T-79.5303 Safety Critical Systems

Case Study 3: B Method - Machine Consistency and Relations

Teemu Tynjälä

February 28, 2008

What is Machine Consistency?

To prove that a machine is consistent, you are ensuring that invariants, etc. are always preserved. Let's take a look at a generic B machine:

```
MACHINE $N$
VARIABLES $v$
INVARIANT $I$
INITIALISATION $T$
OPERATIONS
    $y \longleftarrow op(x) =$
     PRE $P$
     THEN $S$
     END;
    $\dots$
END
```

Consistency of the invariant...

Referring to the general machine earlier, we want to show that there exists some valuation for the variables such that the invariant holds.

In mathematical terms, this is written as:

$$\exists\, v.I$$

A simple example from a Ticket-like machine would be:

$$\exists\, serve, next.\ \ (serve \in \mathbb{N} \wedge next \in \mathbb{N} \wedge serve \leq next)$$

Consistency of the Initialisation...

We require that the the initialisation operation results in a state which satisfies the invariant.

In mathematical terms:

$$[T]I$$

The above logical expression *must* reduce to $TRUE$.

A simple example from the Ticket machine is:

$$[serve, next \ := \ 0,0](serve \in \mathbb{N} \wedge next \in \mathbb{N} \wedge serve \ \leq \ next)$$

After simplification, this reduces to $TRUE$, which was the requirement

Consistency of the Operations...

We must also prove that each operation preserves the invariant. That means, that if the machine is in a state in which I and P are true (invariant and the operations's precondition), then its behaviour S must be guaranteed to reestablish I.

In mathematical terms:

$$I \wedge P \implies [S]I$$

Example from the Ticket machine would be something like:

$I \qquad (serve \in \mathbb{N} \wedge next \in \mathbb{N} \wedge serve \leq next)$

$P \qquad \wedge\ (serve < next)$

$[S]I \qquad \implies [ss, serve := serve+1, serve+1](serve \in \mathbb{N} \wedge next \in \mathbb{N} \wedge serve \leq next)$

After simplification (good exercise...) this reduces to $TRUE$.

Teemu Tynjälä

More General B machine – Part 1

MACHINE $N(p)$
CONSTRAINTS C
SETS St
CONSTANTS k
PROPERTIES B
VARIABLES v

More General B machine – Part 2

```
**INVARIANT** $I$
**INITIALISATION** $T$
**OPERATIONS**
    $y \longleftarrow op(x) \ =$
      **PRE** $P$
      **THEN** $S$
      **END**;
    $\dots$
**END**
```

Machine Parameters...

Machines may be given parameters to statically configure it to a different mode.. I.e. by setting buffer sizes, capacities, etc.

Machines may be given two types of parameters: sets and scalars. Sets are written in CAPITAL LETTERS and scalars in small case.

The set valued parameters must not have a type fixed. The types for scalar parameters are given in the **CONSTRAINTS** clause.

An example of a parameterized machine could be:

$$\textbf{MACHINE}\ Store(ITEM,\ capacity)$$

CONSTRAINTS

The **CONSTRAINTS** clause provides restrictions on the values of parameters. More precisely, we *must* give here

- the types for scalar parameters
- the value ranges for scalar parameters

We may also describe logical constraints on sets, as long as the type of the set is not restricted in any way (i.e. restricting cardinality of a set parameter is OK, but saying that a set parameter is subset of Naturals is not). Also, we can't say that one set parameter is subset of another set parameter.

An example of a **CONSTRAINTS** clause is:

$$\textbf{CONSTRAINTS}\ capacity \in \mathbb{N}_1 \wedge capacity \leq 4096 \wedge card(ITEM) \leq capacity$$

SETS

The sets described here are *fresh types* available for use in the rest of the machine. They are written in upper case, and we may also include enumerated sets here.

Example of a **SETS** clause is:

$$\textbf{SETS}\ REPORT = \{yes, no\};\ NAME$$

Here, $REPORT$ is the set of possible responses to a query, and $NAME$ is just a set whose type is not further specified (it may be defined later on...)

CONSTANTS

Here we define the constants used in the machine. The types of the constants *must* be defined in the **PROPERTIES** clause.

