# Formal development of complex systems

Yamine AIT-AMEUR, Neeraj Kumar SINGH

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

2020

# Plan

# 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

Some problems

- 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

# Objectifs

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

# Propositional logics

Propositional logics operators.

| | |
|---|---|
| $\perp$ | Constant *False* |
| $\top$ | Constant *True* |
| $\neg A$ | Negation |
| $A \wedge B$ | Conjunction |
| $A \vee B$ | Disjunction |
| $A \Rightarrow B$ | Implication |
| $A \iff B$ | Equivalence |

# Propositional logics

| | |
|---|---|
| | $A \rightarrow B = \neg A \vee B$<br>$A \leftrightarrow B = (A \rightarrow B) \ \wedge \ (B \rightarrow A)$ |
| Idempotent | $A \wedge A = A$<br>$A \vee A = A$ |
| | $A \wedge \neg A = \bot$<br>$A \vee \neg A = \top$<br>$A \wedge \bot = \bot$<br>$A \wedge \top = A$<br>$A \vee \bot = A$<br>$A \vee \top = \top$<br>$\neg\neg A = A$ |
| Commutativity | $A \wedge B = B \wedge A$<br>$A \vee B = B \vee A$ |
| Associativity | $(A \wedge B) \wedge C = A \wedge (B \wedge C)$<br>$(A \vee B) \vee C = A \vee (B \vee C)$ |

# Propositional logics

| | |
|---|---|
| Distributivity | $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$<br>$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$ |
| De Morgan | $\neg(A \wedge B) = \neg A \vee \neg B$<br>$\neg(A \vee B) = \neg A \wedge \neg B$<br>$A \vee (\neg A \wedge B) = A \vee B$<br>$A \wedge (\neg A \vee B) = A \wedge B$<br>$A \vee (A \wedge B) = A$<br>$A \wedge (A \vee B) = A$ |

# Proofs and proof system

## Sequent and inference rule

- Sequent

$$list\_of\_hypotheses \quad \vdash \quad conclusion$$

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

- Inference rule. Generic form $\dfrac{A}{C}r$ or $\dfrac{A_1, \ldots A_n}{C}r$
  - $A$ 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.

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

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

# Proofs and proof system

- Definition of inference rules.
- Useful for inferring (deduction) of new sequents

- Implication Elimination (E) and Introduction (I)

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

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

# Proofs and proof system

- And Elimination (E) and Introduction (I)

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

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

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \quad (\mathrm{I}_\wedge)$$

# Proofs and proof system

- Or Elimination (E) and Introduction (I)

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

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

$$\frac{\Gamma \vdash A \vee B \quad \Gamma ; A \vdash C \quad \Gamma ; B \vdash C}{\Gamma \vdash C} \quad (\mathsf{E}_\vee)$$

- Elimination is useful for case base reasoning

# Proofs and proof system

- Handling negation

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A} \quad (\mathsf{E}_\bot)$$

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

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

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

# 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

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

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

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A} \quad (\ \mathrm{I}_\forall) \qquad (\text{for } x \notin FV(\Gamma))$$

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

# Sequents in predicate logic

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

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

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

# 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

$$
\begin{array}{ll}
\textit{Predicate} ::= & \textit{Expression} = \textit{Expression} \\
\textit{Expression} ::= & \ldots \\
\textit{Variable} ::= & \ldots
\end{array}
$$

## 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

# 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 $\{x \mid x \in S \wedge P\}$

These constructs are used to define other set operators.

# Sets. Basic operators

- Inclusion $S \subseteq T$
- Associated axioms

| | | |
|---|---|---|
| $S \subseteq T$ | $\triangleq$ | $S \in \mathbb{P}(T)$ |
| $S = T$ | $\triangleq$ | $S \subseteq T \land T \subseteq S$ |

- 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 | $\cup x.(x \in S \wedge P)$ |
| Generalised Intersection | $\mathtt{inter}(S)$ |
| Quantified Intersection | $\cap x.(x \in S \wedge P)$ |

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

# Binary Relations

| | |
|---|---|
| Binary Relation | $S \leftrightarrow T$ |
| Domain | `dom(r)` |
| Co-domain (Range) | `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 | $S \srel T$ |
| Total binary relation | $S \trel T$ |
| Total Surjective binary relation | $S \strel T$ |

## Axiomatisation

| | |
|---|---|
| $S \srel T$ | If $r \in S \srel T$ then $ran(r) = T$ |
| $S \trel T$ | If $r \in S \trel T$ then $dom(r) = S$ |
| $S \strel T$ | If $r \in S \srel T$ then $dom(r) = S \;\wedge\; ran(r) = T$ |

# Binary Relations

## Manipulation of binary relations

- Restriction and Subtraction

| | |
|---|---|
| Domain Restriction | $S \lhd r$ |
| Range Restriction | $r \rhd T$ |
| Domain Subtraction | $S \unlhd r$ |
| Range Subtraction | $r \unrhd T$ |

## Axiomatisation

| | |
|---|---|
| $S \lhd r$ | $S \lhd r = \{x \mapsto y \mid x \mapsto y \in r \ \wedge \ x \in S\}$ |
| $r \rhd T$ | $r \rhd T = \{x \mapsto y \mid x \mapsto y \in r \ \wedge \ y \in T\}$ |
| $S \unlhd r$ | $S \unlhd r = \{x \mapsto y \mid x \mapsto y \in r \wedge x \notin S\}$ |
| $r \unrhd T$ | $r \unrhd T = \{x \mapsto y \mid x \mapsto y \in r \wedge x \notin T\}$ |

# 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

| | |
|---|---|
| $r[S]$ | $r[S] = \{y \mid \exists.x \in S \land x \mapsto y \in r\}$ |
| $p; q$ | $\forall p, q.p \in S \leftrightarrow T \land q \in T \leftrightarrow U \Rightarrow$ <br> $p; q = \{x \mapsto y \mid (\exists z.\ \ x \mapsto z \in p \land z \mapsto y \in q)\}$ |
| $p \Leftarrow q$ | $p \Leftarrow q = q \cup (dom(q) \lhd r)$ |
| $id(S)$ <br> $S \lhd id$ | $id(S) = \{x \mapsto x \mid x \in S\}$ <br> $id(S) = \{x \mapsto x \mid x \in S\}$ |

