Utilisation de l'atelier B

Maîtrise d'Informatique 2001-2002

4 juin 2002

1 Préliminaires

L'atelier B est installé au bâtiment 308 sur la machine `sol`.

1.1 Affichage

À partir d'une autre machine par exemple `pc21`, il faut autoriser la machine `sol` à afficher sur l'écran de `pc21` ce qui se fait en tapant la commande Unix `xhost + sol.ie2.u-psud.fr` dans une fenêtre de shell sur `pc21`. Il faut ensuite se loger sur la machine `sol`; la variable `DISPLAY` doit être positionnée sur `pc21:0.0`, ce qui se vérifie en tapant `echo $DISPLAY` dans la fenêtre de shell de `sol`. Si la variable `DISPLAY` n'est pas correctement positionnée, on peut utiliser la commande `csh` suivante :
`setenv DISPLAY pc21:0.0`.

1.2 Documentation

La documentation est accessible en ligne à l'adresse :
`file:/308/atelierb/AB/bbin/aide/general.html`

1.3 Licence

Il est nécessaire de positionner une variable `LICENCE_PATH` à la valeur `/308/atelierb/AB/bbin/`, ce qui peut se faire par la commande `csh` suivante :
`setenv LICENCE_PATH /308/atelierb/AB/bbin/`
qui peut être insérée dans un fichier de configuration approprié (`.cshrc`).

1.4 Lancement

L'interface graphique de l'atelier B se lance par la commande suivante :
`/308/atelierb/AB/bbin/lanceAB`
Une fenêtre apparaît qui contient une liste des projets présents dans le système (en particulier le projet `LIBRARY` qui contient des librairies utiles). En double-cliquant sur l'un des projets on obtient la liste des machines qui composent le projet, en double-cliquant sur le nom d'une des machines, une fenêtre d'édition s'ouvre et affiche le source de la machine.

Mode non-graphique Il est également possible d'utiliser l'atelier B à l'aide d'une boucle d'interaction, ce qui peut être utile dans le cas d'une utilisation à distance, la commande est alors :
`/308/atelierb/AB/bbin/lanceBB`
La commande `help` permet d'avoir une liste des commandes possibles, en particulier :

> `op` *nomProjet* : ouvre le projet de nom *nomProjet*
> `af` *nomMachine.suf* : ajoute la machine *nomMachine.suf* au projet (le suffixe dépend de la nature de la machine, cf 2.3)
> `t` *nomMachine* : vérifie le typage de la machine *nomMachine*
> `prove` *nomMachine* n : prouve les obligations de preuve de *nomMachine* avec la force n ($n \in \{0, 1, 2, 3\}$)

2 Création d'un projet

2.1 Initialisation

Pour démarrer un développement dans l'atelier B, il faut créer une structure de répertoire dans laquelle seront stockés différents fichiers créés par l'atelier B. Ceci peut se faire par les commandes Unix :

```
mkdir projet
mkdir projet/BDP
mkdir projet/TRAD
mkdir projet/SRC
```

Le répertoire `BDP` contient des fichiers créés par l'atelier B lors de l'analyse des machines, le répertoire `TRAD` contient les fichiers de code générés à partir des machines B.

2.2 Création d'un projet

Après avoir mis en place les répertoires et lancer l'atelier B en mode graphique, on crée un projet en cliquant dans la fenêtre `Attach` puis en donnant un nom de projet et en indiquant les deux répertoires `BDP` et `TRAD`. Le projet doit alors apparaître dans la fenêtre principale.

2.3 Ajout de machines

Il faut alors double-cliquer sur le nom de votre projet. Vous pouvez alors ajouter des composants à votre projet. Un composant apparaît sous la forme d'un fichier *NomMachine*`.mch` ou *NomMachine*`.ref` ou *NomMachine*`.imp`. Chacun de ces fichiers doit définir une machine dont le nom est *NomMachine*. Le suffixe correspond à la nature de la machine, dans le cas de `.mch` il s'agit d'une machine simple, dans le cas `.ref` d'une machine de raffinement et dans le cas de `.imp` d'une machine d'implantation.
Les composants sont édités séparément et ajoutés au projet à l'aide de la commande `Add` du menu associé au bouton `Components`

3 Analyse d'une machine

Lorsqu'une machine est insérée dans un projet, elle doit être vérifiée.

