B method - Abstract Machines (Jean-Raymond Abrial, BP Research, Oxford Programming Research Group)

- a system is seen as a set of states and operations which modify the state;

- ASSERTIONAL METHOD

  for each operation we define the preconditione and the postcondition

  state=(x,y)
  operatione(x,y)
  precond(x,y)
  postcond(x,y)

- to simplify the development of the system the method uses the same notation for all the stages of the development, that goes on for successive refinement steps

  → verification amounts to check that the preconditions and postconditions of lower level operations verify the preconditions and postconditions of higher level operations

Concetti di base: Set theory, predicate calculus.

The development process can be summarized by the following steps:

1. capturing the functional requirements of the systemin a form that can be mathematically analyzed

2. sistematically producing a design of the system, so that it can be verified with respect to the specification in modo che possa essere sempre verificato rispetto alla specifica

3. automatically generate programs using rules that assure that the software satisfies the specification

- Addresses in particulare the specification of sequntial systems
- Dati are easily formalized
- Formalism on which it is possible to base the whole development process
- There exist automatic design support tools

Abstract Machine (AM)

An Abstract Machine is given by

- STATE
  - variable set
  - STATE INVARIANT, to be permanently verified
- a set of OPERATIONS that can be activate to modify the STATE
  - defined in terms of pre-conditionis and post-conditions on the STATE

PROOF OBLIGATIONS:

→ every time that an operation has been specified, it must be verified that its specification preserves the STATE INVARIANT

The method includes:

- logic system for the expression and the proof of PROOF OBLIGATIONS based on the SUBSTITUTION principle

OPERATIONS:
modify the STATE within the constranints imposed by the INVARIANT

PROOF OBLIGATION

→ it must be proved that the specification of the operaztion preserves the invariant

ASSUMPTION:
the STATE verifies the INVARIANT before the operation

PROOF:
the INVARIANT is satisfied after the operation

Design methodology

1. the B development methodology is based on set-theoretic model and first-order logic;

2. the language that the programmer uses to specify the system is the AMN;

3. the intermediate language that is used by tools is based on substitutions.

    - Proof obligations are the key of correctness of the design.

Programming language	Intermediate language
MACHINE m VARIABLES x INVARIANT I INITIALIZATION T OPERATIONS ... $r \leftarrow$ op(y)= PRE Q THEN S END	Proof Obligations $[T]I$ $I \wedge Q \Rightarrow [S]I$

Refinement

MACHINE	REFINEMENT	
m	r	Proof Obligations
VARIABLES	REFINES	
y	m	$\exists y, z, (I \wedge J)$
INVARIANTS	VARIABLES	$[C]\neg[B]\neg J$
I	z	$\forall y, z, I \wedge J \wedge P \Rightarrow$
INITIALIZATION	INVARIANT	$Q \wedge [L]\neg[K]\neg(J \wedge (r = r'))$
B	J	
OPERATIONS	INITIALIZATION	
$\cdots$	C	
$r \leftarrow op(x) =$	OPERATIONS	
PRE P THEN K END	$\cdots$	
	$r' \leftarrow$ op(x)= PRE Q THEN L END	

Example: Check-departure

Case in which

- the automatic pilot is turned off;
- the train is stopped;
- the SSCS is activated;
- the human pilot pushes the *departure* button.

The operation is called only if the train is stopped.

Then if the departure is not validated, the *emergency-brake* is activated.

Proof Obligation for the specification

$I \wedge Q \Rightarrow [S]I$

MACHINE

Check-departure

VARIABLES

departure, emergency-brake, speed

INVARIANT

$speed \in NAT \wedge$
$emergency - break \in BOOL \wedge$
$departure \in \{Interrupted, Authorized, Asked, NonAsked\} \wedge$
$departure = Interrupted \Rightarrow emergency - brake = TRUE$

INITIALIZATION

$speed := 0 \|$
$emergency - brake := TRUE \|$
$departure := NotAsked$

OPERATIONS

$departure - checking =$
PRE
$speed = 0$
THEN
ANY dp WHERE $dp \in \{Interrupted, Authorized, Asked, NonAsked\}$ THEN
$departure := dp \|$
IF $dp = Interrupted$ THEN
$emergency - brake := TRUE$
END
END
END
END

Refinement of Check-departure

We add to the specification

- how the departure can be validated.

A departure is validated whenever

- a button is kept pushed more than a period
  called the *threshold* duration.

We add information about the physical status of the button in the specification.

MACHINE

Check-departure-button

REFINES

Check-departure

VARIABLES

counter, button-status, emergency-brake, speed

INVARIANT

$counter \in NAT \wedge$
$button - status \in \{Pushed, Released\} \wedge$
$button - status = Released \wedge counter < threshold \wedge counter > 0 \Rightarrow$
$departure = Interrupted$

INITIALIZATION

$button - status := Released \|$
$counter := 0$

OPERATIONS

$departure - checking =$
BEGIN $button - status :\in \{Pushed, Released\}$
IF $button - status = Pushed$ THEN
$counetr := counter + D_0$
ELSEIF $counter < threshold \wedge counter > 0$ THEN
$emergency - brake := TRUE \|$
$counter := 0$
END
END
END

Available tools

FORSE - FORmal Specification Environment

- *type-checker* (to apply before the proof);

- *Proof Obligation Generator - POG*: produces and minimize the proof obligations

- *Prover*, a proof assistance tool which explores all the possible proof paths with respect to a mathematical library.

  The output of Prover is a file where all the unproved predicates are stored.

  The user can iterate again the proof phase by giving some new proof rules.

- all these tools have been developed in the programming language of *B-tool* of Abrial (AMN - Abstract Machine Notation).

Resulting software of *SSCS* is in final on-site test phases and no bugs have been reported.

	Lines ver. code	N. Proof Obl.	*Edu.*(*MM*)	*Real.*(*MM*)
Specification	2001	48	1	1
Refinements	2304	517	–	3
Implementation	3000	–	–	2

Statistics on SSCS

	Lines ver. code	N. Proof Oblig.	*Edu.*(*MM*)	*Real.*(*MM*)
Specification	10051	918	1	20
Refinements	12035	5050	–	14
Implementation	16050	–	–	7

Statistics on KVS