Méthodologie de construction du logiciel (M3301-2)

Méthode B

J. Christian Attiogbé

Université de Nantes

Motivations

Exercice

Construire un logiciel, qui surveille un patient hospitalisé à domicile.
En fonction de l'état constaté du patient, le logiciel contrôle :
- la préparation du traitement chimique approprié, ou
- la modification de la composition d'un traitement chimique, et
- son injection au patient, lorsque le traitement est prêt et le patient dans les bonnes dispositions pour le recevoir.

Quelles méthodes ?

Comment s'y prendre pour développer ou construire le logiciel ?
de plus il faut qu'il soit correct (car santé - critique) !

Plan du cours

Méthode de production d'applications correctes

Logiciel correct ? (application correcte ?)

Un logiciel est défini par ses états corrects (ceux attendus en fonction des fonctionnalités) et **leurs enchaînements**.

Bug ?

Un bug est un état incorrect, indésirable, inattendu d'un logiciel.

Un logiciel fonctionne correctement quand à travers ses fonctionnalités il n'arrive jamais dans un état indésirable (bug)

Méthode de production d'applications correctes

Dans la **Méthode B**, et dans les méthodes formelles en général,
- on spécifie un logiciel par son ensemble d'états corrects et
- on évite que les opérations qui décrivent les fonctionnalités aillent dans un état indésirable.

Pour éviter les états indésirables, on prend des précautions (ce sont les préconditions des opérations).

Pas de risque, on **doit respecter les préconditions**, avant de faire les opérations.

Méthode B

- (..1996) Méthode pour spécifier, concevoir et développer des systèmes informatiques séquentiels.
- (1998..) Event B ... systèmes distribués, concurrents
- (...) méthode en constante évolution, les outils (IDE) aussi
- M. **J-R. Abrial**

Exemples d'application dans le ferroviaire

Figure: Synchronisation ouverture portes palières - Metro Paris (13, ...)

Exemples de systèmes critiques

Un système est critique lorsque son mauvais fonctionnement ou sa défaillance ont des conséquences néfastes en termes de sécurité des personnes, des biens, de l'environnement, ...

- Pilotage par logiciel d'un avion, d'une voiture, d'une centrale nucléaire
- Pilotage d'un robot industriel,
- Contrôle d'un cœur artificiel (*pacemaker*),
- Opérations en bourse
- ...

☞ Eviter les fautes de construction du logiciel (les bugs).
Construire formellement le logiciel.

Méthodes en génie logiciel

Méthode formelle =

- Langage de spécification ou **modélisation** formelle
- Système de **raisonnement** formel

Méthode B =

- Langage de spécification :
  - Logique, théorie des ensembles : langage de données
  - Langage des substitutions généralisées : langage des opérations
- Système formel (de raisonnement)
  - Prouveur de théorèmes

☞ Vous pouvez élaborer la votre !

Développement formel

Développement formel de logiciel =

- Transformation systématique d'un modèle mathématique en code exécutable.

  = Transformation de l'**abstrait** en **concret**

  = Passage des structures mathématiques aux structures informatiques

  = Raffinement jusqu'au code dans un langage de progr.

B : Méthode formelle
+ théorie de raffinement (de machines abstraites)
⇒ méthode de développement formel

