Master Informatique - M2 ALMA
Faculté des Sciences et des Techniques de Nantes

2010/2011
Méthodes formelles - B (C. Attiogbé)
Christian.Attiogbe@univ-nantes.fr

Etudes de cas : N° 1
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

Analyse et spécification

Exercice : système de gestion d'une banque

On considère un système de gestion d'une banque. On veut écrire partiellement la spécification du système. On dispose de deux ensembles : Numéros des comptes des clients et numéros des clients. Ecrire une machine abstraite spécifiant
– qu'un compte appartient à un utilisateur,
– qu'il existe un utilisateur spécial avec un compte spécial.
Compléter la spécification par quelques opérations dans le système bancaire.

Etude de cas : système de gestion mémoire

On veut spécifier un gestionnaire d'espace (Storage Manager) d'un système d'exploitation. L'espace est découpé en blocs et il y a un maximum de *maxblocks*. Un bloc ne peut être alloué qu'à un seul utilisateur à la fois. Un utilisateur doit libérer un bloc avant son allocation à un autre. Les blocs sont identifiés par des entiers entre 1..*maxblocks*. Il y a des utilisateurs, mais aucune représentation n'en est encore décidée (pour le moment) ; on les manipule simplement avec des identificateurs (user).
Soit la machine incomplète suivante :

```
MACHINE   Storman
SETS      USER
CONSTANTS maxblocks
PROPERTIES
          maxblocks : NAT
&         maxblocks > 0
VARIABLES alloc
INVARIANT
          alloc : 1 .. maxblocks +-> USER
INITIALISATION
          alloc := {}
```

Question : Complétez convenablement la partie `OPERATIONS` de la machine, en considérant la spécification informelle suivante :

Acquérir un bloc libre.

En **entrée** on fournit l'identificateur d'un utilisateur. En **sortie** un *bloc*.

Il doit y avoir au moins un bloc libre. La sortie est le numéro de n'importe quel bloc libre et ce bloc est alors alloué à l'utilisateur en entrée.

Libérer un bloc.

En **entrée** on fournit un utilisateur et un numéro de bloc. Il n'y a **aucune sortie**.

Le numéro de bloc en entrée doit être celui d'un bloc préalablement alloué à l'utilisateur en entrée. Le bloc est alors libéré.

Touver le nombre de blocs libres.

Aucune **entrée**. En **sortie** on fournit un nombre.

La sortie est le nombre de blocs libres.

Libérer tout l'espace alloué à un utilisateur.

En **entrée** on a l'identificateur de l'utilisateur. Il n'y a aucune **sortie**.

Tous les blocs alloués à l'utilisateur sont libérés.

Initialement, tous les blocs sont libres. L'opération pour prendre un bloc a la précondition qu'il y a au moins un bloc libre. On peut la tester par la troisième opération.
Un utilisateur ne libère que des blocs qui lui avait été alloués. Il n'y a pas d'opérations pour tester cette précondition. C'est l'utilisateur qui a la charge de garder la trace des blocs qu'on lui alloue.

Etude de cas : heures SSII

On considère une société de services en ingénierie informatique qui fait travailler ses employés, à l'heure, chez des clients.
La facturation des services ou interventions effectués est basée sur le cumul des heures effectuées pour chaque client. On veut spécifier en B l'application de gestion des interventions.
Chaque client a un identificateur (`idcl`). Lorsqu'un employé effectue une intervention (ou un service) chez un client `idcl`, on enregistre les informations relatives à cette intervention en vue de la facturation. L'enregistrement prend la forme `(date, (idcl, nbheures))` et tous les enregistrements se font dans une variable appelée `services`. `date` est un entier qui représente la date à la quelle l'intervention est faite, `idcl` est l'identifiant du client et `nbheures` est un entier qui représente le nombre d'heures effectuées.

1. Ecrivez la machine abstraite (sans les opérations qu'on écrira dans les questions suivantes) qui gère au moins la variable `services`.
2. Ecrivez une opération qui donne la liste des clients pour lesquels des interventions sont effectuées.

3. Ecrivez une opération qui donne toutes les heures effectuées pour un client donné.

Périodiquement, un traitement est lancé et permet de calculer le nombre total d'heures de services pour les différents clients de la société. On veut pour cela écrire une opération qui détermine pour chaque client, le total d'heures d'intervention. Une variable (`heuresdues`) est mise à jour. Elle contient pour chaque client le nombre total d'heures.

4. Complétez convenablement la machine précédente par la variable `heuresdues`.
   Spécifiez une opération nommée `faireCumul` qui, à partir de `services`, met à jour la variable `heuresdues`, puis réinitialise `services`.

