TP 1 : Introduction à l'atelier B

SOMMAIRE

I - Lancement de l'atelier B et création d'un nouveau projet 2
II - Création d'une machine abstraite 2
1 – Machine « search » 2
2 – Obligation de preuve 3
3 – Génération des obligations de preuves POs 3
4 – Statut du projet 3
II - Machine abstraite et implémentation 4
1 – Machine abstraite « utils » 4
2 – Implémentation « utils.imp » 4
3 – Utilisation du « prouveur » 5
III - Introduction au « prouveur » interactif 5
1 – Pouveur automatique 5
2 – Prouveur interactif 6
3 – Preuve d'un lemme mathématique 6
IV - Machine abstraite de réservation 7
1 – Réalisation de la machine 7
2 – Générer les Pos et les prouver 8
3 – Conclusion 8

I - LANCEMENT DE L'ATELIER B ET CREATION D'UN NOUVEAU PROJET

Lors de la **création d'un nouveau projet dans l'atelier B**, nous pouvons remarquer la création de deux nouveaux répertoires : "bdp" et "lang". Ces deux répertoires vont êtres utilisés par l'atelier B de la manière suivante :

- "bdp" : Ce répertoire contient les différents **fichiers outils nécessaires à l'atelier B** pour gérer et appliquer la méthode B à notre « projet ». Il contient également les fichiers de documentations qui pourront être générés par l'atelier B à la demande.

- "lang" : Ce répertoire contient quant à lui l'ensemble des **fichiers sources générés par l'atelier B** dans la langage spécifié pour le projet.

II - CREATION D'UNE MACHINE ABSTRAITE

1 – MACHINE « SEARCH »

```
MACHINE
    search
CONSTANTS
    nn,
    array
PROPERTIES
    nn: NAT &
    array: 0..nn-1 --> NAT
OPERATIONS
    yy <-- searchOp(vv) =
    PRE vv:NAT
    THEN
        IF vv : ran(array) THEN
            yy := TRUE
        ELSE
            yy := FALSE
        END
    END
```

On considère une fonction de recherche d'un élément « v » dans un tableau « t » d'entiers de taille « n ».

L'objectif de cette question est créer une machine abstraite qui représente cette fonction.

Ainsi, en suivant les recommandations de l'énoncé, nous avons réalisé la machine ci-contre.

Nous pouvons donc remarquer les principaux éléments :

- nn, array : des constantes
- searchOp(vv) : l'opération qui réalise la recherche de l'élément « vv » dans le tableau.

2 - OBLIGATION DE PREUVE

```
MACHINE m
CONSTANTS k
PROPERTIES T
VARIABLES v
INVARIANT I
INITIALIZATION L
OPERATIONS
y op(x) ,
PRE P THEN S
```

L'état d'une machine abstraite doit vérifier en tout temps l'**invariant**.

Une **« obligation de preuve »** est ainsi associée à chaque prédicat défini dans l'invariant.

L'obligation de preuve doit s'assurer du **respect de ce prédicat lors de toute modification effectuée** sur les variables considérées dans le prédicat.

Dans l'exemple ci contre, il faudra s'assurer que le prédicat P soit vrai comme « pré-condition » à la réalisation de la substitution S dans l'opération "op(x)".

Si P est vérifié**, l'obligation de preuve veillera à ce que la substitution S préserve l'invariant I à l'issue de sa réalisation**. Si ce n'est pas le cas, S ne sera réalisée.

3 - GENERATION DES OBLIGATIONS DE PREUVES POS

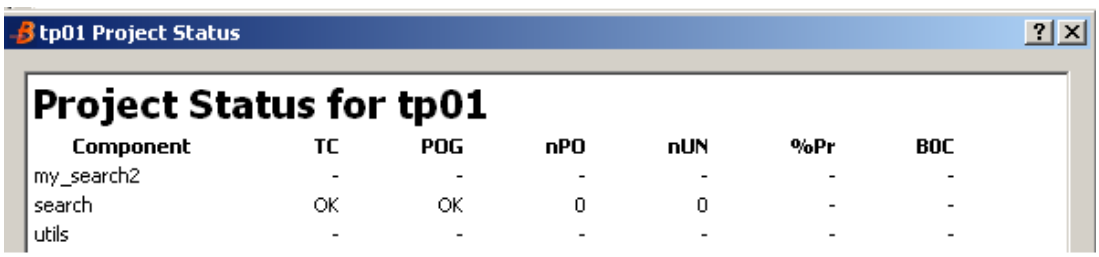
En ce qui concerne les obligations de preuves de notre machine, nous n'avons remarqué aucun problème.

Remarque : En réalisant la machine selon la description de l'énoncé, nous n'avons aucune clause « INVARIANT », de ce fait aucune « obligation de preuve » n'a put être générée.

