Lecture 5

The B Method

Structuring Mechanisms for B Specifications - SEES and USES

Lecture Outline

- Structuring Specifications with SEES and USES
- The SEES mechanism
- The USES mechanism

References

- [1] Abrial, J.-R., *The B Book - Assigning Programs to Meanings*, Cambridge University Press, 1996. (chapter 7)
- [2] Schneider, S., *The B-Method - An Introduction*, Palgrave Macmillan, Cornerstones of Computing series, 2001. (chapter 11)
- [3] Clearsy System Engineering, *AtelierB home page* `http://www.atelierb.eu/en/`
- [4] Clearsy System Engineering, *B Method home page* `http://www.methode-b.com/en/`

Structuring Specifications with SEES and USES

- The AMN `INCLUDES` clause offers a structuring mechanism by which an included machine is considered to be part of and completely under the control of the including machine
- B provides another two ways of structuring large specifications, namely the SEES and USES clauses, both allowing forms of *read-only* access between machines
- These two mechanisms enable a separate definition of a part of the state when several other machines require knowledge of it
  - Since read access does not modify the state of the machine being read, a machine can be accessed in this way by several other machines
- SEES is a special case of USES, that occurs most often in practice
  - The difference between SEES and USES is that USES allows expressing relations between the states of the used and using machine, while SEES does not

The SEES Mechanism

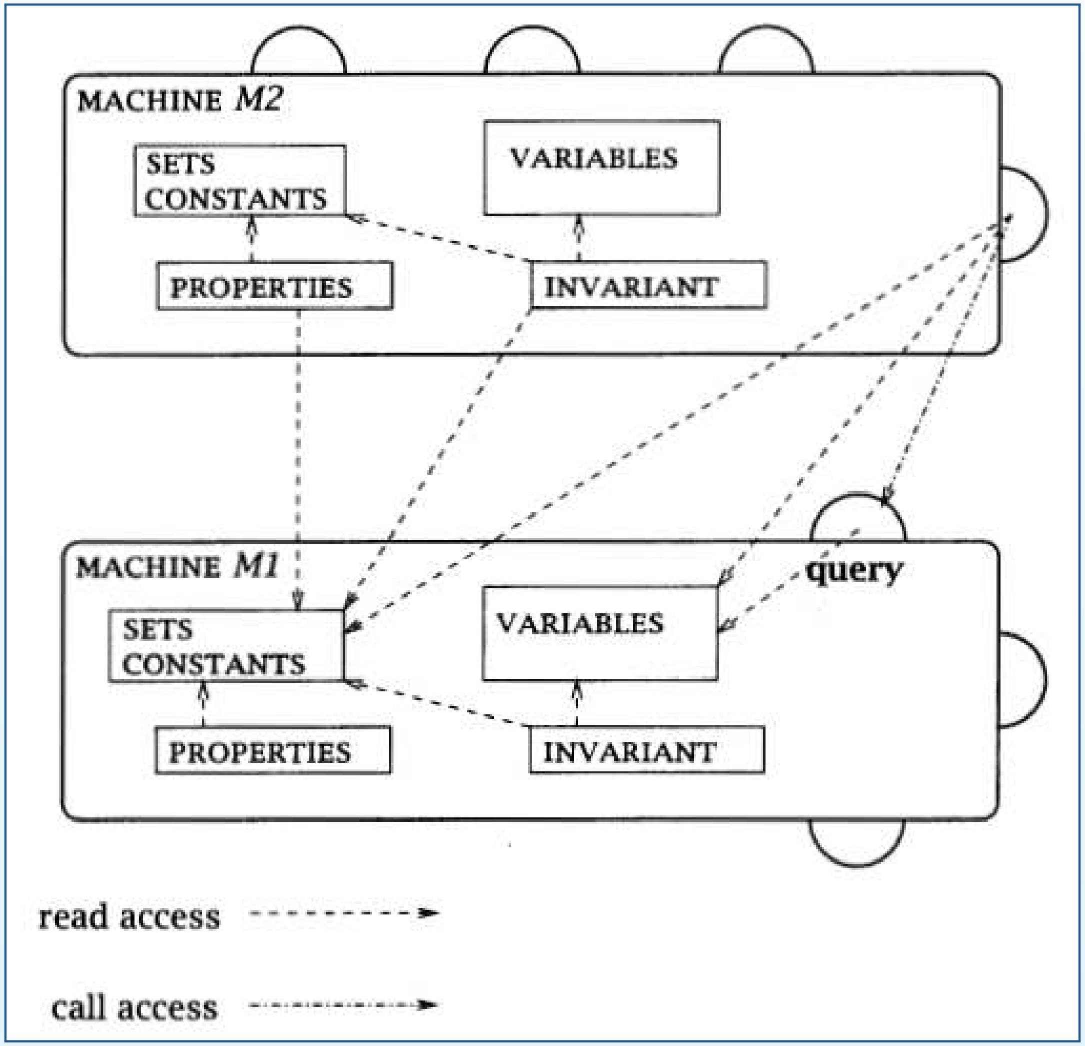
- Ensured by the AMN `SEES` clause
  - A machine `M2` can be provided with read access to a previously built and proved machine `M1`, by means of a `SEES` statement within the definition of `M2`
- If `M1` has parameters then those are not accessible by `M2` and neither are their instantiations by a machine including `M1` within the overall specification
- The sets and constants of `M1` are visible to `M2` and available for use within its properties, invariant, initialisation and operations
- The variables of `M1` are available in read mode within the initialisation and operations of `M2`, but they cannot be referred by its invariant
  - Since `M1` is not under the control of `M2`, its state may be changed by calls from another machine, say `M3`, that may lead to breaking `M2`’ invariant in case this referred to state variables from `M1`

