# Lecture 2

## The B Method

*Introduction to the Abstract Machine Notation (AMN)*
*and Generalised Substitution Language (GSL)*

# Lecture Outline

- References
- B Method Overview
- Formal Specification
- Abstract Machines
- The Statics - State
- The Dynamics - Operations
- Operations’ Specification - Before/After Predicates
- Proof Obligations
- Operations’ Specification - Substitutions
- Pre-conditioned Substitution
- Machine Parameters and Initialization
- Operation Input Parameters
- Operation Output Parameters

## Lecture Outline (cont.)

- Multiple Substitution
- Bounded Choice Substitution
- Conditional Substitution
- Contextual Information: Sets and Constants
- Unbounded Choice Substitution
- Definitions and Assertions
- Abstract Machine Template
- Consistency of an Abstract Machine

# References


- [1] Abrial, J.-R., *The B Book - Assigning Programs to Meanings*, Cambridge University Press, 1996. (chapters 4,5)
- [2] Clearsy System Engineering, *AtelierB home page* `http://www.atelierb.eu/en/`
- [3] Clearsy System Engineering, *B Method home page* `http://www.methode-b.com/en/`

# B Method Overview

- Developed by Jean-Raymond Abrial and based on the work of Hoare and Dijkstra on program correctness
- Follows a model-oriented approach to software construction (similar to Z or VDM)
- Supports the entire lifecycle of a software product (specification, design, code generation, maintainance), using the same notation (AMN - Abstract Machine Notation) throughout all these steps
- AMN is based on set theory and first order predicate logic
- System development starts with the creation of a mathematical model, expressed in terms of one or several abstract machines, which is further refined or specialized until reaching a final implementation
- Consistency of the initial specification and correctness of all refinements steps are guaranteed by mathematical proofs
- Strong tool support (e.g. AtelierB, B Toolkit, ProB, etc.)

# Formal Specification

- A software specification should enclose the *what* and not the *how* of a software system
- Software specifications may be either informal (expressed in natural language) or formal
- Informal specifications carry the inherent disadvantages of natural languages, suffering of ambiguity, inconsistency, incompleteness
- A *formal specification* is a specification expressed in a formal specification language (a language having both a formal syntax and a formal semantics)
- The use of formal specifications
  - involves investing more effort in the early development phases and changes the cost profile of a project
  - forces a deeper analysis of the system requirements which leads to the discovery and elimiation of ambiguities, inconsistencies and incompletenesses
  - reduces the amount of rework due to requirements errors

# Abstract Machines

- Task: formal specification of a software system
  - First concern: how and where do we start from?
- Essential start-up point: a general model of what a software system is supposed to be
  - Approach: fill in the components of the general model with data from the informal requirements specification
- B model of a software system: the *abstract machine* = state (the statics) + operations (the dynamics)
  - Analogy: pocket calculator
    - invisible memory - the state
    - keys - the operations
- Abstract machine = the main structuring mechanism available for B models, allowing system decomposition into independent pieces that interract through well-defined interfaces
- Related concepts: class, abstract data type, module

## The Statics - State

- Is expressed in terms of
  - *state variables*
  - *invariant*, constraining the values of state variables
- The invariant
  - encloses the static laws of the system
  - is defined in terms of the state variables, using predicate calculus and set theory
  - consists of a number of conjoined predicates
  - should ensure at least the typing of each state variable
- Simplistic example - seat reservation system
  - `seat` - number of available seats
  - some machine clauses: `MACHINE`, `VARIABLES`, `INVARIANT`

**MACHINE**
    $Booking$
**VARIABLES**
    $seat$
**INVARIANT**
    $seat \in \mathbb{N}$
**END**

## The Dynamics - Operations

- The purpose of an operation - to modify the state of an abstract machine, within the limits of the invariant
- *Hiding Principle*: the user of a machine cannot access the state directly, he can only activate the operations
- Positive consequence of the hiding principle: the ability to refine a machine (change the state variables and change the definition of the operations correspondingly, while keeping their names)

- Machine clause: `OPERATIONS`
  - `book` - make a reservation
  - `cancel` - cancel a reservation

**MACHINE**
  $Booking$
**VARIABLES**
  $seat$
**INVARIANT**
  $seat \in \mathbb{N}$
**OPERATIONS**
  book $\widehat{=}$ ... ;
  cancel $\widehat{=}$ ...

