T-79.232 Safety Critical Systems

Case Study 4: B Method - Functions, Sequences and Nondeterminism

Teemu Tynjälä

March 13, 2008

Functions in B - what kinds are there?

B provides a rich set of function types in its input language, and we'll describe each one in its turn. The complete list is:

- Partial functions
- Total functions
- Injective functions
- Surjective functions
- Bijective functions
- Lambda notation for functions

Partial functions

Basically, partial functions are relations, so they consist of pairs (s,t) where $s \in S \wedge t \in T$

However, we have the additional requirement, that any member of S is mapped onto *at most* one element of T.

When we allow for some elements in set S not to be mapped onto an element of T we have partial functions. In math,

$$\begin{aligned} S \mathrel{+\!\!\!\rightarrow} T \; = \; \{ f \mid & f \in S \leftrightarrow T \\ & \wedge \forall s, t_1, t_2 . \, (s \in S \wedge t_1 \in T \wedge t_2 \in T \Rightarrow \\ & ((s \mapsto t_1 \in f \wedge s \mapsto t_2 \in f) \Rightarrow t_1 = t_2)) \} \end{aligned}$$

For example, if we say $favourite\_colour \in PERSON \mathrel{+\!\!\!\rightarrow} COLOUR$, we are saying that people have one favourite colour or not at all.

Total Functions

A *total function* is a partial function between sets S and T with the added requirement that every element of S must be mapped to exactly one element of T.

In mathematics,

$$S \rightarrow T \;=\; \{ f \mid f \in S \mapsto T \;\wedge\; dom(f) \;=\; S \}$$

Now, if we declare that $favourite\_colour \in PERSON \rightarrow COLOUR$, we are stating that every person has exactly one favourite colour.

Injective Functions

A function is injective between sets S and T, if it never maps two different members of S into the same element of T. Partial injections are defined as follows:

$$\begin{aligned} S \rightarrowtail\!\!\!\!\!+ T = \{ f \mid & f \in S \mathrel{+\!\!\!\!\rightarrow} T \\ & \wedge \forall s_1, s_2, t.\, (s_1 \in S \wedge s_2 \in S \wedge t \in T \Rightarrow \\ & ((s_1 \mapsto t \in f \wedge s_2 \mapsto t \in f) \Rightarrow s_1 = s_2))\} \end{aligned}$$

For total injections (injections that are also total functions), we have:

$$S \rightarrowtail T = \{ f \mid f \in S \rightarrowtail\!\!\!\!\!+ T \wedge f \in S \rightarrow T \}$$

For example $username \in PERSON \rightarrowtail\!\!\!\!\!+ ID$ associates a username to people in such a way that no two people get the same one. Also, there are some people who have no username at all.

Surjective Functions

A function between sets S and T is *surjective* if every element of set T is reached from some element in set S.

For partial surjections we have:

$$S \mathrel{+\!\!\twoheadrightarrow} T = \{f \mid f \in S \mathrel{+\!\!\rightarrow} T \land ran(f) = T\}$$

For total surjections we have:

$$S \twoheadrightarrow T = \{f \mid f \in S \mathrel{+\!\!\twoheadrightarrow} T \land f \in S \rightarrow T\}$$

For example $attends \in PERSON \mathrel{+\!\!\twoheadrightarrow} SCHOOL$ says that every school is attended by some people, but there may be some people who do not attend any school.

Bijective Functions

Bijective functions are functions which is *total*, *injective* and *surjective*.

In mathematical terms we write,

$$S \rightarrowtail\!\!\!\twoheadrightarrow T = \{f \mid f \in S \rightarrowtail T \wedge f \in S \twoheadrightarrow T\}$$

For example, $married \in husbands \rightarrowtail\!\!\!\twoheadrightarrow wives$ says that there is exactly one wife for every husband, different husbands have different wives and every wife has a husband.

Lambda notation for functions

The lambda notation gets us closer to the 'implementation' language (= equations) of functions. It basically separates two entities - the variables in the function, and the operation that computes the function.

For example, we can define the squaring function of a natural number as follows:

$$square = \lambda x.(x \in \mathbb{N} \mid x^2)$$

The nice thing about lambda notation is that you can add conditions on variables for the operation to occur. It also allows one to separate the domain of a function to disjoint parts.

