T-79.232 Safety Critical Systems

Case Study 2: B Method - Basic Concepts

Teemu Tynjälä

February 21, 2008

B method - what is it about?

B method describes entities called *abstract machines*. What is an abstract machine? An example machine would look something like this:

MACHINE...
VARIABLES...
INVARIANT...
INITIALISATION...
OPERATIONS...
END

Let’s go through each of the constituent parts in the following, together with an example.

MACHINE

The MACHINE keyword gives a name to the specification we are currently modelling. It is possible to include MACHINEs from other MACHINEs, so they can be used as building blocks.

For a ticketing system we may define:

MACHINE *Ticket*

VARIABLES

The VARIABLES keyword is used for introducing variables that define the local state of the system. In this section only the names are given, types are defined later.

For a ticketing system we might have two variables: *serve* and *next*

VARIABLES *serve*, *next*

INVARIANT

The INVARIANT keyword is used for few purposes. The types for variables are defined here, as well as a statement that is to be true for the variables in all system states. In a sense, INVARIANT is an assertion which causes the execution of a machine to stop if it is violated.

For a ticketing system, we may define:

INVARIANT $serve \in \mathbb{N} \wedge next \in \mathbb{N} \wedge serve \leq next$

OPERATIONS

The OPERATIONS keyword defines the ’interface’ between the user and the machine. Output are used to communicate results to users, and these are separate from machine’s own variables. Preconditions control whether an operation is executed at all.

$$outputs \longleftarrow name(inputs)$$

An example of an operation in B is:

$$ss \longleftarrow \textbf{serve\_next} =$$
$$\textbf{PRE}\ \ serve < next$$
$$\textbf{THEN}\ \ ss,\ serve\ \ :=\ \ serve+1,\ serve+1$$
$$\textbf{END}$$

INITIALISATION

The INITIALISATIONS part of a machine gives the initial state of the machine. Each variable defined in the VARIABLES section must be given some value.

In our Ticketing machine, we may define:

INITIALISATION $serve, next \ := \ 0, 0$

My first complete B machine...

```
MACHINE $Ticket$
VARIABLES $serve, next$
INVARIANT $serve \in \mathbb{N} \wedge next \in \mathbb{N} \wedge serve \leq next$
INITIALISATION $serve, next := 0, 0$
OPERATIONS
    $ss \longleftarrow$ serve_next $=$
     PRE $serve < next$
     THEN $ss, serve := serve + 1, serve + 1$
     END ;
    $tt \longleftarrow$ take_ticket $=$
     PRE $true$
     THEN $tt, next := next, next + 1$
     END
END
```

Couple points about sets...

As a little refresher on sets, let's do the following questions (here, $card$ refers to set cardinality, and $\mathbb{P}$ is the notation for powerset):

Assume that we have $MEMBER = \{fred, ginger, harold\}$. Compute the following:
1. $\mathbb{P}\ MEMBER$
2. $MEMBER \times MEMBER$
3. $card(\mathbb{P}(MEMBER \times MEMBER))$
4. $card(\mathbb{P}\mathbb{P}\ MEMBER)$

You should have set cardinalities of 8, 9, 512 and 256 for your answers, respectively...

Little hint on Quantified Logic

You are already familiar with this logic, but the only point worth mentioning is the duality of universal and existential quantification. So, here you go:

$$\neg\forall\, x.(x \in T \Rightarrow P) \;=\; \exists x.(x \in T \wedge \neg P)$$

$$\neg\exists x.(x \in T \wedge P) \;=\; \forall x.(x \in T \Rightarrow \neg P)$$

Substitutions..

B method makes heavy use of substitutions, so let's study it in more detail.

$P[E/x]$ is the same as expression P except that E is substituted for every occurrence of x.

Examples:

$(x < y)[(x+y+z)/y] = (x < x+y+z)$

$(serve < next \wedge next < last - 4)[serve+1/serve] =$
$serve+1 < next \wedge next < last - 4$

$(members \subseteq PERSON)[members \cup \{new\}/members] =$
$members \cup \{new\} \subseteq PERSON$

Hints on Substitutions...

Couple things are worth mentioning about Substitutions when we have quantified logic:

1. Re-name bound variables before substitution, if there is a name clash.

$$(\exists n.(n \in \mathbb{N} \wedge n > limit))[n+3/limit]$$
$$= (\exists q.(q \in \mathbb{N} \wedge q > limit))[n+3/limit]$$
$$= (\exists q.(q \in \mathbb{N} \wedge q > n+3))$$

2. Bound variables are not substituted:

$$(\exists n.(n \in \mathbb{N} \wedge n > limit))[over/n]$$
$$= (\exists m.(m \in \mathbb{N} \wedge m > limit))[over/n]$$
$$= (\exists m.(m \in \mathbb{N} \wedge m > limit))$$

Weakest Preconditions

Changes of state happen through transformations, which are caused by operations.

Sometimes we know the desired outcome of an operation, and the operation to be executed but are unsure of what preconditions will guarantee the desired outcome to occur.

For this purpose, we introduce the concept of Weakest Precondition, which is written as $[S]P$.

P is a logical statement for the desired outcome, S is the operation to be carried out, and $[S]P$ is a logical statement describing the initial states that guarantee P after executing S.

