Introduction au langage B

1 Développement B

- Spécification
- Etapes de raffinement
- Spécification exécutable
- Génération automatique de code

Vérification des étapes de raffinement : chaque étape est conforme avec l'étape précédente.
$\rightsquigarrow$ l'implantation satisfait sa spécification.

1.1 Spécification d'opération

```
res1, ..., resn <-- oper(param1, ..., paramk) =
PRE precondition THEN
  res1, ..., resn : (postcondition)
END
```

La précondition généralise la notion de type en indiqnant les propriétés que doivent satisfaire les paramétres effectifs au moment de l'appel. La postcondition indique la propriété sastifaite par les valeurs retournées.

Exemples

- Ajout de deux valeurs dont la somme est représentable par un entier. L'addition peut ne pas être définie dans le jeu d'instructions cible.

```
res <-- ajouter(x1,x2) =
PRE
  x1: INT & x2: INT & x1+x2: INT
THEN
  res : (res : INT & res = x1 + x2)
END
```

- Symétrique d'un point par rapport à une droite. La spécification indique que le milieu des deux points doit être sur la droite et que la droite et le segment joignant les deux points doivent être orthogonaux.

```
s1,s2 <-- symetrique(p1,p2,a1,a2,a3) =
PRE
  p1: INT & p2: INT & a1: INT & a2: INT & a3: INT
THEN
  s1,s2: (s1: INT & s2: INT &
          a1*(s1+p1)/2 + a2*(s2+p2)/2 + a3 = 0 & /* le milieu est sur la droite */
          a2*(p1-s1) - a1*(p2-s2) = 0) /* droite et segment p-s orthogonaux */
END
```

1.1.1 Spécification par machine abstraite

Les opérations sont définies dans des machines possédant des variables. La machine décrit dans un invariant leurs domaines de valeurs, généralisant ainsi la notion de type. Les opérations peuvent modifier les variables d'état tout en respectant l'invariant.

Exemples Une machine doit toujours se trouver dans un fichier du même nom, avec pour suffixe `.mch`.

- La machine `trajectoire` introduit un point avec ses coordonnées satisfaisant un invariant indiquant que le point reste sur une droite donnée. La machine définit une opération `avancer` modifiant la position du point tout en respectant l'invariant.

MACHINE `trajectoire`
CONCRETE_VARIABLES
`p1,p2`
INVARIANT
`p1: INT & p2: INT & 2 * p1 - 3 * p2 = 0`
INITIALISATION
`p1,p2 :( p1=0 & p2=0)`
OPERATIONS
`avancer = p1, p2 := p1 + 3, p2 + 2`
END

– La machine `symétrie` définit une opération remplaçant la position du point décrit par ses coordonnées par son symétrique par rapport à la droite passée en paramètre. La spécification doit mentionner l'ancienne position et la nouvelle position du point. L'ancienne valeur d'une variable x est notée x$0.

MACHINE `sym`
CONCRETE_VARIABLES
`p1, p2`
INVARIANT
`p1: INT & p2: INT`
INITIALISATION
`p1,p2 :( p1: INT & p2: INT)`
OPERATIONS

```
symetrie(a1,a2,a3) =
PRE a1: INT & a2: INT & a3: INT THEN
  p1, p2: (p1: INT & x2: INT &
           a1*(p1+p1$0)/2 + a2*(p2+p2$0)/2 + a3 = 0 &
           a2*(p1$0-p1) - a1*(p2$0-p2) = 0)
END
```

END

1.2 Implantation

Le langage d'implantation d'une spécification comporte affectation, séquence, sélection et répétition. Une instruction I se définit par la syntaxe suivante :

```
I ::= var := expression
    | skip
    | IF predicat THEN I ELSIF I THEN I ... ELSE I END
    | IF predicat THEN I END
    | VAR var IN I END
    | WHILE predicat DO I INVARIANT predicat VARIANT expression END
    | I ; I
    | BEGIN I END
```

Exemple La machine `test` doit se trouver dans le fichier `test.mch`.