3.1 Vérification de typage

La première analyse concerne la vérification syntaxique et la correction du typage, on l'obtient en cliquant sur la commande `Type-check`. Les erreurs de syntaxe sont détectées et affichées avec un numéro de ligne et de colonne. Lorsque la machine est syntaxiquement correcte, le système procède à la vérification du typage. Chaque identificateur introduit (paramètre de la machine, constante, variables, paramètres des opérations) doit être contraint par un type. Les formules qui permettent de contraindre le type sont essentiellement de la forme $x \in E, X \subseteq E, x = e$. Les paramètres de la machine sont contraints dans la clause **CONSTRAINTS**, les constantes dans la clause **PROPERTIES**, les variables de la machine dans la clause **INVARIANT** et les paramètres d'entrée d'une opération dans la partie PRE de la substitution associée.
Si la machine est mal typée, une fenêtre apparaît, par contre le détail des opérations de typage et les messages d'erreur détaillés apparaissent dans la fenêtre shell dans laquelle a été lancé l'atelier. Il est recommandé de se concentrer sur les premières erreurs détectées.

3.2 Obligations de preuve

Lorsque la machine vérifie les contraintes de typage, cela ne signifie pas qu'elle est correcte. Il faut encore vérifier que les propriétés d'invariant annoncées sont bien satisfaites. L'atelier B engendre des *obligations de preuves* qui sont des formules logiques qui expriment que l'invariant est vérifié après l'initialisation et est préservé lors de l'application des opérations.
La commande `Prove` permet d'engendrer les obligations de preuve et de demander à l'atelier B de prouver automatiquement la correction de ces formules. Le mode `Interactive` permet d'afficher l'ensemble des obligations de preuve. Une fenêtre contient les hypothèses sous laquelle l'obligation doit être démontrée (en particulier un paramètre de la machine en majuscule ou un ensemble abstrait est implicitement considéré comme un ensemble fini non vide d'entiers).
Le mode `Automatic` permet de résoudre certaines obligations, il y a plusieurs niveaux de preuves (0 à 3) qui sont essayés successivement.
Lorsqu'une obligation n'est pas automatiquement prouvée par le système, il est nécessaire de comprendre pourquoi. Si l'obligation ne paraît pas démontrable à partir des hypothèses, il peut s'agir d'une précondition incomplète à une opération; il faudra donc modifier l'opération en conséquent. Si l'obligation semble trivialement déductible des hypothèses, alors il peut être raisonnable de la laisser non-prouvée. Une alternative est de modifier l'écriture de la machine pour rendre les conditions engendrées plus simples à prouver.

4 Utilisation du langage B

L'atelier B nécessite d'utiliser les notations ASCII pour les différentes constructions, celles-ci peuvent se trouver dans l'annexe des notes de cours.

4.1 Notation ASCII pour les opérations

La notation ASCII pour la définition d'une opération ayant deux paramètres d'entrée x et t et deux paramètres de sortie y et z est : `y,z <-- op(x,t) = subst`

4.2 Substitution multiple

Certains exemples donnés dans le cours ne sont pas conformes aux règles de syntaxe de l'atelier B. En particulier une affectation multiple s'écrit impérativement sans parenthèses : `x,y := e,f`

4.3 Somme

Il est possible de sommer un ensemble fini d'expressions. Cela se fait par l'expression : $\mathsf{SIGMA}\ x.(P \mid E)$ x est une variable ou un ensemble de variables séparées par des virgules. P est une formule qui fait intervenir les variables x et qui contient au moins les formule de typage de ses variables. E est une expression qui mentionne x et représente entier. L'ensemble F des x qui vérifient P doit être fini. L'expression $\mathsf{SIGMA}\ x.(P \mid E)$ est un entier qui représente : $\Sigma_{x \in F}\ E$. Si l'ensemble est vide, alors la somme est 0.
On peut par exemple construire la somme des carrés des entiers pairs de 1 à n de la manière suivante :

$$\mathsf{SIGMA}\ x.(x \in 1..n \land x \ \mathtt{mod}\ 2 = 0 \mid x^2)$$

5 Conseils pour la réalisation du projet

5.1 Machines abstraites

Les machines abstraites représentent l'interface des modules et ne doivent comporter que les spécifications des données et des opérations. La substitution qui "implante" l'opération. Il faut donc juste décrire le changement de l'état de la machine (les variables) et la relation entre les entrées et les sorties de l'opération sans chercher à expliquer comment se font les calculs. Les substitutions de séquence, de boucle ou d'affectation locales sont interdites dans les machines abstraites.