# Binary Relations

## Manipulation of binary relations

- Products and projection

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

## Axiomatisation

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

# 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 | $S \rightarrow\!\!\!\!\!+ T$ |
| Total Function | $S \rightarrow T$ |

## Axiomatisation. A Function is a Relation

| | | |
|---|---|---|
| $f \in S \rightarrow\!\!\!\!\!+ T$ | $\triangleq$ | $f \in S \leftrightarrow T \wedge f^{-1}; f = id(ran(f))$<br>$f \in S \leftrightarrow T \wedge f^{-1}; f \subseteq T \lhd id$ |
| $f \in S \rightarrow T$ | $\triangleq$ | $f \in S \rightarrow\!\!\!\!\!+ T \wedge S = dom(f)$ |

# 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

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

- Application of a Function

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

with $a \in S$

## Well definedness

- Let $f$ be a Partial Function, then

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

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

$$\boxed{a \in dom(f)}$$

# Logic notations

## Rewriting logic expressions

- Let us consider the predicate

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

- It can be successfully translated to

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

## Applying rewriting

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

| |
|---|
| $\forall x, y, z \cdot \; x \mapsto y \in f \land x \mapsto z \in f \Longrightarrow y = z$ |

# 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

# 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.
  - $\mathbf{BAA}(\mathbf{x}, \mathbf{x'})$ is a Before-After Predicate characterising the event *ev*. Example.

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

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

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

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

  The $\tau$ 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 $\mathbf{S(x)}$
- 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

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

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

- We write

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

# General notions on Systems

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

- 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 $\tau$ 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.

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

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

- We write

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

# General notions on Systems

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

- 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 $\mathbf{x_k}$ be a state in the trace. Initially $\mathbf{x_k} = \mathbf{x_i}$ (i.e. $\mathbf{k} = \mathbf{i}$.
  - Then, $\mathbf{k}$ increases to reach the suited state
  - When state $\mathbf{x_j}$ is reached, then $\mathbf{x_k} = \mathbf{x_j}$
  - Here, the sequence $\mathbf{j} - \mathbf{k}$ is a decreasing sequence
- This reasoning holds for any liveness property
- Again this proof is an induction. We shall show that
  - $\mathbf{j} - \mathbf{k}$ is a natural number
  - $\mathbf{j} - \mathbf{k}$ 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

# 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 $f := f \mathbin{<\!\!\!+} \{x \mapsto E\}$ |

- 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
  - $\mathbf{BAA(x, x')}$. Example $\mathbf{x' = x + 1}$ pour $\mathbf{x := x + 1}$ or for $\mathbf{x :| (x' = x + 1)}$
- Three types of events

| event_1 = |
|---|
| BEGIN |
| $x :\| BAA(x, x')$ |
| END |
| **Non guarded event** |

| event_2 = |
|---|
| WHEN |
| $G(x)$ |
| THEN |
| $x :\| BAA(x, x')$ |
| END |
| **Guarded Event** |

| event_3 = |
|---|
| ANY $l$ |
| WHERE $G$ |
| $G(l, x)$ |
| THEN |
| $x :\| BAA(x, x', l)$ |
| END |
| **Parameterised Event** |

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 $t$<br>where $G(t,x)$<br>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) | $\Gamma(s, c) \vdash I(x) \Rightarrow (\text{grd}(e_1) \vee \ldots \text{grd}(e_n))$ |
| (SAFE) | $\Gamma(s, c) \vdash I(x) \Rightarrow T(x)$ |
| (FIS) | $\Gamma(s, c) \vdash I(x) \wedge \text{grd}\,(E) \Rightarrow \exists x' \,.\, P(x, x')$ |

# 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

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

- 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*

$$A(s,c) \wedge I(s,c,x) \wedge G(s,c,l,x) \Rightarrow \exists x'.BAA(s,c,l,x,x')$$

- *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
```

$inv1 : sent \subseteq AGENTS \times AGENTS$

$inv2 : got \subseteq AGENTS \times AGENTS$

$inv4 : (got \cup lost) \subseteq sent$

$inv6 : lost \subseteq AGENTS \times AGENTS$

$inv7 : got \cap lost = \varnothing$

```
INITIALISATION
BEGIN
```

$act1 : sent := \varnothing$

$act2 : got := \varnothing$

$act4 : lost := \varnothing$

```
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 =
  any v where
    G(x,v)
  then
    x := E(x,v)
  end
```

```
EVENT =
  any w where
    H(y,w)
  then
    y := F(y,w)
  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 =
  SKIP
  end
```

```
EVENT =
  when
    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

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

**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
```

# Witnesses are helpful in refinement

## Back to parameterised events

The refinement of the left event by the right one

```
EVENT =
  any v where
    G(x,v)
  then
    x := E(x,v)
  end
```

```
EVENT =
  any w where
    H(y,w)
  then
    y := F(y,w)
  end
```

produces existential proof obligations.

## Example : Existential proof obligation for INV preservation

$$I(x) \ \land \ J(x,y) \ \land \ H(y,w) \quad \Longrightarrow \quad \exists v\,.\,(\,G(x,v) \ \land \ J(E(x,v),F(y,w))\,)$$

Their proof, usually consists in producing witnesses $W$ for the existentially quantified parameter using the following proof rule.

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

# Witnesses are helpful in refinement : Example

## Explicit witnesses in a refinement

A refinement offers the capability to give a witness (`WITH` Clause) in order to *help* the prover to discharge the proof obligation.

## Example

