A Light Introduction to Event-B
(With Small Examples)

Dang Van Hung
Based on Jean-Raymond Abrial’s Istanbul Lectures

College of Technology
Model Development with Event-B

- Event-B is not a programming language (even very abstract)

- Event-B is a notation used for developing mathematical models

- Mathematical models of discrete transition systems

- http://www.event-b.org
Such models once finished, can be used to eventually construct:

- sequential programs
- distributed programs
- concurrent programs
- electronic circus
- large systems involving a possibly fragile environment
- etc.
Main Influences

Action Systems developed by the Finnish school (Turku):

- R.J.R. Back and R. Kurki-Suonio, Decentralization of Process Nets with Centralized Control, 2nd ACM SIGACT-SIGOPTS Symposium

Example: A UNITY Program

Program bubblesort
declare
  n: integer,
  A: array [0..n-1] of integer
initially
  n = 20 #
  <# i : 0 <= i and i < n ::
  A[i] = rand() % 100 >
assign
  <# k : 0 <= k < 2 ::
  <|| i : i % 2 = k and 0 <= i < n-1::
  if A[i] > A[i+1] > >
end
The State of a Model

- A discrete model is first made of a state
- The state is represented by some constants and variables
- Constants are linked by some axioms
- Variables are linked by some invariants
- Axioms and invariants are written using set-theoretic expressions
A discrete model is also made of a number of events.

An event is made of a guard and an action.

The guard denotes the enabling condition of the event.

The action denotes the way the state is modified by the event.

Guards and actions are written using set-theoretic expressions.
A Model Schematic View

Variables
- invariants

Events
- guards
- actions

Constants
- axioms

Dynamic Part
(Machines)

Static Part
(Contexts)
Operational Interpretation

- An event execution takes no time
- No two events can occur simultaneously
- When all events have true guards, one of them is chosen non-deterministically and its action modify the state
- The previous phase is repeated (if possible)
Operational Interpretation

Initialize;
while (some events have true guards){
    Choose one such event;
    Modify the state accordingly;
}
Comments: Operational Interpretation

- Stopping is not necessary: a discrete system may run for ever
- This interpretation is just given here for informal understanding
- The meaning of such a discrete system will be given by the proofs which can be performed on it.
Machines and Contexts

Being More Precise:

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

<table>
<thead>
<tr>
<th>Machine</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables</td>
<td>carrier sets</td>
</tr>
<tr>
<td>invariants</td>
<td>constants</td>
</tr>
<tr>
<td>theorems</td>
<td>axioms</td>
</tr>
<tr>
<td>events</td>
<td>theorems</td>
</tr>
</tbody>
</table>

DSE-Seminar, Friday, Jan 2, 2009 – p.12
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: Machines & Contexts

Machine \(\text{sees}\) Context

Machine \(\text{refines}\) Machine

Context \(\text{extends}\) Context
First Simple Example
We are given a non-empty finite array of natural numbers

We like to find the maximum of the range of this array
Requirements

**FUN-1**

We are given a non-empty finite array of natural numbers

**FUN-2**

We like to find the maximum of the range of this array

We want to find that 10 is the greatest element of this array

```
9 3 10 8 3 5
```
Development Strategy

- First, we show an initial model specifying the problem
- Later, we refine our model to an algorithm
- In the initial model we have:
  - a context where the constant array is defined
  - a machine where the maximum is “computed”
Initial Model: The Context

- Constant $n$ denotes the size of the non-empty array
- Constant $f$ denotes the array
- Constant $M$ denotes a natural number

<table>
<thead>
<tr>
<th>constants</th>
<th>$n$</th>
<th>0 &lt; $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$f$</td>
<td>$f \in 1..n \rightarrow 0..M$</td>
</tr>
<tr>
<td></td>
<td>$M$</td>
<td>$\text{ran}(f) \neq \emptyset$</td>
</tr>
</tbody>
</table>

- Mind the reference typing
Initial Model: The Context

- Constant $n$ denotes the size of the non-empty array
- Constant $f$ denotes the array
- Constant $M$ denotes a natural number

<table>
<thead>
<tr>
<th>constants</th>
<th>$n$</th>
<th>axm0_1 : $0 &lt; n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$f$</td>
<td>axm0_2 : $f \in 1..n \rightarrow 0..M$</td>
</tr>
<tr>
<td></td>
<td>$M$</td>
<td>thm0_1 : ran($f$) $\neq \emptyset$</td>
</tr>
</tbody>
</table>

Mind the reference typing
The Context

context
maxi.ctx.0
constants
n
f
M
axioms
axm1: 0 < n
axm2: f ∈ 1..n → 0..M
theorems
thm1: ran(f) ≠ ∅
end
Context Structure

context
  < context_identifier >

set
  < set_identifier >
  ...

constants
  < constant_identifier >
  ...

axioms
  < label >:< predicate >
  ...

theorems
  < label >:< predicate >
  ...

end
Explaining Context Sections

- “sets” lists various sets, which define pairwise disjoint types
- “constants” lists the different constants introduced in the context
- “axioms” defines the properties of the constants
- “theorems” denotes properties to be proved from the axioms
Initial Model: The Machine