Example of a Weakest Precondition calculation

Imagine that $x, y \in \{0, 1, 2\}$, and we have an operation $y := max\{0, y - x\}$.

Now, we are interested in the values for x and y that guarantee that $y > 0$ after the operation is finished.

In mathematical terms, we want to find $[y := max\{0, y - x\}](y > 0)$

Clearly, we see that $(x, y) = (0, 1)$, $(0, 2)$, $(1, 2)$ satisfy the constraint, and thus the weakest precondition in terms of logic is:

$$[y := max\{0, y - x\}](y > 0) = (x = 0 \wedge y = 1) \vee (x = 0 \wedge y = 2) \vee (x = 1 \wedge y = 2)$$

Laws of $[S]P$

There are three simple rules for weakest preconditions which can be used to simplify statements:

$$[S](P \wedge Q) \;=\; [S]P \wedge [S]Q$$

$$[S]P \vee [S]Q \;\Longrightarrow\; [S](P \vee Q)$$

$P \Longrightarrow Q$ implies $[S]P \Longrightarrow [S]Q$

Weakest Preconditions and Assignment Statement

Weakest Precondition for assignment statements has a simple form, but you have to see an example to see how it really works...

$$[x := E]P \;=\; P[E/x]$$

And to see an example, follow along...

$$\begin{aligned}&[serve := serve+1](serve \leq next)\\ \Rightarrow\;\; & serve+1 \leq next\\ \Rightarrow\;\; & serve < next\end{aligned}$$

Similarly

$$\begin{aligned}&[colours := colours \cup \{blue\}](green \notin colours)\\ \Rightarrow\;\; & green \notin colours \cup \{blue\}\\ \Rightarrow\;\; & green \notin colours\end{aligned}$$

Weakest Preconditions and Multiple Assignment

The above rule may be generalized to multiple assignments in a straightforward manner:

$$[x_1, ..., x_n := E_1, ..., E_n]P \;=\; P[E_1, ..., E_n / x_1, ..., x_n]$$

Here’s an example:

$$[serve,\ next \;:=\; serve + 1,\ next - 1](serve \leq next)$$
$$\Rightarrow serve + 1 \;\leq\; next - 1$$

The easiest B statement you'll ever see....

In B, we have a statement which does nothing. The name for it is $skip$.

As you might think, also the weakest precondition rule for this statement is rather simple...

$[skip]P = P$

Weakest Preconditions and Conditional

IF/THEN/ELSE is a very popular construct in programming languages, and B has developed a weakest precondition rule for it.

$$[\textbf{IF}\ E\ \textbf{THEN}\ S\ \textbf{ELSE}\ T\ \textbf{END}]P = (E \wedge [S]P) \vee (\neg E \wedge [T]P)$$

Let's compute an example:

$$\begin{aligned}
&[\textbf{IF}\ x > 7\ \textbf{THEN}\ x := x - 4\ \textbf{ELSE}\ x := x + 3\ \textbf{END}](x > 12)\\
&= (x > 7 \wedge [x := x - 4](x > 12)) \vee (x \leq 7 \wedge [x := x + 3](x > 12))\\
&= (x > 7 \wedge x - 4 > 12) \vee (x \leq 7 \wedge x + 3 > 12)\\
&= (x > 7 \wedge x > 16) \vee (x \leq 7 \wedge x > 9)\\
&= (x > 16)
\end{aligned}$$

Weakest Preconditions and Case clause

Case-clauses are also used widely in programming languages. Here is the rule for it in B...

$$\left[\begin{array}{l}\textbf{CASE } E \textbf{ OF}\\ \textbf{EITHER } e_1 \textbf{ THEN } T_1\\ \textbf{OR } e_2 \textbf{ THEN } T_2\\ \textbf{OR}\ldots\\ \textbf{OR } e_n \textbf{ THEN } T_n\\ \textbf{ELSE } V\\ \textbf{END}\end{array}\right] P = \left(\begin{array}{l} E = e_1 \Rightarrow [T_1]P\\ \wedge E = e_2 \Rightarrow [T_2]P\\ \ldots\\ \wedge E = e_n \Rightarrow [T_n]P\\ \wedge (E \neq e_1 \wedge E \neq e_2 \ldots \wedge E \neq e_n) \Rightarrow [V]P\end{array}\right)$$

There's a lot of similarity here to a IF/ELSEIF/.../ELSE clause, and construction of an example is straightforward.

Last Easy Rule...

Here’s the final rule for weakest preconditions. It is customary for some programs to include a BEGIN/END pair in their language, although it often doesn’t add much meaning.

In weakest precondition terms, such BEGIN/END pairs may be stripped off as follows:

$$[\mathbf{BEGIN}\ S\ \mathbf{END}]P \ = \ [S]P$$

Thank You... You made it to the end...

OK, this was our first look at B method, and some mathematical constructs that are useful there.

Next time we’ll carry on with machine consistency, relations and functions, and we’ll start seeing more complicated B specifications.

References

The material in this presentation has been obtained from

1. the b-method - an introduction. Steve Schneider. Palgrave, 2001. (This book belongs to the *cornerstones of computing* series by the same publisher)