Electronics and
Computer Science

University of Southampton

Introducing Event-B

COMP6226: Software Modelling Tools and Techniques for Critical Systems

Dr A. Rezazadeh (Reza)
Email: ra3@ecs.soton.ac.uk or ar4k06@soton.ac.uk

November 24

Objectives

- Introducing formal modelling using Event-B
- Use Sets and set operators for modelling
  - Modelling a dictionary with sets
- Using Types in Event-B
  - Simple types
  - Constructed types
- Learn about Predicate Logic

University of Southampton

Event-B Development

- System specifications are derived from requirements.
- System specification is an important precursor to design, implementation and testing.
- Event-B is a formal language for writing high-level specifications of computer systems.
- Event-B language includes logic and set theory.
- Formal specification is more precise and consistent than an informal (natural language) specification.
- Event-B typically used in safety-critical applications.

University of Southampton

Modelling in Event-B

- A model in Event-B consist of two parts, namely context and machine

static.

```
context  context_name
sets   <list of carrier sets>
constants   <list of constants>
axioms   <list of labelled
axioms>
end
```

dynamic

```
machine machine_name
sees <list of contexts>
variables <list of variables>
invariants <list of labelled
invariants>
events <list of events>
end
```

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.

Simple Event-B model: Counter

some sets are define before event B

context *CounterContext*
constants *cmax*
axioms $cmax \in \mathbb{N}$ // *cmax* is a natural number
end

machine *Counter*
sees *CounterContext*
variables *ctr*
invariants

$ctr \in \mathbb{Z}$ // *ctr* is an integer
$0 \leq ctr \leq cmax$ // between 0 and *cmax*

Invariants define valid system states.

- A model is representation of the system we want to build
  - Allows to reason about the future system during its design
- To reason about an intended system
  - Define its behaviour (what it does)
  - Incorporate constraints (what it must not do)

Increasing and decreasing the Counter

initialisation $ctr := 0$
events

$Inc \;\hat{=}\;$ **when** $ctr < cmax$ **then** $ctr := ctr + 1$ **end**

$Dec \;\hat{=}\;$ **when** $ctr > 0$ **then** $ctr := ctr - 1$ **end**

Events define changes to the system state.

Events have guards and actions.

Guards must be true for the actions to be executed.

Simple Example: Dictionary

axiom assossiate with

constant

context *DictionaryContext*
sets *Word* // *Word* is a basic type introduced for this model
end

need to define the set

machine *Dictionary*
sees *DictionaryContext*
variables *known*

subset

invariants $known \subseteq Word$ // set of known words

initialisation $known := \{\}$ → $\emptyset$

Adding words to the Dictionary

events

$$
\begin{array}{l}
\mathit{AddWord} \;\hat{=} \\
\quad \textbf{any } w \textbf{ where} \\
\qquad w \in \mathit{Word} \\
\quad \textbf{then} \\
\qquad \mathit{known} := \mathit{known} \cup \{w\} \\
\quad \textbf{end}
\end{array}
$$

$w \notin known$

union

This event has a parameter w representing the word that is added to the set of known words.

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:

$$\boxed{x \in S}$$

Enumeration and Cardinality of Finite Sets

- Finite sets can be expressed by enumerating the elements within braces, for example:

  大括号

$$\{ 3, 5, 8 \}$$

$$\{ a, b, c, d \}$$

- The cardinality of a finite set is the number of elements in that set: 集合的势

$$\boxed{card(S)}$$

- For example

$$\begin{aligned} card(\ \{ 3, 5, 8 \}\ ) &= 3 \\ card(\ \{ a, b, c, d \}\ ) &= 4 \\ card(\ \{\}\ ) &= 0 \end{aligned}$$

Subset and Equality Relations for Sets

- 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:

$$\boxed{S \subseteq T}$$

- 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 $T \subseteq S$.

