# Summary of Event-B Modeling Notation

Jean-Raymond Abrial
(edited by Thai Son Hoang)

Department of Computer Science
Swiss Federal Institute of Technology Zürich (ETH Zürich)

Bucharest DEPLOY 2-day Course, 14th-16th July, 2010

SEVENTH FRAMEWORK PROGRAMME

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Purpose of this Presentation

- Showing the structure of the Event-B modeling notation
- Machines, contexts, and events
- Presenting a small example

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Model Developments with Event-B

- Event-B is not a programming language (even very abstract)
- Event-B is a notation used for developing mathematical models of discrete transition systems
- Event-B is to be used together with the Rodin Platform

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Model Developments with Event-B (cont'd)

- Such models, once finished, can be used to eventually construct:
  - sequential programs,
  - distributed programs,
  - concurrent programs,
  - electronic circuits,
  - large systems involving a possibly fragile environment,
  - etc.
- The underlined statement is an important case.
- In this presentation, we shall construct a small sequential program.

# Machines and Contexts

- A model is made of several components
- A component is either a machine or a context:

| Machine | Context |
|---|---|
| variables | sets |
| invariants | constants |
| events | axioms |
| variant | |

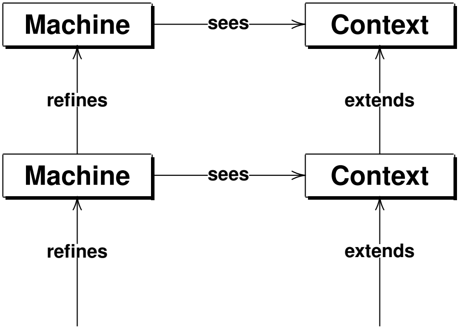
- Machines and contexts have names
- Such names must be distinct in a given model

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Machines and Contexts (cont'd)

- Contexts contain the static structure of a discrete system (constants and axioms)
- Machines contain the dynamic structure of a discrete system (variables, invariants, and events)
- Machines see contexts
- Contexts can be extended
- Machines can be refined

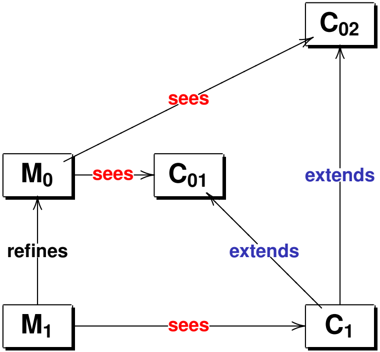
# Relationship Between Machines and Contexts



# Visibility Rules (can be Skipped at First Reading)

- A machine can see several contexts (or no context at all).
- A context may extend several contexts (or no context at all).
- A machine implicitly sees all contexts extended by a seen context.
- A machine only sees a context either explicitly or implicitly.
- A machine only refines at most one other machine.
- No cycle in the “refines” or “extends” relationships.

# Example (can be Skipped at First Reading)




- $\mathbf{M_0}$ sees $\mathbf{C_{01}}$ and $\mathbf{C_{02}}$ explicitly.
- $\mathbf{M_1}$ sees $\mathbf{C_1}$ explicitly.
- $\mathbf{M_1}$ sees $\mathbf{C_{01}}$ and $\mathbf{C_{02}}$ implicitly.

# Context Structure

**context**
$< context\_identifier >$
**extends** $\star$
$< context\_identifier >$
$\dots$
**sets** $\star$
$< set\_identifier >$
$\dots$
**constants** $\star$
$< constant\_identifier >$
$\dots$
**axioms** $\star$
$< label >: \ < predicate >$
$\dots$
**end**

- Sections with “$\star$” might be empty
- All keyword sections are predefined in the Rodin Platform
- All labels are generated automatically by the Rodin Platform (but can be modified)

# Explaining Context Sections

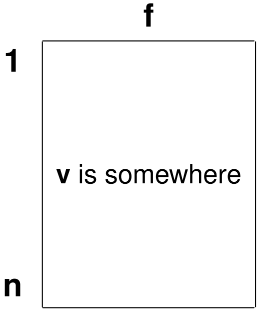
- "sets" lists various carrier sets, which define pairwise disjoint types
- The only property we can assume about a set is that it is not empty
- "constants" lists the different constants introduced in the context
- "axioms" defines the main properties of the constants
- axioms can be marked as "theorems" denotes derived properties (to be proved) from previously declared the axioms.