The constants may be of any types: e.g. types introduced through **SETS**, provided as machine parameters, standard types (e.g. $\mathbb{N}$), and types constructed from all of the above using e.g. Powerset constructor: $\mathbb{P}$, or Product constructor: $\times$

For example, we might have the following **CONSTANTS** clause:

$$\textbf{CONSTANTS}\ total$$

PROPERTIES

This clause describes the conditions that must hold in **SETS** and **CONSTANTS** clauses and possibly machine parameters. Additionally, it *must* give the types of the constants.

An example of a **PROPERTIES** clause is:

$$\textbf{PROPERTIES}\ card(NAME) > capacity \land total \in \mathbb{N}_1 \land capacity < total$$

New Proof Obligations.. **CONSTRAINTS**

In the general B machine, we must ensure that it is possible to find values for machine parameters that satisfy the **CONSTRAINTS** clause.

In mathematical terms:

$$\exists\, p.\, C$$

A concrete example of such a proof obligation is:

$$\exists\, capacity.\ (capacity \in \mathbb{N}_1 \wedge capacity \leq 4096)$$

Proof Obligations for **PROPERTIES**

Next step in proof obligations is to ensure that whenever machine parameters are acceptable, it must always be possible to find legitimate sets and constants which meet the **PROPERTIES** clause B.

In Mathematical terms:

$$C \implies \exists\, St, k.\ B$$

A small example of this is:

$$capacity \in \mathbb{N}_1 \wedge capacity \leq 4096 \implies$$
$$\exists\, NAME, REPORT, total.\ (card(NAME) > capacity \wedge total \in \mathbb{N}_1 \wedge total > 4096)$$

Proof Obligation for **INVARIANT**

The obligation can be expressed informally as: "Once the machine parameters, sets and constants are OK ($B \wedge C$ is satisfied), the machine should have at least one setting of its variables v which satisfies invariant I".

The difference to the simple B machine is that we have added $B \wedge C$ to the left side of the implication.

In mathematical terms:

$$B \wedge C \implies \exists\, v.\, I$$

Proof obligations for **INITIALISATIONS**

Here, once we know that machine parameters, sets and constants are OK, the initialisation statement must establish the invariant.

The difference to the simple B machine is that we have added $B \wedge C$ on the left side of the implication.

In math, we have:

$$B \wedge C \implies [T]I$$

Proof obligations for **OPERATIONS**

This is exactly the same as for the simple B machine, except that the correctness of **CONSTRAINTS** and **PROPERTIES** clauses is prepended to the assumption.

So, we have:

$$(B \wedge C \wedge I \wedge P) \implies [S]I$$

Whew... you made it to the end.... Now, we will go to Relations, but we will have an extended example of a more complex B machine later on in the slides...

Relations

A relation is always defined between two sets, say S and T. The Cartesian Product contains all the possible pairings of elements of S and T. A relation is simply just some subset of all the possible pairings.

So, all possible relations between two sets is defined as:

$$S \longleftrightarrow T = \mathbb{P}(S \times T)$$

Some specific relation, say Rel is found in this Powerset.. So we have:

$$Rel \in \mathbb{P}(S \times T)$$

Example Relation...

Say that we have:

$$PHOTOGRAPHER = \{anna, bob, chris, dave, elizabeth, francis\}$$

and,

$$CAMERA = \{canon, kodak, hasselblad, minolta, olympus, pentax\}$$

We can now define $owns \in PHOTOGRAPHER \longleftrightarrow CAMERA$ as follows:

$$\begin{aligned} owns = \{ & (anna, canon),\ (bob, canon),\ (bob, kodak),\ (chris, hasselblad), \\ & (chris, kodak),\ (chris, pentax),\ (dave, pentax),\ (elizabeth, pentax) \\ & (elizabeth, minolta)\} \end{aligned}$$

Domain of a Relation

The Domain of a Relation is given as:

$$dom(R) = \{s \mid s \in S \wedge \exists t.\ (t \in T \wedge s \mapsto t \in R)\}$$