$$\boxed{S = T}$$

- 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:

$$\boxed{S \cup T}$$

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

$$\boxed{S \cap T}$$

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

$$\boxed{S \setminus T}$$

minus

Example Set Expressions

$$\begin{aligned} \{a, b, c\} \cup \{b, d\} &= ? \\ \{a, b, c\} \cap \{b, d\} &= ? \\ \{a, b, c\} \setminus \{b, d\} &= ? \end{aligned}$$

$$\begin{aligned} \{a, b, c\} \cap \{d, e, f\} &= ? \\ \{a, b, c\} \setminus \{d, e, f\} &= ? \end{aligned}$$

Example Set Expressions

$$
\begin{aligned}
\{a, b, c\} \cup \{b, d\} &= \{a, b, c, d\} \\
\{a, b, c\} \cap \{b, d\} &= \{b\} \\
\{a, b, c\} \setminus \{b, d\} &= \{a, c\}
\end{aligned}
$$

$$
\begin{aligned}
\{a, b, c\} \cap \{d, e, f\} &= \{\} \\
\{a, b, c\} \setminus \{d, e, f\} &= \{a, b, c\}
\end{aligned}
$$

Simple Example: Dictionary

context *DictionaryContext*
sets *Word* // *Word* is a basic type introduced for this model
end

machine *Dictionary*
sees *DictionaryContext*
variables *known*

invariants $known \subseteq Word$ // set of known words

initialisation $known := \{\}$

Adding words to the Dictionary

events

$AddWord \;\hat{=}$

any w **where**

$w \in Word$

then

$known := known \cup \{w\}$

end

This event has a parameter w representing the word that is added to the set of known words.

Checking if a word is in a dictionary: 2 cases

$CheckWordOK \triangleq$
any $w, r!$ **where**
$w \in known$
$r! = TRUE$
then
$skip$ // omit in Rodin
end

$CheckWordNotOK \triangleq$
any $w, r!$ **where**
$w \notin known$
$r! = FALSE$
then
$skip$ // omit
end

Cases are represented by separate events.

In both cases, $r!$ represents a result parameter.

We use the '!' convention to represent result parameters.

Checking if a word is in a dictionary: 2 cases

```
event CheckWordOK
any w result where
    @grd1: w ∈ known
    @grd2: result = TRUE
  // Skip: No actions
end
```

```
event CheckWordKO
any w result where
    @grd1: w ∉ known
    @grd2: result = FALSE
  // Skip: No actions
end
```

Cases are represented by separate events.

In both cases, *result* represents a the output parameter.

B *context* contains

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

B *machine* contains

- **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.

```
context DictionaryContext
sets
    WORD    // WORD is a basic type introduced for this model
end
```

Counting Dictionary

machine *CountingDictionary*
sees *DictionaryContext*
variables *known* *count*
invariants

$$known \subseteq Word$$
$$count = card(known)$$

events

$$AddWord \triangleq$$

any w **where**
$w \in Word$
then
$known := known \cup \{w\}$
$count := count + 1$
end

Counting Dictionary

machine *CountingDictionary*
sees *DictionaryContext*
variables *known* *count*
invariants

$$known \subseteq Word$$
$$count = \underbrace{card(known)}$$ 集合的势

events

$$AddWord \ \hat{=}$$

any w **where**
  $w \in Word$
then
  $known := known \cup \{w\}$
  $count := count + 1$
end

$w \notin known$

- Is this specification of *AddWord* correct?

Word deletion in Counting Dictionary

$$
\begin{array}{l}
\mathit{RemoveWord} \;\; \hat{=} \\
\quad \textbf{any} \;\; w \;\; \textbf{where} \\
\quad\quad w \in \mathit{Word} \\
\quad \textbf{then} \\
\quad\quad \mathit{known} := \mathit{known} \;\setminus\; \{w\} \\
\quad\quad \mathit{count} := \mathit{count} - 1 \\
\quad \textbf{end}
\end{array}
$$

- Is this specification of *RemoveWord* correct?

