



جامعة محمد بوضياف - المسيلة
Université Mohamed Boudiaf - M'sila
University of Mohamed Boudiaf - M'sila



Specification et Verification Formelle

Chapter 03: Method B

Prof. Hichem Debbi

`hichem.debbi@univ-msila.dz`

February 23, 2026

Spreading in Industry:

Method B

The method B is a software development method that aims to specify a software through many versions or steps. Its syntax is based on set theory and first order logic, and employs abstract machines. It was developed by Jean-Raymond Abrial in the 1980s, and it is very related to its predecessor, Z language.

Method B Principle

The B method aims to verify guide software development, so that the resulting final code would be proved to be consistent with the original specification. To do so, Method B uses the Abstract Machine Notation (AMN), which is a famous technique in programming languages and formal proof system for detecting inconsistency.

Introduction to B language

Logical notation

Substitutions

Proof obligations

Refinement

Refinements

Implementation

Structuring Developments

References

In addition to Z of Jean-Raymond Abrial, It has origins in the following approaches

- Pre and post conditions of Jones
- Guarded commands of Dijkstra
- Data Refinement of He Hoare Sanders
- Refinement Calculus Morgan



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

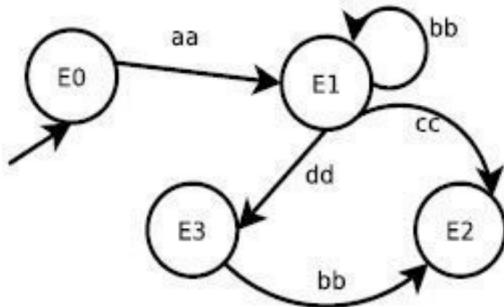
Refinements

Implementation

Structuring
Developments

References

The system developed in B describes a state-transition system, in which the nodes represent the system's states, and the transitions equipped by guards are enabled through operations.



Introduction to B language

Logical notation

Substitutions

Proof obligations

Refinement

Refinements

Implementation

Structuring Developments

References

Spreading in Industry:

- B is extensively used in Industries (by Clearsy and others)
- Clearsy claims to make 30% of its business with B
- The main industrial activity is with train systems
- Alstom and Siemens Transport actively participate in these activities
- Train systems with B in Europe, North and South America, Asia



Introduction to B language

Logical notation

Substitutions

Proof obligations

Refinement

Refinements

Implementation

Structuring Developments

References



Metro line 14 in Paris Formal methods have been used for the automatic train operation system for metro line 14, since 1998.

The specification of running and stopping of trains, and the opening and closing of the train doors as well as platform doors, was performed in method-B .

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Railways

RER line A in Paris Formal methods have been used for railways for the SACEM system, which is an automatic train protection system that controls the speed of all trains. The system permanently ensures the safety of 0.8 million passengers per day. The formal specification was B method to make the informal requirement specifications more precise.

Metro line 14 in Paris Formal methods have been used for the automatic train operation system for metro line 14, since 1998. The specification of running and stopping of trains, and the opening and closing of the train doors as well as platform doors, was performed in method-B .

Railway systems in Denmark Formal methods have been used for computer based interlocking systems for stations. The RAISE formal method has been used for specifying that no derailling or collisions of trains can happen.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- First order logic
- Set theory
- Theory of substitution for operations
- Theory of refinement



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- Set theory
- Theory of substitution for operations
- Theory of refinement