The SEES Mechanism (cont.)

  - Since `M1` is not under control of `M2`, the values of its state variables may change between two consecutive readings by `M2`
- If `M2` sees `M1` and the latter includes some other machines, then the information in these machines (included information for `M1`) will be accessible to `M2`, just as the native information of `M1` is
- The `SEES` relation is not transitive
  - If `M2` sees `M1` and `M3` sees (or includes `M2`), then `M1` is not seen in `M3` by default
  - If `M3` requires read visibility to `M1`, then it should include its own `SEES  M1` clause in this purpose
- When `M2` sees `M1`, they are regarded as distinct machines, thus the latter is not part of the former, as opposed to what happens in case of the inclusion mechanism

The SEES Mechanism (cont.)

- Graphical representation of the relation between machines related by the SEES mechanism (`M2 SEES M1`)



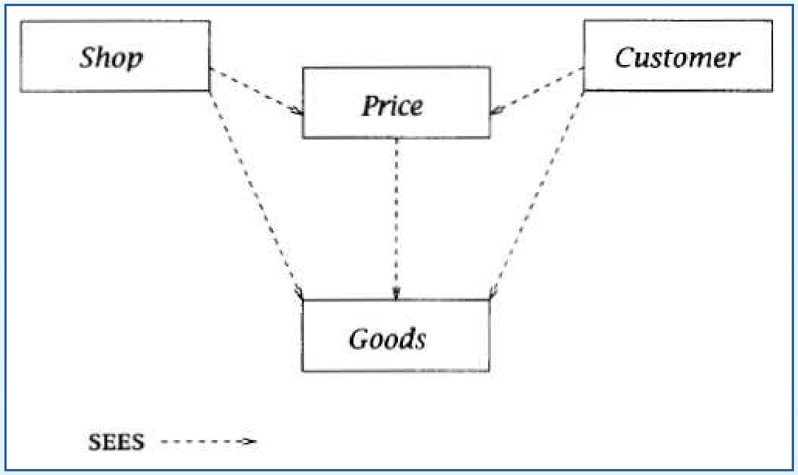
Use of SEES

- For introducing deferred or enumerated sets that should be widely available
  - *Example*: Specification of a shop, containing the distinct machines
    - `Price` - to record products' prices
    - `Shop` - to record takings in the shop
    - `Customer` - to record customers' takings
  - *Problem*: Where to put a set Goods that all machines should make use of?
    - If declared in each machine, the overal development will have multiple copies of it
    - If provided in an included machine, it will only be visible to the including one
  - *Solution*: In a new machine `Goods`, seen by the other three

```
MACHINE
    Goods
SETS
    GOODS
END
```

Use of SEES (cont.)