Correct versions of Add and Remove

$w \in Word$
$w \notin known$

$$
\begin{array}{ll}
AddWord \;\hat{=} & \\
\quad \textbf{any } w \textbf{ where} & \\
\qquad w \in Word \setminus known & \\
\quad \textbf{then} & \\
\qquad known := known \cup \{w\} & \\
\qquad count := count + 1 & \\
\quad \textbf{end} &
\end{array}
\qquad
\begin{array}{ll}
RemoveWord \;\hat{=} & \\
\quad \textbf{any } w \textbf{ where} & \\
\qquad w \in known & \\
\quad \textbf{then} & \\
\qquad known := known \setminus \{w\} & \\
\qquad count := count - 1 & \\
\quad \textbf{end} &
\end{array}
$$

- Both of these events maintain the invariant $count = card(known)$ that links *count* and *known*.

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 recognised by the monitor.
- The system should be aware of whether a recognised user is currently inside or outside the building.

Is there anything missing from this set of requirements?

context *BuildingContext*
sets *User*
end

machine *Building*
sees *BuildingContext*
variables *register*, *in*, *out*
invariants

$inv1 : register \subseteq User$ // set of registered users
$inv2 : in \subseteq register$ // set of registered users who are inside
$inv3 : out \subseteq register$ // set of registered users who are outside
$inv4 : in \cap out = \{\}$ // no users can be both inside and outside
$inv5 : register = in \cup out$ // all registered users must be
// either inside or outside

Entering and Leaving the Building

initialisation $in,\ out,\ register\ :=\ \{\},\ \{\},\ \{\}$

events

register 随着 in 和 out 同时变化

$Enter\ \ \widehat{=}$
any s **where**
$s \in out$
then
$in := in \cup \{s\}$
$out := out \setminus \{s\}$
end

$Leave\ \ \widehat{=}$
any s **where**
$s \in in$
then
$in := in \setminus \{s\}$
$out := out \cup \{s\}$
end

Adding New Users

New users cannot be registered already.

$$
\begin{array}{l}
\mathit{NewUser} \quad \hat{=} \\
\quad \textbf{any } s \textbf{ where} \\
\qquad s \in (\mathit{User} \setminus \mathit{register}) \\
\quad \textbf{then} \\
\qquad \mathit{register} := \mathit{register} \cup \{s\} \\
\quad \textbf{end}
\end{array}
$$

Adding New Users

New users cannot be registered already.

$$
\begin{array}{l}
\mathit{NewUser} \quad \widehat{=} \\
\quad \textbf{any } s \textbf{ where} \\
\qquad s \in (\mathit{User} \setminus \mathit{register}) \\
\quad \textbf{then} \\
\qquad \mathit{register} := \mathit{register} \cup \{s\} \\
\quad \textbf{end}
\end{array}
$$

Can anyone spot an error in this specification?

neither inside nor out:

Adding New Users – Correct Version

$$
\begin{array}{l}
\mathit{NewUser} \ \ \widehat{=} \\
\quad \textbf{any } s \textbf{ where} \\
\qquad s \ \in \ (\mathit{User} \setminus \mathit{register}) \\
\quad \textbf{then} \\
\qquad \mathit{register} := \mathit{register} \cup \{s\} \\
\qquad \mathit{out} := \mathit{out} \cup \{s\} \\
\quad \textbf{end}
\end{array}
$$

Newly registered users must be added either to in or out to preserve inv5.

Types

All variables and expressions in B must have a type.
Types are represented by sets.

Let T be a set and x a constant or variable.
$x \in T$ specifies that x is of type T.

Examples:

$$\begin{array}{lll} a & \in & \mathbb{N} \quad //Integer \\ b & \in & \mathbb{Z} \\ w & \in & Word \end{array}$$

define in context

What are the types of the following expressions?

$$(a + b) \times 3$$

$$unix$$

operating system

Types in B

- Predefined Types:
  $\mathbb{Z}$ Integers
  $\mathbb{B}$ Booleans $\{$ TRUE, FALSE $\}$
- Basic Types (or Carrier Sets):
  sets *Word* *Name*

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

Note: $\mathbb{N}$ is a subet of $\mathbb{Z}$ representing all non-negative integers (including 0).