Méthode B : Approche globale

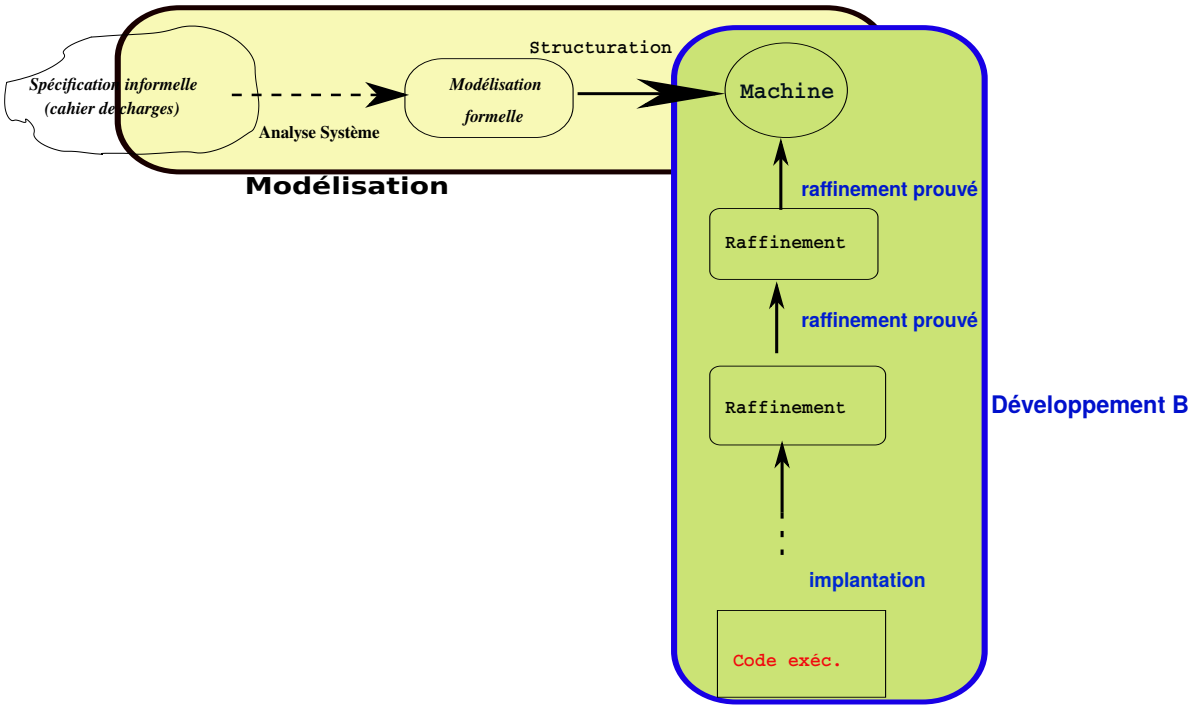

Figure: Analyse et développement B

Machine abstraite : offre des opérations

Une machine abstraite offre des opérations qui sont appelables à partir d'autres opérations/programmes externes.

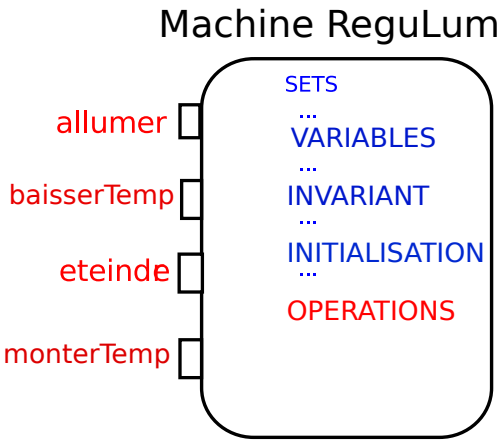

Figure: Les opérations sont appelables de l'exterieur

☛ Une opération d'une machine ne peut pas en appeler une autre

Exemple de machine abstraite : régulation lumière

```
MACHINE ReguLum
SETS
MODEJ = {jour, nuit}
; ETATLUM = {eteint, allume}

VARIABLES
mode
, lumiere
, temp
INVARIANT
mode : MODEJ
& lumiere : ETATLUM
& temp : ENTIER
& ... /* d'autres propriétés */
INITIALISATION
mode := jour || temp := 20
|| lumiere := eteint
```

```
OPERATIONS
changerMode =
    CHOICE mode := jour
    OR mode := nuit
    END
;
allumer =
BEGIN lumiere := allume END
;
eteindre =
lumiere := eteint
;
baisserTemp = temp := temp - 1
;
monterTemp = temp := temp +1
END
```


Machine abstraite : exemple de la jauge

Exemple

Soit à **contrôler la valeur d'une variable (jauge) entre 2 et 45** alors qu'on y effectue des opérations d'incrémentation et de décrémentation.