```
EVENT  =
   any  n  where
      n ∈ NAT
   then
      x := n
   end
```

```
EVENT  =
   WITH
      n : n = 4
   then
      x := 4
   end
```

The witness helps to prove the existential proof obligation.

# Complete representation of Event-B components

| **Context** | **Machine** | **Refinement** |
|---|---|---|
| CONTEXT Ctx | MACHINE $M^A$ | MACHINE $M^C$ |
| SETS $s$ | SEES Ctx | REFINES $M^A$ |
| CONSTANTS $c$ | VARIABLES $x^A$ | VARIABLES $x^C$ |
| AXIOMS $A$ | INVARIANTS $I^A(x^A)$ | INVARIANTS $J(x^A, x^C)$ |
| THEOREMS $T_{ctx}$ | THEOREMS $T_{mch}(x^A)$ | ... |
| END | VARIANT $V(x^A)$ | EVENTS |
| | EVENTS | EVENT $evt^C$ |
| | EVENT $evt^A$ | REFINES $evt^A$ |
| | ANY $\alpha^A$ | ANY $\alpha^C$ |
| | WHERE $G^A(x^A, \alpha^A)$ | WHERE $G^C(x^C, \alpha^C)$ |
| | THEN | WITH |
| | $x^A :\mid BAP^A(\alpha^A, x^A, x^{A'})$ | $x^{A\prime}, \alpha^A$: $W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C'})$ |
| | END | THEN |
| | ... | $x^C :\mid BAP^C(\alpha^C, x^C, x^{C'})$ |
| | | END |
| | | ... |
| (a) | (b) | (c) |

TABLE – Event B : Context, Machine and Refinement components

Note the presence of the `WITH` clause introducing witnesses using a before-after predicate for the variables and the parameters.

# Refinement Proof Obligations

| | | |
|---|---|---|
| (6) | Initialisation<br>(INIT) | $A \wedge G^C(\alpha^C)$<br>$\wedge W(\alpha^A, \alpha^C, x^{A\prime}, x^{C\prime})$<br>$\wedge BAP^C(\alpha^C, x^{C'}) \Rightarrow J(x^{A'}, x^{C'})$ |
| (7) | Invariant<br>preservation<br>(INV) | $A \wedge G^C(x^C, \alpha^C)$<br>$\wedge W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C\prime})$<br>$\wedge BAP^C(x^C, \alpha^C, x^{C'}) \wedge$<br>$I^A(x^A) \wedge J(x^A, x^C) \Rightarrow J(x^{A'}, x^{C'})$ |
| (8) | Event<br>Simulation<br>(SIM) | $A \wedge G^C(x^C, \alpha^C)$<br>$\wedge I^A(x^A) \wedge J(x^A, x^C)$<br>$\wedge W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C\prime})$<br>$\wedge BAP^C(x^C, \alpha^C, x^{C'})$<br>$\Rightarrow BAP^A(x_A, \alpha^A, x^{A'})$ |
| (9) | Guard<br>Strengthening<br>(GS) | $A \wedge I^A(x^A) \wedge J(x^A, x^C)$<br>$\wedge W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C\prime})$<br>$\wedge G^C(x^C, \alpha^C) \Rightarrow G_A(x^A, \alpha^A)$ |

TABLE – Refinement Proof obligations

# 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
  $ALL\_ORDERS \neq \varnothing$
variables
  $orders, stock, invoiced\_orders, reference, quantity$
invariant
  $orders \subseteq ALL\_ORDERS \;\wedge$
  $stock \in PRODUCTS \longrightarrow \mathbb{N} \;\wedge$
  $invoiced\_orders \subseteq orders \;\wedge$
  $quantity \in orders \longrightarrow \mathbb{N}^* \;\wedge$
  $reference \in orders \longrightarrow PRODUCTS$
initialisation
  $stock, invoiced\_orders, orders, quantity, reference := PRODUCTS \times \{0\}, \varnothing, \varnothing, \varnothing, \varnothing$
events
...
END

# Refinement in Event-B. An example

```
invoice_order =
  ANY
    o
  WHERE
```
$$o \in orders - invoiced\_orders \quad \wedge$$
$$quantity(o) \leq stock(reference(o))$$
```
  THEN
```
$$invoiced\_orders := invoiced\_orders \cup \{o\}$$
$$stock(reference(o)) := stock(reference(o)) - quantity(o)$$
```
  END;
```

```
delivery_to_stock =
  BEGIN
```
$$stock \ :| \ (stock' \ \in \ PRODUCTS \longrightarrow \mathbb{N})$$
```
  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

# Proof in logic

Proof strategy for a sequent $S$

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

While $K$ is not empty

- **CHOOSE** an inference rule $\frac{A}{C}\mathbf{r}$ in $\mathcal{T}$ such that its conclusion $C$ is in $K$
- **SUBSTITUTE** $C$ in $K$ by the hypotheses $A$ (if there are)

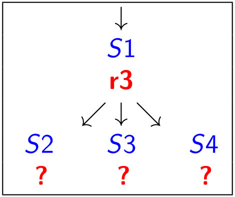
End

- 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 $\dfrac{S_2, S_3, S_4}{S_1}$ **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

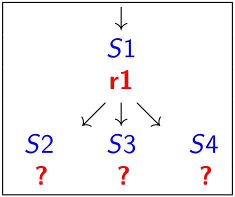
Let us consider the following inference rules

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

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

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

Our objective is to prove the sequent $S_1$



# The Proof Tree. An example

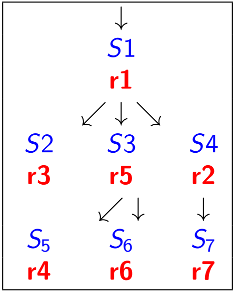
Let us consider the following inference rules

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

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

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

Our objective is to prove the sequent $S_1$



# Plan

# Proof in logics

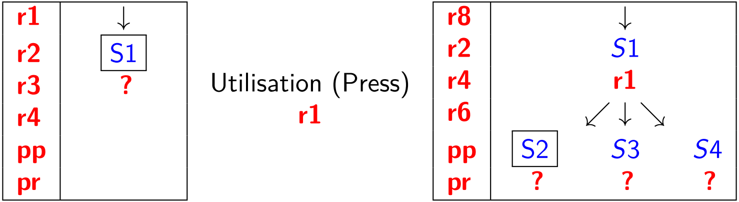
A proof in sequent calculus is a tree.

