Master Informatique (M2) - ALMA 2013/2014
Faculté des Sciences et des Techniques de Nantes FSE / Méthodes formelles - B (C. Attiogbé)
Christian.Attiogbe@univ-nantes.fr

Cahier d'exercices : N° 2
spécification formelle en B, développement formel

Références

C. Attiogbé, Notes de cours (*B : méthode de développement formel de logiciels*)
J-R. Abrial, *The B Book*, Cambridge University Press, 1996
J. B. Wordsworth, *Software Engineering with B*, Addison-Wesley, 1996
H. Habrias, *Spécification formelle avec B*, Hermès, 2001
J-R. Abrial, *Modeling in Event- B : System and Software Engineering*, Cambridge University Press, 2010

Spécifications B : prise en main

Le livre de référence *The B-Book* de J-R Abrial contient une série d'exemples de machines abstraites que nous recommandons en priorité.

Exercice : Machine *Data_Base*

(voir BBook - Chapitre 4)

Exercice : Machines *Product, Client, Invoice*

(voir BBook - Chapitre 8)

Exercice : A Lift Control System

(voir BBook - Chapitre 8)

Exercice 1 : le b a ba

Soit la machine abstraite suivante :

```
MACHINE
      RMan(RES)
VARIABLES
      rfree
INVARIANT
      rfree <: RES
INITIALISATION
      rfree := {}
OPERATIONS
      alloc(rr) =
      PRE
         rr : rfree
      THEN
         rfree := rfree - {rr}
      END
;     free(rr) =
      PRE  rr : RES
      &    rr /: rfree
      THEN
         rfree := rfree \/ {rr}
      END
;      setfree(rrs) =
      PRE
        rrs <: RES
      THEN
        rfree := rrs
      END
END
```

1. Commentez cette machine abstraite. Retrouvez la spécification informelle.
2. Ecrivez les obligations de preuve pour l'initialisation et les opérations.

Remarques : Les modèles formels peuvent être dédiés à une étude de propriétés avec ou sans implantation. Lorsqu'on envisage un modèle devant aller jusqu'à la génaration de code, on doit prendre les précautions nécessaires ; par exemples, les ensembles doivent avoir une taille définie. Un ensemble abstrait ne peut pas servir de type de paramètre.

Exercice 2 : les bases

Soit la machine abstraite suivante :

```
MACHINE
      Students
SETS
      STUDENT
CONSTANTS
      max_students /* pour borner le sous-ensemble de travail */
PROPERTIES
      max_students : NAT1 & max_students < 100
```

```
VARIABLES
      studentset
INVARIANT
      studentset <: STUDENT
&     card(studentset) <= max_students
INITIALISATION
      studentset := {}
OPERATIONS

      enter(st) =           /* ajoute un etudiant st a l'ensemble */
      PRE
          ...               /* a completer */
      THEN
          studentset := studentset \/ {st}
      END
;
      remove(st) =         /* enleve l'etudiant st de l'ensemble */
      PRE
      ...                  /* a completer */
      THEN
          studentset := studentset - {st}
      END
END
```

1. Complétez les préconditions des opérations,
2. Ecrivez les obligations de preuve pour la correction (cohérence) de la machine.

Exercice 3 : les bases, encore

Les objets existent indépendamment de leurs caractéristiques !

```
MACHINE
      ProcNumber /* serveur de numéros de processus */
DEFINITIONS
      PROCESS == 1000..9999 /* serait mieux dans un contexte, et vu */
VARIABLES
      curProc
,     processes  /* un sous-ens de processus déjà utilisés */
INVARIANT
      curProc : PROCESS
&     processes <: PROCESS
INITIALISATION
      curProc :: PROCESS
||    processes := {}
OPERATIONS
res <--  newProc =           /* fournit un numero de processus disponible */
      PRE
          card(processes) <= 9999 /* places encore dispo */
      THEN
          ANY pp WHERE
              pp : PROCESS & pp /: processes  /* un num tout frais */
          THEN
```

```
            res := pp
        ||  processes := processes \/ {pp}
        END
    END
;
    freeProcess(nn) =         /* libere un numero précédemment pris */
    PRE
        ...                   /* a completer */
    THEN
        ...                   /* a completer */
    END
END
```

1. Analysez cette machine et proposez des améliorations
2. Completez la machine.

Exercice 4 : organisation des données et opérations

Soit à écrire une machine abstraite B (ProcessManager) pour la gestion des processus dans un système d'exploitation.
La gestion des processus intègre par exemple la fonctionalité suivante :

F1 : CreerProcess	Pour créer un nouveau processus : Prendre un nouveau numéro unique, pour le processus ; ajouter ce processus aux courants ; Mettre le processus dans un état donné (exemple prêt).
F2 : ...	...

1. Esquissez l'architecture des machines abstraites du module gestionnaire de processus.
2. Quelles leçons en tirer par rapport à la structuration des données et traitements en machines et opérations ?