4 - STATUT DU PROJET

Signification des acronymes du statut :

- TC : Type Checked, vérification du typage des variables utilisées
- POG : PO Genereated , état de la génération des preuves (terminée et réussie ou non)
- nPO : Proof Obligations, nombre d'obligations de preuve générées suite à l'invariant
- nUn : Unproved, nombre de PO non prouvées
- Pr : Proved : nombre d'obligations de preuves vérifiées
- BOC : BO checked, état au niveau B0



II - MACHINE ABSTRAITE ET IMPLEMENTATION

1 – MACHINE ABSTRAITE « UTILS »

```
MACHINE
    utils
OPERATIONS
    yy <-- minOp(aa,bb) =
    PRE
        aa : NAT &
        bb : NAT
    THEN
        yy := min({aa,bb})
    END;

    zz <-- maxOp(aa,bb,cc) =
    PRE
        aa,bb,cc : NAT*NAT*NAT
    THEN
        zz := min({aa,bb,cc})
    END
```

Remarques de syntaxe :

Nous avons utilisé deux exemples de typage des variables. Dans la première opération, nous avons défini les variables « aa » et « bb » séparément. Dans la seconde, et puisque « aa », « bb » et « cc » appartiennent au même ensemble NAT, nous avons pu utiliser la notation par produit cartésien.

Petite remarque, Les fonctions intégrées « min » et « max » travaillent sur des ensembles d'entiers, d'où la nécessité d'utiliser les accolades « {} ».

Remarque sur les Pos :

Encore une fois, nous n'avons pas utilisé d'invariant explicite et donc il n'y a aucune obligation de preuve à cet instant

2 – IMPLEMENTATION « UTILS.IMP »

```
IMPLEMENTATION
  utils_i
REFINES
  utils
OPERATIONS
   yy <-- minOp(aa,bb) =
   IF aa >= bb THEN yy:= bb
   ELSE yy:=aa
   END;

   zz<-- maxOp(aa,bb,cc) =
   IF (aa >= bb) & (aa >= cc) THEN zz:= aa
   ELSIF (bb >=aa) & (bb >= cc) THEN zz:= bb
   ELSE zz:= cc END
END
```

Pour générer ces implémentations, nous nous sommes laissé influencer très fortement par la syntaxe et la logique de programmation du langage C. De ce fait, nous pouvons remarquer l'utilisation de doubles conditions dans les clauses IF.

Remarque sur les Pos :

A ce niveau, l'atelier B nous a générer 10 preuves pour vérifier que notre implémentation effectue bien les spécifications énoncées dans la machine abstraite.

3 – UTILISATION DU « PROUVEUR »

Nous avons essayé de prouver ces obligations de preuve à l'aide du **« prouveur » automatique de force successives 0, 1, 2 et 3**. Nous avons ainsi pu remarquer que l'utilisation de « force » de niveau supérieur permet, de temps à autre, de prouver certaines preuves qui ne l'étaient pas au niveau inférieur.

Cependant, nous pouvons ajouter que nous n'avons pas le même comportement selon les conditions que nous indiquons dans notre implémentation.

Par exemple, lors de l'utilisation de de symboles de comparaison « strict », le prouveur chercher à montrer que (b+1 <= a). Or « b » appartenant à l'ensemble NAT et ce dernier étant borné, il se peut que b+1 dépasse cette limite si « b » égale MAXINT. De ce fait, il ne pourra prouver que « a » appartient bien à NAT avec la condition (b+1 <= a).

Nous avons également pu remarquer que le prouveur rencontre quelques difficultés lorsque nous indiquons plusieurs conditions dans les clauses IF.

Et enfin, dans le cas où tous les nombres, « aa », « bb », « cc » dont égaux, nous devons en choisir un arbitrairement pour le désigner comme valeur maximale de l'ensemble. Ceci est également un élément que le prouveur a du mal a vérifier.

Ainsi, nous pouvons dire que pour qu'un maximum d'obligation de preuve soit vérifié, il faut les détaillées un maximum dans l'implémentation pour faciliter le travail « automatique » du prouveur.

III - INTRODUCTION AU « PROUVEUR » INTERACTIF

1 – POUVEUR AUTOMATIQUE

```
MACHINE
   Ex1
VARIABLES
   few, many
INVARIANT
   few <: NATURAL &
   many <: NATURAL &
   few <: many
INITIALISATION
   few,many := {1,2,3},{2,3,4}
END
```

Si nous regardons cet exemple, nous pouvons lire :

- La déclaration de deux variables « few » et « many »
- Les prédicats d'invariant tel que « few » et « many » sont des sous ensemble de NATURAL
- Et « few » est même un sous ensemble de « many »
- L'initialisation spécifie les valeurs comprises dans les intervalles « few » et « many »

Le prouveur automatique de force 0, 1, 2 et 3 cherche donc à vérifier l'invariant restrictif : few <: many

Pour ce faire, il cherche à vérifier que chaque élément de « few » est bien inclus dans « many ».