- L'application of inference rules $\rightsquigarrow$ a proof tree.
- Two types of reasoning
  - Forward reasoning.

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

    Bottom-Up Application of inference rules $\dfrac{A}{C}\uparrow$
- 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.

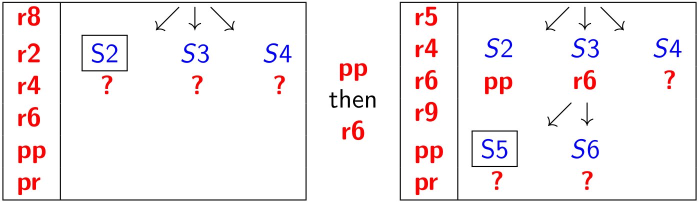
# Prover Interface : A Tree and a Palette of buttons the inference rules




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)

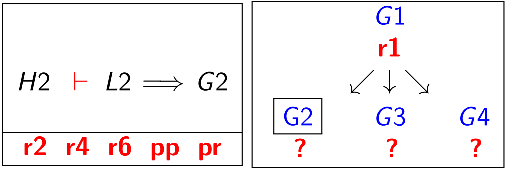
General form of a sequent for the proof of invariant

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

# 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

$H2 \vdash L2 \Longrightarrow G2$

**r2 r4 r6 pp pr**

$G1$
**r1**
↙ ↓ ↘
G2 | $G3$ | $G4$
**?** | **?** | **?**

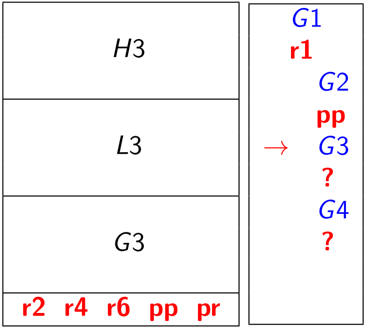
Use of (Press) **pp**

$H3 \vdash L3 \Longrightarrow G3$

**r2 r5 r8 pp pr**

$G1$
**r1**
↙ ↓ ↘
$G2$ | G3 | $G4$
**pp** | **?** | **?**

# More realistic windows



# More elaborated sequents

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

$$\begin{cases} hidden & \text{Non visible in the sequent window} \\ searched & \text{Visible after search in } hidden \\ cached & \text{Visible but is no more part of the conclusion} \\ local & \text{Visible and part of the conclusion} \end{cases}$$

# Plan

# The Rodin Platform

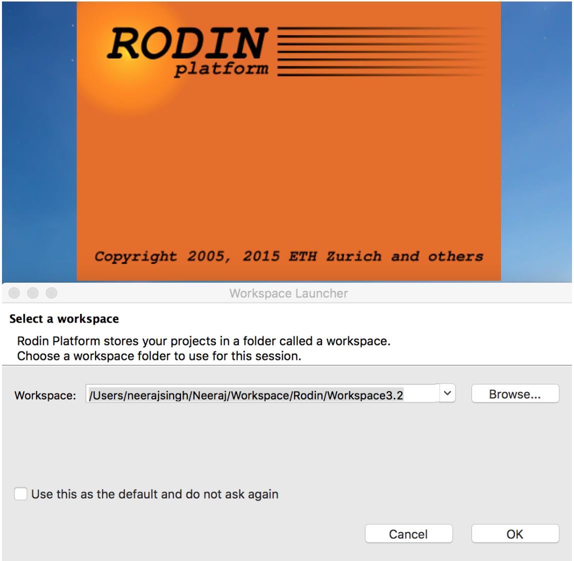
It is an application developed on Eclipse for the management and development of system models supporting verification of system model correctness.

Download `http://www.event-b.org`

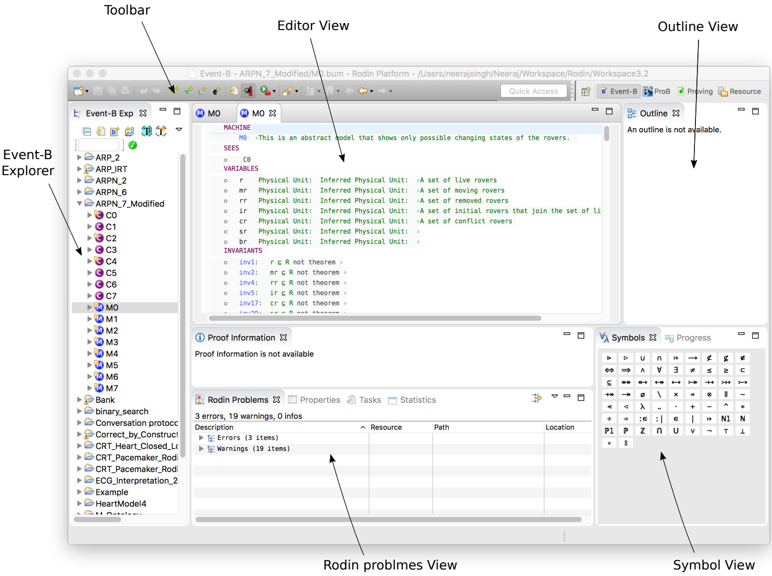
## Rodin Platform and Plug-in Installation

| Name | Installation |
| --- | --- |
| Rodin platform | • Requires Java 1.6<br>• 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.<br>• Start Rodin<br>• **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.<br>• **Details on plug-ins.**<br>• Install the Atelier B Provers plugin from the Atelier B Provers Update site to take full advantage of Rodin proof capabilities<br>• 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