Type for sets?

- $w \in \mathit{WORD}$ means that the type of w is WORD.

Type for sets?

- $w \in WORD$ means that the type of w is $WORD$.
- $known \subseteq WORD$ - what is the type of *known*?

Powersets

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

$$\boxed{\mathbb{P}(S)}$$

Example

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

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

Sets are themselves elements – so we can have sets of sets. $\mathbb{P}(\ \{a, b, c\}\ )$ 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: $\{3, 4, 5\} \in \mathbb{P}(\mathbb{Z})$.
So the type of $\{3, 4, 5\}$ is $\mathbb{P}(\mathbb{Z})$

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

$$x \in \mathbb{P}(T) \qquad \text{or} \qquad x \subseteq T$$

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: $\{3, 4, 5\} \in \mathbb{P}(\mathbb{Z})$.
So the type of $\{3, 4, 5\}$ is $\mathbb{P}(\mathbb{Z})$

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

$$x \in \mathbb{P}(T) \qquad \text{or} \qquad x \subseteq T$$

- $\mathit{known} \subseteq \mathit{WORD}$ - so type of *known* is $\mathbb{P}(\mathit{WORD})$.

Checking Types

Assume S and T have type $\mathbb{P}(M)$. What are the types of:

$$S \cup T \quad ?$$
$$S \cap T \quad ?$$

Type of $\{\ \{3, 4\},\ \{4, 6\},\ \{7\}\ \}$?

Checking Types

$S \sqsubseteq M$

$T \sqsubseteq M$

Assume S and T have type $\mathbb{P}(M)$. What are the types of:

union

intersection

don't change set.

$$S \cup T \quad \in \mathbb{P}(M)$$

$$S \cap T \quad \in \mathbb{P}(M)$$

Type of $\{\ \{3,4\},\ \{4,6\},\ \{7\}\ \}\ \in \boxed{\mathbb{P}(\mathbb{P}(\mathbb{Z}))}$

$\sqsubseteq \mathbb{P}(\mathbb{Z})$

Expressions which are incorrectly typed are meaningless:

$$\{\ 4, 6, alice\ \}$$

$$\{\ alice, bob\ \}\ \cup\ \{\ red, green, blue\ \}$$

Classification of Types

Simple Types:

- $\mathbb{Z}$, $\mathbb{B}$
- Basic types (e.g., *Word*, *Name*)

Constructed Types:

- $\mathbb{P}(T)$

$\mathbb{P}(T)$ is a type that is constructed from T.

We will see more constructed types later.

Why Types?

- Types help to structure specifications by differentiating objects.
- Types help to prevent errors by not allowing us to write meaningless things.
- Types can be checked by computer.

Predicate Logic

mathmatical

Basic predicates: $x \in S, \quad S \subseteq T, \quad S = T, \quad x < y, \quad x \leq y$

Predicate operators:

logic.

- Negation: $\boxed{\neg P}$ P does not hold
- Conjunction: $\boxed{P \wedge Q}$ both P and Q hold
- Disjunction: $\boxed{P \vee Q}$ either P or Q holds
- Implication: $\boxed{P \implies Q}$ if P holds, then Q holds
- Universal Quantification: $\boxed{\forall x \cdot P}$ P holds for all x.
- Existential Quantification: $\boxed{\exists x \cdot P}$ P holds for some x.

Defining Set Operators with Logic

Predicate	Definition
$x \notin S$	$\neg\,(x \in S)$
$x \in S \cup T$	$x \in S \quad \lor \quad x \in T$
$x \in S \cap T$	$x \in S \quad \land \quad x \in T$
$x \in S \setminus T$	$x \in S \quad \land \quad x \notin T$
$S \subseteq T$	$\forall x \cdot x \in S \implies x \in T$

YOUR QUESTIONS