Modelling and verification with Event-B

Michael Butler

SETSS 2016, Chongqing
V model of software development

- Specification
- Architecture design
- Module design
- Coding
- Unit testing
- Integration testing
- System testing
B Method (Abrial, from 1990s)

- **Model** using set theory and logic (following Z notation)
- **Analyse models** using proof, model checking, animation
- Refinement-based development
  - verify conformance between *higher-level* and *lower-level* models
  - chain of refinements
- Commercial tools, :
  - *Atelier-B* (ClearSy, FR) - used mainly in railway industry
  - *B-Toolkit* (B-Core, UK)
B evolves to Event-B (from 2004)

• B Method was designed for software development

• Realisation that it is important to reason about system behaviour, not just software

• Event-B is intended for modelling and refining system behaviour

• Refinement notion is more flexible than B
  • Same set theory and logic

• Rodin tool for Event-B (www.event-b.org)
  • Open source, Eclipse based, open architecture
    – Range of plug-in tools (provers, ProB model checker, UML-B,...)
Industrial uses of Event-B

• Event-B in Railway Interlocking
  – Alstom, Systerel
• Event-B in Smart Grids
  – Selex, Critical Software
• External Adopters:
  – AWE: Experience of Applying Rodin in an Industrial Environment

www.advance-ict.eu/industry_days
SETSS lectures on Event-B

• Modelling with sets and invariants

• Model verification with Rodin prover

• Modelling with relations, class diagrams

• Refinement
  • model extension
  • data refinement
Example Requirements for a Building Control System

- Specify a system that monitors users entering and leaving a building.
- A person can only enter the building if they are a registered user.
- The system should be aware of whether a registered user is currently inside or outside the building.
Venn Diagram

Intersection
Disjoint sets

Invariants:

inv1: register in

inv2: register = in \[ out \] // all registered users must be either inside or outside

inv3: in \setminus out = {} // no user can be inside and outside

Invariant: \( \text{in} \cap \text{out} = \{\} \)
Registered users are either \textit{in} or \textit{out}

\[ \text{register} = \text{in} \cup \text{out} \]
Carrier Set: type for users

\[ \text{register} \subseteq \text{USER} \]
Event: user *enters* building

Guard: \( s \in out \)

Action:
\[
\begin{align*}
in &:= in \cup \{s\} \\
out &:= out \setminus \{s\}
\end{align*}
\]
Event: user *leaves* building

Guard:

\[ s \in in \]

Action:

\[ \begin{align*}
    in & := in \setminus \{s\} \\
    out & := out \cup \{s\}
\end{align*} \]
Basic Set Theory

- A set is a collection of elements.
- Elements of a set are not ordered.
- Elements of a set may be numbers, names, identifiers, etc.
- Sets may be finite or infinite.
- Relationship between an element and a set: is the element a member of the set.

For element $x$ and set $S$, we express the membership relation as follows:

$$x \in S$$
A set $S$ is said to be subset of set $T$ when every element of $S$ is also an element of $T$. This is written as follows:

$$S \subseteq T$$

For example:

$$\{ 5, 8 \} \subseteq \{ 4, 5, 6, 7, 8 \}$$

A set $S$ is said to be equal to set $T$ when $S \subseteq T$ and $T \subseteq S$.

$$S = T$$

For example:

$$\{ 5, 8, 3 \} = \{ 3, 5, 5, 8 \}$$
Operations on sets

- **Union** of $S$ and $T$: set of elements in either $S$ or $T$:

  \[ S \cup T \]

- **Intersection** of $S$ and $T$: set of elements in both $S$ and $T$:

  \[ S \cap T \]

- **Difference** of $S$ and $T$: set of elements in $S$ but not in $T$:

  \[ S \setminus T \]
Example Set Expressions

\[\{a, b, c\} \cup \{b, d\} = \{a, b, c, d\}\]
\[\{a, b, c\} \cap \{b, d\} = \{b\}\]
\[\{a, b, c\} \setminus \{b, d\} = \{a, c\}\]

\[\{a, b, c\} \cap \{d, e, f\} = \{\}\]
\[\{a, b, c\} \setminus \{d, e, f\} = \{a, b, c\}\]
context BuildingContext
sets USER
end

machine Building
variables register in out
invariants

inv1: register ⊆ USER  // set of registered users
inv2: register = in ∪ out  // all registered users must be
                         // either inside or outside
inv3: in ∩ out = {}   // no user can be inside and outside
Entering and Leaving the Building

initialisation \( in, out, register := \{\}, \{\}, \{\} \)

events

\[
\begin{align*}
\text{Enter} & \triangleq \\
\text{any } s \text{ where } & s \in out \\
\text{then } & in := in \cup \{s\} \\
& out := out \setminus \{s\} \\
\text{end} \\
\text{Leave} & \triangleq \\
\text{any } s \text{ where } & s \in in \\
\text{then } & in := in \setminus \{s\} \\
& out := out \cup \{s\} \\
\text{end}
\end{align*}
\]
Event-B context