**END**

# Operations' Specification - Before/After Predicates

- Specification of an operation = description of the key properties that the execution of that operation (the corresponding state modification) must ensure
- Classical approach = *before/after predicates* relating the values of the state variables prior and following the execution of the operation in question
- Approach undertaken by formal methods such as Z or VDM
- The state values after the execution are denoted by priming the corresponding variables
- Example: before/after predicate used to specify the `cancel` operation of machine `Booking`

$$seat' = seat + 1$$

# Proof Obligations

- An abstract machine is *consistent* only if each operation specification preserves the invariant (provided the invariant is true before the execution, it will also be true after)
- This ensures that the operations themselves (which are supposed to be refined so as to satisfy their specifications) will not break the static laws of the system
- Proof obligation for the `cancel` operation

$$seat \in \mathbb{N} \Rightarrow (\forall) seat' \cdot (seat' = seat + 1 \Rightarrow seat' \in \mathbb{N}) \tag{1}$$

# Operations' Specification - Substitutions

- Concept of substitution: $[x := E]P$ - the result of replacing all free occurences of $x$ in $P$ with $E$, where $P$ is a formula, $x$ is a variable and $E$ is an expression
- One point rule: $\forall x \cdot (x = E \Rightarrow P) \Leftrightarrow [x := E]P$, if $x$ has no free occurences in $E$
- Equivalent rewrittings for the proof obligation (1)

$$seat \in \mathbb{N} \Rightarrow [seat' := seat + 1]seat' \in \mathbb{N} \tag{2}$$

$$seat \in \mathbb{N} \Rightarrow seat + 1 \in \mathbb{N} \tag{3}$$

$$seat \in \mathbb{N} \Rightarrow [seat := seat + 1]seat \in \mathbb{N} \tag{4}$$

- General proof obligation for a substitution $S$ and an invariant $I$

$$\boxed{I \Rightarrow [S]I}$$

- Specification of `cancel` using a substitution

$$\boxed{\text{cancel} \mathrel{\hat{=}} \text{BEGIN} \;\; seat := seat + 1 \;\; \text{END}}$$

## Pre-conditioned Substitution

- Specification of book
  - ?

$$\boxed{\text{book} \mathrel{\widehat{=}} \text{BEGIN } seat := seat \text{ - } 1 \text{ END}}$$

  - Proof obligation (! not true when $seat = 0$)

$$seat \in \mathbb{N} \Rightarrow [seat := seat - 1]seat \in \mathbb{N} \quad \Leftrightarrow$$

$$seat \in \mathbb{N} \Rightarrow seat - 1 \in \mathbb{N}$$

- *Pre-conditioned substitution*
  - Construct: $P|S$, read "$P$ pre $S$" ($P$ - predicate, $S$ - substitution)
  - Alternative syntax: `PRE` $P$ `THEN` $S$ `END`
  - Definition (by post-condition establishment): $[P|S]R \Leftrightarrow P \wedge [S]R$
  - Proof obligation: $I \wedge P \Rightarrow [P|S]I$, or $\boxed{I \wedge P \implies [S]I}$

# Pre-conditioned Substitution (cont.)

- Specification of book revisited
  - Using pre-conditioned substitution

```
book ≙
PRE
    0 < seat
THEN
    seat := seat - 1
END
```

  - Proof obligation (true)

$$seat \in \mathbb{N} \wedge 0 < seat \Rightarrow [seat := seat - 1] seat \in \mathbb{N} \quad \Leftrightarrow$$

$$seat \in \mathbb{N} \wedge 0 < seat \Rightarrow seat - 1 \in \mathbb{N}$$

# Machine Parameters and Initialization

- Machine *parameters*
  - Allow for future machine instantiations
  - Implicit constraints: they can be either simple scalars or finite and non-empty sets
  - Explicit constraints: introduced by means of the `CONSTRAINTS` clause, as a list of conjoined predicates (e.g. typing of scalar parameters)
  - Set formal parameters are independent sets
- `INITIALISATION` clause
  - Allows the assignment of initial values to the machine variables
  - Initializations are substitutions that should ensure the machine invariant

```
MACHINE
    Booking(max_seat)
CONSTRAINTS
    max_seat ∈ ℕ
VARIABLES
    seat
INVARIANT
    seat ∈ 0 .. max_seat
INITIALISATION
    seat := max_seat
OPERATIONS

    book ≙
    PRE
        0 < seat
    THEN
        seat := seat - 1
    END;

    cancel ≙
    PRE
        seat < max_seat
    THEN
        seat := seat + 1
    END

END
```