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

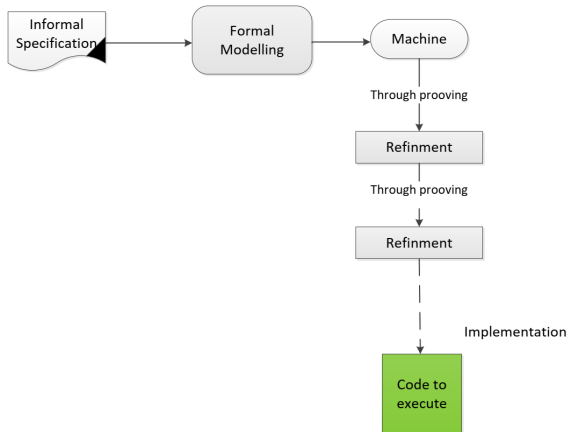
Refinement

Refinements

Implementation

Structuring
Developments

References



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

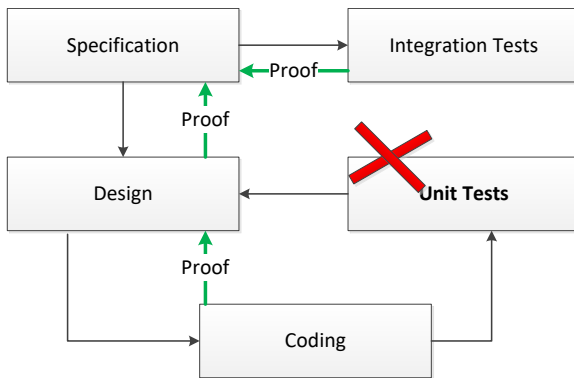
Refinement

Refinements

Implementation

Structuring
Developments

References



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

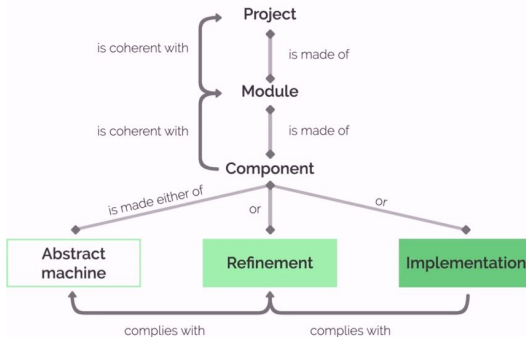
Implementation

Structuring
Developments

References



What is proved



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References



MACHINE . . .
SETS . . .
VARIABLES
. . . .
INVARIANT
. . . predicate
INITIALISATION
. . .
OPERATIONS
. . .
END

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

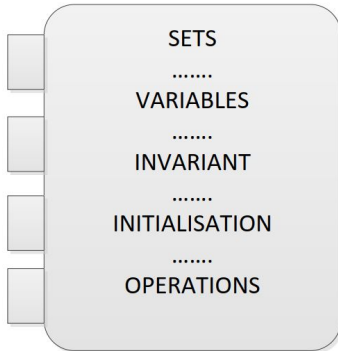
Structuring
Developments

References



Machine Regulum

Switch_on/
allumer
Reduce_temp/
baissérTemp
Switch_off/
eteindre
Raise_temp/
monterTemp



Introduction to B language

Logical notation

Substitutions

Proof obligations

Refinement

Refinements

Implementation

Structuring Developments

References

```
MACHINE ReguLum
  SETS MODEJ = {jour, nuit}; ETATLUM = {eteint, allume}
  VARIABLES mode, lumiere, temp
  INVARIANT mode : MODEJ & lumiere : ETATLUM & temp : INT
  INITIALISATION mode := jour || temp := 20 || lumiere := eteint
  OPERATIONS
    changerMode = CHOICE mode := jour OR mode := nuit END;
    allumer = lumiere := allume;
    eteindre = lumiere := eteint;
    baisserTemp = temp := temp - 1;
    monterTemp = temp := temp + 1
END
```

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

IMPLEMENTATION ReguLum_i

REFINES ReguLum

CONCRETE_VARIABLES mode, lumiere, temp

INITIALISATION mode := jour; temp := 20; lumiere := eteint

OPERATIONS

changerMode = **IF** mode = jour **THEN**

mode := nuit

ELSE

mode := jour

END;

allumer = lumiere := allume;

eteindre = lumiere := eteint;

baissertemp = temp := temp - 1;

montertemp = temp := temp + 1

END

- Each machine has a name
- *SETS* clause considers abstract or enumerable sets, which are used for variables typing. Predefined sets: NAT, INTEGER, BOOL, etc.
- *VARIABLES* clause lists the variables
- *INVARIANT* clause lists the predicates that describe properties invariants of the abstract machine.
- *INITIALISATION* clause initializes all the listed variables. Modifications can be done later through OPERATIONS.
- *Operations* clause aim to describe changes of the machine's state, through logical substitutions $:=$. Postconditions must always respect the invariants.

- This machine has four(04) main operations: (Switch_on, Reduce_temp, Switch_off, Raise_temp) and another operation: changeMode.
- Invariants on the machine's behavior can be expressed such as:
 - Switchon only at night
 - The temperature should not exceed 29 degree by day.

