Méthode B :
mémento de raffinement et de génération de code en C
(avec AtelierB V4 / ComenC)
Dut Info et Master Alma
version provisoire

C. Attiogbé

UNIVERSITÉ DE NANTES

Janvier 2011

Table des matières

1 Ensembles

Du point de vue théorique, on peut manipuler des ensembles de taille quelconque dans un modèle ; pratiquement on utilise des ensembles de taille connue (bornée) lorsqu'on est préoccupé par un modèle pour la construction de programmes ou de logiciels.

Remarque : Les cas suivants sont complètement prouvés et la génération de code effectuée.

1.1 Cas : Raffinement, implantation d'un sous-ensemble d'un intervalle

Notes : Dans la version V4.0 de l'AtelierB,

- Les intervalles (ensembles) utilisés comme domaine de fonction/relation doivent être définis à partir de 0.
- Dans les implantations, les fonctions doivent être des fonctions totales. Il faut implanter les relations et autres fonctions comme des fonctions totales.

Un exemple de machine (`Resrc.mch`) :

```
/*-------------------------------------------------
 * Author: Christian Attiogbé
 * Université de Nantes (IUT/LINA)
 * Creation date: lun. mars 8 2010
 -------------------------------------------------*/
MACHINE
    Resrc
DEFINITIONS
    RESC  == 0..200 /* un ensemble abstrait */
    /* attention : l'intervalle doit démarrer de 0 */
CONSTANTS
    maxRes
PROPERTIES
    maxRes : 201..MAXINT
VARIABLES
    rsc
INVARIANT
    rsc <: RESC  &  card(rsc) <= maxRes
INITIALISATION
    rsc := (1..0)   /* ens vide */
 OPERATIONS
     addRsc(rr) = /* ajout de l'elt rr dans l'ensemble */
    PRE
        rr : RESC & rr /: rsc & card(rsc) < maxRes
    THEN
        rsc := rsc \/ {rr}
    END

;   rmvRsc(rr) = /* retrait de l'elt rr de l'ensemble */
    PRE  rr : RESC & rr : rsc
         & card(rsc) > 1
    THEN
         rsc := rsc - {rr}
    END
END
```

et son implantation (Resrc_i.imp) :

```
/*-------------------------------------------------
 * Author: Christian Attiogbé
 * Université de Nantes (IUT/LINA)
 * Creation date: lun. janv. 3 2011
 -------------------------------------------------*/
IMPLEMENTATION
   Resrc_i
REFINES
    Resrc
    /*
    Conception : On choisit les éléments de l'ens RESC dans une plage
    les elts choisis sont notés ok, les autres sont ko
    */
SETS
    OKKO = {ok, ko}
DEFINITIONS
    RESC == 0..200 /* la même définition que dans l'abstraite */

VALUES
    maxRes = 210

CONCRETE_VARIABLES
    c_rsc /* ressources concretes qui sont utilisees ou non */
INVARIANT
    c_rsc : (RESC) --> OKKO   /* fonction totale necessaire */
    & card(c_rsc) <= maxRes   /* contrainte reconduite */
    & rsc = c_rsc~[{ok}]  /* invariant de liaison abstrait concret */

INITIALISATION
    c_rsc := (0..200)*{ko}  /* solution simple : aucun n'est utilisé */

OPERATIONS
   addRsc ( rr ) =   /* ajout d'un elt ==>  rr |-> ok   */
   BEGIN
       c_rsc(rr) := ok   /* c_rsc \/ { rr }*/
   END
   ;
   rmvRsc ( rr ) =   /* retrait d'un elt  ==> rr |-> ko  */
   BEGIN
       c_rsc(rr) := ko  /*  c_rsc - { rr } */
   END

END
```