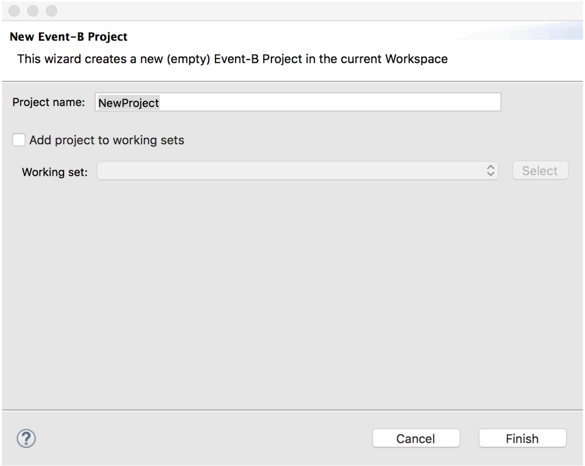


# The Rodin Platform. The Rodin interface (GUI)



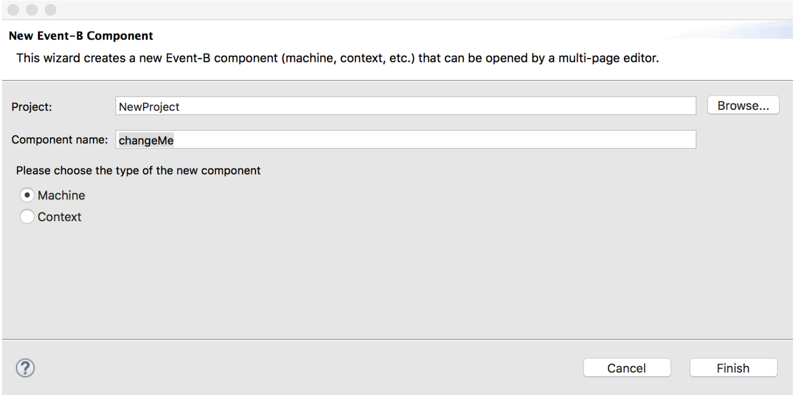
# The Rodin Platform. Creating a new Project)

File > New > Event-B Project

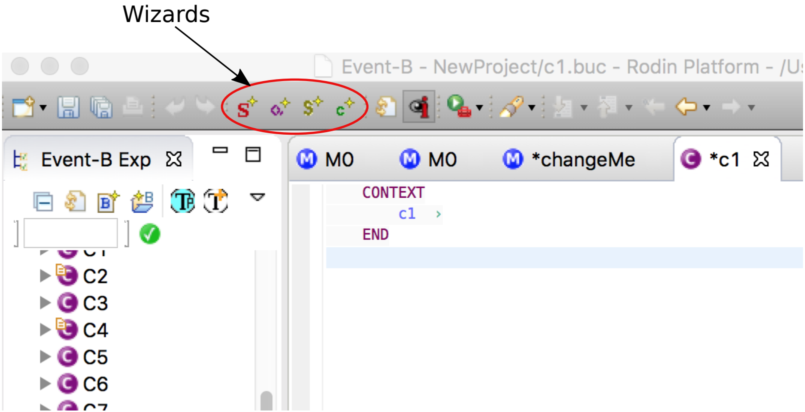


# The Rodin Platform. Creating Event-B Components

File > New > Event-B Components



# The Rodin Platform. Construction of contexts (Context component)



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

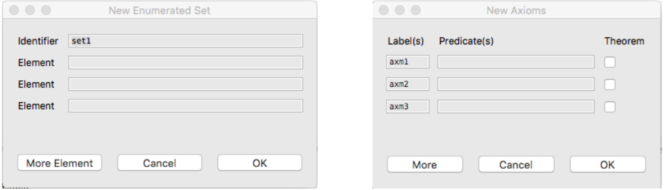



FIGURE – New sets (Enumerated Set) and new axioms (Axioms)

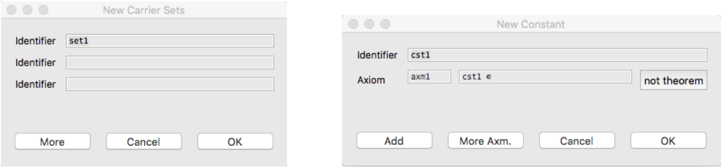
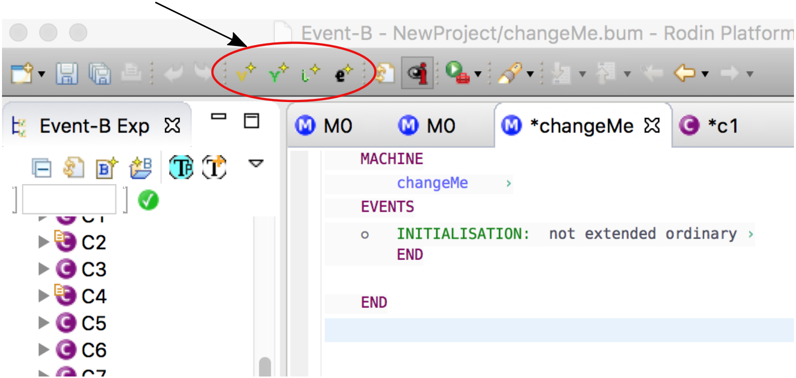



FIGURE – New sets (New Carrier Sets) and constants definitions (Constants)