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- Logical substitution is an operation that transforms a state into another correct state
- Transforming a state = transforming a predicate.
- Operation = Substitution = Predicate transformer



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- An operation can not be accessed from another operation in the same machine (Respecting the precondition)
- From an external machine, we can not call or access two operations of the same machine in the same time (e.g. Increment/Decrement)
- A machine can include secondary operations for verifying preconditions of principal operations.
- The caller of an operation must verify its preconditions (e.g. division by zero)
- Through *Refinement* we try to weaken the preconditions until they disappear.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Logical context

In the logic Hoare/Floyd/Dijkstra and triplet of Hoare , we have the concept of state, space states, commands and execution.

We write

$$\{P\}S\{R\}$$

Where R refers to the result's predicate of S , and $P = wp(S, R)$ characterizes the set of all states before executing S , such that . executing S results in a state satisfying R .

We consider $wp(S, R)$ as the weakest precondition of S with respect to R

The weakest precondition was introduced by Dijkstra in 1975.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Examples

- Let S be an affectation and R the predicate $i \leq 1$ then:
 $wp(i := i + 1, i \leq 1) = (i \leq 0)$
- Let S be the conditional: *if* $x \geq y$ **then** $z := x$ *else* $z := y$ and
 R the predicate $z = \max(x, y)$, then $wp(S, R) = \text{true}$.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Generalized Substitutions

- Simple substitution: semantically: $S[R] : S$ transforms R
- Multiple substitution: $x, y := E, F$: semantically $[x, y := E, F]R$.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References



Substitution simple

BEGIN
S
END

Simultaneous Substitution

Let S and T two substitutions. S being $x := E$ and T being $y := F$
we note $S || T$

Sequential Substitution

Let S and T two substitutions. S being $x := E$ and then T being
 $y := F$ we note $S ; T$

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Substitution with precondition ($P|S$)

```
PRE
P
THEN
S
END
```

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References



Bounded choice ($S \parallel T$)

Choice

S

OR

T

END

Substitution with guard ($S \parallel T$)

IF P

THEN T

ELSE S

END

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Non-deterministic ($@x.(P \Rightarrow S)$)

ANY X
WHERE P
THEN S
END



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Select Substitution

```
SELECT P1 THEN S1 WHEN P2 THEN S2 ... ELSE Sn END
```

Select Example

```
SELECT x > 10 THEN x := x - 10  
WHEN x < 10 & x > 0 THEN x := 2 * (x - 10)  
ELSE x := x - 1  
END
```

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

While loop

While substitution ::=

WHILE Condition DO Instruction

INVARIANT Predicate

VARIANT Expression

END

Let **P** be a predicate, **S** a substitution, **I** and **R** predicates and **V** an expression (V must be an integer).

WHILE P DO S INVARIANT I VARIANT V END



While loop – Example

MACHINE

While

varLoc := 0;

cpt := 0

WHILE cpt < 5 **DO**

varLoc := varLoc + 1 ;

cpt := cpt + 1

INVARIANT

cpt : **NAT** &

cpt <= 5 &

varLoc & **NAT** &

varLoc = var1 + cpt

VARIANT

5 - cpt

END

END

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

MACHINE

M (prm) /* Name of the machine and its parameters*/

CONSTRAINTS

C /* list of clauses such as: uses, sees, includes,*/

SETS

S /* list Of Sets*/

CONSTANTS

K /* list of Identifiers or constants */

PROPERTIES

P /* Predicates on K*/

VARIABLES

V /* list of Variables */

DEFINITIONS

D /* LIST OF Definitions*/

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

INVARIANT

I /* LIST OF Invariants*/

INITIALISATION

U

OPERATIONS

u $\leftarrow$ O(pp) =

PRE

Q

THEN

Subst

END;

. . .

END

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Constraints

Parameters values should satisfy the constraints.

$$\exists prm.C$$

Properties

Sets and constants should satisfy the properties of the machine.

$$C \implies \exists (S, K). P$$

Invariants

There exists a state satisfying the invariant.

$$P \wedge C \implies \exists V. I$$



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References



Initialization

The Initialization establishes the Invariant.

$$P \wedge C \implies [U]I$$

Operations

For each operation,

$P \wedge C \wedge I \wedge Pre \implies [Subs]I$ Every called operation preserves the invariant under its precondition

Introduction to B
language

Logical notation

Substitutions

**Proof
obligations**

Refinement

Refinements

Implementation

Structuring
Developments

References

Are a set of predicates, which helps to guarantee the consistency of the abstract machine mathematically.

Two essential types of obligations :

- The INITIALIZATION establish the Invariant.
- Each operation, should preserve the Invariant under its Precondition.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Are a set of predicates, which helps to guarantee the consistency of the abstract machine mathematically.