# Context Example

**context**
  **ctx_0**
**sets**
  $D$
**constants**
  $n$
  $f$
  $v$
**axioms**
  **axm1** : $n \in \mathbb{N}$
  **axm2** : $f \in 1..n \rightarrow D$
  **axm3** : $v \in \mathrm{ran}(f)$
  **thm1** : $n \in \mathbb{N}_1$
**end**

- A set $D$ is defined in context **ctx_0**
- Moreover, three constants, $n$, $f$, and $v$, are defined in this context:
  - $n$ is a natural number (**axm1**)
  - $f$ is a total function from the interval $1 \,..\, n$ to the set $D$ (**axm2**)
  - $v$ is supposed to belong to the range of $f$ (**axm3**)
- A theorem is proposed: $n$ is a positive number (**thm1**)

# Pictorial Representation of the Context



# Machine Structure

**machine**
  $<$ *machine_identifier* $>$
**refines** $\star$
  $<$ *machine_identifier* $>$
**sees** $\star$
  $<$ *context_identifier* $>$
  $\ldots$
**variables**
  $<$ *variable_identifier* $>$
  $\ldots$
**invariants**
  $<$ *label* $>$: $<$ *predicate* $>$
  $\ldots$
**events**
  **initialisation** $\ldots$
  $\ldots$
**variant** $\star$
  $<$ *variant* $>$
**end**

- Each machine has exactly one **initialisation** event
- All keyword sections are predefined in the Rodin Platform
- All labels are generated automatically by the Rodin Platform (but can be modified)

# Explaining Machine Sections

- "**variables**" lists the state variables of the machine
- "**invariants**" states the properties of the variables
- Invariants are defined in terms the seen sets and constants
- invariants can be marked as "**theorems**" which are derivable from previously declared invariants and seen axioms
- "**events**" defines the dynamics of the transition system (slide 17)
- "**variant**" is explained later (slide 29)

# Machine (and Context) Example

```
machine
   m_0a
sees
   ctx_0
variables
   i
invariants
   inv1 :  i ∈ 1 .. n
events
   . . .
end
```

```
context
   ctx_0
sets
   D
constants
   n
   f
   v
axioms
   axm1 : n ∈ ℕ
   axm2 : f ∈ 1..n → D
   axm3 : v ∈ ran(f)
   thm1 : n ∈ ℕ1
end
```

- Machine **m_0a** sees the previously defined context **ctx_0**
- A variable $i$ is defined
- $i$ is a member of the interval $1 \,..\, n$ (**inv1**)
- **events**: next slide

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Event Structure

$<$ *event_identifier* $>$ $\widehat{=}$
  **status**
    $\{$*ordinary*, *convergent*, *anticipated*$\}$
  **refines** $\star$
    $<$ *event_identifier* $>$
    ...
  **any** $\star$
    $<$ *parameter_identifier* $>$
    ...
  **where** $\star$
    $<$ *label* $>$: $<$ *predicate* $>$
    ...
  **with** $\star$
    $<$ *label* $>$: $<$ *witness* $>$
    ...
  **then** $\star$
    $<$ *label* $>$: $<$ *action* $>$
    ...
  **end**

- Notice that keyword "**where**" becomes "**when**" in the Rodin Platform Pretty Print when there is no "**any**".
- Notice that keyword "**then**" becomes "**begin**" in the Rodin Platoform Pretty Print when there are no "**any**" and no "**where/when**".
- Again, all keyword sections are predefined in the Rodin Platform.
- All labels are generated automatically by the Rodin Platform (but can be modified)

# Explaining Event Sections

- An event is a state transition in a discrete dynamic system.
- "**refines**" contains the name(s) of the refined event(s) (if any)
- Can be skipped at first reading:
  - Several refined events are possible in case of a merging refining event concentrating more than one refined event
  - Merged events must have the same actions