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

# The Rodin Platform. Building Machines

Wizards



# The Rodin Platform. Building Machines with Wizards

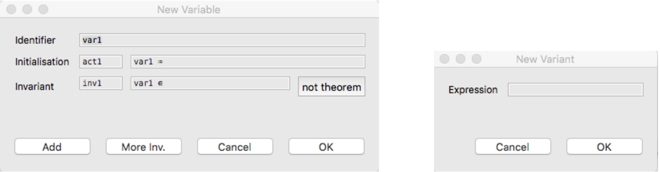



FIGURE – New Variables, Invariants and Variants

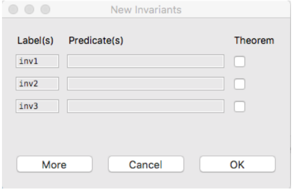



FIGURE – Adding Invariants

# The Rodin Platform. New events (Events)

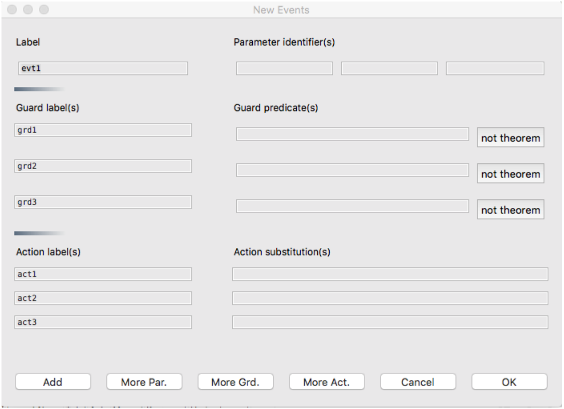
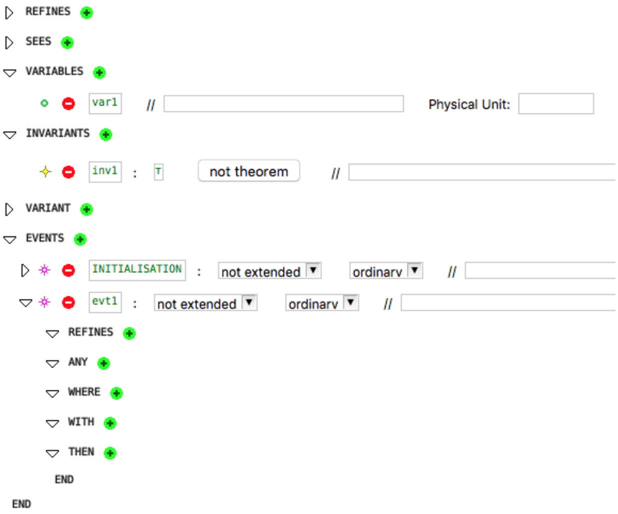


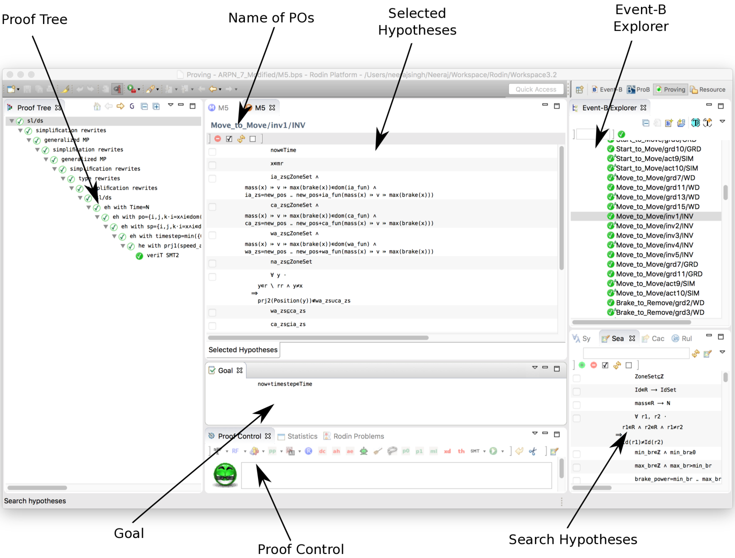

FIGURE – New Events)

# The Rodin Platform. Rodin Editor for the Machine component



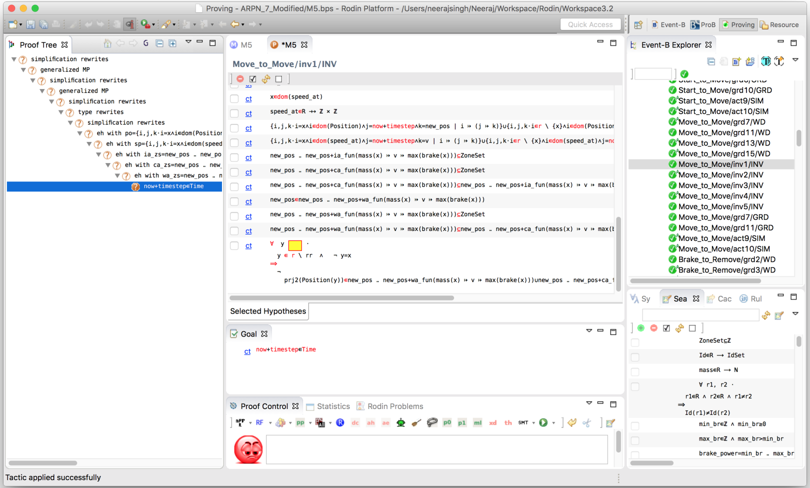
# The RODIN Prover

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

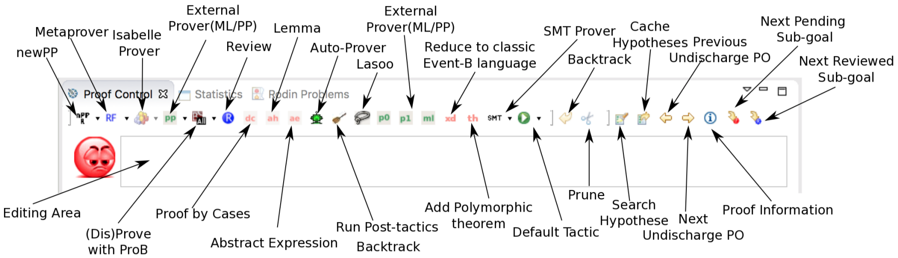


# The Rodin Platform. Interface (GUI) of the Rodin Prover

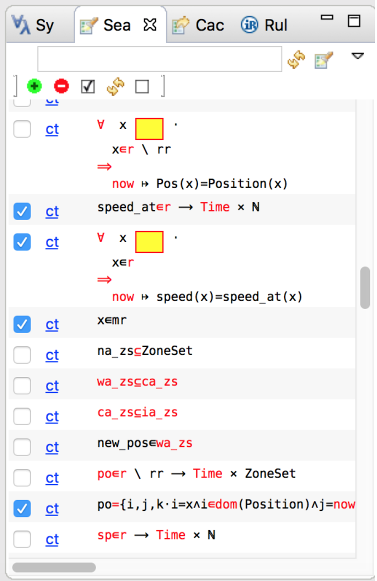
## Interface (GUI) for an unsuccessful proof