Variable $m$ denotes the results:

- **variable:** $m$
- **inv0_1:** $m \in \mathbb{N}$

Next are the two events:

- **INIT**
  - ```
    begin
    m := 0
    end
  ```

- **maximum**
  - ```
    begin
    m := \text{max}(\text{ran}(f))
    end
  ```

Event maximum presents the final intended result (in one shot)
Machine Example

Machine
variables
invariants
theorems
events

```plaintext
machine maxi.0 sees maxi.ctx.0 variables i invariants inv1 : i ∈ 1..n events . . . end
```
Machine and Context

machine maxi.0
sees maxi.ctx.0
variables
  \( m \)
invariants
  inv1 : \( m \in \mathbb{N} \)
events
  \ldots
end

context maxi.ctx.0
constants
  \( n \)
  \( f \)
  \( M \)
axioms
  axm1: 0 < n
  axm2: \( f \in 1..n \rightarrow 0..M \)
theorems
  thm1: \text{ran}(f) \neq \emptyset
end
Machine Structure

```
machine
  < machine_identifier >
sees
  < context_identifier >
variables
  < variable_identifier >
  ...
invariants
  < label >::< predicate >
theorems
  < label >::< predicate >
events
  ...
end
```
Explaining Machine Sections

- “variables” lists the state variables of the machine
- “invariants” states the properties of the variables
- “theorems” are provable from invariants and seen axioms and thms
- “events” defines dynamics of the transition system
Event

- An event defines a transition of our discrete system

- An event is made of a Guard $G$ and an Action $A$

- $G$ defines the enabling conditions of the transition

- $A$ defines parallel assignment of the variables
Kinds of Events

- **no guard**
  - begin
  - $A$
  - end

- **Simple guard**
  - begin
  - when $G$
  - then $A$
  - end

- **Quantified guard**
  - any $x$ where $G(x)$
  - then $A(x)$
  - end
Kinds of Events

<table>
<thead>
<tr>
<th>no guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin $A$</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

Simple guard

<table>
<thead>
<tr>
<th>no guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin $G$</td>
</tr>
<tr>
<td>then $A$</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

Quantified guard

<table>
<thead>
<tr>
<th>any $x$ where $G(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>then $A(x)$</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

Our event (so far) has no guards

INIT

<table>
<thead>
<tr>
<th>begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m := 0$</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

maximum

<table>
<thead>
<tr>
<th>begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m := \max(\text{ran}(f))$</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>
**Summary**

**constants:**
- \( n \)
- \( f \)
- \( M \)

**variable:** \( m \)

**INIT**
- `begin`
- \( m := 0 \)
- `end`

**axm0.1:** \( 0 < n \)

**axm0.2:** \( f \in 1..n \rightarrow 0..M \)

**thm0.1:** \( \text{ran}(f) \neq \emptyset \)

**inv0.1:** \( m \in \mathbb{N} \)

**maximum**
- `begin`
- \( m := \max(\text{ran}(f)) \)
- `end`
Our Task is not Completed

We have to perform some proofs:

- **thm0.1** holds
- invariant **inv0.1** is established by event “INIT”
- invariant **inv0.1** is maintained by event “maximum”
- Expression “\( \text{max}(\text{ran}(f)) \)” is well-defined
Summary of what is to be Proved

- Stated theorems
- Invariant maintenance
- Well-definedness
The Rodin Platform Kernel Tools

Model

Static Checker -> Proof Obligation Generator -> Prover

Proofs
Automatic and Interactive Modes

Proof Obligations ➔ Prover ➔ Proof

Proof Obligations ➔ Prover ➔ Proof

Human Intervention
Refinement: the New Variables

We introduce two new variables in our model:

Variables $p$ and $q$ denote two indices in the domain of $f$.

- $p \in 1..n$
- $q \in 1..n$
The maximum is always in “between” $p$ and $q$. 

\[ \begin{array}{cccccc} 9 & 3 & 10 & 8 & 3 & 5 \\ \end{array} \]
The maximum is always in “between” $p$ and $q$. 
The maximum is always in “between” $p$ and $q$.
Interval $p..q$ is never empty (inv1.3)

The maximum is always in the image of $p$ and $q$ under $f$ (inv1.4)

**inv1.1:** $p \in 1..n$

**inv1.2:** $q \in 1..n$

**inv1.3:** $p \leq q$

**inv1.4:** $\max(\text{ran}(f)) \in f(1..n)$

**inv1.4:** is the main invariant
Refinement: Initial and Final Events

INIT
begin
  \( m := 0 \)
  \( p := 1 \)
  \( q := n \)
end

maximum
when
  \( p = q \)
then
  \( m := f(p) \)
end
Refinement: Two New Events

**INIT**

```
begin
  m := 0
  p := 1
  q := n
end
```

**maximum**

```
when
  p = q
then
  m := f(p)
end
```

**increment**

```
when
  p < q
  f(p) ≤ f(q)
then
  p := p + 1
end
```

**decrement**

```
when
  p < q
  f(q) < f(p)
then
  q := q − 1
end
```
9  3  10  8  3  5  

5 < 9 (decrement)
Trace

\[
\begin{array}{cccccc}
9 & 3 & 10 & 8 & 3 & 5 \\
\end{array}
\]

\[8 < 9 \text{ (decrement)}\]
Trace

9 3 10 8 3 5

10 > 3 (increment)
Abstract and Concrete Traces

```
INIT  dec  dec  dec  inc  inc  maxi
```

DSE-Seminar, Friday, Jan 2, 2009 – p.40
Old Events Refine Their Abstractions
How about New Events?