MACHINE
`test`
CONCRETE_VARIABLES
`p1,p2`
INVARIANT
`p1 : INT & p2 : INT`
INITIALISATION

```
    p1,p2 :( p1: INT & p2: INT)
OPERATIONS
echange = p1,p2 := p2,p1
END
```

L'implantation `test_i` doit se trouver dans le fichier `test_i.imp`.

```
IMPLEMENTATION
    test_i
REFINES
    test
INITIALISATION
    p1 := 0; p2 := 0
OPERATIONS
echange =
    VAR aux IN
        aux := p1
    ;   p1 := p2
    ;   p2 := aux
    END
END
```

2 Types et expressions B

B repose sur la théorie des ensembles. Une déclaration de type est donc remplacée par l'appartenance à un ensemble. Il est ainsi possible de préciser (dans une précondition, une postcondition, un invariant) un domaine de valeur pour une ou plusieurs variables à l'aide d'une formule de la logique des prédicats

2.1 Les prédicats

Notation mathématique	Notation ascii
x1 = x2	x1 = x2
x1 ≠ x2	x1 /= x2
xx ∈ EE	xx : EE
¬ pp	not(pp)
p1 ∧ p2	p1 & p2
p1 ∨ p2	p1 or p2
p1 ⇒ p2	p1 => p2
p1 ⇔ p2	p1 <=> p2
∃ xx.(p)	♯xx.(p)
∀ xx.(p)	!xx.(p)

Exemple

$$\forall ii.(ii \in \mathbb{N} \Rightarrow \exists jj.(jj \in \mathbb{N} \wedge (ii = 2 * jj \vee ii = 2 * jj + 1)))$$

```
!ii.(ii : NATURAL => ♯jj.(jj : NATURAL & (ii = 2 * jj or ii = 2 * jj + 1)))
```

2.2 Les booléens

Les booléens sont introduits par l'ensemble énuméré prédéfini BOOL = {TRUE,FALSE}. La fonction `bool` permet de transformer un prédicat en booléen.

```
bb <-- carre(nn) = /* retourne TRUE si nn est un carré */
PRE nn: INT THEN
  bb := bool(∃ rr.(rr ∈ INT ∧ rr*rr = nn))
END
```

2.3 Les entiers

Les entiers relatifs sont introduits via l'ensemble INTEGER. D'autres types entiers sont définis par des sous-ensembles prédéfinis ou des intervalles. On distingue les entiers mathématiques et les entiers implantables dont la valeur peut être représentée par un mot en machine.

- INTEGER = $\mathbb{Z}$
- NATURAL = $\mathbb{N}$
- NATURAL1 = $\mathbb{N}^*$
- INT = -MAXINT .. MAXINT
- NAT = 0..MAXINT
- NAT1 = 1..MAXINT

2.4 Les ensembles

Notation matématique	Notation ascii	
$\emptyset$	{}	ensemble vide
$\{x_1, \ldots, x_n\}$	{x1,. . .,xn}	ensemble en extension
{xx | P }	{xx | P }	ensemble en compréhension
xx $\in$ EE	xx : EE	appartenance
xx $\notin$ EE	xx /: EE	non appartenance
e1 $\subseteq$ e2	e1 <: e2	inclusion
e1 $\subset$ e2	e1 <<: e2	inclusion stricte
e1 $\cup$ e2	e1 \/ e2	union
e1 $\cap$ e2	e1 /\ e2	intersection
e1 $\times$ e2	e1 * e2	produit carthésien

2.5 Relations et fonctions

En B, et contrairement aux langages fonctionnels, une fonction de A dans B est introduite comme une relation binaire (un ensemble de couples) telle que tout élément de A a au plus une image dans B. La fonction est totale si tout élément de A a exactement une image. Sinon elle est dite partielle. Son domaine de définition est l'ensemble des éléments de A ayant une image.

Notation mathématique	Notation ascii	
$A1 \longrightarrow$ A2	A1 –> A2	fonctions totales de A dans B
$A1 \nrightarrow$ A2	A1 +-> A2	fonctions partielles de A dans B
dom(ff)	dom(ff)	domaine de définition de ff
ff(xx)	ff(xx)	image de xx par la fonction ff
$\{i_1 \mapsto v_1, \ldots, i_n \mapsto v_n\}$	{i1 |-> v1,. . .,in |-> vn}	fonction définie point à point
f1 <+ f2	f1 <+ f2	surcharge de f1 par f2

Axiome (f1 <+ f2)(xx) = si xx $\in$ dom(f2) alors f2(xx) sinon f1(xx)

Un tableau est une fonction dont le domaine est un intervalle d'entiers.