

Introducing Event-B – Modelling with Sets

© Michael Butler

University of Southampton

May 22, 2010

Simple Event-B model: Counter

context *CounterContext*

constants *max*

axioms $max \in \mathbb{N}$ // *max is a natural number*

end

machine *Counter*

sees *CounterContext*

variables *ctr*

invariants

$ctr \in \mathbb{Z}$ // *ctr is an integer*

$0 \leq ctr \leq max$ // *between 0 and max*

Invariants define **valid** system states.

initialisation $ctr := 0$

Increasing and decreasing the Counter

events

```
Inc  $\hat{=}$  when  
    ctr < max  
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

context *DictionaryContext*

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

machine *Dictionary*

variables *known*

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

initialisation $known := \{\}$

Adding words to the Dictionary

events

```
AddWord  $\hat{=}$   
  any  $w$  where  
     $w \in \text{Word}$   
  then  
     $\text{known} := \text{known} \cup \{w\}$   
  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 $\hat{=}$
any w , *result* **where**
 $w \in \textit{known}$
 result = *TRUE*
then
 skip
end

CheckWordNotOK $\hat{=}$
any w , *result* **where**
 $w \notin \textit{known}$
 result = *FALSE*
then
 skip
end

Cases are represented by **separate events**.

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

B *context* contains

- ▶ **Sets:** abstract types used in specification
- ▶ **Constants:** logical variables whose values 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 *CountingDictionary*

variables *known count*

invariants

$$\textit{known} \subseteq \textit{Word}$$

$$\textit{count} = \textit{card}(\textit{known})$$

events

AddWord $\hat{=}$

any *w* **where**

$$w \in \textit{Word}$$

then

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

$$\textit{count} := \textit{count} + 1$$

end

Counting Dictionary

machine *CountingDictionary*

variables *known count*

invariants

$$\textit{known} \subseteq \textit{Word}$$

$$\textit{count} = \textit{card}(\textit{known})$$

events

AddWord $\hat{=}$

any *w* **where**

$$w \in \textit{Word}$$

then

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

$$\textit{count} := \textit{count} + 1$$

end

- ▶ Is this specification of *AddWord* correct?

Word deletion in Counting Dictionary

```
RemoveWord  $\hat{=}$   
  any  $w$  where  
     $w \in \textit{Word}$   
  then  
     $\textit{known} := \textit{known} \setminus \{w\}$   
     $\textit{count} := \textit{count} - 1$   
  end
```

- ▶ Is this specification of *RemoveWord* correct?

Correct versions of Add and Remove

AddWord $\hat{=}$

any w **where**

$w \notin \textit{known}$

then

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

$\textit{count} := \textit{count} + 1$

end

RemoveWord $\hat{=}$

any w **where**

$w \in \textit{known}$

then

$\textit{known} := \textit{known} \setminus \{w\}$

$\textit{count} := \textit{count} - 1$

end

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

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:

$$x \in S$$

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:

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

- ▶ For example

$$\text{card}(\{ 3, 5, 8 \}) = 3$$

$$\text{card}(\{ a, b, c, d \}) = 4$$

$$\text{card}(\{ \}) = 0$$

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:

$$S \subseteq T$$

- ▶ 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

$$\{a, b, c\} \cup \{b, d\} = ?$$

$$\{a, b, c\} \cap \{b, d\} = ?$$

$$\{a, b, c\} \setminus \{b, d\} = ?$$

$$\{a, b, c\} \cap \{d, e, f\} = ?$$

$$\{a, b, c\} \setminus \{d, e, f\} = ?$$

Example Set Expressions

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

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

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

$$\{a, b, c\} \cap \{d, e, f\} = \{\}$$

$$\{a, b, c\} \setminus \{d, e, f\} = \{a, b, c\}$$

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  $\subseteq$  in  $\cup$  out // all registered users must be  
                        // either inside or outside
```

Entering and Leaving the Building

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

events

Enter $\hat{=}$

any s **where**

$s \in out$

then

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

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

end

Leave $\hat{=}$

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.

```
NewUser  $\hat{=}$   
  any s where  
     $s \in (User \setminus register)$   
  then  
     $register := register \cup \{s\}$   
  end
```

Adding New Users

New users cannot be registered already.

```
NewUser  $\hat{=}$   
  any s where  
     $s \in (User \setminus register)$   
  then  
     $register := register \cup \{s\}$   
  end
```

Can anyone spot an error in this specification?

Adding New Users – Correct Version

```
NewUser  $\hat{=}$   
  any s where  
     $s \in (User \setminus register)$   
  then  
     $register := register \cup \{s\}$   
     $out := out \cup \{s\}$   
  end
```


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:

$$a \in \mathbb{N}$$

$$b \in \mathbb{Z}$$

$$w \in \textit{Word}$$

$$\textit{sys} \in \textit{OperatingSystem}$$

What are the types of the following expressions?

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

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 subset 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 :

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

Example

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

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

Powersets

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

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

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

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

Checking Types

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

$$S \cup T ?$$

$$S \cap T ?$$

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

Expressions which are incorrectly typed are meaningless:

$$\{ 4, 6, \textit{unix} \}$$

$$\{ \textit{windows}, \textit{mac} \} \cup \{ \textit{bwm}, \textit{tata}, \textit{ford}, \textit{toyota} \}$$

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: $\neg P$ P does **not** hold
- ▶ Conjunction: $P \wedge Q$ both P **and** Q hold
- ▶ Disjunction: $P \vee Q$ 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

Predicate	Definition
$x \notin S$	$\neg (x \in S)$
$x \in S \cup T$	$x \in S \vee x \in T$
$x \in S \cap T$	$x \in S \wedge x \in T$
$x \in S \setminus T$	$x \in S \wedge x \notin T$
$S \subseteq T$	$\forall x \cdot x \in S \implies x \in T$