INIT maxi dec dec dec inc inc maxi

INIT dec dec dec inc inc maxi
New Events Refine “Skip”

INIT  skip  skip  skip  skip  skip  maxi

INIT  dec  dec  dec  inc  inc  maxi
To be Proved

- Invariant maintenance
- Event refinement
  - guard strengthening
  - concrete action simulates the abstract one
- Well-definedness
Pathologies which Must be Avoided

- Early deadlock

```
INIT  dec  dec  dec  maxi
```
Pathologies which Must be Avoided

- Early deadlock
  
  ![Diagram of Early Deadlock]

- Divergence
  
  ![Diagram of Divergence]
To be Proved more

- Invariant maintenance
- Event refinement
  - guards strengthening
  - concrete action simulates the abstract one
- Well-definedness
- Trace refinement
  - Disjunction of guards must hold (no early deadlock)
  - New events must be convergent (must decrease a variant)
Towards the Final Construction

INIT
begin
\[ m := 0 \]
\[ p := 1 \]
\[ q := n \]
end

maximum
when
\[ p = q \]
then
\[ m := f(p) \]
end

increment
when
\[ p \neq q \]
\[ f(p) \leq f(q) \]
then
\[ p := p + 1 \]
end

decrement
when
\[ p \neq q \]
\[ f(q) < f(p) \]
then
\[ q := q - 1 \]
end
Statements for a Pidgin PL

```plaintext
while condition do statement end
if condition then statement else statement end
statement; statement
variable_list:= expression_list
```
IF Merging Rule

when \( P \) then \( S \) end

when \( Q \) then \( T \) end

when \( \neg Q \) then \( S \) else \( T \) end end

\( \rightarrow \)

when \( P \) then

if \( Q \) then \( S \) else \( T \) end end

M.IF
Applying Rule M.IF

increment

when

\[ p \neq q \]
\[ f(p) \leq f(q) \]
then

\[ p := p + 1 \]
end

decrement

when

\[ p \neq q \]
\[ f(q) < f(p) \]
then

\[ q := q - 1 \]
end

decrement_increment

when

\[ p \neq q \]
then

if

\[ f(q) < f(p) \]
then

\[ q := q - 1 \]
else

\[ p := p + 1 \]
end
end
WHILE Merging Rule (spec.)

| when \( Q \) then \( S \) end | when \( \neg Q \) then \( T \) end | while \( Q \) do \( S \) end; \( T \) end | M.WHILE |
Applying Rule M.WHILE

decrement_increment

when \( p \neq q \)
then if \( f(q) < f(p) \) then
  \( q := q - 1 \)
else \( p := p + 1 \) end
end

maximum

when
  \( p = q \)
then
  \( m := f(p) \)
end


decrement_increment_maximum

when \( p \neq q \)
then if \( f(q) < f(p) \) then \( q := q - 1 \)
else \( p := p + 1 \) end
end; \( m := f(p) \)
WHILE Merging Rule (gen.)

<table>
<thead>
<tr>
<th>when $P$ then $S$ end</th>
<th>when $P$ then $T$ end</th>
<th>when $P$ then while $Q$ do $S$ end; $T$ end</th>
</tr>
</thead>
</table>

- $P$ must be invariant under $S$
- The first event must have been introduced at one refinement step below the second one
Final Rule M.INIT

- Once we have obtained an event without guard
- We add to it the event INIT by sequential composition
- We then obtain the final program
The Program: Putting Event Together

**INIT**

begin
  \( m := 0 \)
  \( p := 1 \)
  \( q := n \)
end

**decrement**

when
  \( p \neq q \)
  \( f(q) < f(p) \)
then
  \( q := q - 1 \)
end

**increment**

when
  \( p \neq q \)
  \( f(p) \leq f(q) \)
then
  \( p := p + 1 \)
end

**maximum**

when
  \( p = q \)
then
  \( m := f(p) \)
end
\[ m, p, q := 0, 1, n; \]

\[ \text{INIT} \]

\[ \text{while } p < q \text{ do} \]

\[ \text{if } f(q) < f(p) \text{ then} \]

\[ q := q - 1 \]

\[ \text{decrement} \]

\[ \text{else} \]

\[ p := p + 1 \]

\[ \text{increment} \]

\[ \text{end} \]

\[ \text{end}; \]

\[ m := f(p) \]

\[ \text{maximum} \]