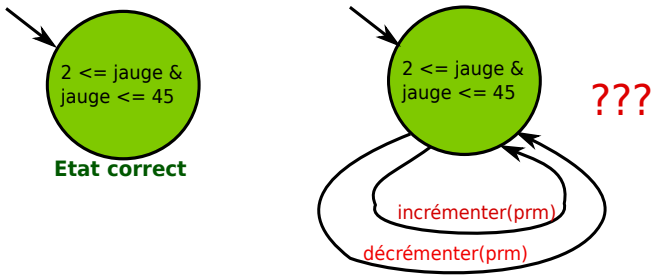

Figure: Rester dans un état correct

Exprimer les états corrects par un prédicat (invariant)

Machine abstraite : exemple de la jauge

```
MACHINE MaJauge
VARIABLES
    jauge
INVARIANT
    jauge : NAT
&    jauge >= 2
&    jauge <= 45
INITIALISATION
jauge := 1 // quoi ???
```

```
OPERATIONS
decrementer1 =
PRE jauge > 2
THEN jauge := jauge - 1
END
; decrementer(pas) =
PRE pas : NAT
& jauge - pas >= 2
THEN
     jauge := jauge - pas
END
...
incrementer ...
...
END
```


Machine abstraite : exemple de la jauge

```
MACHINE MaJauge
VARIABLES
    jauge
INVARIANT
    jauge : NAT & ....
INITIALISATION
    jauge := 1 // quoi ???
OPERATIONS
   decrementer1 = ...
;  decrementer(pas) = ...
;  incrementer ...
;  ...
END
```

Avec la méthode B,
on transforme cette machine abstraite en programme (C) exécutable ; la transformation se fait à l'aide de raffinements.

Exemple : construction du programme de PGCD

De la machine abstraite vers son raffinement en code exécutable.

Du modèle mathématique (abstrait) → modèle informatique (concret)

Développement du PGCD : machine abstraite

```
MACHINE
    pgcd1 /* pour le PGCD de deux entiers, JCA, U. Nantes */
            /* pgcd(x,y) is d | x mod d = 0 ∧ y mod d = 0
            ∧ ∀ other divisors dx, d > dx
            ∧ ∀ other divisors dy, d > dy */
OPERATIONS
        rr <-- pgcd(xx,yy) = /* SORTIE : rr ; ENTREE xx, yy */
            ...
END
```

Développement du PGCD : machine abstraite

```
OPERATIONS
rr <-- pgcd(xx,yy) = /* spécification du pgcd */
PRE
    xx : INT & xx >= 1 & xx < MAXINT
&   yy : INT & yy >= 1 & yy < MAXINT
THEN
    ANY dd WHERE
    dd : INT
    & (xx - (xx/dd)*dd) = 0 /* d is a divisor of x */
    & (yy - (yy/dd)*dd) = 0 /* d is a divisor of y */
        /* and the other common divisors are < d */
    & !dx.((dx : INT & dx < MAXINT
        & (xx- (xx/dx)*dx) = 0 & (yy-(yy/dx)*dx)=0) => dx < dd)
    THEN rr := dd
    END
END
```


Développement du PGCD : raffinement

```
REFINEMENT /* raffinement de ...*/
    pgcd1_R1
REFINES pgcd1 /* la machine précédente */
OPERATIONS
rr <-- pgcd (xx, yy) = /* l'interface ne change pas */
    BEGIN
        ... Corps de l'opération raffinée
    END
END
```

Développement du PGCD : raffinement

```
rr <-- pgcd (xx, yy) = /* opération raffinée */
    BEGIN    VAR cd, rx, ry, cr IN
             cd := 1
             ; WHILE ( cd < xx & cd < yy) DO
                 ; rx := xx - (xx/cd)*cd ; ry := yy - (yy/cd)*cd
                 IF (rx = 0 & ry = 0)
                 THEN /* cd divise x et y, possible GCD */
                     cr := cd /* possible rr */
                 END
                 ; cd := cd + 1 /* on tente plus grand */
             INVARIANT
                 xx : INT & yy : INT & rx : INT & rx < MAXINT
                 & ry : INT & ry < MAXINT & cd < MAXINT
                 & xx = cr*(xx/cr) + rx   &   yy = cr*(yy/cr) + ry
             VARIANT
                 xx - cd
             END
        rr := cr    END     END
```


La méthode B

Dans la suite nous allons étudier :

- les concepts de base utilisés dans la méthode B : la logique, la théorie des ensembles, pour modéliser les données et les traitements, de façon abstraite ;
- les bases de la méthode B : espaces d'états et propriétés invariants, les substitutions généralisées pour modéliser les opérations
- les bases du raffinement

Références

- J-R. Abrial, The B-Book, Cambridge University Press, 1996
- J. Wordsworth, Software Engineering with B, Addison-Wesley, 1996
- J-R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press
- H. Habrias, Spécification formelle avec B, Hermès - Lavoisier, 2001