For $owns$:

$$\begin{aligned} owns = \{ & (anna, canon),\ (bob, canon),\ (bob, kodak),\ (chris, hasselblad), \\ & (chris, kodak),\ (chris, pentax),\ (dave, pentax),\ (elizabeth, pentax) \\ & (elizabeth, minolta)\} \end{aligned}$$

we have

$$dom(owns) = \{\ anna, bob, chris, dave, elizabeth\ \}$$

Range of a Relation

The Range of a relation is given as:

$$ran(R) = \{\ t \mid t \in T\ \wedge\ \exists\, s.\ (s \in S\ \wedge\ s \mapsto t \in R)\ \}$$

For *owns*:

$$\begin{aligned} owns = \{\ & (anna, canon),\ (bob, canon),\ (bob, kodak),\ (chris, hasselblad),\\ & (chris, kodak),\ (chris, pentax),\ (dave, pentax),\ (elizabeth, pentax)\\ & (elizabeth, minolta)\} \end{aligned}$$

We have

$$ran(owns) = \{\ canon, kodak, hasselblad, minolta, pentax\ \}$$

Domain Restriction

This operation allows us to restrict the relation to certain elements of the Domain. We have

$$U \lhd R = \{ s \mapsto t \mid s \mapsto t \in R \wedge s \in U \}$$

For *owns*:

$$\begin{aligned} owns = \{ & (anna, canon), (bob, canon), (bob, kodak), (chris, hasselblad), \\ & (chris, kodak), (chris, pentax), (dave, pentax), (elizabeth, pentax) \\ & (elizabeth, minolta)\} \end{aligned}$$

We have

$$\{chris\} \lhd owns = \{(chris, kodak), (chris, hasselblad), (chris, pentax)\}$$

Domain Antirestriction

This is dual of the restriction operator... Notice, I use my own symbol here, as the Latex I use did not possess the correct symbol.. The correct symbol is triangle pointing to the left, but with the addition of a horizontal line from peak to the base...

We have

$$U \blacktriangleleft R = \{\, s \mapsto t \mid s \mapsto t \in R \,\land\, s \notin U \,\}$$

For *owns*, we actually have:

$$\{chris\} \lhd owns = \{anna, bob, dave, elizabeth\} \blacktriangleleft owns$$

Range restriction

We can also restrict the relation to contain those pairs who map only to certain elements in the Range.

We have

$$R \rhd V = \{ s \mapsto t \mid s \mapsto t \in R \wedge t \in V \}$$

For *owns*:

$$\begin{aligned} owns = \{ & (anna, canon),\ (bob, canon),\ (bob, kodak),\ (chris, hasselblad), \\ & (chris, kodak),\ (chris, pentax),\ (dave, pentax),\ (elizabeth, pentax) \\ & (elizabeth, minolta) \} \end{aligned}$$

We have:

$$owns \rhd \{kodak\} = \{ (bob, kodak),\ (chris, kodak) \}$$

Range anti-restriction

This is the dual of the range restriction... The same thing applies to the symbol used here as in the domain anti-restriction...

We have:

$$R \blacktriangleright V = \{\, s \mapsto t \mid s \mapsto t \in R \land t \notin V \,\}$$

Once again, to see the duality of the operators in *owns*, we have:

$$owns \rhd \{kodak\} = owns \blacktriangleright \{\, canon, hasselblad, minolta, pentax \,\}$$

Relational Image

This operator gives those elements in the Range, to which the elements in a certain subset of the domain map to. More formally,

$$R[U] = \{\, t \mid s \mapsto t \in R \wedge s \in U \,\}$$

For *owns*:

$$\begin{aligned} owns = \{\, &(anna, canon),\ (bob, canon),\ (bob, kodak),\ (chris, hasselblad),\\ &(chris, kodak),\ (chris, pentax),\ (dave, pentax),\ (elizabeth, pentax)\\ &(elizabeth, minolta)\} \end{aligned}$$

We have:

$$owns[\{chris, elizabeth\}] = \{\, kodak, hasselblad, minolta, pentax \,\}$$

Relational Inverse

This is an operator that ’turns around’ each ordered pair that represents a relation. In a diagrammatic form, the direction of each arrow is switched. We have:

$$R^{-1} = \{ t \mapsto s \mid s \mapsto t \in R \}$$

$$\begin{aligned} owns = \{ & (anna, canon),\ (bob, canon),\ (bob, kodak),\ (chris, hasselblad), \\ & (chris, kodak),\ (chris, pentax),\ (dave, pentax),\ (elizabeth, pentax) \\ & (elizabeth, minolta) \} \end{aligned}$$

Therefore,

$$\begin{aligned} owns^{-1} = \{ & (canon, anna),\ (canon, bob),\ (kodak, bob),\ (hasselblad, chris), \\ & (kodak, chris),\ (pentax, chris),\ (pentax, dave),\ (pentax, elizabeth) \\ & (minolta, elizabeth) \} \end{aligned}$$

Relational Composition... The final one...

