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, maj novembre 2012

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 (AnnuaireR)

Nous utilisons le contexte suivant pour la machine abstraite AnnuaireR.

```
/*-------------------------------------------------
 * Author: Christian Attiogbé
 * Université de Nantes / LINA
 ------------------------------------------------*/
MACHINE
    Ctx_ANNUAIRE
CONSTANTS
```

```
    maxPers,
    maxNums
PROPERTIES
    maxPers : 0..MAXINT
    &maxNums : 0 ..MAXINT
    & maxNums > 2 & maxNums < MAXINT
SETS
    PERSONNE
    ; NUMERO
DEFINITIONS
    NUMERO2 == 0..maxNums
END
```

Voici la machine abstraite qui modélise l'annuaire :

```
/*-------------------------------------------------
 * Author: Christian Attiogbé
 * Université de Nantes / LINA
 -------------------------------------------------*/
MACHINE
    AnnuaireR
SEES
    Ctx_ANNUAIRE
DEFINITIONS
    NUMERO2 == 0..maxNums
VARIABLES
    membres
    , telephones
    /* variables pour resultats (ensemble) des operations */
    , resRecup /* resultat de l'operation recupNum */
INVARIANT
    membres   <: PERSONNE
& telephones : membres <-> NUMERO2
&   resRecup  <: NUMERO2
INITIALISATION
    membres := {}
    || telephones := {}
    || resRecup := {}
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
;
ajoutNumMembre(mm, num) = /* ajouter le num d'un membre */
PRE
    mm : PERSONNE & mm : membres
    & num : NUMERO2
    & (mm,num) /: telephones
THEN
    telephones := telephones \/ {mm |-> num}
END
;
/* ress <-- recupNumMembre(mm) = ensuite ress est passe en global */
recupNumMembre(mm) = /* recuperer le numero d'un membre */
PRE
    mm : PERSONNE & mm : membres & mm : dom(telephones)
THEN
    resRecup := telephones[{mm}] /* plusieurs */
END
;
/*-------------------- operations auxilliaires -----------*/

res <-- estMembre(lenom) = /* dejaMembre */
PRE
lenom : PERSONNE /* & lenom : membres */
THEN
res := bool(lenom : membres)
END
;
 res <-- pasMembre(lenom) = /* la meme qu'au dessus */
  PRE
lenom : PERSONNE /* & lenom /: membres */
THEN
res := bool(lenom /: membres)
END
END
```

Premier raffinement de la machine AnnuaireR. Nous avons introduit un nouveau contexte.

```
/*-------------------------------------------------
 * Author: Christian Attiogbé
 -------------------------------------------------*/
MACHINE
    CtxR_ANNUAIRE
SETS
    OKKO = {ok, ko} /* pour faire les raffinements */
END
```

L'implantation de ce contexte est :

```
IMPLEMENTATION
   CtxR_ANNUAIRE_i
REFINES
   CtxR_ANNUAIRE
END
```

Voici le raffinement de la machine Annuaire.

```
/*-------------------------------------------------
 * Author: Christian Attiogbé
 * Université de Nantes / LINA
 -------------------------------------------------*/
REFINEMENT
   AnnuaireR_r
REFINES
   AnnuaireR
SEES
    Ctx_ANNUAIRE
    , CtxR_ANNUAIRE /* contexte supplementaire pour raffiner */
                    /* on y voit OKKO */
DEFINITIONS
    NUMERO2 == 0..maxNums /* astuce pour avoir la compatibilité entre les valeurs ... et NUMERO, sinon

VARIABLES
    resRecup /* non encore raffinée */
  ,  c_telephones /* raffinement de telephones */
CONCRETE_VARIABLES
   c_membres

INVARIANT
    c_membres : PERSONNE --> OKKO
    & membres = c_membres~[{ok}] /* liaison abstrait concret */
    & c_telephones : (PERSONNE * (NUMERO2)) +-> 0..1
     /* une relation peut etre encodee par une matrice */
    & telephones = dom(c_telephones |> {1}) /* liaison abstrait concret */
    /* les elements dans telephone sont ceux  pour les quels
      la matrice contient '1', les autres sont à '0' */
    & membres <: PERSONNE /* pas necessaire */
INITIALISATION
   c_membres := (PERSONNE)*{ko} ;
   c_telephones := (PERSONNE*(NUMERO2))*{0};
   resRecup :={}

OPERATIONS
   ajoutMembre ( lenom ) =
   BEGIN
       c_membres(lenom) := ok /* membres := membres \/ { lenom } */
   END
   ;
   supprimerMembre ( lenom ) =
   BEGIN
      c_membres(lenom) := ko /* membres := membres - { lenom } */
      ; /*  telephones := { lenom } <<| telephones */
      c_telephones := c_telephones <+ ({lenom}*(NUMERO2))*{0}
   END
   ;

   ajoutNumMembre(mm, num) = /* ajouter le num d'un membre */
   BEGIN
   c_telephones(mm |-> num) := 1
   END
   ;
```

```
   recupNumMembre(mm) = /* recuperer les numeros d'un membre */
   BEGIN
resRecup := ran(
        dom((({mm}<| dom(c_telephones)) <| c_telephones) |>{1})) /* ceux à 1 */
   END
;
/*-------------------- operations auxilliaires -----------*/
   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
```