Machines abstraites : structuration

Le système de gestion d'un annuaire téléphonique

Nous considérons un système de gestion d'un annuaire. Le système est vu comme le gestionnaire d'une base de données contenant des noms de personnes avec leurs numéros de téléphone. Le système gére les personnes et les numéros de téléphones associés à ces personnes. Les opérations classiques d'ajout, suppression, modification doivent être offertes par le système.
On se propose d'étudier ici la spécification en B du système. On donne pour ce faire quelques éléments de travail qu'il va falloir compléter.

Architecture type

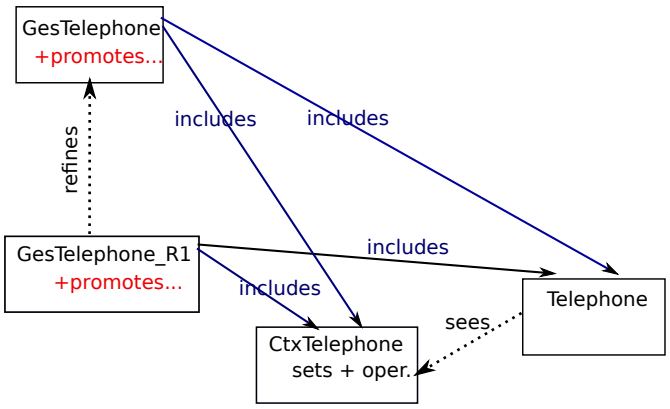

FIGURE 1 – Une architecture type de développement en B

Exemples de machines pour démarrer

```
MACHINE
    Ctx_ANNUAIRE
CONSTANTS
    maxPers,
    maxNums
PROPERTIES
    maxPers : 0..MAXINT
    &maxNums : 0 ..MAXINT
SETS
    MESSAGE = {Ok, PasOk}

DEFINITIONS
    PERSONNE == 0..maxPers
    ; NUMERO == 0..maxNums
OPERATIONS
 res <-- Succes = /* quand une operation est effectuee */
      BEGIN
        res := Ok
      END
;
 res <-- Echec = /* quand une operation est non effectuee */
BEGIN
res := PasOk
      END
END

MACHINE
      Annuaire
SEES
    Ctx_ANNUAIRE

DEFINITIONS   /* les memes que dans le contexte */
    PERSONNE == 0..maxPers
```

```
    ; NUMERO == 0..maxNums


VARIABLES
      membres, telephones
INVARIANT
      membres <: PERSONNE
&     telephones : membres <-> NUMERO
INITIALISATION
      membres, telephones := {},{}
OPERATIONS
      ajoutMembre(lenom) =
      PRE
         lenom : PERSONNE & lenom /: membres
      THEN
         membres := membres \/ {lenom}
      END
/* ;
    autres operations
*/
END
```

Opérations à écrire pour compléter la machine précédente

```
MACHINE
      Annuaire     /* la suite */
...

OPERATIONS
 lesnums <-- rechercherNumeros(lenom) =
/* ici, on renvoie un ensemble (donc abstrait) */
/* en vue d'une implantation, il faut renvoyer un type implantable,
comme une sequence ... à faire plus tard */
       PRE
         lenom : PERSONNE & lenom : dom(telephones)
       THEN
         lesnums := telephones[{lenom}]
       END
;
       ajoutCoordonnees(lenom, lenum) =
/* deux hypotheses :
(a) soit le nom est membre,
(b) soit le nom n'est pas membre */
       ...
;
       supprimerMembre(lenom) = /* hypothese, lenom est pas dans 'membres' */
       ...
;
       supprimerCoordonnees(lenom, lenum) = /* enleve le couple (lenom, lenum) */
       ...
;
res <-- rechercherNoms(numero) = /* recherche des personnes ayant 'numero' */
       ...
```

```
;

/*---------- Operations auxilliaires : pour "tester" les PRE --------------*/
bb <--  dejaMembre(lenom) = /* signale que le nom est dans 'membres' */
       ... /* on renvoie un booleen True / False */
;
bb <--   pasMembre(lenom) =  /* signale que le nom n'est pas dans 'membres'*/
       ...
;
 res <-- coordonneesDejaDefini(lenom, lenum) =
       ...
;
 res <-- coordonnesInconnues(lenom, lenum) =
       ...
;
 res <-- numeroInconnu(lenum) = /* booleen : lenum n'est pas dans annuaire */
       ...
END
```

Toutes ces opérations sont regroupées dans la même machine car elles ont le même invariant.