# Explaining Event Sections (cont'd)

- "**status**" is either:
  - ordinary,
  - convergent: it has to decrease the variant (slide 29),
  - anticipated: to be convergent later in a refinement.
- "**any**" contains the parameters of the event (might be empty)
- "**where**" (or "**when**") contains the various guards of the event
- A guard is a necessary condition for an event to be enabled
- Guards can be marked as "theorems" which are derivable from invariants, seen axioms and previously declared guards.
- "**actions**" see next slide

# Explaining Action Section

- An action describes the ways one or several state variables are modified by the occurrence of an event

- An action might be either deterministic or non-deterministic

# Deterministic Action (Example)

- Here is the form of some deterministic actions on variables $x$, $y$ and $z$:

$$\begin{aligned} x &:= x + y \\ y &:= y - x - z \end{aligned}$$

- Notice that $x$ and $y$ should be distinct.

- Actions are supposed to be “performed” in parallel

- Variables $x$ and $y$ are assigned to $x + y$ and $y - x - z$ respectively

- Variable $z$ is used but not modified by these actions

# First Form of Non-deterministic Action (Example)

$$x, y :\mid\ x' > x \ \wedge\ y' < x'$$

- On the LHS of operator $:\mid$, we have two distinct variables
- On the RHS, we have a, so-called, before-after predicate
- The RHS contains occurrences of $x$ and $y$ (before values) and primed occurrences $x'$ and $y'$ (after values)
- As a result (in this example):
  - $x$ is assigned a value greater than its previous value
  - $y$ is assigned a value smaller than that, $x'$, assigned to $x$

$$x :\in \{x+1, y-2, z+3\}$$

- Here $x$ is assigned any value from the set $\{x+1, y-2, z+3\}$

# The Most General Form of an Action

- The second form of non-deterministic action is equivalent to the following first form:

$$x :\mid \; x' \in \{x+1, y-2, z+3\}$$

- Likewise, a deterministic action has an equivalent non-deterministic form:

$$x, y :\mid \; x' = x + y \;\wedge\; y' = y - x - z$$

- The non-det. first form can thus always be assumed (by the tools)

# Event Examples of Machine **m_0a**

- This machine is the model specification of a searching program

**machine**
  **m_0a**
**sees**
  **ctx_0**
**variables**
  $i$
**invariants**
  **inv1** : $i \in 1 \,..\, n$
**events**
  $\ldots$
**end**

**initialisation** $\widehat{=}$
  **status**
    **ordinary**
  **begin**
    **act1** : $i := 1$
  **end**

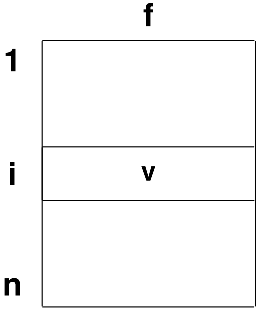
**search** $\widehat{=}$
  **status**
    **ordinary**
  **any**
    $k$
  **where**
    **grd1** : $k \in 1 \,..\, n$
    **grd2** : $f(k) = v$
  **then**
    **act1** : $i := k$
  **end**

- Event **search** assigns to $i$

- any value $k$ such that $f(k) = v$,

- provided $k$ is in interval $1 \,..\, n$

# Pictorial Representation of the State after “search”



# Another Machine m_0b

**machine**
  **m_0b**
**sees**
  **ctx_0**
**variables**
  $i$
**invariants**
  **inv1** : $i \in 1\,..\,n$
**events**
  $\ldots$
**end**

**initialisation** $\widehat{=}$
  **status**
    **ordinary**
  **begin**
    **act1** : $i := 1$
  **end**

**search** $\widehat{=}$
  **status**
    **ordinary**
  **begin**
    **act1** : $i :\mid i' \in 1\,..\,n \;\wedge\; f(i') = v$
  **end**

- The only difference between **m_0a** and **m_0b** is in event **search**
- $i$ is assigned non-deterministically a values $i'$ such that $i' \in 1\,..\,n$ and $f(i') = v$
- Notice that event **search** has no guard

# Explaining Event Sections (cont'd)

- "**with**" contains the witnesses of a refining event.

- A witness has to be provided in a refining event
  - for each disappearing parameter of the refined event (see m_1a)
  - after value of each disappearing variable.