For example, the following function divides the domain $\mathbb{N}$ into two separate parts and performs a different operation on the input variable depending on whether is even or odd.

$$f = \lambda x.(x \in \mathbb{N} \wedge x \, mod \, 2 = 1 \mid 3x + 1\}$$
$$\cup \lambda x.(x \in \mathbb{N} \wedge x \, mod \, 2 = 0 \mid x/2\}$$

B machine with Functions - 1

MACHINE $Reading$

SETS $READER$; $BOOK$; $COPY$; $RESPONSE = \{yes, no\}$

CONSTANTS $copyof$

PROPERTIES $copyof \in COPY \twoheadrightarrow BOOK$

VARIABLES $hasread$, $reading$

INVARIANT

$hasread \in READER \leftrightarrow BOOK$

$\wedge\ reading \in READER \rightarrowtail\!\!\!\!\!+ COPY$

$\wedge\ (reading ; copyof) \cap hasread = \{\}$

INITIALISATION $hasread := \{\} \parallel reading := \{\}$

So, we have a machine where we have a number of COPIES of every BOOK, and every READER is reading a different COPY at any moment, as well as nobody is allowed to read a book a second time.

B machine with Functions - 2

OPERATIONS

$\textbf{start}(\ rr, cc\ )\ =$

$\textbf{PRE}$

$rr \in READER \wedge cc \in COPY \ \wedge copyof(cc) \notin hasread[\ \{rr\}\ ]$

$\wedge\ rr \notin dom(\ reading\ ) \wedge cc \notin ran(\ reading\ )$

$\textbf{THEN}\ reading\ := \ reading \cup \{\ rr \mapsto cc\ \}$

$\textbf{END}$;

$\textbf{finished}(\ rr, cc\ )\ =$

$\textbf{PRE}\ rr \in READER \wedge cc \in COPY \ \wedge cc\ =\ reading(\ rr\ )$

$\textbf{THEN}\ hasread\ :=\ hasread \cup \{\ rr \mapsto copyof(cc)\ \}$

$\|\ reading\ :=\ \{\ rr\ \} \lhd\!\!\!\!\blacktriangleleft reading$

$\textbf{END}$;

Teemu Tynjälä

B machine with Functions - 3

$resp \longleftarrow$ **precurrentquery**$(\ rr\ )\ =$
PRE $rr \in READER$
THEN
IF $rr \in dom(\ reading\ )$
THEN $resp\ :=\ yes$
ELSE $resp\ :=\ no$
END
END ;

$bb \longleftarrow$ **currentquery**$(\ rr\ )\ =$

PRE $rr \in READER \wedge rr \in dom(\ reading\ )$

THEN $bb\ :=\ copyof(\ reading(\ rr\ )\ )$

END ;

B machine with Functions - 4

$resp \longleftarrow$ **hasreadquery**$(\, rr, bb\,)\ =$

PRE $rr \in READER \wedge bb \in BOOK$

THEN

IF $bb \in hasread[\,\{\, rr \,\}\,]$

THEN $resp\ :=\ yes$

ELSE $resp\ :=\ no$

END

END

END

Sequences - 1

Sequences are very useful in modelling some situations where we have a list with a definite order. B language provides a rich set of operations that are sequence specific, which will be given in the following:

Sequences may be formed by simply listing the elements as follows:

$$prime_1 := [Wilson, Heath, Wilson, Callaghan]$$
$$prime_2 := [Thatcher, Major]$$

To concatenate two sequences we may use the $\frown$ - operator:

$$prime_1 \frown prime_2 = [Wilson, Heath, Wilson, Callaghan, Thatcher, Major]$$

Sequences - 2

Sequences may be reversed as well:

$$rev(prime_1) = [Callaghan, Wilson, Heath, Wilson]$$

If we want to append an element to the front of the list, we use the $\rightarrow$ operator:

$$Callaghan \rightarrow prime_2 = [Callaghan, Thatcher, Major]$$

Similarly we may ask the $first$ element and $tail$ of a sequence:

$$first(\ prime_1\ ) = Wilson$$

$$tail(\ prime_1\ ) = [Heath, Wilson, Callaghan]$$

Sequences - 3

We have a 'dual' operator pair for $first$ and $tail$ – namely $front$ and $last$:

$$front(\ prime_1\ ) = [Wilson, Heath, Wilson]$$
$$last(\ prime_1\ ) = Callaghan$$