Two essential types of obligations :

- The INITIALIZATION establish the Invariant.
- Each operation, should preserve the Invariant under its Precondition.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

$n_rsrc \in 0..100$
 $n_rsrc = \text{cardinal Of the set}$

reserve $\rightarrow$ - 1 element
free $\rightarrow$ + 1 element



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

```
MACHINE
  Reservation
VARIABLES
  n_rsrc
INVARIANT
  n_rsrc : 0..100
INITIALISATION
  n_rsrc := 100
OPERATIONS
  reserve =
    PRE n_rsrc > 0
    THEN
      n_rsrc := n_rsrc - 1
    END;

  free =
    PRE n_rsrc < 100
    THEN
      n_rsrc := n_rsrc + 1
    END;
  bb <-- disponibilite =
  bb := bool(0 < n_rsrc)
END
```

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

We have to prove that the INITIALISATION respects the invariant :

$[n_rsrc := 100](n_rsrc \in 0..100)$

We have to prove that:

$100 \in 0..100$



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

We have to prove that each operation once called, its precondition should satisfy the invariant.

Reserve

We have to prove:

$$n_rsrc \in 0..100 \wedge 0 < n_rsrc \implies n_rsrc - 1 \in 0..100$$

Free

We have to prove:

$$n_rsrc \in 0..100 \wedge n_rsrc < 100 \implies n_rsrc + 1 \in 0..100$$

Disponibility

We have to prove:

$$n_rsrc \in 0..100 \wedge (n_rsrc > 0 \vee \neg(n_rsrc > 0)) \implies n_rsrc \in 0..100$$



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

REFINEMENT

Reservation_r

REFINES

Reservation

VARIABLES

n_rsrc, r_libres, r_occupees |

INVARIANT

r_libres : POW(INTEGER)

& r_occupees : POW(INTEGER)

& r_libres /\ r_occupees = {}

& n_rsrc = card(r_libres) // invariant de liaison

INITIALISATION

r_libres, r_occupees, n_rsrc := 1..100, {}, 100

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

OPERATIONS

reserver = // Redefine with new variables

ANY ss **WHERE**

ss : r_libres

THEN

r_libres := r_libres - {ss}

|| r_occupees := r_occupees \ / {ss}

|| n_rsrc := n_rsrc - 1

END

;

liberer = // Redefine with new var

ANY ss **WHERE**

ss : r_occupees

THEN

r_libres := r_libres \ / {ss}

|| r_occupees := r_occupees - {ss} || n_rsrc :=
n_rsrc + 1

END

;

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

```
bb <-- disponibilite = // Redefine using
                        //If Else

IF 0 < n_rsrc
THEN
bb := TRUE
ELSE
bb := FALSE
END
END
```



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

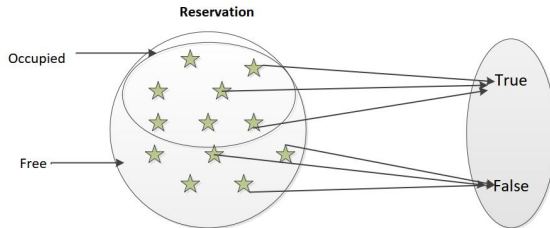
Structuring
Developments

References



100

Array of resources



We have to prove that each operation once called, its precondition should satisfy the invariant.

Definition

Refinement is a technique of transforming abstract models of a software into more concrete model. we call the new resulting model a refinement, and should preserve the same behavior of the previous model.

The concrete model or refinement should be characterized by:

- It introduces more details on the specification
- It is closer to the final implementation
- It reduces Non-determinism

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Through refinement we can reach the specification SP_n starting from SP_1 progressively:

$$SP_1 \rightarrow SP_2 \rightarrow \dots \rightarrow SP_n$$

By applying the proof obligations at each step: SP_{i+1} must preserve the behavior of SP_i .

Types of refinement:

- Data Refinement
- Control Refinement
- Algorithmic refinement



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- Introduction of more concrete variables and sets
- Define operations employing concrete variables
- Define abstraction relation between the variables of abstract machine and the variables of the refined one through the linkage invariant.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- The operations conserve the same signature: same operations and same parameters
- Weaken the precondition: A new precondition of the substitution T should be weaker than the previous substitution S
- Reducing the non-determinism: T must be more deterministic than S .



