ApplicaZion - gestion d'un mini-dictionnaire
la Méthode B : du modèle abstrait au code en C

Christian Attiogbé
Université de Nantes, Décembre 2021

1 Introduction de l'exercice

On veut écrire un programme correct qui construit et gère un mini-mini-dictionnaire. Notre programme exploite un ensemble de couples (mot, signification) nommé dico qui représente le dictionnaire. La signification est ici prise sous une forme très abstraite : un mot n'a qu'une et une seule signification, mais on admet que deux mots peuvent avoir une même signification.

Au début du programme, le dictionnaire sera vide. On propose les opérations suivantes à travers notre programme.

- AjoutMot(mm, signif) : cette opération ajoute le mot mm et sa signification (signif) dans dico ;
- RetraitMot(mm) : cette opération enlève le mot mm du dictionnaire ;
- existeMot(mm) : vérifie si le mot mm est dans le dictionnaire (dico) ou non ; l'opération renvoie une valeur booléenne ;
- sensDuMot(mm) : recherche et renvoie la signification d'un mot donné en paramètre.

On nous demande d'écrire une machine abstraite B nommée DicoMot qui spécifie l'état du dictionnaire et modélise les différentes opérations décrites précédemment.

On utilisera pour ce faire un ensemble abstrait MOT qui représente l'ensemble de tous les mots possibles imaginables et un ensemble abstrait SIGNIFIK qui représente (les abstractions de) toutes les significations possibles imaginables.

2 Analyse et modélisation en B

L'exercice est relativeent simple ; il ressort de l'analyse qu'on peut modéliser simplement ce dictionnaire par une fonction de l'ensemble des mots vers l'ensemble des significations. Nous avons le choix de modélisation de ces ensembles ; considérons comme indiqué dans l'énoncé un ensemble de mots abstraits MOT, mais le dictionnaire manupuleraun sous-ensemble fini de ces mots. Considérons aussi un ensemble de significations énumérées (SIGNIFIK).

2.1 Modélisation

Nous proposons une machine abstraite DicoMo qui modélise le mini-dictionnaire ;

Un composant abstrait (machine)

```
MACHINE
    dicoMot
SEES Ctx_Dico // Ctx_Dico est une machine qui contient les ens+constantes
              // utilisées ici mais peuvent être partagées avec d'autres machines
VARIABLES
    mots /* sous ensemble de mots */
,    dico
INVARIANT
    mots : FIN(MOT) /* sous ens fini de mots utilisés, borné */
    & card(mots) <= maxMots
&    dico : mots --> SIGNIFIK
INITIALISATION
    mots := {} ||
    dico := {} /* : mots --> SIGNIFIK */
OPERATIONS
    ajoutMot(mm, signif) = /*?---- Ajouter un mot et sa signification */
    PRE mm : MOT & mm /: mots
        & signif : SIGNIFIK
        & (mm |-> signif) /: dico
        & card(mots) < maxMots
    THEN
        mots := mots \/ {mm}
        || dico(mm) := signif
        /* dico := dico \/ { mm |-> signif}  */
    END
;
    RetraitMot(mm) = /*?---- Retirer un mot du dico */
PRE  mm : MOT & mm : mots // = m : dom(dico)
    & card(mots) >=1
    THEN
        mots := mots - {mm}
    ||  dico := {mm} <<| dico
   END
;
res <-- rechercheSignifMot(mm) = /*?---- Trouver la signification d'un mot */
    PRE mm : MOT & mm : dom(dico)
    THEN
     res := dico(mm)
    END
;
/*----fonctions auxilliaires : pour tester les PREcond., avant les appels */

bb <-- existeMot(mm) = /* est-ce que le mot mm existe dans le dico */
    PRE  mm : MOT
```

```
    THEN
      bb := bool(mm : dom(dico))
    END
;
rb <-- placeDisponible = /* est-ce qu'il y a encore de la place  */
    BEGIN
        rb := bool(card(mots) < maxMots)
    END
;
rb <-- estceAjoutPossible (mm, signi) = /*?---- pour la PRE de ajoutMot */
    PRE
        mm : MOT
        & signi : SIGNIFIK
    THEN
        rb := bool(mm /: mots // & (mm |-> signi) /: dico
                & card(mots) < maxMots)
    END
END
```