1.2 Cas : Raffinement, implantation d'un sous-ensemble abstrait

Un exemple de machine (Resrc2.mch) :

```
/*-------------------------------------------------
 * Author: Christian Attiogbé
 *  Université de Nantes (IUT / LINA)
 * Creation date: mar. janv. 4 2011
 -------------------------------------------------*/
```

```
MACHINE
    Resrc2
SETS
    RESC    /* ensemble abstrait */
CONSTANTS
    maxRes /* borne de l'ensemble qu'on utilsera */
PROPERTIES
    maxRes : 201..MAXINT /* un entier borné au moins par 201 */
VARIABLES
    rsc
INVARIANT
    rsc <: RESC   /* un sous-ensemble de RESC */
&    card(rsc) <= maxRes  /* borné */
INITIALISATION
    rsc := {}   /* ens vide  */
 OPERATIONS
     addRsc(rr) = /* ajout de l'elt rr dans l'ensemble */
    PRE
        rr : RESC & rr /: rsc & card(rsc) < maxRes
    THEN
        rsc := rsc \/ {rr}
    END
    ;
    rmvRsc(rr) = /* retrait de l'elt rr de l'ensemble */
    PRE
        rr : RESC & rr : rsc
        & card(rsc) > 1
    THEN
        rsc := rsc - {rr}
    END
END
```

et son implantation (`Resrc2_i.imp`) :

```
/*-----------------------------------------------------
 * Author: Christian Attiogbé / Université de Nantes
 * Creation date: mar. janv. 4 2011
 -----------------------------------------------------*/
IMPLEMENTATION
   Resrc2_i
REFINES
    Resrc2
    /*
    Conception : On choisit les éléments de l'ens RESC dans une plage
    les elts choisis sont notés ok, les autres ko
    */
SETS
   OKKO = {ok, ko}
DEFINITIONS
    PLAGE_RESC == 0..200  /* necessaire comme domaine */

VALUES
   maxRes = 210
   ;  RESC = PLAGE_RESC  /* bizarrement RESC = 0..200  ne marche pas */
```

```
CONCRETE_VARIABLES
   c_rsc
INVARIANT
   c_rsc : PLAGE_RESC --> OKKO   /* fonction totale necessaire */
   & card(c_rsc) <= maxRes   /* contrainte reconduite */
   & rsc = c_rsc~[{ok}]  /* invariant de liaison abstrait concret */

INITIALISATION
    c_rsc := RESC*{ko}  /* solution simple : aucun n'est utilisé */

OPERATIONS
   addRsc ( rr ) =   /* ajout d'un elt ==>  rr |-> ok   */
   BEGIN
       c_rsc(rr) := ok   /* c_rsc \/ { rr }*/
   END
   ;
   rmvRsc ( rr ) =   /* retrait d'un elt  ==> rr |-> ko  */
   BEGIN
       c_rsc(rr) := ko  /*  c_rsc - { rr } */
   END
```

2 Fonctions partielles et totales

2.1 Cas : modélisation d'un dictionnaire simple de mots (fonction totale)

Un exemple de machine abstraite (dicoMot.mch) :

```
/* %
* Author: Christian Attiogbé
 * Université de Nantes (IUT / LINA)
 * Creation date: nov. 2009
 */
MACHINE
    dicoMot
SETS
    MOT /* un ensemble abstrait de mots */
    ; SIGNIFIK = {s0,s1,s2}/* un ensemble abstrait, des significations */
CONSTANTS
    maxMots  /* borne */
PROPERTIES
    maxMots : 1..MAXINT
VARIABLES
    mots /* sous ensemble de mots */
,    dico /* représente le dictionnaire */
INVARIANT
    mots <: MOT /* sous ens de mots utilisés, borné */
    & card(mots) <= maxMots
&    dico : mots --> SIGNIFIK

INITIALISATION
    mots := {} ||
    dico := {} /* : mots --> SIGNIFIK */
```

```
OPERATIONS
    ajoutMot(mm, signif) =
    PRE mm : MOT & mm /: mots
        & signif : SIGNIFIK
        & (mm,signif) /: dico
        & card(mots) < maxMots
    THEN
        mots := mots \/ {mm}
        || dico(mm) := signif
    END
;   RetraitMot(mm) =
PRE  mm : MOT & mm : dom(dico)
    & card(mots) > 1
    THEN
        mots := mots - {mm}
    ||  dico := {mm} <<| dico
   END
;
    bb <-- existeMot(mm) =
    PRE  mm : MOT
    THEN
      bb := bool(mm : dom(dico))
    END
;
    res <-- rechercheSignifMot(mm) = /* trouver la signification d'un mot */
    PRE mm : MOT & mm : dom(dico)
    THEN
     res := dico(mm)
    END
END
```

