System Modelling and Design
Machine Composition

Revision: 1.3, August 26, 2005

Ken Robinson

26th August 2005

mailto::k.robinson@unsw.edu.au

Contents

1 INCLUDES 2
  1.1 Include Once Only ......................................................... 2
  1.2 PROMOTES ................................................................. 3
  1.3 EXTENDS .................................................................. 3
  1.4 SEES ........................................................................ 3
  1.5 Comparison between INCLUDES and SEES ................................. 4
  1.6 USES ....................................................................... 4
  1.7 Comparison between SEES and USES ....................................... 4
  1.8 Inconsistent use of operations ................................................ 4
  1.9 Global Sets, Constants and Variables ...................................... 5

2 Examples 5
  2.1 A Date Machine ............................................................... 5
  2.2 Example: A Time Machine .................................................... 6

Objectives of this Lecture

To introduce the constructs offered by B Method (B) for composing machines.
To discuss the rules and restrictions imposed by the different composition constructs.
To give some examples of the use of each composition construct.
1 INCLUDES

The INCLUDES mechanism allows a machine to be embedded in another machine with read/write access to the variables of the included machine.

\[
\begin{align*}
\text{MACHINE A} & \quad \text{(formal-params-A)} \\
\text{MACHINE B} & \quad \text{(formal-params-B)} \\
\text{MACHINE C} & \quad \text{(formal-params-C)} \\
\text{INCLUDES A} & \quad \text{(actual-params-A)} \\
\text{INCLUDES B} & \quad \text{(actual-params-B)}
\end{align*}
\]

- The state of machine \( A \) is prepended to the state of machine \( B \).
- Machine \( B \) has full access to the constants, sets, variables and operations of \( A \).
- The state of machine \( A \) can be modified by machine \( B \), but only by using operations of \( A \).
- Within a single development, any single machine may be included only once.

- INCLUDES is transitive: machine \( A \) is also included in machine \( C \).
- The operations of an included machine are not automatically exported by the including machine. See PROMOTES and EXTENDS.
- The parameters of an included machine must be instantiated at the point of inclusion. Often the instantiation is to parameters of the including machine.

1.1 Include Once Only

Development 1

\[
\begin{align*}
\text{MACHINE A} & \quad \text{INCLUDES} \quad \text{MACHINE B} \\
\text{MACHINE B} & \quad \text{INCLUDES} \quad \text{MACHINE C} \\
\text{MACHINE C} & \quad \text{INCLUDES} \quad \text{MACHINE E}
\end{align*}
\]

Development 2

\[
\begin{align*}
\text{MACHINE A} & \quad \text{INCLUDES} \quad \text{MACHINE B} \\
\text{MACHINE B} & \quad \text{INCLUDES} \quad \text{MACHINE D} \\
\text{MACHINE D} & \quad \text{INCLUDES} \quad \text{MACHINE E}
\end{align*}
\]

\textit{Developments 1 & 2 are valid.}

Development 3

\[
\begin{align*}
\text{MACHINE A} & \quad \text{INCLUDES} \quad \text{MACHINE B} \\
\text{MACHINE B} & \quad \text{INCLUDES} \quad \text{MACHINE C} \\
\text{MACHINE C} & \quad \text{INCLUDES} \quad \text{MACHINE E}
\end{align*}
\]

Development 4

\[
\begin{align*}
\text{MACHINE A} & \quad \text{INCLUDES} \quad \text{MACHINE B} \\
\text{MACHINE B} & \quad \text{INCLUDES} \quad \text{MACHINE D} \\
\text{MACHINE D} & \quad \text{INCLUDES} \quad \text{MACHINE E}
\end{align*}
\]

\textit{Development 4 is invalid.}
Development 3 is invalid, as machine A is included twice.

Within a single development, any single machine may be included only once. Different instantiations of a machine may be created by renaming using the following dot notation:

machinename.modifier

All the variables and operations of the machine are similarly qualified, but not the sets and constants.

1.2 PROMOTES

PROMOTES is used in the including machine to promote an operation of the included machine to the interface of the including machine.

Clarification: all operations of the included machine are available for the including machine to use, but they do not automatically become operations in the interface of that machine. If machine B INCLUDES machine A, then without promotion the only operations in the interface of machine B will be the operations defined in B.

1.3 EXTENDS

EXTENDS is used in place of INCLUDES to include a machine and promote all operations of the included machine.

If machine B EXTENDS machine A then the interface of the new machine will contain all the operations defined in both A and B.

1.4 SEES

SEES provides readonly access to the SETS, CONSTANTS and VARIABLES of the seen machine.