This operator allows us to chain many relations into a new one. In mathematical terms:

$$R_0;R_1 = \{\, s \mapsto u \mid s \in S \wedge u \in U \wedge \exists\, t.\, (t \in T \wedge s \mapsto t \in R_0 \wedge t \mapsto u \in R_1)\}$$

Relational Composition example..

Imagine we have a new relation:

$$\begin{aligned} takes = \{ & (canon, 35mm),\ (kodak, disc),\ (hasselblad, 120\ roll), \\ & (minolta, 35mm),\ (pentax, APS),\ (olympus, 35mm)\ \} \end{aligned}$$

and $owns$:

$$\begin{aligned} owns = \{ & (anna, canon),\ (bob, canon),\ (bob, kodak),\ (chris, hasselblad), \\ & (chris, kodak),\ (chris, pentax),\ (dave, pentax),\ (elizabeth, pentax) \\ & (elizabeth, minolta)\} \end{aligned}$$

Now, $owns\ ;\ takes$ is:

$$\begin{aligned} owns\ ;\ takes = \{ & (anna, 35mm),\ (bob, 35mm),\ (bob, disc),\ (chris, 120\ roll),\ (chris, 35mm), \\ & (chris, APS),\ (dave, APS),\ (elizabeth, APS),\ (elizabeth, 35mm)\ \} \end{aligned}$$

More Complex B machine - 1

MACHINE $Access$

SETS $USER;\ PRINTER;\ OPTION;\ PERMISSION = \{\ ok, noaccess\ \}$

CONSTANTS $options$

PROPERTIES $options \in PRINTER \longleftrightarrow OPTION\ \wedge$

$\qquad \mathrm{dom}(options) = PRINTER\ \wedge\ \mathrm{ran}(options) = OPTION$

VARIABLES $access$

INVARIANT $access \in USER \longleftrightarrow PRINTER$

INITIALISATION $access := \{\}$

More Complex B machine - 2

```
OPERATIONS
    add(uu, pp) =
      PRE $uu \in USER \wedge pp \in PRINTER$
      THEN $access := access \cup \{ uu \mapsto pp \}$
      END ;
    block(uu, pp) =
      PRE $uu \in USER \wedge pp \in PRINTER$
      THEN $access := access - \{ uu \mapsto pp \}$
      END ;
    ban(uu) =
      PRE $uu \in USER$
      THEN $access := \{uu\} \blacktriangleleft access$
      END ;
```

More Complex B machine - 3

```
$\textbf{unify}(u1, u2) =$
 $\textbf{PRE}\ u1 \in USER \wedge u2 \in USER$
 $\textbf{THEN}\ access := access \cup \{ u1 \} \times access[\{ u2 \}]$
                    $\cup \{ u2 \} \times access[\{ u1 \}]$
 $\textbf{END}\ ;$
$ans \longleftarrow \textbf{optionquery}(uu, oo) =$
 $\textbf{PRE}\ uu \in USER \wedge oo \in OPTION$
 $\textbf{THEN IF}\ uu \mapsto oo \in (\ access\ ;\ options\ )$
    $\textbf{THEN}\ ans := ok$
    $\textbf{ELSE}\ ans := noaccess$
    $\textbf{END}$
 $\textbf{END}\ ;$
```

More Complex B machine - 4

```
  $nn \longleftarrow$ **printnumquery**$(pp) =$
    **PRE** $pp \in PRINTER$
    **THEN** $nn :=$ card$(\ access \rhd \{pp\}\ )$
    **END** ;
**END**
```

Study the above machine, and see how we use the relational operations in it.... This is it for today... This was a long one...

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)