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References



Reduce nondeterminism

Nondeterminism is about having a choice, where any of the outcomes might be satisfactory, thus we can choose those among these values.

Simple example

$S_Res \leftarrow \text{ChoiceValue} = \text{result} \in \{1, 2, 3\}$ can be refined by
 $T_Res \leftarrow \text{ChoiceValue} = \text{result} := 3$

Concrete Example

If an architecture wants to design a home, he let some details concerning the materials not defined until further steps in design. For instance, Always there is always choice between tiled roofs and slates. Such a final decision might be identified just before construction.

Machine abstraite	Machine raffinement
MACHINE MaisonAbstraite SETS TYPE_TOIT = {ardoises, tuiles} VARIABLES toit INVARIANT toit : TYPE_TOIT INITIALISATION toit :: TYPE_TOIT OPERATIONS choix_toit = CHOICE toit := ardoises OR toit := tuiles END END	REFINEMENT MaisonRaff REFINES MaisonAbstraite VARIABLES letoit INVARIANT letoit = toit INITIALISATION letoit := ardoises OPERATIONS choix_toit = letoit := tuiles END

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

Weak precondition

given that the behavior of a refined operation is specified on the assumption of its precondition, the refinement can do anything outside of the precondition

Example

```
result  $\leftarrow$  Divide ( n1 , n2 )  $\hat{=}$   
  PRE  n1  $\in \mathbb{N} \wedge$  n2  $\in \mathbb{N} \wedge$  n2  $\neq 0$   
  THEN result := n1 / n2  
  END
```

```
result  $\leftarrow$  Divide ( n1 , n2 )  $\hat{=}$   
  IF  n2  $\neq 0$  THEN result := n1 / n2  
  ELSE result := 27  
  END
```

Concrete Example

Machine abstraite	Machine raffinement
MACHINE MAbstraite VARIABLES xx INVARIANT xx : 1.. 10 INITIALISATION xx := 1 OPERATIONS plus1 = PRE xx < 10 THEN xx := xx + 1 END END	REFINEMENT MoinsAbstraite REFINES MAbstraite VARIABLES yy INVARIANT yy = xx INITIALISATION yy := 1 OPERATIONS plus1 = PRE yy < 15 THEN yy := yy + 1 END END

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

$$\left\{ \begin{array}{l} \text{MACHINE } MA \\ \text{SETS } C \\ \text{VARIABLES } a \\ \text{INVARIANT } I \\ \text{INITIALISATION } Init \\ \text{OPERATIONS} \\ \quad Op = P|S \\ \text{END} \end{array} \right\} \sqsubseteq \left\{ \begin{array}{l} \text{REFINEMENT } MB \text{ REFINES } MA \\ \text{SETS } D \\ \text{VARIABLES } b \\ \text{INVARIANT } J \\ \text{INITIALISATION } Init' \\ \text{OPERATIONS} \\ \quad Op = P'|S' \\ \text{END} \end{array} \right\}$$

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

The concrete machine should not contradict the abstract machine.

Proof obligation

$$INV_{abst} \wedge INV_{conc} \Rightarrow [Subst_{conc}] \neg ([Subst_{abst}] \neg (INV_{abst}))$$



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- Abstract sets in MA are implicitly present in MB
- the abstract variables a are refined by the concrete variables b
 - Typing the concrete variables introduced by the refinement
 - Express properties on the concrete variables
 - introduce the *linking invariant*, which relates the concrete variables to the abstract ones
- The concrete initialisation $Init'$ refines the abstract one $Init$
- The operation Op is refined by the operation Op' preserving the same signature.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- Non-determinism is not allowed, since Implementations must contain only constructs that can be executed.
- simultaneous substitution ($||$)
- preconditioned substitutions
- Assignments involving abstract variable types



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- Enumerable sets
- Integers: **NAT1**, **NAT** or **INT** between MININT and MAXINT
- Booleans: **BOOL**
- subsets of concrete types
- Concrete integer intervals
- **Arrays**: a total function of the form $T_1 \times T_2 \times \dots \times T_n \rightarrow T$ where each T_i is a concrete set.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

```
MACHINE
  MA
SETS
  POSITION ;
  MOVE = {Stop, Forward, Reverse} ;
  DIRECTION = {North, South, East, West}
...
END
```

```
IMPLEMENTATION
  MA_i
REFINES
  MA
IMPORTS
  MB
SETS
  SPEED ;
  SIGNAL = {Red, Orange, Green}
VALUES
  POSITION = 0 .. 100 ;
  SPEED = -10 .. 10
...
END
```

```
MACHINE
  MB
SETS
  MOVE = {Stop, Forward, Reverse} ;
...
END
```