Cette machine abstraite, est analysée syntaxiquement puis prouvée (obligations de preuve issues de l'invariant de la machine) avec l'outil AtelierB. Les statistiques de preuve sont ci-après (Figure 1)

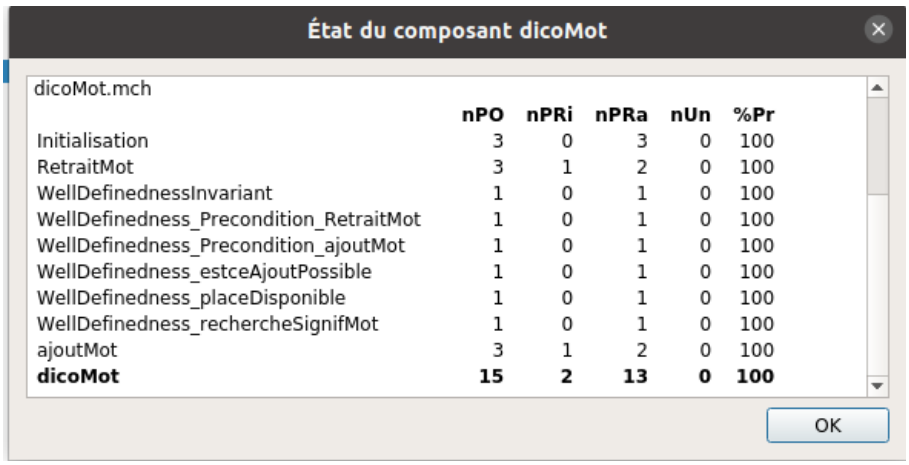

Figure 1: Statistiques PO et preuves

La machine `Ctx_Dico` est présentée ci-après ; nous y avons introduit des constantes supplémentaires (des mots m1, m2, m3) juste pour le besoin de l'application adhoc que nous allons construire par la suite.

Machine abstraite

```
/* Machine Ctx_Dico
 * Author: attiogbe-c
 * Creation date: 12/2021
 */
MACHINE
    Ctx_Dico
SETS
    MOT /* un ensemble abstrait de mots */
    ; SIGNIFIK = {s0,s1,s2} /* ensemble abstrait des significations */
CONSTANTS
    maxMots // borne du dictionnaire
    , m1, m2, m3 // des mots (pour tester dans une appli par exemple)
PROPERTIES
    maxMots : 1..MAXINT
    & m1 : MOT
    & m2 : MOT
    & m3 : MOT
END
```

2.2 Raffinement

Nous raffinons en une seule étape (donc directement une impléentation) la machine abstraite DicoMot.

Une impplémentation

```
/* Implementation dicoMot_i
 * Author: Christian Attiogbé
 * Creation date: 12/2021
 */
IMPLEMENTATION
   dicoMot_i
REFINES
   dicoMot
SEES
    Ctx_Dico
    , Ctx_Impl_Dico // où est défini l'ens OKKO
CONCRETE_VARIABLES
    c_mots,    /* nouvelles variables, concretes */
    c_dico,
    nbMots
INVARIANT
    c_mots : MOT --> OKKO   /* mots utilisés (ok) ou non (ko) */
```

```
    & mots = c_mots~[{ok}]  /* liaison abstrait concret       */
    & c_dico : MOT --> SIGNIFIK
    & dico = (mots <| c_dico) /* liaison abstrait concret     */
    & nbMots : 0..maxMots
    & nbMots = card(mots)
INITIALISATION
   c_mots := (MOT)*{ko}; /* aucun mot n'est encore utilisé */
   c_dico := (MOT)*{s0};  /* qui est vide */
   nbMots := 0
OPERATIONS
   ajoutMot ( mm , signif ) =
   BEGIN
       c_mots(mm) := ok  /* mots := mots \/ { mm } */
   ;   c_dico (mm) := signif
   ;   nbMots := nbMots + 1
   END
;
   RetraitMot ( mm ) =
   BEGIN
       c_mots(mm) := ko  /* mots := mots - { mm } */
   ;   nbMots := nbMots - 1
   END
;
res <-- rechercheSignifMot ( mm ) =
   BEGIN
       res :=  c_dico(mm)   /* res := dico ( mm ) */
   END
;
//------fonctions auxilliaires : pour test des PREconds avant les appels
bb <-- existeMot ( mm ) =
   BEGIN
       /* bb := bool ( mm : dom (dico))*/
       /* /!\ dans dom de dico, => ok ou ko dans le ran de c_mots */
       VAR okko IN
           okko := c_mots(mm);
           IF okko = ok /* donc dans le dom et ok */
           THEN bb := TRUE
           ELSE bb := FALSE /* dans le dom et ko */
           END
       END
   END
;
rb <-- placeDisponible =
    BEGIN
        rb := bool(nbMots < maxMots)
    END
```

```
;
rb <-- estceAjoutPossible (mm, signi) =
    BEGIN
       // rb := bool(mm /: mots & (mm |-> signi) /: dico & card(mots) < maxMots)
       VAR okko IN
           okko := c_mots(mm);
           IF (okko = ko & nbMots < maxMots) //  mm /: mots donc nbMots < maxMot
           THEN rb := TRUE
           ELSE rb := FALSE
           END
       END
    END
END
```