Deuxième raffinement de la machine AnnuaireR. Il s'agit d'une implantation :

```
/*-----------------------------------------------------
 * Author: Christian Attiogbé
 * Université de Nantes / LINA
 -----------------------------------------------------*/
IMPLEMENTATION
   AnnuaireR_i
REFINES
   AnnuaireR_r

SEES
    Ctx_ANNUAIRE
    , CtxR_ANNUAIRE /* pour voir OKKO = {ok, ko} */

DEFINITIONS
    NUMERO2 == 0..maxNums /* astuce pour avoir la compatibilité entre les valeurs ... et NUMERO, sinon

CONCRETE_VARIABLES
     c_resRecup
    ,cf_telephones /* final concrete telephone */
    /* var auxilliaires devant etre typées dans les operations */
```

```
    ,ii

INVARIANT
   /* c_membres : PERSONNE --> OKKO    & */
   cf_telephones : (PERSONNE * (NUMERO2)) --> (0..1)
   & (cf_telephones |> {1}) = c_telephones |> {1}
   & (cf_telephones |> {0})  = c_telephones |> {0}
   & dom(cf_telephones) = (dom(c_membres))*(NUMERO2)
     /* une relation peut etre encodee par une matrice */
   & c_resRecup : (NUMERO2) --> OKKO
     /* var auxil, pour besoin de typage */
   & ii : NUMERO2
INITIALISATION

   c_membres := (PERSONNE)*{ko} ;
   cf_telephones := ((PERSONNE)*(NUMERO2))*{0};
   c_resRecup := (NUMERO2)*{ko};
   ii := 0

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

   supprimerMembre ( lenom ) =
   BEGIN
      c_membres(lenom) := ko /* membres := membres - { lenom } */
      /*  telephones := { lenom } <<| telephones */
      /* c_telephones := ((dom(dom(c_telephones))-{lenom})  <|  dom(c_telephones) ) <| c_telephones */
      ;
      ii := 0; /* ii dans NUMERO2 */
      WHILE (ii < maxNums) DO
      cf_telephones(lenom |-> ii) := 0
      ; ii := ii + 1
  INVARIANT
      ii : NUMERO2
  &    cf_telephones : ((PERSONNE) * (NUMERO2)) --> (0..1)
  &    (lenom |-> ii) : dom(cf_telephones)
   /*&   (lenom |-> ii) : dom(c_telephones)*/
  VARIANT
      maxNums - ii
  END /* while */
 ;
  IF (ii = maxNums)
  THEN
      cf_telephones(lenom |-> ii) := 0
  END

   END
   ;

ajoutNumMembre(mm, num) = /* ajouter le num d'un membre */
```

```
BEGIN
    cf_telephones(mm |-> num) := 1
END
;
recupNumMembre(mm) = /* recuperer les numeros d'un membre */
    /* ress := ran(
    dom((({mm}<| dom(c_telephones)) <| c_telephones) |>{1})) */
    /* ceux à 1 */
VAR vc  IN
        ii := 0; /* dans NUMERO2 */
   WHILE (ii < maxNums) DO
       vc := cf_telephones(mm |-> ii);
       IF (vc = 1)
       THEN
         c_resRecup(ii) := ok
       END
         ; ii := ii + 1
   INVARIANT
       ii : NUMERO2 & ii : dom(c_resRecup)
      & c_resRecup : (NUMERO2) --> OKKO
   VARIANT
      maxNums - ii
   END
END
;
/*-------------------- operations auxilliaires -----------*/
   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
```