2 - PROUVEUR INTERACTIF

En utilisant le prouveur interactif, nous remarquons que le prouveur n'arrive pas à vérifier l'invariant tel que

---
"`Check that the invariant (few <: many) is established by the initialisation - ref 3.3"' => 1: {2,3,4}

En saisissant la commande « pr » nous obtenons le message suivant :

Bfalse

En effet, le résultat de l'initialisation est faux et ne vérifie pas l'invariant. Si nous ne changeons pas les valeurs fournies, nous ne pourrons rien faire d'autre.

3 - PREUVE D'UN LEMME MATHEMATIQUE

Prouvez à l'aide du prouveur interactive le lemme suivant :

Si a ∈ 1, 2, ..., 10 et b ∈ 2, ..., 10 et c ∈ 3, ..., 10 alors max(a, b, c) ∈ 1, 2, ..., 10

```
MACHINE
   Exemple2

VARIABLES
   aa, bb, cc,dd

INVARIANT
   aa : 1..10 &
   bb : 2..10 &
   cc : 3..10 &
   dd:1..10 &
   dd=max({aa,bb,cc})

INITIALISATION
   aa:=1||
   bb:=2||
   cc:=3||
   dd:=2
END
```

Pour traduire ce lemme, nous avons déclaré les variables aa, bb, cc et dd qui représente la valeur du max.

Dans l'invariant, nous avons directement traduit les conditions d'appartenance des variables aux différents ensembles.

Nous avons ensuite défini la condition sur « dd » qui devra appartenir au même ensemble que « aa » et qui représentera le max de ces trois variables.

Pour tester notre lemme, nous avons initialisé les variables et lancer le prouveur interactif. De manière automatique au niveau 0, il réussie à nous prouver 3 obligations sur 5 : les initialisations de « aa, « bb », et « cc ».

Nous utilisons ensuite la commande « pr » sur les deux obligations successives et obtenons :

"`Check that the invariant (dd = max({aa,bb,cc})) is established by the initialisation - ref 3.3'" => 2 = max({1,2,3})

En effet, « dd » appartient bien à l'intervalle spécifié mais ne satisfait pas la condition d'être le max de « aa », « bb » et « cc ».

En modifiant la valeur de « dd » à 3 par exemple, il n'y a plus de problème.

IV - MACHINE ABSTRAITE DE RESERVATION

1 – REALISATION DE LA MACHINE

```
MACHINE
   reservation
VARIABLES
   nbPlaceLibre
INVARIANT
   nbPlaceLibre : NATURAL
INITIALISATION
   nbPlaceLibre := 10

OPERATIONS
   reserver =
   PRE
      nbPlaceLibre > 1
   THEN
      nbPlaceLibre := nbPlaceLibre - 1
   END ;

   annuler =
   nbPlaceLibre := nbPlaceLibre + 1
END
```

Nous avons compléter l'exemple fourni par les instructions en gras.

Dans l'opération « reserver », aucune valeur n'est passée en paramètre, il faut donc juste s'assurer qu'il reste encore une place disponible avant d'effectuer la soustraction, bien que cela soit vérifié par l'invariant (en effet, si nbPlaceLibre = 0, la réservation réalisera la substitution et nbPlaceLibre prendra la valeur de -1, or nbPlaceLibre est un entier naturel qui ne peux être négatif).

Dans l'opération « annuler », nous n'avons aucune pré-condition à vérifier puisque NATURAL n'est pas borné.

2 – GENERER LES POS ET LES PROUVER

Obligations de preuves de « reserver » :

"`Check that the invariant (nbPlaceLibre: NATURAL) is preserved by the operation - ref 3.4'" => nbPlaceLibre-1: INTEGER

"`Check that the invariant (nbPlaceLibre: NATURAL) is preserved by the operation - ref 3.4'" => 0<=nbPlaceLibre-1

Obligations de preuves de « annuler » :

"`Check that the invariant (nbPlaceLibre: NATURAL) is preserved by the operation - ref 3.4'" => nbPlaceLibre+1: INTEGER

"`Check that the invariant (nbPlaceLibre: NATURAL) is preserved by the operation - ref 3.4'" => 0<=nbPlaceLibre+1

3 – CONCLUSION

Il n'y a eu aucun problème lors de la vérification des obligations de preuves.

Lors du raffinement de notre machine et avec l'utilisation d'ensembles concrets comme NAT, nous devrons nous assurer lors de l'opération « annuler » que nous ne dépasserons pas le nombre de place originale qui pourra être défini dans une constante.