Nous disposons à ce stade d'un composant prouvé, raffiné et pour lequel, on peut générer le code en C avec Atelier.

La machine `Ctx_Impl_Dico` est construite comme suit :

Composant

```
MACHINE
    Ctx_Impl_Dico
SETS
    OKKO = {ok, ko} /* indique si mot utilise ou non */
END
```

L'implémentation de la machine `Ctx_Dico` est effectuée comme suit :

Implementation Ctx_Dico_i

```
IMPLEMENTATION Ctx_Dico_i
REFINES Ctx_Dico
DEFINITIONS
    PLAGE_MOT == 0..20  /* une plage pour implanter l'ens des mots */
VALUES
    MOT = (0..20)  /* implantation de l'abstrait MOT */
    ; maxMots = 22   /* une valeur quelconque ici */
    ; m1 = 10  /* quelques mots */
    ; m2 = 12
    ; m3 = 13
 END
```

3 Construction d'une application utilisant le composant dicoMot

A présent, nous allons créer une application adhoc (juste un scénario), pour montrer comment réutiliser le composant `dicoMot`, et aussi montrer comment se servir de la Méthode B pour créer le squelette de l'application, prouver que ce squelette est correct, puis en générer le code en C. Ce code en C, peut être ensuite étendu par des ajouts de code (non critiques) ; c'est ce qui a été fait avec des traces d'exécution.

3.1 Aperçu de la démarche

Pour le principe, un programme ou une application logicielle, importe des composants (machines raffinées et implémentées) déjà développés.

Nous créons ici une application (nommée `MyAppli`), qui va importer la machine `dicoMot`, et qui se servira de ces opérations. La machine abstraite est simple, on y prévoit une seule opération (`main`) et il n'y a aucun espace d'états à modifier.

Machine abstraite MyAppli, un scénario d'utilisation du composant dicoMot.

```
MACHINE
    MyAppli
OPERATIONS
    main = skip // ici on ne fait rien, on ne modifie rien,
END
```

Raffinons à présent cette machine, en important la machine `dicoMot` pour accéder aux opérations qu'elle offre.

Implémentation de MyAppli

```
/* Implementation MyAppli_i
 * Author: Christian Attiogbe
 * Creation date: 12/2021
 */
IMPLEMENTATION MyAppli_i
REFINES MyAppli
SEES    Ctx_Dico // pour recuperer les mots et signification
IMPORTS dicoMot   // pour importer toutes les opérations offertes
OPERATIONS
  main =
  BEGIN
      VAR lc, lb IN
```

```
            lb <-- estceAjoutPossible(m1, s2);
            IF (lb = TRUE)
            THEN
                ajoutMot(m1, s2)
            END;
            lb <-- estceAjoutPossible(m2, s1);
            IF (lb = TRUE)
            THEN
                ajoutMot(m2, s1)
            END;
            lb <-- existeMot(m1);
            IF (lb = TRUE)
                THEN
                  RetraitMot(m1)
              END
              ;
      lc <-- existeMot(m1)
      END
  END
END
```

Nous avons maintenant une application complète ; toutes les machines sont analysées synytaxiquement et prouvées en utilisant AtelierB. Les statistiques des PO et preuves sont indiquées ci-après (Figure 2).