- For factoring out a part of a system's state which is required by several other parts
  - *Problem*: Both `Shop` and `Customer` need read access to the prices of goods
  - *Solution*: Isolate the price information in a new machine `Price`, seen by the other two
  - Resulting machine architecture



- Machine `Price`

MACHINE
  Price
SEES
  Goods
VARIABLES
  price
INVARIANT
  $price \in GOODS \rightarrow \mathbb{N}_1$
INITIALISATION
  $price :\in GOODS \rightarrow \mathbb{N}_1$
OPERATIONS
  setprice$(gg, pp) \mathrel{\widehat{=}}$
  PRE
    $gg \in GOODS \wedge pp \in \mathbb{N}_1$
  THEN
    $\mathrm{price}(gg) := pp$
  END;

  $pp \leftarrow$ pricequery$(gg) \mathrel{\widehat{=}}$
  PRE
    $gg \in GOODS$
  THEN
    $pp := price(gg)$
  END

END

Use of SEES (cont.)

- Machine Shop

$$
\begin{array}{l}
\textbf{MACHINE} \\
\quad \mathit{Shop} \\
\textbf{SEES} \\
\quad \mathit{Goods},\ \mathit{Price} \\
\textbf{VARIABLES} \\
\quad \mathit{takings} \\
\textbf{INVARIANT} \\
\quad \mathit{takings} \in \mathbb{N} \\
\textbf{INITIALISATION} \\
\quad \mathit{takings} := 0 \\
\textbf{OPERATIONS} \\
\quad \mathrm{sale}(gg) \mathrel{\widehat{=}} \\
\quad \textbf{PRE} \\
\qquad gg \in \mathit{GOODS} \\
\quad \textbf{THEN} \\
\qquad \mathit{takings} := \mathit{takings} + \mathit{price}(gg) \\
\quad \textbf{END}; \\
\\
\quad tt \leftarrow \mathrm{total} \mathrel{\widehat{=}} \\
\quad \textbf{BEGIN} \\
\qquad tt := \mathit{takings} \\
\quad \textbf{END} \\
\\
\textbf{END}
\end{array}
$$