et son implantation (`dicoMot_i.imp`) :

```
/*-----------------------------------------------------
* Author: Christian Attiogbé
* Université de Nantes (IUT / LINA)
 * Creation date: mar. janv. 4 2011
 -----------------------------------------------------*/
IMPLEMENTATION
   dicoMot_i
REFINES
   dicoMot
DEFINITIONS
    PLAGE_MOT == 0..20  /* une plage pour implanter l'ens des mots */
SETS
    OKKO = {ok, ko} /* indique si mot utilise ou non */
VALUES
    MOT = PLAGE_MOT  /* implantation de l'abstrait MOT */
    ; maxMots = 22   /* une valeur quelconque ici */

CONCRETE_VARIABLES
    c_mots,    /* nouvelles variables */
    c_dico
```

```
INVARIANT
    c_mots : PLAGE_MOT --> OKKO   /* mots utilisés ou non */
    & mots = c_mots~[{ok}]  /* liaison abstrait concret */
    & c_dico : PLAGE_MOT --> SIGNIFIK
    & dico = (mots <| c_dico) /* liaison abstrait concret */
INITIALISATION
   c_mots := (PLAGE_MOT)*{ko}; /* aucun n(est encore utilisé */
   c_dico := (PLAGE_MOT)*{s0}  /* qui est vide */

OPERATIONS
   ajoutMot ( mm , signif ) =
   BEGIN
       c_mots(mm) := ok ; /* mots := mots \/ { mm } */
       c_dico (mm) := signif
   END
   ;

   RetraitMot ( mm ) =
   BEGIN
       c_mots(mm) := ko  /* mots := mots - { mm } */
   END
   ;

   bb <-- existeMot ( mm ) =
   BEGIN
       /* bb := bool ( mm : dom (dico) )*/
       VAR okko IN
           okko := c_mots(mm);
           IF okko = ok
           THEN bb := TRUE
           ELSE bb := FALSE
           END
       END
   END
   ;
   res <-- rechercheSignifMot ( mm ) =
   BEGIN
       res :=  c_dico(mm)   /* res := dico ( mm ) */
   END
END
```

Remarque : Les 2 machines sont complètement prouvées mais la génération de code n'a pas aboutit (raison inconnue pour le moment).

2.2 Cas : Annuaire de numéros

Définissons d'abord une machine de contexte pour rassembler des données communes dans un projet.

Un exemple de machine abstraite (`Ctx_ANNUAIRE.mch`) :

```
/*-------------------------------------------------
 * Author: Christian Attiogbé
 *  Université de Nantes (IUT / LINA)
 * Creation date: mar. janv. 4 2011
```

```
 ------------------------------------------------------*/
MACHINE
    Ctx_ANNUAIRE
CONSTANTS
    maxPers,     /* borne des sous-ensembles */
    maxNums
PROPERTIES
    maxPers : 0..MAXINT
    &maxNums : 0 ..MAXINT
DEFINITIONS
    PERSONNE == 0..maxPers
    ; NUMERO == 0..maxNums
END
```

et son implantation (`Ctx_ANNUAIRE_i.imp`) :

```
IMPLEMENTATION
   Ctx_ANNUAIRE_i
REFINES
   Ctx_ANNUAIRE

DEFINITIONS
   PERSONNE == 0 .. maxPers ;
   NUMERO == 0 .. maxNums
VALUES
    maxPers = 10 ;
    maxNums = 8
END
```

Définissons maintenant la machine abstraite qui modélise la gestion (simplifiée ici) d'un annuaire de numéros de personnes.