Etude

1. Complétez la spécification des opérations de la machine `Annuaire`
2. Ecrivez et prouvez l'obligation de preuve de l'opération `ajoutMembre`

Pour compléter la spécification du système de gestion de l'annuaire, nous introduisons la machine `GesTelephone`.

```
MACHINE
      GestAnnuaire
INCLUDES
      CtxANNUAIRE,
      Annuaire,
            // accès aux VARIABLES, SETS, CONSTANTS
            // avec possibilite d'utiliser les operations de ces Machines
            // pour modifier leurs VARIABLES

OPERATIONS
 res <-- ajoutMembreTotal (lenom) =
       PRE
           lenom : PERSONNE
       THEN
         CHOICE
                // ajoutMembre(lenom) sera fait dans ce cas
         ||  res <-- Succes
         OR
               // on n'ajoute pas  ici
         ||  res <-- Echec
END
/*-----
```

```
 cette operation pourrait etre rafinee/implantee par
 res <-- ajoutMembreToral(lenom) =
    PRE ...
        bl := dejaMembre(lenom) /* bl : var locale à introduire */
      ; IF (bl = FALSE)
        THEN
            ajoutMembre(lenom)
        ;   res <-- Succes
        ELSE
            res <-- Echec
        END
    END
------*/

;
res <-- supprimerMembreTotal (lenom) =
        ...
;
res <-- ajoutCoordonneesTot (lenom, lenum) =
        ...
;
res <-- supprimerCoordonneesTot (lenom, lenum) =
        ...
;
res, numeros <-- rechercherNumerosTotal (lenom) =
        ...
;
res,lesnoms <-- rechercherNomsTotal (lenum) =
        ...
END
```

Suite de l'étude

1. Complétez les machines proposées et surtout`GestAnnuaire`
2. Proposez des raffinements des différentes machines
3. Analysez formellement avec l'Atelier B.

Machines abstraites : obligations de preuves de correction

Exercice 1

1. Effectuez les substitutions indiquées dans les prédicats suivants :
   - $[xx := 1]xx \neq yy$
   - $[xx := yy]xx = yy$
   - $[xx := 1](\forall xx.(xx \in \mathbb{N} \Rightarrow xx \geqslant yy) \vee xx \geqslant 0)$
   - $[xx := 0]yy > 0$
   - $[xx := xx + 1]xx > 0$
2. Effectuez les substitutions indiquées dans les prédicats suivants :

- $[PRE\ xx > 0\ THEN\ xx := xx - 1\ END]\, xx > yy$
- $[PRE\ xx > yy\ THEN\ xx := yy\ END]\, xx > 0$
- $[PRE\ xx \in \mathbb{N}\ THEN\ xs := xs \cup \{xx\}\ END]\, xs \subseteq \mathbb{N}$
- $[PRE\ xx \in \mathbb{N}\ THEN\ yy := xx\ END]\, \forall xx.(xx \in \mathbb{N} \Rightarrow yy > zz)$

Exercice 2

1. Montrez que `[ANY xx WHERE xx : NAT & xx > 5 THEN yy := xx END] yy > 5`
2. Montrez que `[ANY xx WHERE xx : NAT & xx > 0 THEN yy := xx + 1 END] yy > 1`
3. Montrez que `[ANY xx WHERE xx : NAT & xx > yy THEN zz := xx END] zz > yy`
4. Montrez que `[CHOICE xx := 1 OR xx := 2 END] xx > 0`
5. Montrez que `[xx :: {yy}] xx = yy`
6. Montrez que `[xx :: yy /\ zz] xx : zz`

Exercice 3

Soit à étudier la machine suivante :

```
MACHINE       RRMan(RESOURCE, max_res)
CONSTRAINTS   card(RESOURCE) >= max_res
INCLUDES      RMan(RESOURCE),
              bkup.RMan(RESOURCE)
INVARIANT
              card(rfree) <= max_res
&             card(bkup.rfree) <= max_res
INITIALISATION
              rfree, bkup.rfree := {}, {}
OPERATIONS
      res <-- rec_alloc =
              PRE      rfree /= {}
              THEN
                 ANY   rr  WHERE rr : rfree
                 THEN
                       res := rr || alloc(rr)
                 END
              END
END
```

1. Commentez les différentes clauses de la machine.
2. Quelles sont les variables de la machine `RRMan` ?
3. Montrez que `rec_alloc` préserve l'invariant. (Ecrivez et prouvez le théorème).
4. Commentez les opérations suivantes :

```
rec_free(rr) =
PRE
   rr : RESOURCE - rfree
```

```
                    &  card(rfree) < max_res
                    THEN
                       free(rr)
                    END
    ;
                    rep <-- is_free(rr) =
                    PRE
                        rr : RESOURCE
                    THEN
                        rep := bool(rr : rfree)
                    END
    ;
                    rec_backup =
                    BEGIN
                        bkup.setfree(rfree)
                    END
```

Exercice 4 : preuve de correction

Soit la machine suivante :

```
MACHINE   RMan(RES)        /* resource manager */
VARIABES  rfree
INVARIANT
          rfree <: RES
INITIALISATION
          rfree := {}
OPERATIONS
          alloc(rr) =
          PRE  rr : rfree
          THEN rfree := rfree - {rr}
          END
;
          free(rr) =
          PRE   rr : RES & rr /: rfree
          THEN  rfree := rfree \/ {rr}
          END
;
          setfree(rrs) =
          PRE   rrs <: RES
          THEN  rfree := rrs
          END
END
```

1. Ecrivez et prouvez le théorème qui garantit que l'opération `free` préserve l'invariant de `RMan`.
2. Ecrivez et prouvez le théorème qui garantit que l'opération `setfree` préserve l'invariant de `RMan`.