Appending to the back of the sequence is accomplished by $\leftarrow$ operator:

$$prime_2 \leftarrow Blair = [Thatcher, Major, Blair]$$

To extract the first n elements of a sequence we use the $\uparrow$ operator:

$$prime_1 \uparrow 3 = [Wilson, Heath, Wilson]$$

To extract all but the first n elements of a sequence we use the $\downarrow$ operator:

$$prime_1 \downarrow 3 = [Callaghan]$$

Sequences - 4

The set of all possible sequences on a set S is defined as $seq(S)$ (in other words, the infinite union of total functions from the set $1..N$ to the set S, where N grows without bounds):

$$seq(S) = \bigcup_{N=0}^{\infty}(1..N \rightarrow S)$$

A more restrictive sequence is the injective sequence $iseq(S)$. Here we are not allowed to repeat elements of S in the sequence, but we are not forced to include every element of S there:

$$iseq(S) = seq(S) \cap \mathbb{N} \rightarrowtail\!\!\!\!\!+ S$$

Finally, a useful sequence is one where every element of set S appears exactly once $perm(S)$. For this to make sense, S has to be finite:

$$perm(S) = 1..N \rightarrowtail\!\!\!\!\twoheadrightarrow S, \text{ where } S \text{ is finite}$$

B machine with Sequences - 1

MACHINE $Results$
SETS $RUNNER$
VARIABLES $finish$
INVARIANT $finish \in iseq(\ RUNNER\ )$
INITIALISATION $finish\ :=\ [\,]$
OPERATIONS

finished$(\ rr\ )\ =$
  PRE $rr \in RUNNER \wedge rr \notin ran(\ finish\ )$
  THEN $finish\ :=\ finish \leftarrow rr$
  END ;

$rr \longleftarrow$ **query**$(\ pp\ )\ =$
  PRE $pp \in \mathbb{N}_1 \wedge pp \leq size(\ finish\ )$
  THEN $rr\ :=\ finish(\ pp\ )$
  END ;

B machine with Sequences - 2

$$\begin{array}{l} \textbf{disqualify}(\ pp\ ) \ = \\ \quad \textbf{PRE}\ pp \in \mathbb{N}_1 \wedge pp \leq size(\ finish\ ) \\ \quad \textbf{THEN}\ finish := finish \uparrow (pp-1) \frown (\ finish \downarrow pp\ ) \\ \quad \textbf{END}\ ; \end{array}$$

$$\begin{array}{l} ss \longleftarrow \textbf{medals} \ = \\ \quad ss := finish \uparrow 3 \end{array}$$

END

Nondeterminism in B machines

Nondeterminism is very important concept when modelling and verification is considered. A system has to work correctly on any input, and no matter what the sequence of correct and incorrect signals between communicating entities, a protocol must not deadlock.

B introduces ANY, $CHOICE$ and $SELECT$ statements to help in specifying non-determinism.

ANY has the least restrictions on non-determinism, $CHOICE$ narrows down a potentially huge amount of alternatives by introducing many branches of alternatives, and $SELECT$ allows one to control when particular 'branches' of alternatives are active.

ANY x WHERE Q THEN T END

x is a new variable disjoint from any other variables defined in the system. Q is a predicate which must contain the type of x and how it may/may not relate to other variables in the system. T is a B statement that can use the value of x and other variables inside the machine. Notice that the value of x that is used in T is nondeterministically picked, but the choice must respect the predicate Q.

For example,

$$\textbf{ANY}\ n\ \textbf{WHERE}\ n \in \mathbb{N}_1\ \textbf{THEN}\ total\ :=\ total \times n\ \textbf{END}$$

This statement multiplies the machine variable $total$ by some nondeterministically picked natural number.