- Machine `Customer`

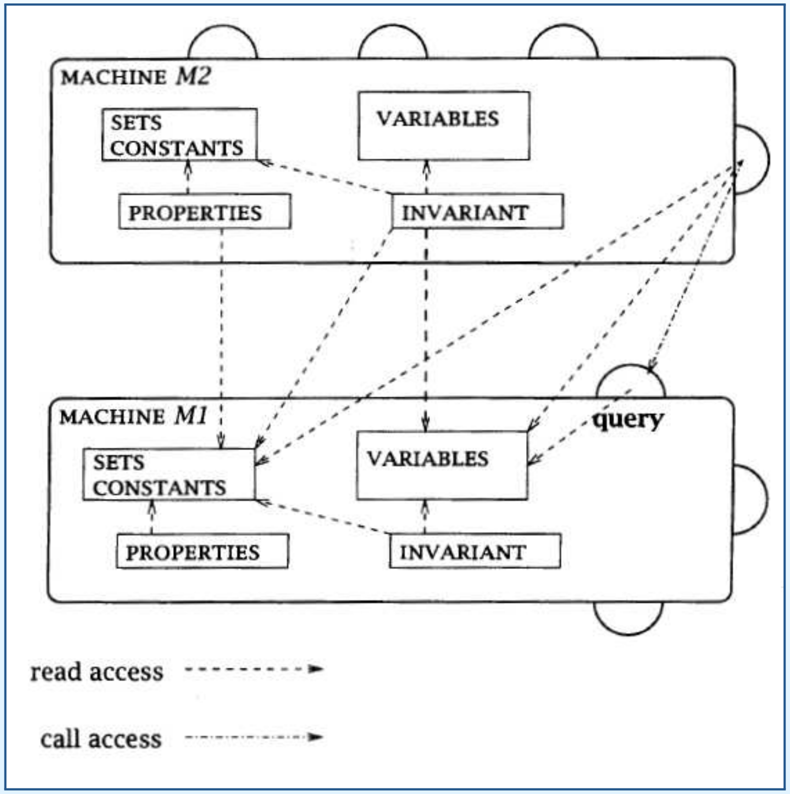
$$
\begin{array}{l}
\textbf{MACHINE} \\
\quad \mathit{Customer} \\
\textbf{SEES} \\
\quad \mathit{Goods},\ \mathit{Price} \\
\textbf{CONSTANTS} \\
\quad \mathit{limit} \\
\textbf{PROPERTIES} \\
\quad \mathit{limit} \in \mathit{GOODS} \rightarrow \mathbb{N}_1 \\
\textbf{VARIABLES} \\
\quad \mathit{purchases} \\
\textbf{INVARIANT} \\
\quad \mathit{purchases} \subseteq \mathit{GOODS} \\
\textbf{INITIALISATION} \\
\quad \mathit{purchases} := \emptyset \\
\textbf{OPERATIONS} \\
\quad pp \leftarrow \mathrm{buy}(gg) \mathrel{\widehat{=}} \\
\quad \textbf{PRE} \\
\qquad gg \in \mathit{GOODS} \wedge \mathit{price}(gg) \leq \mathit{limit}(gg) \\
\quad \textbf{THEN} \\
\qquad \mathit{purchases} := \mathit{purchases} \cup \{gg\} \ || \\
\qquad pp := \mathit{price}(gg) \\
\quad \textbf{END} \\
\\
\textbf{END}
\end{array}
$$

The USES Mechanism

- Ensured by the AMN `USES` clause
  - A machine `M2` wanting to use a previously built and proved machine `M1` does that by means of an `USES M1` clause in its definition
- The `USES` mechanism is a generalization of `SEES`
- The only difference between the access enabled by `USES` and the one enabled by `SEES` is that, in the former case, the invariant of the using machine may refer to state variables of the used one
- Since a used machine `M1` is not under the control of the using one, `M2`, calls to operations of the former may breake the invariant of the latter
- The requirement that the invariant of `M2` should be preserved by any operation of `M1` is not a proof obligation of either `M1` (defined independently of `M2`) or `M2` (that does not have any control over `M1`)

The USES Mechanism (cont.)

- The discharge of such proof obligations should happen within a machine `M3`, including both `M1` and `M2` within the overal development
- Graphical representation of the relation between machines related by `USES` (`M2 USES M1`)



USES example - Registrar System

- Functionalities of a registrar
  - Recording births and deaths
  - Recording marriages
- Machine `Life` (recording births and deaths)

MACHINE
    Life
SETS
    $PERSON; SEX = \{boy, girl\}$
VARIABLES
    $male, female$
INVARIANT
    $male \subseteq PERSON \wedge female \subseteq PERSON \wedge male \cap female = \emptyset$
INITIALISATION
    $male := \emptyset \parallel female := \emptyset$

OPERATIONS

    born$(nn, ss) \mathrel{\hat=}$
    PRE
        $nn \in PERSON \wedge nn \notin (male \cup female) \wedge ss \in SEX$
    THEN
        IF $ss = boy$
        THEN
            $male := male \cup \{nn\}$
        ELSE
            $female := female \cup \{nn\}$
        END
    END;

    die$(nn) \mathrel{\hat=}$
    PRE
        $nn \in PERSON \wedge nn \in male \cup female$
    THEN
        IF $nn \in male$
        THEN
            $male := male - \{nn\}$
        ELSE
            $female := female - \{nn\}$
        END
    END

END

USES example - Registrar System (cont.)