5.2 Découpage du projet

Il est recommandé de découper le projet en machines indépendantes.

5.2.1 Représentation d'une base de données

Une machine pourra encapsuler un ensemble de données (le stock, les clients, ...) Une telle machine peut se construire sur le modèle suivant :
- Un ensemble abstrait de "clés" est introduit dans la clause **SETS**
- Une variable représentant l'ensemble des clés présentes dans le système est introduite ainsi que des variables pour chaque champ de la base. Ces champs sont représentés par des fonctions totales de l'ensemble des clés stockées vers le type des champs. On peut en particulier utiliser le type `STRING` des chaînes de caractère.
- Des opérations d'ajout, de retrait et de consultation peuvent être introduits.

5.2.2 Opération agissant sur plusieurs bases

Si une opération doit faire intervenir deux bases (par exemple les pièces ajoutées à une réparation doivent être retirées du stock) alors il peut être judicieux d'avoir deux opérations indépendantes : une opération de retrait dans la machine gérant le stock, une opération d'ajout dans la machine qui gère les réparations puis de construire l'opération qui ajoute dans la réparation en retirant dans le stock dans une machine qui encapsule le service réparation.

5.3 Représentation d'une somme disjointe

Supposons qu'une base B doive contenir des informations provenant de deux bases différentes, l'une indicée sur un ensemble de clés K_1 et l'autre sur l'ensemble de clés K_2.
Une manière de construire la base jointe est d'introduire un nouvel ensemble de clés `key` et d'introduire une fonction totale `tag` qui à chaque clé de `key` associe le type des objets (représenté par un type énuméré

à deux valeurs t_1 et t_2. Ensuite on a deux fonctions partielles l'une, f_1, de `key` dans $\texttt{key}_1$ qui représente les objets de B qui sont dans B_1 et l'autre, f_2 de `key` dans $\texttt{key}_2$ qui représente les objets de B qui sont dans B_2. On peut ajouter dans l'invariant que les objets qui ont une image par f_1 sont ceux dont le `tag` est t_1 et de même pour f_2. En B, cela s'écrit :

SETS
$\quad\texttt{KEYS};\ \texttt{TYPEBASE} = \{t_1, t_2\}$
VARIABLES
$\quad\texttt{key}, \texttt{tag}, f_1, f_2$
INVARIANTS
$\quad\texttt{key} \subseteq \texttt{KEYS}$
$\quad\wedge\ \texttt{tag} : \texttt{key} \longrightarrow \texttt{TYPEBASE}$
$\quad\wedge\ f_1 : \texttt{key} \nrightarrow \texttt{key}_1$
$\quad\wedge\ f_2 : \texttt{key} \nrightarrow \texttt{key}_2$
$\quad\wedge\ \texttt{tag}^{-1}[\{t_1\}] = \texttt{dom}(f_1)$
$\quad\wedge\ \texttt{tag}^{-1}[\{t_2\}] = \texttt{dom}(f_2)$

6 Problèmes systèmes

6.1 Manque de licences

La sortie brutale de l'atelier B, laisse des licences inutilement occupées. Pour les "nettoyer" manuellement il faut :

- Se loger comme utilisateur `atelierb` sur `sol`
- Trouver dans le fichier `/308/atelierb/AB/bbin/licence.sta` les utilisateurs qui bloquent des licences
- Aller dans le répertoire `/308/atelierb/AB/bbin/sun5.6` et exécuter la commande
  `lic_retrieve` "Atelier B" "Principal" user mach mach pid
  Les arguments de `lic_retrieve` peuvent être copiés à partir de la ligne apparaissant dans le fichier de licence.

Il est recommandé d'essayer autant que possible de sortir de l'atelier B "proprement". Si vous pensez ne pas l'avoir fait, vérifiez le fichier `licence.sta` et demandez le nettoyage.

6.2 Composant "occupé"

Il peut arriver qu'un composant ne puisse être traité car le système considère qu'il est en cours de modification. On peut alors supprimer le fichier de suffixe `.lock` qui se trouve dans le répertoire `projet/BDP`.