Etude de cas : système de gestion de fichiers en B

Nous allons spécifier un système de fichiers Unix particulier appelé ici SysFic. Le système de fichiers travaille avec les noms des fichiers et ne s'intéresse pas à leur contenu.
SysFic (Système de Fichiers Unix) est une version allégée mais réaliste du système de fichier Unix. C'est un système de fichiers hiérarchique et arborescent qui contient des répertoires (directoires) et des fichiers. Chaque répertoire peut contenir un nombre quelconque de fichiers et de sous-répertoires (ce sont aussi des répertoires).
SysFic effectue diverses opérations sur les fichiers et répertoires : copie, création, suppression, renommage, déplacement, listage. Il connait à tout moment le répertoire courant.
Il y a un répertoire initial particulier nommé *racine* (root).
Les noms des fichiers gérés par SysFic peuvent être absolus ou relatifs. Un nom absolu décrit un nom de fichier en incluant tous les fichiers (répertoires) situés entre la racine et le fichier lui-même.
Un nom relatif est donné par rapport au répertoire courant.

Description informelle du système SysFic

Nous donnons ici la syntaxe et la sémantique de chaque opération ou commande du système de fichiers SysFic.
Les opérations peuvent avoir un ou plusieurs arguments parmis lesquels les noms de fichiers ou répertoires.
Ces noms sont composés de caractères alphabétique et des symboles '-', '+','=','_'.

Nous notons les arguments des opérations entre < et >.

`InitSysFich`
Cette opération permet d'initialiser le système de fichiers. Il est alors vide.
`creerep <repert>`
Créer le répertoire <repert> comme sous-répertoire du répertoire courant. Un message d'erreur est affiché lorsque <repert> est le nom d'un répertoire qui existe déjà.
`suprep <reper>`
détruit (supprime) le répertoire donné en argument s'il existe sous le répertoire courant et est vide, sinon un message d'erreur est affiché.
`chgrep <repert>`
Le répertoire <repert> devient répertoire courant s'il est un sous-répertoire du répertoire courant.
`quelrep`
affiche le nom absolu du répertoire courant.

`creefich <nomfich>`
crée un fichier nomme <nomfich> sur le répertoire courant à condition qu'aucun fichier ou répertoire du répertoire courant ne porte déjà ce nom. Un message d'erreur est affiché le cas échéant.
`suprfich <nomfich>`
détruit (supprime) le fichier dont le nom est donné en argument s'il existe sous le répertoire courant, sinon un message d'erreur est affiché.
`copifich <ancien> <nouveau>`
réalise une copie du fichier nommé <ancien>. La copie est nommée <nouveau>. Le contenu du fichier n'étant pas géré le fichier est vide. Un message d'erreur approprié est affiché si <ancien> n'existe pas ou si <nouveau> existe déjà.
`renomfich <ancien> <nouveau>`
change le nom du fichier <ancien> en <nouveau>. Un message d'erreur approprié est affiché si <ancien> n'existe pas ou si <nouveau> existe déjà.
`listrep`
affiche les noms des répertoires situés sur le répertoire courant.
`listfich`
affiche les noms des fichiers situés sur le répertoire courant.

Spécification

1. Donnez la spécification en B du système SysFic.
2. Utilisez l'AtelierB pour analyser votre spécification.

Exercice : Analyse de spécifications B (relations *vs* fonctions)

Soit à étudier les spécifications à l'aide de *fonction* ou à l'aide de *relation*. Pour ce faire on considère une étude de cas où il s'agit de classes et d'étudiants. L'hypothèse de travail est qu'une classe peut contenir plusieurs étudiants et qu'un étudiant est dans une seule classe.
On utilise *CLASSE* et *ETUDIANT* comme ensembles de base. La relation suivante exprime qu'une classe donnée contient un ou plusieurs étudiants.

$$membre : CLASSE \leftrightarrow ETUDIANT$$

A la place de la relation *membre* on peut aussi bien utiliser la fonction partielle *etudiants* qui exprime qu'une classe donnée contient un ensemble d'étudiants.

$$etudiants : CLASSE \nrightarrow \mathbb{F}\, ETUDIANT$$

1. Explicitez le(s) lien(s) entre *membre* et *etudiants*. Faites des diagrammes de Euler-Wenn au besoin.
2. Pour spécifier des opérations sur les classes et les étudiants membres des classes, on a le choix entre deux modélisations (une avec *membre* et une autre avec *etudiants*). On voudrait justifier le choix final en expérimentant les deux spécifications possibles, puis en tirant les conclusions.
   Ecrivez, en vous servant de *etudiants*, une machine avec les opérations décrites ci-après. Ecrivez ensuite une autre machine avec les mêmes opérations mais qui se sert elle de *membre*.
   creerClasse : pour créer une nouvelle classe étant donné les étudiants qui la constituent.
   *supprimerEtudiant*1 : pour supprimer un étudiant donné d'une classe donnée.
   *supprimerEtudiant*2 : pour supprimer un étudiant donné comme seul paramètre.
3. Comparez les deux modélisations à travers les machines et opérations. Quelles sont les conclusions qui s'imposent ?