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- MA defines the *POSITION* deferred set and the enumerated sets *MOVE* and *DIRECTION*.
- MA_i defines a new *SPEED* deferred set and a new enumerated set *SIGNAL*.
- The two deferred sets in MA_i are valued in the *VALUES* clause
- The MA_i implementation imports the MB machine.
- The enumerated set *MOVE* is identical to the one MA_i



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

```
MACHINE
  MA
CONCRETE_CONSTANTS
  PosMin,
  PosMax
ABSTRACT_CONSTANTS
  PosInit
PROPERTIES
  PosMin ∈ INT ∧
  PosMax ∈ NAT ∧
  PosInit ∈ INT
...
END
```

```
IMPLEMENTATION
  MA_i
REFINES
  MA
CONCRETE_CONSTANTS
  PosAverage,
  PosInit
PROPERTIES
  PosAverage ∈ INT
VALUES
  PosMin = -100 ;
  PosMax = 100 ;
  PosAverage = 0 ;
  PosInit = 50
...
END
```

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- The implementation MA_i of MA defined the new concrete constants $PosAverage$ and $PosInit$.
- $PosInit$ is refined by MA_i
- All of the constants of $MA_i(PosMin, PosMax, PosAverage \text{ and } PosInit)$ are valued in the implementation.



Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

```
MACHINE data_show_reserv
SETS DataShows /* abstract set of datashows */
;
reserved = {ok, ko} /* datashow used or not */
CONSTANTS max_Rsrc /* limit */
PROPERTIES max_Rsrc : 1..MAXINT
VARIABLES reservedDataShows /* reserved datashows */
INVARIANT reservedDataShows <: DataShows /*subset*/
    & card(reservedDataShows) <= card(DataShows)
INITIALISATION reservedDataShows := {}
```

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

OPERATIONS

```
    reserve(ds) =  
  PRE ds : DataShows  
    & card(reservedDataShows) < card(DataShows)  
  THEN  
    reservedDataShows := DataShows \ {ds}  
  END ;  
  free(ds) =  
  PRE ds : reservedDataShows & card(DataShows) > 1  
  THEN  
    reservedDataShows := reservedDataShows - {ds}  
  END ;  
  bb <-- isReservedDatashow(ds) =  
  PRE ds : DataShows  
  THEN  
    bb := bool(ds : (reservedDataShows))  
  END ;  
  END
```

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

```
IMPLEMENTATION data_show_reserv_i  
REFINES data_show_reserv
```

```
DEFINITIONS NBR_Rsrc == 0..100 /* a range for  
implementing the number of datashows */
```

```
VALUES DataShows = NBR_Rsrc
```

```
; max_Rsrc = 100 /* some value */
```

```
CONCRETE_VARIABLES c_reservedDataShows
```

```
INVARIANT c_reservedDataShows : NBR_Rsrc --> reserved  
    & reservedDataShows = c_reservedDataShows~[{ok}]  
/* linkage invariant */
```

```
INITIALISATION c_reservedDataShows := (NBR_Rsrc)*{ko};
```

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

OPERATIONS

reserve (ds) =

BEGIN

c_reservedDataShows(ds) := ok ;

END ;

free (ds) =

BEGIN

c_reservedDataShows(ds) := ko

END ;

bb <-- isReservedDatashow (ds) =

BEGIN

VAR okko **IN**

okko := c_reservedDataShows(ds) ;

IF okko = ok **THEN** bb := **TRUE** **ELSE** bb := **FALSE** **END**

END

END ;

END

Introduction to B
language

Logical notation

Substitutions

Proof
obligations

Refinement

Refinements

Implementation

Structuring
Developments

References

- The B-Book: Assigning Programs to Meanings, J. R. Abrial
- Specification en B Support de cours Ecole des Jeunes Chercheurs en Programmation EJCP 2007
- La Methode B, Christian Attiogbe Faculte des sciences – Universite de Nantes
- METHODE B – CLEARSY
- [http://www.computing.surrey.ac.uk/personal/st/H.Treharne/te](http://www.computing.surrey.ac.uk/personal/st/H.Treharne/)
- M. Bourahla, Cours en methode B. Université de M'sila