# 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

Proof Information

```
      act2: mr ≔ mr \ {x}
     END
• Event in M1
    Moves_to_Stop_by_Controller:
     REFINES
      Moves_to_Stop_by_Controller
     ANY
      x, ia_zs, wa_zs, ca_zs, na_zs, new_pos
     WHERE
      grd1: x ∈ mr
      grd2: ia_zs ⊆ ZoneSet
      grd3: ca_zs ⊆ ZoneSet
      grd4: wa_zs ⊆ ZoneSet
      grd5: na_zs ⊆ ZoneSet
      grd6: ∃y· y∈r\rr ∧ y≠x ∧ Pos(y) ∈ (wa_zs ∪ ca_zs)
      grd7: wa_zs ⊆ ca_zs
      grd8: ca_zs ⊆ ia_zs
      grd9: new_pos ∈ wa_zs
     THEN
      act1: sr ≔ sr ∪ {x}
      act2: mr ≔ mr \ {x}
      act3: ia ≔ ia  {i↦j|j∈ia_zs ∧ i=x}
      act4: ca ≔ ca  {i↦j|j∈ca_zs ∧ i=x}
      act5: wa ≔ wa  {i↦j|j∈wa_zs ∧ i=x}
      act6: na ≔ na  {i↦j|j∈na_zs ∧ i=x}
      act7: Pos(x) ≔ new_pos
     END
• Invariant in M1
    inv7: ∀x· x∈r ∧ x ∉ rr ⇒ ran({x} ◁ wa) ⊆ ran({x} ◁ ca)
```

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

Symbols | Type Env | Rule Det

| Identifier | Type |
|---|---|
| Pos | ℙ(R×ℤ) |
| Pos' | ℙ(R×ℤ) |
| R | ℙ(R) |
| ZoneSet | ℙ(ℤ) |
| br | ℙ(R) |
| ca | ℙ(R×ℤ) |
| ca' | ℙ(R×ℤ) |
| ca_zs | ℙ(ℤ) |
| cr | ℙ(R) |
| ia | ℙ(R×ℤ) |
| ia' | ℙ(R×ℤ) |
| ia_zs | ℙ(ℤ) |
| ir | ℙ(R) |
| mr | ℙ(R) |
| mr' | ℙ(R) |
| na | ℙ(R×ℤ) |
| na' | ℙ(R×ℤ) |
| na_zs | ℙ(ℤ) |
| new_pos | ℤ |
| r | ℙ(R) |
| rr | ℙ(R) |

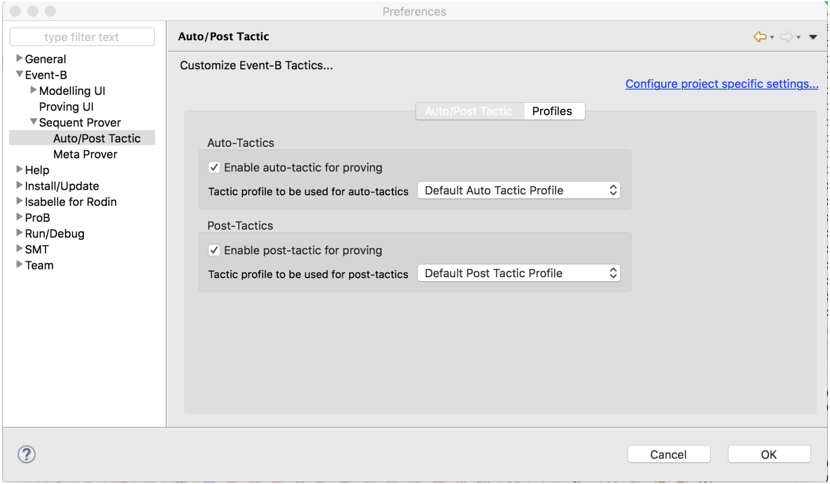
# The Rodin Platform. Interface (GUI) of the Rodin Prover. The used inference rules perspective

Symbols | Type Environment | **Rule Details**

```
Rule: PP

Input Sequent:
     ¬x0∈rr
     x∈mr
     ∀x·x∈r∧¬x∈rr⇒ran({x} ◁ wa)⊆ran({x} ◁ ca)
     ca_zs⊆ZoneSet
     ca_zs⊆ia_zs
     wa_zs⊆ca_zs
     new_pos∈wa_zs
     ∃y·y∈r ∖ rr∧¬y=x∧Pos(y)∈wa_zs∪ca_zs
     ia_zs⊆ZoneSet
     na_zs⊆ZoneSet
     x0∈r
     wa_zs⊆ZoneSet
  ⊢ ran({x0} ◁ (wa⩥{i,j·j∈wa_zs∧i=x | i ↦ j}))⊆ran({x0} ◁ (ca⩥{i,j·j∈ca_zs∧i=x | i ↦ j})
```

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



# Plan

# 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

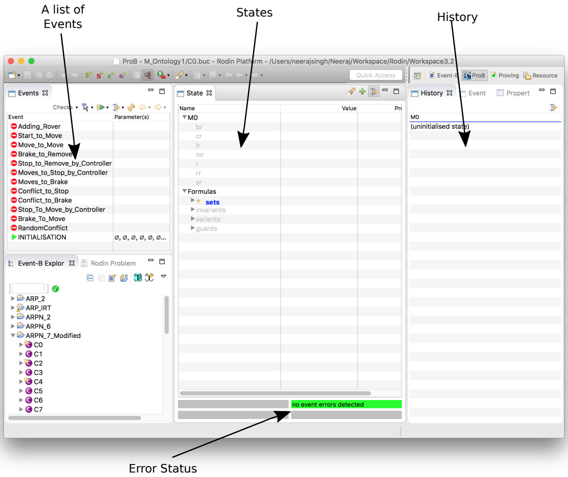
- Exhaustive verification if the models are finite
- Assistance to the design of Event-B models by identifying counter-examples

# The ProB model-checker

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

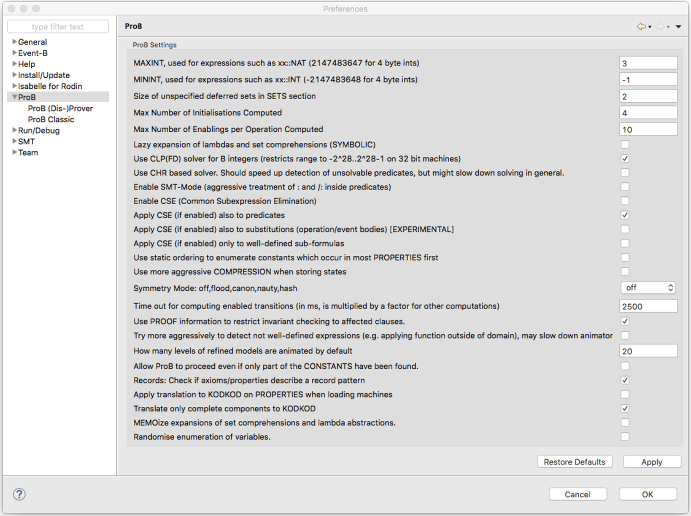
A useful tool for analysing and debugging Event-B models.