5 Autres projets et génération de code

5.1 Calculette 8 bits

On développe une petite calculette ayant 2 registres (RP et RS) de 8 bits, et proposant des opérations élémentaires : stockage d'une valeur dans un registre ; addition/soustraction des valeurs dans les registres, ...

```
/*-------------------------------------------------
 * Author: Christian Attiogbé / Université de Nantes
 * Creation date: mars 2009, maj oct 2010
 -------------------------------------------------*/
MACHINE
    Calculette8
VARIABLES
    rp, /* registre principal */
    rs  /* registre secondaire */
INVARIANT
    rp : NAT
    & rs : NAT
    & 0 <= rp & rp <= 255 /* 8 bits */
    & 0 <= rs & rs <= 255  /* 8 bits */
INITIALISATION
    rp :=0 || rs := 0
OPERATIONS
    inc1 = /* incrementer le registre principal de 1 */
    PRE
        rp +1 <= 255
    THEN
        rp := rp + 1
    END
    ;
    dec1 = /* decrementer le registre principal de 1 */
    PRE
        rp - 1 >= 0
    THEN
        rp := rp - 1
    END
    ;
    storeRP(vv) =
    PRE
        vv : NAT & 0<= vv & vv <= 255
    THEN
     rp := vv
    END
    ;
    storeRS(val) =
    PRE
        val : NAT & val <= 0 & val <= 255
    THEN
        rs := val
    END
    ;
    res <-- cmp = /* comparer les contenus de RP et RS : 1 = egalite ; 0 different*/
    res := bool(rs = rp)
```

```
    ;
    res <-- getRP = /* recuperer la valeur de RP */
    res := rp ;

    res <-- getRS = /* recuperer la valeur de RS */
    res := rs

END
```

L'implantation de la calculette est comme suit :

```
/*-----------------------------------------------
 * Author: Christian Attiogbé
 * Université de Nantes / LINA
 * Creation date: mer. nov. 7 2012
 -----------------------------------------------*/
IMPLEMENTATION
   Calculette8_i
REFINES
    Calculette8
    CONCRETE_VARIABLES
   rp ,
   rs
INITIALISATION
   rp := 0 ;  rs := 0
OPERATIONS
   inc1 =
   BEGIN
       rp := rp + 1
   END
 ;
   dec1 =
   BEGIN
       rp := rp - 1
   END
 ;
   storeRP ( vv ) =
   BEGIN
       rp := vv
   END
;
   storeRS ( val ) =
   BEGIN
       rs := val
   END
;
   res <-- cmp =
   res := bool ( rs = rp )
;
   res <-- getRP =
   res := rp
;
   res <-- getRS =
   res := rs
END
```

Le code est généré pour cette calculettre

5.2 Exemple de génération d'application pour utiliser le code généré

On fabrique une machine dont la seule opération est main ; ici dans la machine elle ne fait rien ; dans le raffinement, on importera notre calculette puis on se servira de ses opérations dans l'implantation du main.

```
MACHINE
    Application
OPERATIONS
    main = skip
END
```

Implantation pour creer un main (en langage C)

```
IMPLEMENTATION
   Application_i
REFINES
   Application
IMPORTS
   Calculette8

OPERATIONS
    main =
    VAR val IN
            val <-- getRP;
            /* ici on fera un printf("Val = %d",val); dans le code genere*/
            /* storeRS(8) */
            val <-- getRP
            /* ici on fera un printf("Val = %d",val);*/
   END
END
```

Le code est généré pour cette application.

5.3 Division euclidienne

```
 /*-------------------------------------------
 Euclide  - Correct Program Development
 NaBLa Project - LINA / University de Nantes
  Christian Attiogbe
-------------------------------------------*/
MACHINE
euclide
OPERATIONS

 reste, quot <-- calculReste (divis, divid) =
 PRE
divis : NAT
 &   divid : NAT
 &   divis > 0
 &      divis <= divid /* sinon b le trouve */
 THEN
  ANY vq, vr WHERE
```

```
  vq  : NAT
&  vr  : NAT
&  divid = vq*divis + vr
THEN
  quot  :=  vq
||  reste  :=  vr
  END
 END
END
```

et voici le raffinement de la machine Euclide

```
IMPLEMENTATION
euclide_I1_1
REFINES
euclide

OPERATIONS
reste, quot <-- calculReste(divis, divid) =
VAR rr, mm, dd IN
  dd := 0         /* le  multiple de divis qui nous interesse*/
; mm := dd*divis  /* dd*divis*/
; rr := divid     /* le reste courant, initialement c'est donc divid  */
        ;WHILE     /* Maintenant, calcul du quot+reste par soustractions successives
des multiples de mm de rr (rr = rr - (dd*divis))   */
    rr >= divis
DO
dd := dd + 1      /* pour calculer prochain multiple */
; rr := rr - divis  /* le calcul du reste progresse    */
INVARIANT
     dd : NAT
     &    rr : NAT
     & divid : NAT & divid <= MAXINT
&    0 <= rr
&    dd <= MAXINT
&    divid = dd*divis + rr  /* mm = dd*divis */
VARIANT
     rr      /* décroit jusqu'a etre < divis */
END
;       reste := rr
; quot:= dd
END
END
```

