# Introducing Event-B

© Michael Butler

University of Southampton

January 23, 2013

# Simple Event-B model: Counter

**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.

# Increasing and decreasing the Counter

**initialisation** $\quad ctr := 0$

**events**

$$
\begin{array}{lll}
Inc & \triangleq & \textbf{when} \\
 & & \quad ctr < cmax \\
 & & \textbf{then} \\
 & & \quad ctr := ctr + 1 \\
 & & \textbf{end}
\end{array}
\qquad
\begin{array}{lll}
Dec & \triangleq & \textbf{when} \\
 & & \quad ctr > 0 \\
 & & \textbf{then} \\
 & & \quad ctr := ctr - 1 \\
 & & \textbf{end}
\end{array}
$$

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

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

**machine** *Dictionary*
**variables** *known*

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

**initialisation** $known := \{\}$

# Adding words to the Dictionary

**events**

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

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:

$$\{\ 3, 5, 8\ \}$$
$$\{\ a, b, c, d\ \}$$

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

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

- 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 $T$:

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

# 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

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

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

# Simple Example: Dictionary

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

**machine** *Dictionary*
**variables** *known*

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

**initialisation** $known := \{\}$

# Adding words to the Dictionary

**events**

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

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

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

$CheckWordNotOK \;\hat{=}$
**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.

# 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.

# Counting Dictionary

**machine** $\mathit{CountingDictionary}$
**variables** $\mathit{known}$ $\mathit{count}$
**invariants**

$$\mathit{known} \subseteq \mathit{Word}$$
$$\mathit{count} = \mathit{card}(\mathit{known})$$

**events**

$$\mathit{AddWord} \;\hat{=}$$

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

# Counting Dictionary

**machine** *CountingDictionary*
**variables** *known* *count*
**invariants**

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

**events**

$$AddWord \;\; \widehat{=}$$

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

- Is this specification of *AddWord* correct?

# Word deletion in Counting Dictionary

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

- Is this specification of *RemoveWord* correct?

# Correct versions of Add and Remove

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

$RemoveWord \;\hat{=}$
**any** $w$ **where**
$w \in known$
**then**
$known := known \setminus \{w\}$
$count := count - 1$
**end**

- 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*
**variables** *register* *in* *out*
**invariants**

$register \subseteq User$ // set of registered users

$in \subseteq register$ // set of registered users who are inside

$out \subseteq register$ // set of registered users who are outside

$in \cap out = \{\}$ // no users can be both inside and outside

$register = in \cup out$ // all registered users must be
// either inside or outside

# Entering and Leaving the Building

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

**events**

$Enter \quad \widehat{=}$

**any** $s$ **where**

$s \in out$

**then**

$in := in \cup \{s\}$

$out := out \setminus \{s\}$

**end**

$Leave \quad \widehat{=}$

**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.

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

# Adding New Users

New users cannot be registered already.

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

Can anyone spot an error in this specification?

# Adding New Users – Correct Version

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

# 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:

$$\begin{aligned} a &\in \mathbb{N} \\ b &\in \mathbb{Z} \\ w &\in \mathit{Word} \\ sys &\in \mathit{OperatingSystem} \end{aligned}$$

What are the types of the following expressions?

$$(a + b) \times 3$$
$$\mathit{unix}$$

# 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).

# Powersets

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

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

Example

$$\begin{aligned}\mathbb{P}(\ \{a, b, c\}\ ) \quad = \quad & \{\ \{\},\ \{a\},\ \{b\},\ \{c\}, \\ & \{a, b\},\ \{a, c\},\ \{b, c\},\ \{a, b, c\}\ \}\end{aligned}$$

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

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

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

# 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 $\{\ \{3,4\},\ \{4,6\},\ \{7\}\ \}$ ?

Expressions which are incorrectly typed are meaningless:

$$\{\ 4, 6, unix\ \}$$
$$\{\ windows, mac\ \}\ \cup\ \{\ bwm, tata, ford, toyota\ \}$$

# 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

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

**Predicate operators:**

- 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 |
|---|---|
| $x \notin S$ | $\neg (x \in S)$ |
| $x \in S \cup T$ | $x \in S \lor x \in T$ |
| $x \in S \cap T$ | $x \in S \land x \in T$ |
| $x \in S \setminus T$ | $x \in S \land x \notin T$ |
| $S \subseteq T$ | $\forall x \cdot x \in S \implies x \in T$ |