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



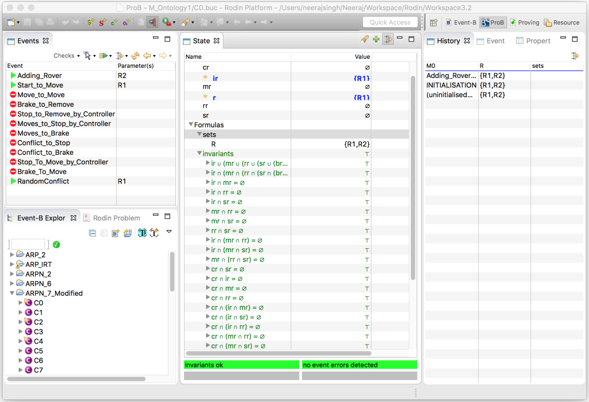
# Animation of Event-B models. Model checking with ProB ?

Rodin Platform > Preference > ProB



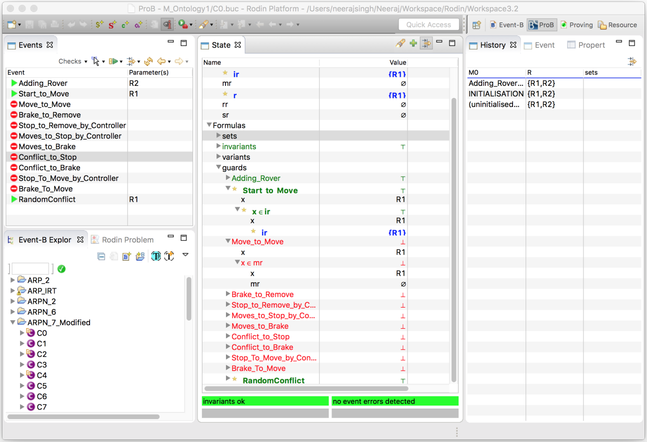
# Animation of Event-B models. Model checking with ProB ? Verification of Invariants

- Enable Events
- Check Invariants



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

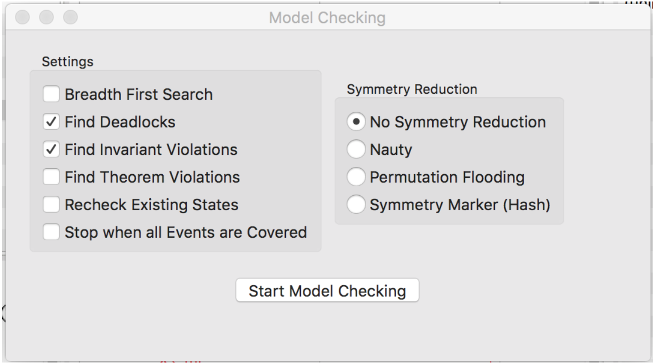
- Guards Checking



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

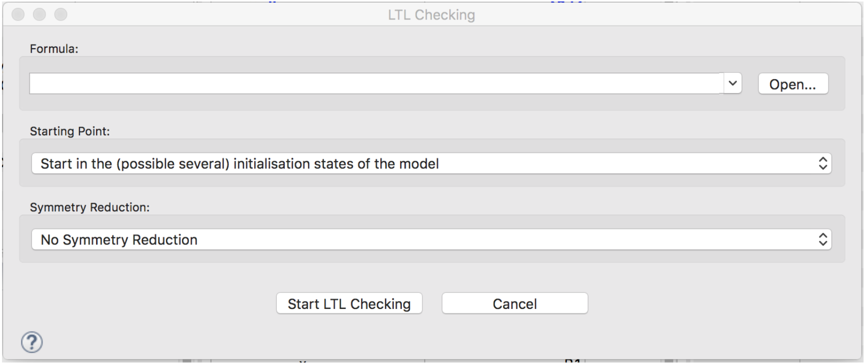
- Deadlock Freedom

Checks > Model Checking



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

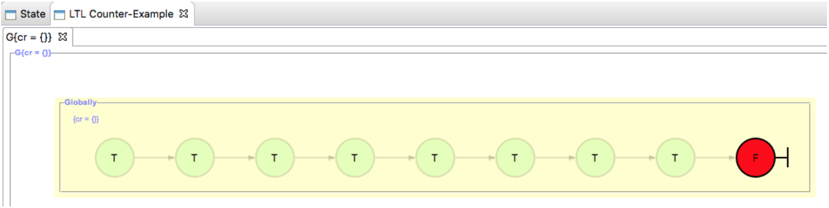
Checks > LTL Model Checking



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

The following formula can be verified.

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

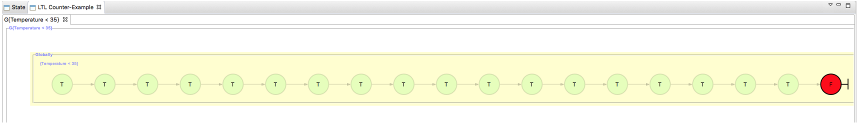



Adding_Rover → Adding_Rover → Start_to_Move → Start_to_Move → Move_to_Stop_by_Controller → Stop_to_Remove_by_Controller → Move_to_Stop_by_Controller → Random_Conflict

# 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