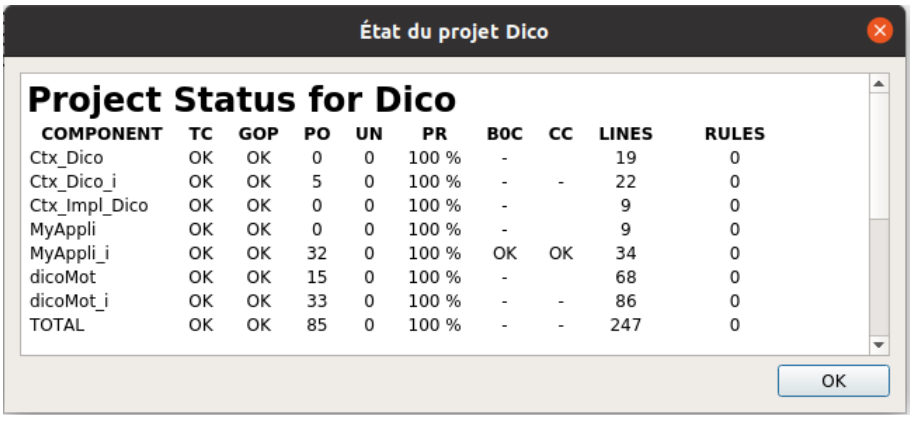

Figure 2: Statistiques PO et preuve sur le projet complet

Le code C est généré pour l'application entière. Il a été complété par quelques instructions d'entrée sortie, pour voir la trace du déroulement de l'application en cours d'exécution (Figure 3).

```
./resultat

----------------------------
Execution en cours
----------------------------
Ajout de m1
Ajout de m2
Resultat de existeMot m1 =   true
Retrait de m1
Resultat de existeMot :  m1 =   false
-----Fin execution---------
```

Figure 3: Trace d'exécution de MyAppli

3.2 Architecture globale de l'application

En guise de synthèse voici une architecture globale du projet complet (Figure 4).

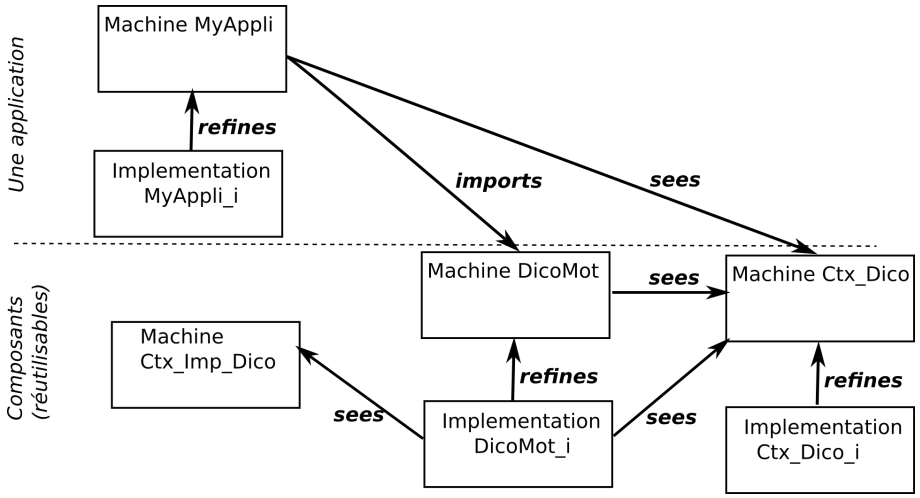

Figure 4: Architecture de l'application

4 Conclusion

Nous avons présenté dans ce document une petite étude de cas avec la Méthode B, allant de la modélisation jusqu'à la réalisation d'une application, dont le code a été généré. L'AtelierB (4.5.1) a été utilisé comme outil support de la méthode, à toutes les étapes. Nous avons illustré différents aspects de la méthode, concernant par exemple, le partage de partie de modèle (en utilisant des machines qui sont vues/partagées par d'autres), la structuration des machines avec des opérations auxilliaires, la réutilisation de composants. Nous n'avons pas abordé ici, les détails sur les preuves de cohérence des machines abstraites et les preuves de raffinement.

Cette étude de cas complète est un guide petit méthodologique pour aider à la compréhension et à la pratique de la méthode B.

Références

- Pages du cours (DUT Info) : http://pagesperso.ls2n.fr/˜ attiogbe-c/mespages/methodologie-production-applications.html
- Cours Méthode B (Master Info) : http://pagesperso.ls2n.fr/˜ attiogbe-c/mespages/enseignements-pcf.html