- The witness for parameter $a$ is defined as follows $a : P(a)$ where $P(a)$ is a predicate involving $a$

- The witness for after value of variable $b$ is defined as follows $b' : P(b')$ where $P(b')$ is a predicate involving $b'$

- For a deterministic witness $P(x)$ is $x = E$ (with $E$ free of $x$)

# Variant

- The variant of a machine is either a natural number expression or a finite set expression
- It has to be present in any machine with convergent events
- A numeric variant must be decreased by all convergent events
- A set variant must be made strictly included in its previous value by all convergent events

# Refinement Machine m_1a Refining Machine m_0a

**machine**
  **m_1a**
**refines**
  **m_0a**
**sees**
  **ctx_0**
**variables**
  $i$
  $j$
**invariants**
  **inv1** : $j \in 0 \,..\, n-1$
  **inv2** : $v \notin f[1 \,..\, j]$
  **thm1** : $v \in f[j+1 \,..\, n]$
**variant**
  $n - j$
**events**
  ...
**end**

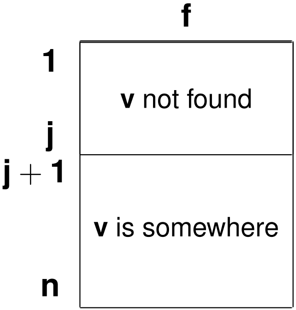
**initialisation** $\widehat{=}$
  **status** **ordinary**
  **begin**
    **act1** : $i := 1$
    **act2** : $j := 0$
  **end**

**search** $\widehat{=}$
  **status** **ordinary**
  **refines**
    **search**
  **when**
    **grd1** : $f(j+1) = v$
  **with**
    $k$ : $j+1 = k$
  **then**
    **act1** : $i := j+1$
  **end**

**progress** $\widehat{=}$
  **status** **convergent**
  **when**
    **grd1** : $f(j+1) \neq v$
  **then**
    **act1** : $j := j+1$
  **end**

- A new variable $j$ is introduced
- Notice invariant **inv2** and theorem **thm1**
- Notice the **with** section in event **search**
- A new **convergent** event **progress** is introduced
- Notice the numeric **variant** $n - j$

# Pictorial Representation of the State



# Refinement Machine m_1b Refining Machine m_0b

**machine**
  **m_1b**
**refines**
  **m_0b**
**sees**
  **ctx_0**
**variables**
  $i$
  $j$
**invariants**
  **inv1** : $j \in 0 \,..\, n-1$
  **inv2** : $v \notin f[i \,..\, j]$
  **thm1** : $v \in f[j+1 \,..\, n]$
**variant**
  $j \,..\, n$
**events**
  ...
**end**

**initialisation** $\widehat{=}$
  **status** **ordinary**
  **begin**
    **act1** : $i := 1$
    **act2** : $j := 0$
  **end**

**search** $\widehat{=}$
  **status** **ordinary**
  **refines**
    **search**
  **when**
    **grd1** : $f(j+1) = v$
  **then**
    **act1** : $i := j+1$
  **end**

**progress** $\widehat{=}$
  **status** **convergent**
  **when**
    **grd1** : $f(j+1) \neq v$
  **then**
    **act1** : $j := j+1$
  **end**

- The **with** section in event **search** is not needed
- Notice the finite set **variant** $j \,..\, n$
- These are the only differences with refining machine **m_1a**

# Constructing the Final Program

- A sequential program can be constructed from **m_1a** (or **m_1b**)
- This is done by applying a number of event merging rules (NOT DEFINED HERE)
- The application of these rules yields the following program:

| | |
|---|---|
| $i, j := 1, 0$ ; | **initialisation** |
| **while** $f(j+1) \neq v$ **do** | |
| $\quad j := j + 1$ | **progress** |
| **end** ; | |
| $i := j + 1$ | **search** |

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Exercise

- Modify refinement **m_1a** (or **m_1b**) in order to obtain the following final program from the same specification **m_0a** (or **m_0b**):

| | |
|---|---|
| $i, j := 1, n+1$ ; | **initialisation** |
| **while** $f(j-1) \neq v$ **do** | |
| $\quad j := j - 1$ | **progress** |
| **end** ; | |
| $i := j - 1$ | **search** |