Modélisation et spécification – Master Informatique
TP 1 : CADP

1 Manuel concis de CADP

CADP (`www.inrialpes.fr/vasy/cadp`) est un environnement de modélisation et vérification qui fournit, parmi d'autres outils, les moyens pour modéliser, spécifier et vérifier en utilisant les systèmes de transitons étiquetés (STE).

Pour utiliser CADP, il faut affecter des variables d'environnement suivantes (on utilise la syntaxe `bash`) :

```
# .bashrc
CADP=/usr/local/cadp
export CADP
PATH=$PATH:/usr/local/bin:$CADP/bin.mac86:$CADP/com
export PATH
```

1.1 Formats de description de STE

CADP offre plusieurs formats de description des STE : AUT et DOT sont des formats textuels, BCG est un format binaire.

Format aut La première ligne du fichier `.aut` spécifie le numéro de l'état initial du STE, le nombre de transitions et le nombre d'états. Les états sont numérotés à partir de 0, sans discontinuité. Après cette ligne sont énumérées toutes les transitions ; le format utilisé est état source, étiquette (chaîne de caractères en **majuscules** ou `"i"` pour l'action interne τ) et état cible.

Par exemple, un tampon à une place "buf.aut" est décrit comme suit en format AUT :

```
des(0,2,2)
(0, "IN", 1)
(1, "OUT", 0)
```

L'outil `bcg_io` permet de transformer ce format vers les autres formats pour sa visualisation ou son analyse. (faire `man bgc_io` pour comprendre son fonctionnement). La visualisation des STE peut être faite :

- en utilisant le format BCG et l'outil `bcg_draw` ou
- en utilisant le format DOT et l'outil GRAPHVIZ.

1.2 Composition des STE et SVL

Le langage de script SVL (Script Verification Language) permet de composer les STE décrits en différents formats avec les opérateurs de composition parallèle, de renommage ou d'abstraction.

Renommage des actions Il est effectué en utilisant la construction `rename`. On doit spécifier la liste des actions à renommer et leur nouveaux noms, ainsi que le STE sur lequel l'opération est faite. Le résultat peut être mis dans un fichier au format souhaité (indiqué par l'extension du fichier). Par exemple :

```
"buf_1.aut" = rename OUT -> A in "buf.aut" ;
```

ou

```
"buf_2.aut" = rename IN -> A in "buf.aut" ;
```

Abstraction ou renommage des actions vers l'action interne τ (`"i"` pour CADP) est faite en utilisant l'opérateur `hide`. Par exemple, la construction ci-dessous permet de abstraire (rendre interne) l'action `A` du STE `buf_2.aut` :

```
"out.aut" = hide A in "buf_2.aut" ;
```

Composition parallèle est un opérateur *binaire* ayant différentes variantes en fonction du nombre des actions à synchroniser :

- Si on veut synchroniser sur toutes les actions (non-internes) des deux STE opérandes, alors on utilise l'opérateur `||`.
- Si aucune synchronisation doit être faite, on utilise l'opérateur `|||`.
- Si on doit synchroniser les deux STE sur les actions d'une liste L, on utilise l'opérateur `|[`L`]|`.

Afin de obtenir le STE résultant de cette composition, il faut utiliser la construction `generation of`.

Par exemple, pour composer deux STE représentant chacun un tampon à une place afin d'obtenir un tampon à deux places, on utilise le script SVL suivant :

```
"buf2.aut" = generation of ("buf_1.aut" |[A]| "buf_2.aut");
```

Ensuite, on peut abstraire (rendre interne) l'action `A` :

```
"buf2i.aut" = hide A in "buf2.aut";
```

La composition parallèle est un opérateur binaire, mais elle peut se transformer en un opérateur n-aire en enchaînant les compositions parallèles. Par exemple, pour que l'action `A` soit effectué en même temps par trois STE, on écrira :

```
("ste1.aut" |[A]| "ste2.aut")  |[A]| "ste3.aut"
```

Toutefois, si on souhaite n'avoir que des synchronisation binaires sur la même action, il faut bien séparer les STE qui sont synchronisés des STE qui le sont pas. Par exemple, la synchronisation entre les travailleurs de l'atelier de Milner et les outils est binaire :

```
("perso1.aut" ||| "perso2.aut")  |[PUTM,GETM]| "marteau.aut"
```

1.3 Minimisation et comparaison de STE

CADP fournit plusieurs outils pour la minimisation et la comparaison des STE par des relations d'équivalence et de pré-ordre. Les relations vues en cours ont la syntaxe suivante :

- `trace` pour équivalence ou inclusion de traces,
- `weaktrace` pour équivalence ou inclusion de traces avec actions internes,
- `strong` pour bisimulation ou simulation forte,
- `observational` pour bisimulation ou simulation faibles.

La minimisation est faite par la construction `reduction` et elle doit spécifier la relation d'équivalence utilisée. Par exemple :

```
"buf2min.aut" = observational reduction of "buf2i.aut";
```

La comparaison est faite de façon similaire par la construction `comparison` qui doit spécifier le type de comparaison (équivalence `==`, inclusion `<=`) et la relation à utiliser. Par exemple :

```
"diag_buf.seq" = strong comparison "buf.aut" == "buf2min.aut";
```

teste l'équivalence des deux STE en utilisant la bisimulation forte ; en cas de non satisfaction de la relation proposée (ici l'égalité), une séquence de contre-exemple est donnée dans le fichier `.seq`.

Exercice 1 : Décrire puis visualiser les STE de base des exercices 1 et 2 faits au TD 1.

Exercice 2 : Décrire puis visualiser toutes les compositions des STE des exercices 1 et 2 faits au TD 1.

Exercice 3 : Montrer, en utilisant CADP, les différentes relation d'équivalence demandées dans l'exercice 1 du TD 2.