Un exemple de machine abstraite (`Annuaire.mch`) :

```
*------------------------------------------------
* Author: Christian Attiogbé
 * Université de Nantes (IUT / LINA)
 * Creation date: mar. janv. 4 2011
 ------------------------------------------------------*/
MACHINE
    Annuaire
SEES
    Ctx_ANNUAIRE

DEFINITIONS   /* les memes que dans le contexte, mais on peut inclure un fichier  */
    PERSONNE == 0..maxPers
    ; NUMERO == 0..maxNums
VARIABLES
    membres
    , telephones

INVARIANT
    membres <: PERSONNE
& telephones : membres +-> NUMERO
```

```
INITIALISATION
    membres := {}
    || telephones := {}

OPERATIONS
ajoutMembre(lenom) =
    /*? ajout un membre qui n'y est pas encore */
PRE
    lenom : PERSONNE & lenom /: membres
    & card(membres) < maxPers
THEN
membres := membres \/ {lenom}
END
;
supprimerMembre(lenom) =
     /*? retrait d'une personne qcq */
PRE
    lenom : PERSONNE   & lenom : membres
    & card(membres) > 1
THEN
membres := membres - {lenom} ||
telephones := {lenom} <<| telephones
END
;
res <-- estMembre(lenom) = /* dejaMembre */
PRE
lenom : PERSONNE
THEN
res := bool(lenom : membres)
END
;
 res <-- pasMembre(lenom) = /* la meme qu'au dessus */
  PRE
lenom : PERSONNE
THEN
res := bool(lenom /: membres)
END
END
```

et son implantation (`Annuaire_i1.imp`) :

```
/*-------------------------------------------------
* Author: Christian Attiogbé
* Université de Nantes (IUT / LINA)
 * Creation date: mar. janv. 4 2011
 -------------------------------------------------*/
IMPLEMENTATION
   Annuaire_i1
REFINES
    Annuaire
SEES
    Ctx_ANNUAIRE
SETS
    OKKO = {ok, ko} /* indique si element utilise ou non */
```

```
DEFINITIONS
   PERSONNE == 0..maxPers ;
   NUMERO == 0..maxNums

CONCRETE_VARIABLES
   c_membres ,
   c_telephones

INVARIANT
    c_membres : PERSONNE --> OKKO
    & membres = c_membres~[{ok}] /* liaison abstrait concret */
    & c_telephones : PERSONNE --> NUMERO
    & telephones = (membres <| c_telephones) /* liaison abstrait concret */

INITIALISATION
   c_membres := (PERSONNE)*{ko} ;
   c_telephones := (PERSONNE)*{0}

OPERATIONS
   ajoutMembre ( lenom ) =
   BEGIN
       c_membres(lenom) := ok /* membres := membres \/ { lenom }*/
       ; c_telephones(lenom) := 0
   END
   ;

   supprimerMembre ( lenom ) =
   BEGIN
      c_membres(lenom) := ko /* membres := membres - { lenom } */
       /* telephones := { lenom } <<| telephones*/
   END
   ;

   res <-- estMembre ( lenom ) =
   BEGIN
       /* res := bool ( lenom : membres ) */
       VAR okko IN
           okko := c_membres(lenom);
           IF okko = ok
           THEN res := TRUE
           ELSE res := FALSE
           END
       END
   END
   ;

   res <-- pasMembre ( lenom ) =
   BEGIN
       /* res := bool ( lenom /: membres ) */
       VAR okko IN
           okko := c_membres(lenom);
           IF okko = ok
           THEN res := FALSE
           ELSE res := TRUE
```

```
            END
        END
    END

END
```

Notes : Il reste une preuve non déchargée dans l'implantation (avant la génération de code).

3 Autres cas d'implantation avec génération de code

3.1 Cas : grille avec des valeurs prises dans un ensemble abstrait

Un exemple de fonction avec comme ensemble de départ un produit cartésien et comme ensemble d'arrivée un ensemble énuméré.