- A machine may be seen by any number of machines in a development.
- SEES is not transitive: if B SEES A, and C INCLUDES or SEES B, then C does not automatically see A.
- The variables of a seen machine may be referenced in the seeing machine, but not in the invariant.
- Operations of the seen machine that do not change the seen machine’s state may be used by the seeing machine. Such operations provide mathematical functions.
- The parameters of a seen machine are not instantiated by the seeing machine.
- A seen machine must be implemented, and imported into the development.
1.5 Comparison between INCLUDES and SEES

<table>
<thead>
<tr>
<th>INCLUDES</th>
<th>SEES</th>
</tr>
</thead>
<tbody>
<tr>
<td>read/write access</td>
<td>readonly access</td>
</tr>
<tr>
<td>parameters must be instantiated</td>
<td>parameters must not be instantiated</td>
</tr>
<tr>
<td>all operations accessible</td>
<td>only functional operations accessible</td>
</tr>
<tr>
<td>transitive</td>
<td>not transitive</td>
</tr>
<tr>
<td>may not be implemented</td>
<td>must be implemented</td>
</tr>
</tbody>
</table>

1.6 USES

USES provides read-only access by the using machine to the variables of the used machine.

- The using machine may reference the variables of the used machine in its invariant. This is sometimes convenient, but is insecure and gives rise to the following requirement.
- The used and using machines must be included in a larger machine.

1.7 Comparison between SEES and USES

<table>
<thead>
<tr>
<th>SEES</th>
<th>USES</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>readonly</td>
</tr>
<tr>
<td>parameters must not be instantiated</td>
<td>must not be instantiated</td>
</tr>
<tr>
<td>variables not referenced in invariant</td>
<td>may be referenced in invariant</td>
</tr>
<tr>
<td>operations only functional accessible</td>
<td>only functional accessible</td>
</tr>
<tr>
<td>not transitive</td>
<td>not transitive</td>
</tr>
<tr>
<td>must be implemented</td>
<td>cannot be implemented</td>
</tr>
</tbody>
</table>

1.8 Inconsistent use of operations

It is clear that the substitution \( xx, xx := xx + 1, xx - 1 \) is inconsistent.

Thus B insists that the variables on the lhs of a multiple substitution are disjoint.

Of course, \( xx, xx := xx + 1, xx - 1 \) is equivalent to \( xx := xx + 1 \mid \mid xx := xx - 1 \), so concurrently composed substitutions must modify different components of the state.

Any single operation of a machine that changes the state will generate a proof obligation that the operation restores the invariant. Proof of this obligation assumes that the only changes are those made by this operation. Composing any two operations in parallel, each of which independently maintains the invariant, can be expected, in general, to produce inconsistent state changes, and is therefore forbidden.

For the above reasons the following restrictions apply to use of operations:

1. Operations of a single machine may not be referenced within other operations of the same machine. If this is desired, then it can be achieved, when it is safe, by splitting the machine into two, including one machine in the other and using the operation of the included machine in the including machine.

Notice that splitting a machine into two machines demonstrates that the original single machine contains two independent sub-states, and operations on each of those sub-states cannot interfere.
2. Although a machine $B$ may have access to the operations of another machine $A$, for example via includes, machine $B$ may not compose two or more operations of $A$ concurrently.

1.9 Global Sets, Constants and Variables

In the simple library case study we needed to have access to a global set $BOOK$, the set of all books. We used a machine parameter, but there was a footnote saying that this was not the correct way of representing a global set.

The reason is that machine parameters are instantiated at the point where the machine is introduced into a development.

Thus the headers
MACHINE A(BOOK) MACHINE B(BOOK)

do not guarantee that both $A$ and $B$ see the same set. Parameter names are dummy names and do not imply anything about the identity of the actual values used to instantiate the parameter.

The correct way to represent a universal set is to embed it in a machine, for example

MACHINE Book_TYPE (maxbook)
CONSTRAINTS maxbook $\in \mathbb{N}_1$
SETS BOOK
PROPERTIES card (BOOK) = maxbook
END

and to see that machine using SEES.

2 Examples

2.1 A Date Machine

MACHINE Date (maxDate)
CONSTRAINTS maxDate $\in \mathbb{N}_1$
VARIABLES today
INVARIANT today $\in$ DATE
INITIALISATION today := DATE

OPERATIONS
newday $	riangleq$
  PRE today $\neq$ maxDate
  THEN today := today + 1
END
DEFINITIONS DATE $	riangleq$ 0 . . maxDate
END
2.2 Example: A Time Machine

MACHINE Time
SEES Bool_TYPE
SETS TIME
CONSTANTS
  FUTURE ,
  PAST ,
  BigCrunch ,
  BigBang

PROPERTIES

The future, modelled by $FUTURE$, is a relation that relates any time to all times in its future.

$FUTURE \in TIME \rightarrow TIME$ ∧

The past, modelled by $PAST$, is simply the inverse of the future.

$PAST = FUTURE^{-1}$ ∧

The relation $FUTURE$ is total: any two unequal times $t_1$ and $t_2$ must be related in $FUTURE$.

$\forall (t_1,t_2). ( t_1 \in TIME \land t_2 \in TIME \land t_1 \neq t_2 \Rightarrow$

$t_1 \rightarrow t_2 \in FUTURE \lor t_2 \rightarrow t_1 \in FUTURE )$ ∧

$FUTURE$ is irreflexive: no time is in its own future. We model this by saying that $FUTURE$ and the identity relation on $TIME$, $id(TIME)$, are disjoint.