# Operation Input Parameters

- Extensions to `book` and `cancel` to allow reservation/canceling of several seats

**book**$(ns) \mathrel{\widehat{=}}$
**PRE**
  $ns \in \mathbb{N} \land$
  $ns \leq seat$
**THEN**
  $seat := seat - ns$
**END**;

**cancel**$(ns) \mathrel{\widehat{=}}$
**PRE**
  $ns \in \mathbb{N} \land$
  $seat + ns \leq max\_seat$
**THEN**
  $seat := seat + ns$
**END**;

- Proof obligation for `cancel`

$$
\begin{array}{l}
seat \in 0..max\_seat \land \\
ns \in \mathbb{N} \land \\
ns + seat <= max\_seat \\
\Rightarrow \\
[seat := ns + seat]seat \in 0..max\_seat
\end{array}
$$

$$
\begin{array}{l}
seat \in 0..max\_seat \land \\
ns \in \mathbb{N} \land \\
ns + seat <= max\_seat \\
\Rightarrow \\
ns + seat \in 0..max\_seat
\end{array}
$$

# Operation Output Parameters

- Accessor operation for *seat*, enabling the testing of the various operation preconditions by external machine users

$$
\begin{array}{|l|}
\hline
value \leftarrow \mathbf{val\_seat} \mathrel{\widehat{=}} \\
\mathbf{BEGIN} \\
\quad value := seat \\
\mathbf{END} \\
\hline
\end{array}
$$

## Multiple Substitution

- Construct: $[x, y := E, F]P$
  - Meaning: $P$, with all the free occurences of $x$ and $y$ simultaneously replaced by $E$ and $F$.
  - $P$ if a formula (predicate or expression), $x$ and $y$ are variables and $E$ and $F$ are expressions
- Alternative syntax: $x := E \mid\mid y := F$
- It may be extended to any number of parallel substitutions

# Bounded Choice Substitution

- A notation expressing the choice between two substitutions
- Construct: $S \; [] \; T$, read "$S$ choice $T$", where $S$ and $T$ are substitutions
- Introduces bounded non-determinism - the implementer has the freedom to choose to implement either $S$ or $T$
- Definition (by post-condition establishment):
  $[S \; [] \; T]R \Leftrightarrow [S]R \wedge [T]R$
- Alternative syntax: `CHOICE` $S$ `or` $T$ `END`
- It may be extended to any number of choices

**MACHINE**
    *Sequence*(*VALUE*)
**SETS**
    $REPORT = \{good, bad\}$
**VARIABLES**
    *sequence*
**INVARIANT**
    $sequence \in \mathbf{seq}(VALUE)$
**INITIALISATION**
    $sequence := []$
**OPERATIONS**
    $report \leftarrow \mathbf{push}(vv) \mathrel{\widehat{=}}$
    **PRE**
        $vv \in VALUE$
    **THEN**
        **CHOICE**
            $report := good \; ||$
            $sequence := sequence \leftarrow vv$
        **OR**
            $report := bad$
        **END**

    **END**
**END**

# Conditional Substitution

- Guarded substitution
  - A substitution performed under the assumption of a predicate
  - Construct: $P \Longrightarrow S$, read "$P$ guards $S$", where $P$ is a predicate and $S$ a substitution
  - Definition (by post-condition establishment): $[P \Longrightarrow S]R \Leftrightarrow (P \Rightarrow S[R])$
- Conditional substitution
  - Syntax: `IF` $P$ `THEN` $S$ `ELSE` $T$ `END`
  - Definition: $(P \Longrightarrow S)[](\neg P \Longrightarrow T)$
  - Small conditional: `IF` $P$ `THEN` $S$ `END`
  - Definition of small conditional: `IF` $P$ `THEN` $S$ `ELSE` $skip$ `END` ($skip$ is a substitution with no effect)

```
report ← book(ns) ≙
PRE
    ns ∈ ℕ
THEN
    IF ns ≤ seat THEN
        seat := seat - ns ||
        report := good
    ELSE
        report := bad
    END
END;
```

