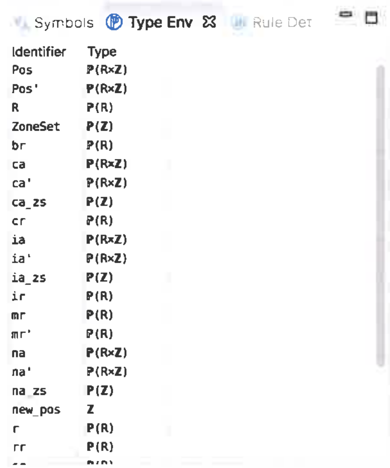
The Rodin Platform. Interface (GUI) of the Rodin Prover. Search of Hypotheses



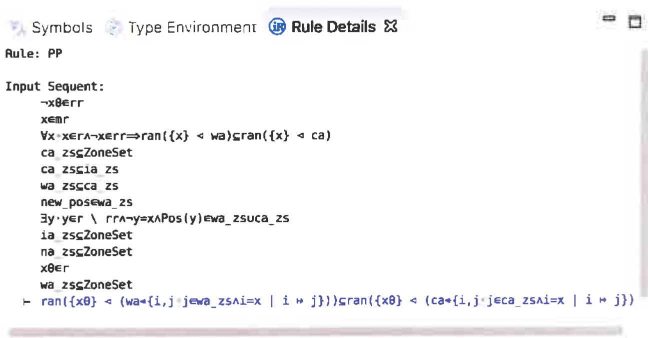
The Rodin Platform. Interface (GUI) of the Rodin Prover. The information perspective on proofs



The Rodin Platform. Interface (GUI) of the Rodin Prover. The used Types perspective



The Rodin Platform. Interface (GUI) of the Rodin Prover. The used inference rules perspective



The Rodin Platform. Interface (GUI) of the Rodin Prover. The preference perspective for tactics (Auto/Post)

Plan

9. Animation of Event-B models

Animation of Event-B models. Use of model checking

- The Rodin platform is equipped with a model checker offering the capability to
  - animate Event-B models at all refinement levels
  - *model check* properties expressed in temporal logic

Benefits

- Exhaustive verification if the models are finite
- Assistance to the design of Event-B models by identifying counter-examples

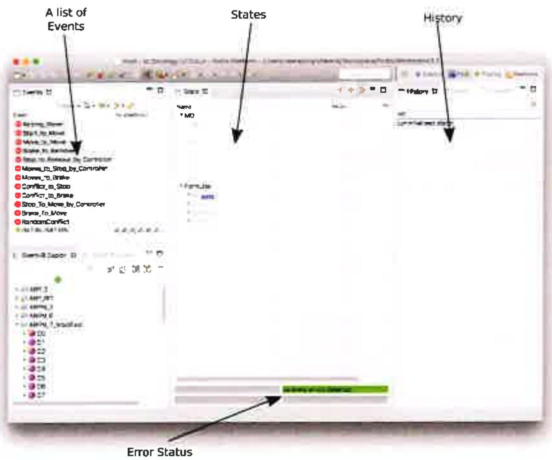
Animation of Event-B models. Use of model checking

The ProB model-checker

Animation of Event-B models. Use of model checking. What is ProB ?

A useful tool for analysing and debugging Event-B models.

$\Longrightarrow$ It is required to bound models (if they are not) to use ProB

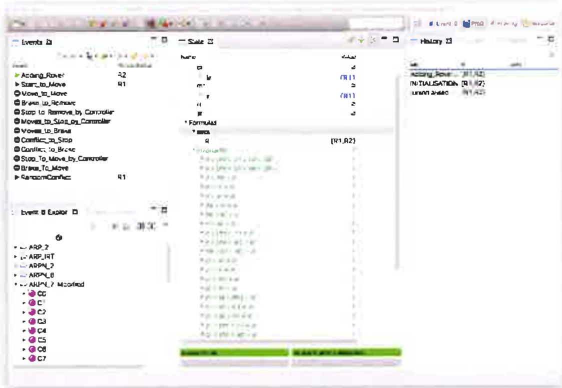


Animation of Event-B models. Model checking with ProB ?