- **Carrier Sets**: abstract types used in specification
- **Constants**: logical variables whose value remain constant
- **Axioms**: constraints on the constants. An axiom is a logical predicate.
Event-B *machine*

- **Sees:** one or more contexts
- **Variables:** state variables whose values can change
- **Invariants:** constraints on the variables that should always hold true. An invariant is a logical predicate.
- **Initialisation:** initial values for the abstract variables
- **Events:** guarded actions specifying ways in which the variables can change. Events may have parameters.
Adding New Users

New users cannot be registered already.

\[
\text{NewUser} \triangleq \\
\text{any } s \text{ where } \\
s \in (\text{USER} \setminus \text{register}) \\
\text{then} \\
\text{register} := \text{register} \cup \{s\} \\
\text{end}
\]

What is the error in this specification?
Adding New Users – Correct Version

NewUser $\triangleq$

any $s$ where
$$s \in (\text{USER} \setminus \text{register})$$
then
$$\text{register} := \text{register} \cup \{s\}$$
$$\text{out} := \text{out} \cup \{s\}$$
end

Newly registered users must be added either to $in$ or $out$ to preserve to $inv2$. 
Rodin demo

• Animation with ProB

• Checking for invariant violations with ProB
Types in Event-B

- **Predefined Types:**
  - \( \mathbb{Z} \) Integers
  - \( \mathbb{B} \) Booleans \{ TRUE, FALSE \}

- **Basic Types (or Carrier Sets):**
  - \texttt{sets} \quad \texttt{WORD} \quad \texttt{NAME}

Basic types are introduced to represent the entities of the problem being modelled.

Note: \( \mathbb{N} \) is a subset of \( \mathbb{Z} \) representing all non-negative integers (including 0).
Type for sets?

- \( w \in \text{WORD} \) means that the type of \( w \) is \( \text{WORD} \).

- \( \text{known} \subseteq \text{WORD} \) - what is the type of \( \text{known} \)?
Powersets

The powerset of a set $S$ is the set whose elements are all subsets of $S$:

$$\mathcal{P}(S)$$

Example

$$\mathcal{P}(\{a, b, c\}) = \{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$$

Note $S \in \mathcal{P}(T)$ is the same as $S \subseteq T$

Sets are themselves elements – so we can have sets of sets. $\mathcal{P}(\{a, b, c\})$ is an example of a set of sets.
Types of Sets

All the elements of a set must have the same type.

For example, \{3, 4, 5\} is a set of integers.
More Precisely: \{3, 4, 5\} \in \mathcal{P}(\mathbb{Z}).
So the type of \{3, 4, 5\} is \mathcal{P}(\mathbb{Z})

To declare \(x\) to be a set of elements of type \(T\) we write either

\[ x \in \mathcal{P}(T) \quad \text{or} \quad x \subseteq T \]

- \(\text{known} \subseteq \text{WORD}\) - so type of \(\text{known}\) is \(\mathcal{P}(\text{WORD})\)
Predicate Logic

Basic predicates: \( x \in S \), \( S \subseteq T \), \( x \leq y \)

Predicate operators:

- Negation: \( \neg P \) \( P \) does not hold
- Conjunction: \( P \land Q \) both \( P \) and \( Q \) hold
- Disjunction: \( P \lor Q \) either \( P \) or \( Q \) holds
- Implication: \( P \implies Q \) if \( P \) holds, then \( Q \) holds
- Universal Quantification: \( \forall x \cdot P \) \( P \) holds for all \( x \).
- Existential Quantification: \( \exists x \cdot P \) \( P \) holds for some \( x \).
## Defining Set Operators with Logic

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \notin S$</td>
<td>$\neg (x \in S)$</td>
</tr>
<tr>
<td>$x \in S \cup T$</td>
<td>$x \in S \lor x \in T$</td>
</tr>
<tr>
<td>$x \in S \cap T$</td>
<td>$x \in S \land x \in T$</td>
</tr>
<tr>
<td>$x \in S \setminus T$</td>
<td>$x \in S \land x \notin T$</td>
</tr>
<tr>
<td>$S \subseteq T$</td>
<td>$\forall x \cdot x \in S \implies x \in T$</td>
</tr>
</tbody>
</table>