- Machine `Marriage` (recording marriages)

MACHINE
  $Marriage$
USES
  $Life$
VARIABLES
  $marriage$
INVARIANT
  $marriage \in male \rightarrowtail\!\!\!\!\!\rightarrow\!\!\!\!\!\!\shortmid\; female$
INITIALISATION
  $marriage := \emptyset$
OPERATIONS
  $\mathbf{wed}(mm, ff) \mathrel{\widehat{=}}$
  PRE
    $mm \in male \wedge ff \in female \wedge$
    $mm \notin \mathbf{dom}(marriage) \wedge ff \notin \mathbf{ran}(marriage)$
  THEN
    $\mathrm{marriage}(mm) := ff$
  END;

  $\mathbf{part}(mm, ff) \mathrel{\widehat{=}}$
  PRE
    $mm \in male \wedge ff \in female \wedge$
    $marriage(mm) = ff$
  THEN
    $marriage := marriage - \{mm \mapsto ff\}$
  END;

  $pp \leftarrow \mathbf{partner}(nn) \mathrel{\widehat{=}}$
  PRE
    $nn \in \mathbf{dom}(marriage) \cup \mathbf{ran}(marriage)$
  THEN
    IF $nn \in \mathbf{dom}(marriage)$
    THEN $pp := marriage(nn)$
    ELSE $pp := marriage^{-1}(nn)$
    END
  END

END

USES example - Registrar System (cont.)

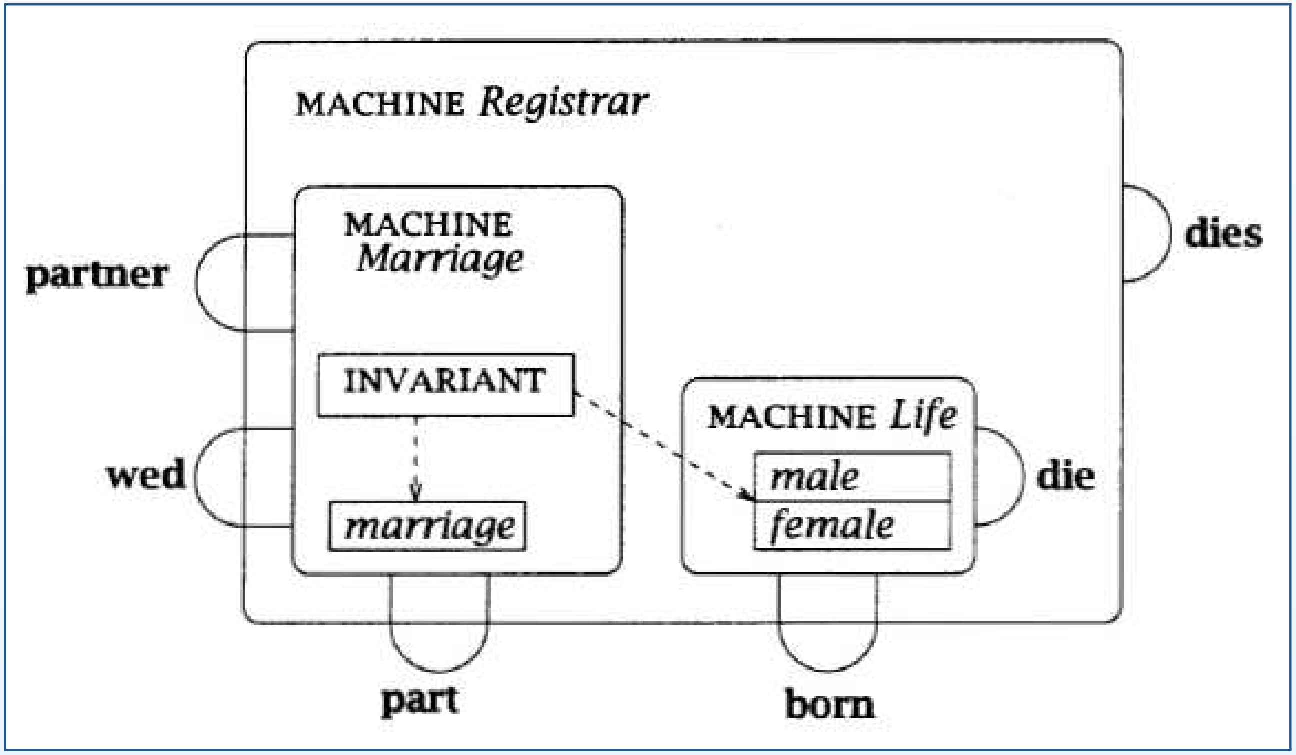
- Machine `Registrar` (final system)

```
MACHINE
    Registrar
EXTENDS
    Marriage
INCLUDES
    Life
PROMOTES
    born
OPERATIONS
    dies(nn) =
    PRE
        nn ∈ male ∪ female
    THEN
        die(nn) ||
        IF (nn ∈ dom(marriage)) THEN part(nn, marriage(nn))
        ELSIF (nn ∈ ran(marriage)) THEN part(marriage ⁻¹ (nn),nn)
        END
    END

END
```