5.4 Contrôle de l'état d'une jauge

D'abord nous définissons un contexte qui rassemble les ensembles et constantes communs au projet Jauge.

```
/*-----------------------------------------------------
 * Author: Christian Attiogbé
 * Université de Nantes / LINA
 -----------------------------------------------------*/
MACHINE
    ContextJauge
SETS
```

```
    ETATJ = {vert, rouge, orange}
CONSTANTS
    SeuilV, SeuilO, SeuilR, maxVal
PROPERTIES
    maxVal : 0..100 & maxVal = 100
    &    SeuilV :  0..maxVal & SeuilV = 40
    & SeuilO : 0..maxVal & SeuilO = 80
    & SeuilR :  0..maxVal & SeuilR = 80
END
```

Ce contexte est raffiné (implanté) de la façon suivante :

```
IMPLEMENTATION
   ContextJauge_i
REFINES
   ContextJauge
VALUES
     maxVal = 100
  ;  SeuilV = 40
  ;  SeuilO  = 80
  ;  SeuilR = 80
END
```

Voici maintenant la machine abstraite :

```
/*-----------------------------------------------------
 * Author: Christian Attiogbé
 * Université de Nantes / LINA
 -----------------------------------------------------*/
MACHINE
    Jauge
SEES
    ContextJauge
VARIABLES
      valJauge /* valeur de la jauge */
    , etatJauge /* etat de la jauge
                vert : valde jauge < seuilV
                orange : entre seuilv et seuilOrgange
                rouge : au dela de seuilOrange */
INVARIANT
    valJauge : 0..maxVal
    & etatJauge : ETATJ
    & ((valJauge < SeuilV) => etatJauge = vert)
    & ((SeuilV <= valJauge & valJauge <= SeuilO) => etatJauge = orange)
    & ((valJauge > SeuilO) => etatJauge = rouge)
INITIALISATION
    valJauge := 10
    || etatJauge := vert
OPERATIONS
    incrJaugeV(pas) =
    PRE
        pas : 0..maxVal
        & valJauge+pas : 0..(SeuilV-1)
    THEN
         valJauge := valJauge+pas
```

```
        || etatJauge := vert
    END
;    incrJaugeO(pas) =
    PRE
        pas : 0..maxVal
        & valJauge+pas : SeuilV..SeuilO
    THEN
           valJauge := valJauge+pas
        ||  etatJauge := orange

    END
;    incrJaugeR(pas) =
    PRE
        pas : 0..maxVal
        & valJauge+pas : (SeuilO+1)..maxVal
    THEN
       valJauge := valJauge+pas
        || etatJauge := rouge
     END
;
     incrJauge(pas) = /* celle plus abstraite ; bien mieux */
     PRE pas : 0..maxVal &
         valJauge+pas : 0..maxVal
     THEN
         valJauge, etatJauge :(
     valJauge : 0..maxVal &
        etatJauge : ETATJ &
     ((valJauge < SeuilV) => etatJauge = vert)  &
       ((SeuilV <= valJauge & valJauge <= SeuilO) => etatJauge = orange) &
      ((valJauge > SeuilO) => etatJauge = rouge))
       END
END
```

Le raffinement (implantation)

```
/*-----------------------------------------------------
 * Author: Christian Attiogbé
 * Université de Nantes / LINA
 -----------------------------------------------------*/
IMPLEMENTATION
   Jauge_i
REFINES
   Jauge
SEES
    ContextJauge
CONCRETE_VARIABLES
   valJauge ,
   etatJauge

INITIALISATION
   valJauge := 10 ;
   etatJauge := vert

OPERATIONS
   incrJaugeV ( pas ) =
   BEGIN
```

```
        valJauge := valJauge + pas ;
        etatJauge := vert
    END
    ;
    incrJaugeO ( pas ) =
    BEGIN
        valJauge := valJauge + pas ;
        etatJauge := orange
    END
    ;
    incrJaugeR ( pas ) =
    BEGIN
        valJauge := valJauge + pas ;
        etatJauge := rouge
    END
    ;
    incrJauge ( pas ) =
    BEGIN
        VAR valAux IN
        BEGIN
        valAux := valJauge + pas;
        IF (valAux  < SeuilV )
        THEN valJauge := valAux ;
            etatJauge := vert
        ELSE
            IF ( SeuilV <= valAux & valAux <= SeuilO )
            THEN etatJauge := orange ;
                valJauge := valAux
            ELSE
                IF ( valAux > SeuilO ) THEN
                    etatJauge := rouge;
                    valJauge := valAux
                END
            END
        END
    END
    END
END
END
```

5.5 Etc Etc

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