Un exemple de machine abstraite (`ExempleMot1.mch`) :

```
/*------------------------------------------------
* Author: Christian Attiogbé
 * Université de Nantes (IUT / LINA)
 * Creation date: mar. nov. 30 2010
 ------------------------------------------------*/
MACHINE
    ExempleMot1
DEFINITIONS
    INDICE2 == 0..10
SETS
    MOTS = {faa, foo, fuu, fee, fii}
CONCRETE_VARIABLES   /* parce qu'on veut les garder jusqu'à l'implantation */
    tab1, tab2
INVARIANT  /* on peut utiliser des DEFINITIONS pour 0..2 et 0..5 */
    tab1 : (0..2)*(0..5) --> MOTS
 &   tab2 : INDICE2 --> INT
 &   max(ran(tab2)) > 10   /* une propriete */
INITIALISATION
    tab1 := (0..2)*(0..5)*{faa}
||    tab2 := (INDICE2)*{11}
OPERATIONS
    r1 <--getT1 (ii,jj) = /* recuperer un mot */
    PRE
        ii : 0..2 & jj : 0..5
    THEN
        r1 := tab1(ii,jj)
    END
    ;
    setT2(ii,vv) = /* ranger un entier */
    PRE
        ii : INDICE2
        &    vv : INT
        & vv > 10
  THEN
    tab2(ii) := vv
    END
END
```

et son implantation (`ExempleMot1_i.imp`) :

```
IMPLEMENTATION
   ExempleMot1_i
REFINES
   ExempleMot1

DEFINITIONS
   INDICE2 == 0 .. 10

INITIALISATION
   tab1 := ( 0 .. 2 ) * ( 0 .. 5 ) * { faa } ;
   tab2 := ( 0 .. 10 ) * { 11 }

OPERATIONS
   r1 <-- getT1 ( ii , jj ) =
   BEGIN
       r1 := tab1 ( ii , jj )
   END
   ;
   setT2 ( ii , vv ) =
   BEGIN
       tab2 ( ii ) := vv
   END
END
```

Notes : Machines prouvées et code C généré.

3.2 Cas : variations sur le dictionnaire

Un exemple de machine abstraite (`TradMot1.mch`) :

```
/*-----------------------------------------------------
 * Author: Christian Attiogbé / Université de Nantes
 * Creation date: mar. nov. 30 2010
 -----------------------------------------------------*/
MACHINE
    TradMot1
    /* Simple exemple pour la generation de code avec des fonctions */
SETS
    SIGNIF = {oof, iif, uuf, aaf} /* des significations */
DEFINITIONS
    MOTS == 0..3
    /* Dans l'optique de generer du code,
       il faut utiliser un intervalle comme domaine
       Cela n'est pas genant puisqu'on peut ranger des mots
       par rapport aux indices */
CONCRETE_VARIABLES /* parce qu'on veut les garder jusqu'à l'implantation */
    traducMot  /* des traductions de mots */

INVARIANT
    traducMot : (MOTS) --> SIGNIF /* signification de certains mots */
                /* les parenthèses sont necessaires ici */
INITIALISATION
    traducMot :: MOTS --> SIGNIF
OPERATIONS
```

```
    r1 <--getMot (ii) =  /* recuperer le mot à l'indice ii */
    PRE
        ii : MOTS
    THEN
        r1 := traducMot(ii)
    END

; setT2(mot,sens) = /* definir la traduction du mot */
   PRE
        mot : MOTS
     &    sens : SIGNIF
  THEN
     traducMot(mot) := sens
    END
END
```

et son implantation () :

```
*-------------------------------------------------
 * Author: Christian Attiogbé / Université de Nantes
 * Creation date: mer. déc. 1 2010
 * Code en C OK
 -------------------------------------------------*/
IMPLEMENTATION
   TradMot1_i
REFINES
   TradMot1
DEFINITIONS
    MOTS == 0..3
   /* L'ensemble intervalle comme ens depart,  marche à la generation de code */
INITIALISATION
   traducMot := (MOTS)*{iif}

OPERATIONS
   r1 <-- getMot ( ii ) = /* recuperer un mot */
   BEGIN
       r1 := traducMot ( ii )
   END
   ;
   setT2 ( mot , sens ) = /* mettre un mot + signif */
   BEGIN
       traducMot ( mot ) := sens
   END
END
```

Notes : Machines prouvées et code C généré.