## Contextual Information: Sets and Constants

- Sets
  - Introduce new types within an abstract machine
  - Can be either enumerated or deferred (left unspecified, but assumed finite and non-empty)
  - Listed within a `SETS` clause
  - The sets, as well as the set machine parameters are all independent types
- Constants
  - Can be either scalar constants of a set or subsets of a scalar set or total functions from a set to a set
  - Do not obey to the Hiding Principle, cannot be refined
  - Listed within a `CONSTANTS` clause
- Properties
  - Conjoined predicates involving the constants and sets
  - Listed within a `PROPERTIES` clause
  - Should ensure the typing of each constant

# Contextual Information: Sets and Constants (cont.)

**MACHINE**
  $Database$
**SETS**
  $PERSON$;
  $SEX = \{male, female\}$;
  $STATUS = \{living, dead\}$
**CONSTANTS**
  $max\_pers$
**PROPERTIES**
  $max\_pers \in \mathbb{N}_1 \wedge$
  $\mathbf{card}(PERSON) = max\_pers$
**VARIABLES**
  $person, sex, status$
**INVARIANT**
  $person \subseteq PERSON \wedge$
  $sex \in person \rightarrow SEX \wedge$
  $status \in person \rightarrow STATUS$
**INITIALISATION**
  $person, sex, status := \emptyset , \emptyset , \emptyset$
**OPERATIONS**
  $\mathbf{death}(pp) \mathrel{\widehat{=}}$
  **PRE**
    $pp \in person \wedge$
    $status(pp) = living$
  **THEN**
    $status(pp) := dead$
  **END**
**END**

## Unbounded Choice Substitution

- A generalization of the bounded choice substitution
- Construct: $@z \cdot S$, read "Any $z$ $S$", where $S$ is a substitution depending on the variable $z$
- Definition (by post-condition establishment): $[@z \cdot S]R \Leftrightarrow \forall z \cdot [S]R$
- Usual syntax: `ANY` $z$ `WHERE` $P$ `THEN` $S$ `END`, defined as $@z \cdot (P \Longrightarrow S)$

```
baby ← newborn(sx) ≙
PRE
    sx ∈ SEX ∧
    PERSON - person ≠ ∅
THEN
    ANY bb WHERE
        bb ∈ PERSON - person
    THEN
        person := person ∪ {bb} ||
        sex(bb) := sx ||
        status(bb) := living ||
        baby := bb
    END
END;
```

# Definitions and Assertions

- Definitions
  - Macros allowing to increase the readability of an abstract machine
  - Introduced by means of the `DEFINITIONS` clause, whose position in the machine is irrelevant

**DEFINITIONS**

$$unborn \mathrel{\hat{=}} PERSON - person$$

- Assertions
  - Introduced by means of the `ASSERTIONS` clause containing a number of conjoined predicates
  - The assertion predicates are supposed to be deducible from those in the invariant and properties (corresponding proof obligations are generated)
  - The role of assertions is to ease the invariant preservation proofs

# Abstract Machine Template

```
MACHINE
  M(X, x)
CONSTRAINTS
  C
SETS
  S;
  T = {a, b}
(ABSTRACT_)CONSTANTS
  c
PROPERTIES
  P
DEFINITIONS
  D
(CONCRETE_)VARIABLES
  v
INVARIANT
  I
ASSERTIONS
  J
INITIALIZATION
  U
OPERATIONS
  u ← O(w) ≙
   PRE
     Q
   THEN
     V
   END;
  ...
END
```

## Consistency of an Abstract Machine

- Establishing the internal consistency of an abstract machine involves proving that
  - its context and invariant ensure the assertions

$$A \wedge B \wedge C \wedge P \wedge I \Rightarrow J \tag{5}$$

  - within the context in question, the initialization ensures the invariant and each of the operations preserves it

$$A \wedge B \wedge C \wedge P \Rightarrow [U]I \tag{6}$$

$$A \wedge B \wedge C \wedge P \wedge I \wedge J \wedge Q \Rightarrow [V]I \tag{7}$$

- Contextual abbreviations used by the above proof obligations

| **Abbreviation** | **Definition** |
|---|---|
| A | $X \in \mathbb{P}_1(\mathrm{INT})$ |
| B | $S \in \mathbb{P}_1(\mathrm{INT}) \wedge T \in \mathbb{P}_1(\mathrm{INT}) \wedge T = \{a, b\} \wedge a \neq b$ |