Weakest Precondition for **ANY**

The proof obligation for the ANY statement will involve universal quantification, so that we prove that the invariant will be preserved no matter what value for x is chosen out of the possible ones:

$$[\,\mathbf{ANY}\ x\ \mathbf{WHERE}\ Q\ \mathbf{THEN}\ T\ \mathbf{END}]P \;=\; \forall x.\ (Q \Rightarrow [T]P)$$

For example, we see that the following precondition is identically true:

$$[\,\mathbf{ANY}\ n\ \mathbf{WHERE}\ n \in \mathbb{N} \wedge n < 50\ \mathbf{THEN}\ total := n \times 2](total < 100)$$
$$\forall n\,.((n \in \mathbb{N} \wedge n < 50) \Rightarrow [total := n \times 2](total < 100))$$
$$\forall n\,.((n \in \mathbb{N} \wedge n < 50) \Rightarrow (n \times 2 < 100))$$
$$\forall n\,.((n \in \mathbb{N} \wedge n < 50) \Rightarrow (n < 50))$$

$\textbf{ANY}\ e\ \textbf{WHERE}\ e \in S\ \textbf{THEN}\ x\ :=\ e\ \textbf{END}$

This construct is very heavily used in B, and sometimes it is called *nondeterministic assignment*.

It has a special symbol in B, written as follows: $x :\in S$

The proof obligation for this is derived from the general ANY clause and it's the following:

$$[x :\in S]P \ = \ \forall x \,.\, (x \in S \Rightarrow P) \quad x \text{ not free in } S$$

For example:

$$\begin{aligned}[x \in S](x \neq 3) \ &= \ \forall x.\, (x \in S \Rightarrow x \neq 3) \\ &= 3 \notin S\end{aligned}$$

CHOICE S OR T OR ... OR U END

This allows us to make a non-deterministic choice of a statement to execute. Each S, T,... is a valid B statement, and we could use such a construct e.g. to send a correct message or an incorrect message in a protocol.

For the proof obligation we get:

$$[\,\textbf{CHOICE}\ S\ \textbf{OR}\ T\ \textbf{END}\,]P \;=\; [S]P \wedge [T]P$$

For example, the following weakest precondition is identically false:

$$[\,\textbf{CHOICE}\ x := 3\ \textbf{OR}\ x := 5\ \textbf{END}\,](x = 4)$$

SELECT statement

This statement allows us to control which 'branches' of the options are active at one time, rather than having all branches active as in the **CHOICE** statement. The optional **ELSE** clause will be executed if none of the conditionals Q_n are satisfied. The syntax is as follows:

SELECT Q_1 **THEN** T_1
WHEN Q_2 **THEN** T_2
WHEN $\ldots$
WHEN Q_n **THEN** T_n
ELSE V
END

Weakest Precondition for **SELECT**

$$\left[\begin{array}{l}\textbf{SELECT } Q_1 \textbf{ THEN } T_1 \\ \textbf{WHEN } Q_2 \textbf{ THEN } T_2 \\ \ldots \\ \textbf{WHEN } Q_n \textbf{ THEN } T_n \\ \textbf{END}\end{array}\right] P = \left(\begin{array}{l} Q_1 \Rightarrow [T_1]P \\ \wedge\, Q_2 \Rightarrow [T_2]P \\ \ldots \\ \wedge\, Q_n \Rightarrow [T_n]P \end{array}\right)$$

B machine with Nondeterminism - 1

MACHINE *Jukebox*

SETS $TRACK$

CONSTANTS $limit$

PROPERTIES $limit \in \mathbb{N}_1$

VARIABLES $credit,\ playset$

INVARIANT $credit \in \mathbb{N} \wedge credit \leq limit \wedge playset \subseteq TRACK$

INITIALISATION $credit := 0 \parallel playset := \{\}$

B machine with Nondeterminism - 2

OPERATIONS

  pay(cc) $=$

   PRE $cc \in \mathbb{N}_1$

   THEN $credit := min(\{ credit + cc, limit \})$

   END ;

  select(tt) $=$

   PRE $credit > 0 \land tt \in TRACK$

   THEN

    CHOICE $credit := credit - 1 \parallel playset := playset \cup \{ tt \}$

    OR $playset := playset \cup \{ tt \}$

    END

   END ;

B machine with Nondeterminism - 3

$tt \longleftarrow$ **play** $=$

PRE $playset \neq \{\}$

THEN

ANY tr

WHERE $tr \in playset$

THEN $tt := tr \parallel playset := playset - \{ tr \}$

END

END ;

B machine with Nondeterminism - 4

penalty $=$

SELECT $credit > 0$ **THEN** $credit := credit - 1$

WHEN $playset \neq \{\}$ **THEN**

ANY pp

WHERE $pp \in playset$

THEN $playset := playset - \{ pp \}$

END

ELSE *skip*

END

END

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)