3.3 Cas : variation sur dictionnaire

Un exemple de machine abstraite (`TradMot3.mch`) :

```
/*-------------------------------------------------
 * Author: Christian Attiogbé / Université de Nantes
 * Creation date: mar. nov. 30 2010
 -------------------------------------------------*/
```

```
MACHINE
    TradMot3
SETS
    MOTS = {foo, fii, fuu, faa} /* des mots */
    ; SIGNIF = {oof, iif, uuf, aaf} /* des significations */
CONSTANTS
    maxMot  /* un parametre */
PROPERTIES
    maxMot : NAT

DEFINITIONS
     LOCATION == 0..maxMot  /* une abstraction des mots */

CONCRETE_VARIABLES  /* parce qu'on veut les garder jusqu'à l'implantation */
    lesMots,
    traducMot /* des traductions de mots */

INVARIANT
    lesMots : LOCATION --> MOTS /* de l'abstrait vers le mot*/
&   traducMot : LOCATION --> SIGNIF /* signification de certains mots */
/* Pour generer du code :
    Ens. de depart sous forme d'intervalle necesaire ;
    fonction (totale) necessaire */

INITIALISATION
    traducMot :: LOCATION --> SIGNIF
    || lesMots :: LOCATION --> MOTS

OPERATIONS
    r1 <--getMot (ii) =  /* recuperer le mot à ii, jj */
    PRE
        ii : LOCATION & ii : dom(traducMot)
    THEN
        r1 := traducMot(ii)
    END

; setT2(mot,sens) = /* definir la traduction du mot */
   PRE
        mot : LOCATION  & mot /: dom(traducMot)
     &    sens : SIGNIF
  THEN
     traducMot(mot) := sens
    END
    /* les operations auxilliaires pour tester les preconditions */
;
res <-- bonneLocation (ii) = /* est ce qu'un indice est dans les bornes */
    PRE
        ii : NAT
    THEN
        res := bool(ii <= maxMot)
    END
END
```

et son implantation (`TradMot3_i.imp`) :

```
/*-------------------------------------------------
 * Author: Christian Attiogbé / Université de Nantes
 * Creation date: mer. déc. 1 2010
 -------------------------------------------------*/
IMPLEMENTATION
   TradMot3_i
REFINES
   TradMot3

DEFINITIONS
   LOCATION == 0..maxMot
VALUES
    maxMot = 5 /* instanciation du parametre */
INITIALISATION
   traducMot := (LOCATION) * {oof} ; /* les ( ) sont necessaires */
   lesMots := (LOCATION) * {foo}

OPERATIONS
   r1 <-- getMot ( ii ) = /* recuperer la traduc d'un mot */
   BEGIN
       r1 := traducMot ( ii )
   END
   ;

   setT2 ( mot , sens ) = /* mettre le sens d'un mot */
    BEGIN
       traducMot ( mot ) := sens
   END
;
res <-- bonneLocation (ii) = /* est ce qu'un indice est dans les bornes */
    BEGIN
        res := bool(ii <= maxMot)
    END
END
```

4 Relations

La particularité ici est d'implanter une relation comme une fonction totale. L'idée est simple : une relation $ra : A \leftrightarrow B$ est implantée par
une fonction totale $rc : A * B --> (0..1)$.
rc est une matrice avec $rc(a, b) = 1$ ou 0 selon que le couple (a, b) appartient à ra ou pas.

L'invariant de liaison entre ra et rc est : $rr = dom(ra \triangleright \{1\})$ ou bien $rr = ra{\sim}[\{1\}]$

Les exemples montrant des variations sur le dictionnaire en sont des illustrations.

4.1 Annuaire avec une relation

à suivre ...

5 Autres exemples

- Calculette 8 bits
- division euclidienne
- ...

Avertissement

Ces modèles et raffinements/implantations sont mis au point pour la version 4.0 de l'AtelierB et notament le générateur de code ComenC qui pose quelques restrictions. Ces modèles et raffinements/implantations vont être améliorés. Vous pouvez consulter les pages de cours pour avoir les versions les plus à jour de ce mémento et le source des modèles.

Références

- *Cours Méthode B*, C. Attiogbé
- *Manuel de référence du langage B*, ClearSy, www.atelierb.eu/php/manuels-atelier-b-fr.php
- *Documentations B*, ClearSy, www.tools.clearsy.com/index.php5 ?title=Comenc_Project