Rodin Platform > Preference > ProB

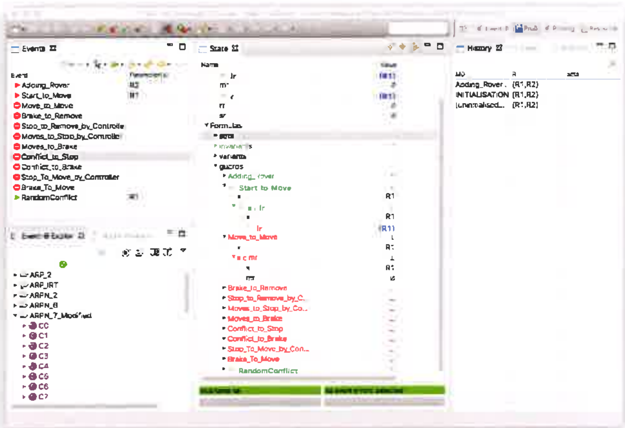
Animation of Event-B models. Model checking with ProB ?
Verification of Invariants

- Enable Events
- Check Invariants



Animation of Event-B models. Model checking with ProB ?
Guards Checking

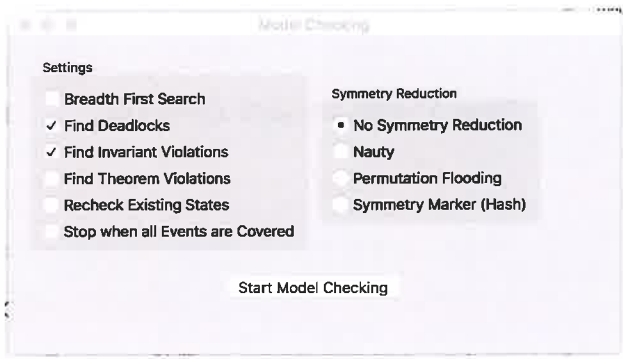
- Guards Checking



Animation of Event-B models. Model checking with ProB.
Deadlocks Freeness

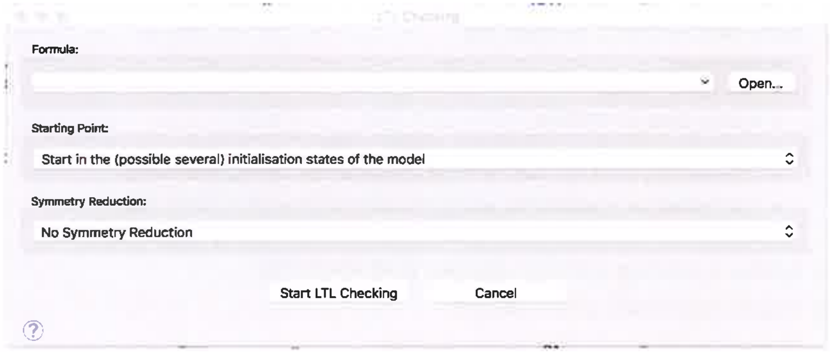
- Deadlock Freedom

Checks > Model Checking



Animation of Event-B models. Model checking with ProB. LTL in ProB

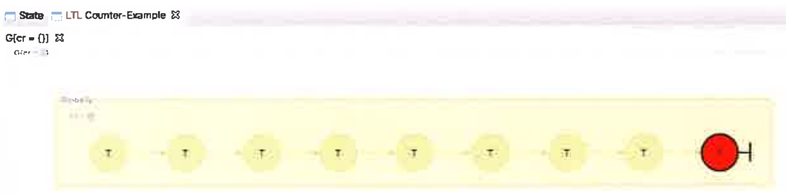
Checks > LTL Model Checking



Animation of Event-B models. Model checking with ProB. LTL in ProB

The following formula can be verified.

$$G\{cr = \varnothing\}$$



Adding_Rover → Adding_Rover → Start_to_Move → Start_to_Move → Move_to_Stop_by_Controller → Stop_to_Remove_by_Controller → Move_to_Stop_by_Controller → Random_Conflict

Animation of Event-B models. Model checking with ProB. LTL in ProB

The following formulas can be verified.

$G\{Temperature >= 7 \& Temperature <= 35\}$ No Counterexample
$G\{Temperature < 35\}$ Counterexample

G {Temperature > 7 } Counterexample
X {Temperature < 7 } Counterexample

The RODIN paltform

- It is an IDE (Integrated Development Environment) for the development of Event-B models
- Developed on top of Eclipse
- Many associated tools
  - Model editors, refinement
  - Proofs, model animation
  - Validation of models using model checking (ProB)
  - Code generation
  - Many PlugIns (UML, BPEL2B, THEORY, Prouveurs, etc.)
- Available on `http://www.event-b.org`

Used for TP at ENSEEIHT

Plan

Conclusion

- Event-B is a system modelling method which uses refinement and proof
- Incremental development approach
- Many developments in areas like
  - transport,
  - electronic cards,
  - cyber-physical systems,
  - embedded systems, pacemaker, insulin pump,
  - information systems, web services composition, voting machines,
  - mathematical engineering, proof and demonstration of theorems,
  - etc.
- System modelling and high abstract level reasoning
- Simple mathematical foundations
- Availability of a tool with many plug-ins

FIN

yamine@enseeiht.fr

nsingh@enseeiht.fr