

Le Système SMV

Le matériel nécessaire à la réalisation de ce TP est accessible à l'url suivante :

http://www.labri.fr/~ly/enseignement/smv/tp_smv.tgz

Les instructions d'installation sont détaillées dans le fichier `readme.txt`.

1. Introduction

Le système SMV implémente la vérification automatique de propriétés temporelles de modèles finis (cf. <http://www-2.cs.cmu.edu/~modelcheck/smv.html>). Il comporte

- Un langage de modélisation qui permet la description modulaire de systèmes finis, sous forme de machine à états (« modèles de Kripke »).
- Un langage de spécification : CTL (« Computation Tree Logic ») qui permet la spécification de propriétés.
- Un module de vérification automatique de spécifications.

Langage de modélisation.

La modélisation d'un système se fait par la définition d'un ensemble d'états et d'un ensemble de transitions. Nous donnons ici l'exemple de la description d'un compteur à 2 bits, tiré du manuel de référence¹:

```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := value + carry_in mod 2;
DEFINE
  carry_out := value & carry_in;
```

Le système, décrit par le module `main` est formé de deux bits, eux-mêmes décrits par le module `counter_cell`. Chacun de ces bits comporte un état matérialisé par la variable booléenne `value`, déclarée dans la partie « VAR » de la définition du module. Les transitions sont décrites dans la partie `ASSIGN` : l'état initial (d'une instance du module `counter_cell`) est donné par l'expression `init(value) := 0`; qui affecte 0 à

la variable `value`. Les transitions sont décrites par l'expression `next(value) := value + carry_in mod 2` ; ou `carry_in` désigne une valeur extérieure au module déclarée comme paramètre de son instanciation. Enfin la partie « DEFINE » de la description permet de définir des macros à l'usage du développeur.

Ainsi vous pourrez vérifier que l'état initial du système consiste à initialiser chaque bit à 0, et que les transitions consistent à incrémenter le compteur.

Exercice.

Modifiez le code précédent pour obtenir un compteur qui « compte » jusqu'à 15 (en binaire).

Spécifications.

Les spécifications sont déclarées dans le langage CTL qui permet de décrire des propriétés portant sur les états et sur les trajectoires du systèmes (suites infinies d'états). Un exemple de spécification du système précédent est le suivant :

```
SPEC
  AG AF bit2.carry_out
```

L'expression `bit2.carry_out` est une formule atomique portant sur les états du système. Elle exprime le fait que dans l'instance `bit2` du module `counter_cell`, l'expression booléenne `carry_out` est vraie, i.e., égale à 1. Cette formule atomique est vraie pour tout état du système satisfaisant cette condition. D'une façon générale, les formules atomiques sont construites à partir de variables booléennes, d'égalité et des connecteurs classiques (&, |, -, >, <-, !, opérateurs arithmétiques, etc. Cf. Manuel de référence fourni dans la distribution du TP)

Une formule de la forme `AF ϕ` , où ϕ est une formule spécifiant un ensemble d'état, spécifie l'ensemble des états à partir desquels toute trajectoire rencontre un état satisfaisant ϕ . Ainsi, l'expression `AF bit2.carry_out` spécifie l'ensemble des états à partir desquels toute trajectoire rencontre au moins un état pour lequel `bit2.carry_out` est vraie.

Enfin, une formule de la forme `AG ϕ` définit l'ensemble des états à partir desquels toute trajectoire ne comporte que des états satisfaisant ϕ . Ainsi, la spécification ci-dessus exprime le fait qu'à partir de tout état de toute trajectoire du système (commençant dans un état initial), le système accèdera à un état dans lequel `bit2.carry_out` est vraie.

Mise en œuvre de SMV.

A partir du fichier `counter.smv` qui contient la modélisation précédente, lancer la vérification par la commande suivante :

```
> smv -f counter.smv
(ou bien C-c-C-f sous emacs, ce qui est recommandé)
```

Exercice.

Vérifiez la spécification précédente à l'aide du système.

De même, vérifiez que la valeur 8 sera toujours atteinte par votre système.

Le langage CTL comporte d'autres opérateurs :

- `AX ϕ` : ϕ est vraie pour tout état successeur.

¹ cf. <http://www-2.cs.cmu.edu/~modelcheck/smv.html>

Introduction au système SMV

- $A[\varphi \cup \varphi']$: pour toute trajectoire, φ est vraie tant que φ' est fausse, et φ' sera vraie au bout d'un temps fini.
- $EG \varphi$: il existe une trajectoire dont tous les états satisfont φ .
- $EF \varphi$: il existe une trajectoire dont un état satisfait φ .
- $EX \varphi$: φ est vraie pour au moins un successeur.
- $E[\varphi \cup \varphi']$: il existe une trajectoire sur laquelle φ est vraie tant que φ' est fausse, et φ' sera vraie au bout d'un temps fini.

Exercice.

Vérifiez que dans votre compteur, la valeur 0 succède à la valeur 15.

Exercice.

Vérifiez que la valeur 0 apparaît infiniment souvent.

Exercice.

En ajoutant un paramètre au module `counter_cell` pour spécifier la valeur d'initialisation, modifiez votre code de telle sorte que votre compteur commence à partir de la valeur 5 (lisez la documentation du système).