$FUTURE \cap id(TIME) = \{\}$ ∧

$FUTURE$ is anti-symmetric: if $t_2$ is in the future of $t_1$ then $t_1$ is not in the future of $t_2$.

$FUTURE \cap PAST = \{\}$ ∧

$FUTURE$ is transitive, if $t_2$ is in the future of $t_1$ and $t_3$ is in the future of $t_2$ then $t_3$ is in the future of $t_1$

$\forall (t_1,t_2,t_3). ( t_1 \in TIME \land t_2 \in TIME \land t_3 \in TIME \land ( t_2 \in FUTURE [ \{ t_1 \} ] \land t_3 \in FUTURE [ \{ t_2 \} ] ) \Rightarrow$

$t_3 \in FUTURE [ \{ t_1 \} ] )$ ∧

The following cosmology is concerned with time in a finite universe.

There are two extreme times, $BigBang$ and $BigCrunch$

$BigBang \in TIME \land BigCrunch \in TIME$ ∧

they are not equal

$BigBang \neq BigCrunch$ ∧

no time is in the future of $BigCrunch$,
BigCrunch \notin \text{dom} (\text{FUTURE}) \land \\

and no time is in the past of \text{BigBang}.

\text{BigBang} \notin \text{dom} (\text{PAST}) \land \\

All times, except \text{BigBang}, are in the future of \text{BigBang},

\text{FUTURE} [ \{ \text{BigBang} \} ] = \text{TIME} - \{ \text{BigBang} \} \land \\

and all times, except \text{BigCrunch}, are in the past of \text{BigCrunch}.

\text{PAST} [ \{ \text{BigCrunch} \} ] = \text{TIME} - \{ \text{BigCrunch} \} \\

The machine state has one variable, \text{currenttime}, that records the last time given by an operation of this machine.

\text{VARIABLES} \ \text{currenttime} \\
\text{INVARIANT} \ \text{currenttime} \in \text{TIME} \\
\text{INITIALISATION} \ \text{currenttime} :\in \text{TIME} \\

\text{OPERATIONS} \\
Operation \text{Clock} \ gives \ a \ time \ that \ is \ in \ the \ future \ of \ the \ current \ value \ of \ \text{currenttime}.

\text{time} \leftarrow \text{Clock} \triangleq \\
\text{PRE} \ \text{currenttime} \neq \text{BigCrunch} \\
\text{THEN ANY now} \\
\ \ \ \ \ \ \text{WHERE now} \in \text{FUTURE} [ \{ \text{currenttime} \} ] \\
\ \ \ \ \ \ \ \ \text{THEN} \ \text{time} := \text{now} \ \| \ \text{currenttime} := \text{now} \\
\ \ \ \ \ \ \ \ \text{END} \\
\ \ \ \ \ \ \ \ \text{END} ;

Operation \text{Tick} \ returns \ the \ next \ time \ tick \ after \ the \ current \ time. \ This \ is \ modelled \ as \ the \ “least” \ time \ in \ the \ future \ of \ \text{currenttime}.

\text{time} \leftarrow \text{Tick} \triangleq \\
\text{PRE} \ \text{currenttime} \neq \text{BigCrunch} \\
\text{THEN ANY now} \\
\ \ \ \ \ \ \text{WHERE now} \in \text{FUTURE} [ \{ \text{currenttime} \} ] \land \\
\ \ \ \ \ \ \ \ \text{FUTURE} [ \{ \text{currenttime} \} ] \cap \text{PAST} [ \{ \text{now} \} ] = \{\} \\
\ \ \ \ \ \ \ \ \text{THEN} \ \text{time} := \text{now} \ \| \ \text{currenttime} := \text{now} \\
\ \ \ \ \ \ \ \ \text{END} \\
\ \ \ \ \ \ \ \ \text{END} ;

Operation \text{Later}(t1,t2) \ returns \ TRUE \ if \ t1 \in \text{FUTURE}[[t2]] \ and \ FALSE \ otherwise.
\[
\text{later} \leftarrow \text{Later}(t1, t2) \equiv \\
\begin{array}{ll}
\text{PRE} & t1 \in \text{TIME} \land t2 \in \text{TIME} \\
\text{THEN} & \text{later} := \text{bool}(t1 \in \text{FUTURE} \{t2\}) \\
\end{array}
\]
END;

Operation Earlier(t1, t2) returns TRUE if \( t1 \in PAST[t2] \) and FALSE otherwise.

\[
\text{earlier} \leftarrow \text{Earlier}(t1, t2) \equiv \\
\begin{array}{ll}
\text{PRE} & t1 \in \text{TIME} \land t2 \in \text{TIME} \\
\text{THEN} & \text{earlier} := \text{bool}(t1 \in \text{PAST} \{t2\}) \\
\end{array}
\]
END