USES example - Registrar System (cont.)

- Registrar architecture



General Usage Pattern

MACHINE	MACHINE
$M(X, x)$	$M_1(X_1, x_1)$
CONSTRAINTS	CONSTRAINTS
C	C_1
USES	SETS
M_1	S_1;
SETS	$T_1 = \{a_1, b_1\}$
S;	(ABSTRACT_)CONSTANTS
$T = \{a, b\}$	c_1
(ABSTRACT_)CONSTANTS	PROPERTIES
c	P_1
PROPERTIES	(CONCRETE_)VARIABLES
P	v_1
(CONCRETE_)VARIABLES	INVARIANT
v	I_1
INVARIANT	ASSERTIONS
$I \wedge I'$	J_1
ASSERTIONS	INITIALIZATION
J	U_1
INITIALIZATION	OPERATIONS
U	...
OPERATIONS	END
$op \hat{=}$	
PRE	
Q	
THEN	
V	
END;	
...	
END	

Usage Proof Obligations

- The invariant of the using machine `M` has been broken into I, supposed to be independent of the state of $\mathtt{M}_1$ and I', supposed to depend on the state of $\mathtt{M}_1$
- Following are, from top to bottom, the corresponding proof obligations for initialization, assertions and operations

$$\overbrace{A_1 \wedge B_1 \wedge C_1 \wedge P_1}^{Used} \wedge \overbrace{A \wedge B \wedge C \wedge P}^{Using} \Rightarrow [U]I$$

$$\overbrace{A_1 \wedge B_1 \wedge C_1 \wedge P_1 \wedge I_1 \wedge J_1}^{Used} \wedge \overbrace{A \wedge B \wedge C \wedge P \wedge I \wedge I'}^{Using} \Rightarrow J$$

$$\overbrace{(A_1 \wedge B_1 \wedge C_1 \wedge P_1 \wedge I_1 \wedge J_1)}^{Used} \wedge \overbrace{(A \wedge B \wedge C \wedge P \wedge I \wedge I' \wedge J \wedge Q)}^{Using} \Rightarrow [V]\,I$$

- Only the part of the invariant not concerned with the state of the used machine can be proved locally, the other one will be proved at the level of the machine in which the two will be simultaneously included

Usage Proof Obligations (cont.)

- Proof obligations are very similar to those given for a simple machine, with the difference that the machine $\mathtt{M}_1$ (except its operations) is put together with machine $\mathtt{M}$
- Contextual abbreviations A, A_1, B, B_1 are defined as in Lecture 2
- Proof obligations for the SEES clause are similar