En utilisant la forme $A[\varphi \cup \varphi']$, vérifiez que la valeur 8 apparaît avant la valeur 0.

Exercice (subsidaire).

L'opérateur « weak until » : $\mathbf{j} \ W \ \mathbf{j}'$ exprime le fait que \mathbf{j} est vraie tant que \mathbf{j}' est fausse, et \mathbf{j}' est éventuellement toujours fausse. Cet opérateur n'est pas reconnu par le système SMV.

Montrer que l'on a : $A[\varphi \ W \ \varphi'] == !E[!\varphi' \cup !\varphi]$

2. Modélisation de Feux de Signalisation

On considère un système de feux de signalisation pour réguler le trafic routier à l'intersection de deux voies à sens unique.

Le système est formé de deux sous systèmes, chacun d'eux étant une instance d'un module SMV modélisant le comportement d'un feu. Un feu comporte trois champs booléens modélisant les lampes rouge, orange et verte.

On considère également un bouton d'appel qui permet d'envoyer une requête à un feu pour qu'il passe au rouge. Ce bouton d'appel sera modélisé par un module SMV dont les transitions sont aléatoires, i.e. non déterministes.

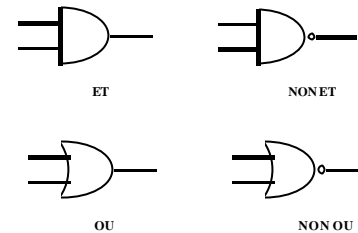
Vous développerez ces modèles, et vous développerez également leurs spécifications :

- L'activation du bouton d'appel est suivie après un temps fini du passage au rouge du feu concerné.
- Les deux ne sont jamais au vert en même temps.

Dans un second temps, vous ferez évoluer votre modèle en incluant une temporisation, basée sur les compteurs vus précédemment.

3. Modélisation de circuits logiques

On considère des circuits formés de portes logiques connectées entre elles. De façon précise, on considèrera les portes logiques classiques : ET, OU, NON-ET, NON-OU :



On donne ici la modélisation d'une porte ET ainsi que sa spécification :

```
MODULE and-gate (entry1, entry2, init-state)
VAR State : boolean;
ASSIGN
  init(State) := init-state;
  next(State) := entry1.State & entry2.State;
SPEC
  AG ((entry1.State & entry2.State) <-> (AX State))
```

Expliquez la spécification, notamment l'utilisation de l'opérateur AX.

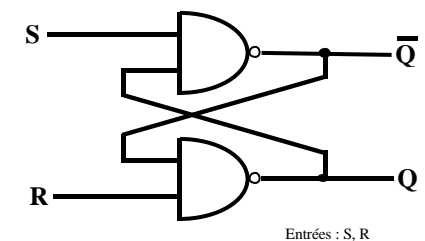
Une entrée de circuit sera modélisé par un état prenant les valeurs 0 et 1 de façon non-déterministe :

```
MODULE entry (value)
VAR State : boolean;
ASSIGN
  init(State) := value;
  next(State) := { 0, 1 };
```

Utilisez ces modules pour construire la modélisation et la spécification d'un système comportant 3 portes ET et 4 entrées, réalisant le ET logique des 4 entrées.

Attention au délai de calcul.

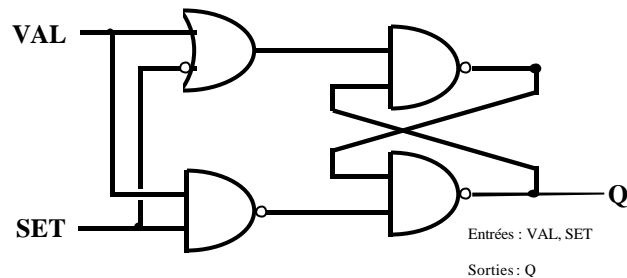
Le schéma ci-contre représente une bascule. Les entrées S et R sont par défaut à 1. Le circuit s'utilise en envoyant des impulsions sur l'une des 2 entrées, i.e., en faisant passer



Introduction au système SMV

l'une des deux entrées à l'état 0, puis en la ramenant à l'état 1. Une impulsion sur l'entrée S fait passer la sortie Q à l'état 0 et la sortie \bar{Q} à 1. Les sorties restent alors dans cet état, quelle que soit la valeur de S, tant qu'aucune impulsion n'est envoyée sur R. Réciproquement une impulsion sur l'entrée R fait passer la sortie \bar{Q} à l'état 0 et la sortie Q à 1. Le circuit « mémorise » les impulsions envoyées sur les entrées.

Un autre exemple est fourni par le schéma ci-contre qui représente un circuit implémentant une *mémoire élémentaire*. Par défaut, l'entrée SET est à 0. Lorsque SET passe à l'état 1, la sortie Q « mémorise » la valeur de l'entrée VAL. C'est-à-dire que lorsque SET repasse à l'état 0, la sortie Q passe dans un état stable égal à VAL. Q reste dans cet état quelque soit les variation de VAL, tant que SET reste à l'état 0.



En utilisant la méthode précédente, modélisez et spécifiez ces deux circuits.