DESS - Semaine du 28/10/2002

# 1 Rappels sur Spin

Le Schema du modele en Spin est

```
#define trans(garde,action,etat) d_step{garde -> action} goto etat

/* Variables globales */
int t[10];
bool test;

proctype A(int n ...){
...
}
proctype B(char b,...){
...
}

init        /* pid 0 */
{
    atomic {
    .... ;      /* par exemple les initialisations des variables globales */
    .... ;
    run A(3);   /* pid 1 */
    run A(2);   /* pid 2 */
    run B('a')  /* pid 3 */
    }
}
```

Dans chaque proctype on represente l'automate du processus. Ses etats sont declares par des labels. Les transitions sont énumérés comme des choix non déterministes, (`if`) sous la forme (*garde*, *action*, *nouveletat*), exprimees grace au `#define trans(,,)`.

```
etat1:
        if
        :: trans(x!=y,x=y,etat2)
        :: trans(k<3,skip,etat3)
        fi;

etat2:
...
```

On lance Spin avec la commande `xspin`. Vous pouvez controler votre syntaxe dans l'onglet `run - run syntax check`.

# 2 Exercices

**<u>Exercice</u>** 1 : On reprend le TD precedent.

```
Boolean array b[0..1] initialized to true;
Integer k;

Process(i) with i either 0 or 1
begin
1 : the core program
    b[i] := false;
2 : if (k!=i) then begin
3 : if (not b[1-i]) then goto 3 else k:=i;
    goto 2 end;
    Critical Section
    b[i] := true; goto 1;
end
```

Avec pour atomicite : les affectations, les evaluations de conditions, les goto.

1. Modeliser le protocole sous SPIN.
2. Visualisez l'automate d'un process (onglet `Run - View Spin automaton` )
3. Simuler le protocole (onglet `Run - Set simulation Parameter - Start`), en tirant les transitions au hasard.
4. Idem, mais a la main, peut on atteindre le "livelock", c.a.d une execution dans laquelle les deux processus restent indefiniment en attente sans qu'aucun ne rentre en section critique.
5. Effectuer l'execution qui amene les 2 process en SC. Attention, les identites donnees aux process ne correspondent pas toujours.
6. Simplifier l'automate, en fusionnant les transitions des goto avec les transitions qui les precedent. Normalement cet automate ne verifie toujours pas la propriete de Section Critique, simuler a nouveau un de ces comportements.
7. On va chercher a le retrouver automatiquement. Pour cela, aller sur l'onglet `Run - Set verification parameter - Verify an LTL propperty`. Dans la nouvelle fenetre, inserer dans la fenetre des definitions :

   ```
   #define sc0 (user[1]@section_critique)
   #define sc1 (user[2]@section_critique)
   ```

   Puis ecrivez la formule

   ```
   ! <>  (sc0  && sc1)
   ```

   qui signifie "il n'est pas possible d'avoir dans le futur et au meme moment (sc0 et sc1)". Cliquez sur `generate` et observez l'automate. Dans le futur vous cliquerez directement sur `Run Verification`. Constatez le contre exemple sur la simulation, est il plus cours que le votre ? Cherchez dans les options du verificateur l'option qui trouve le plus petit contre exemple.

   Vous avez un generateur d'automate a partir de formule a l'adresse :

   `http://verif.liafa.jussieu.fr/cgi-bin/binbin/ltl2ba.shtml`

# L'algorithme auto-stabilisateur de Dijkstra (74)

Il s'agit d'un algorithme d'exclusion mutuelle pour des systèmes distribués disposés en anneaux, sur lequel on reviendra a l'occasion d'autres TP. Les $n$ processus, $P_0 \dots P_{n-1}$ peuvent chacun demander des informations à leur voisin de droite ($P_{i-1 \ mod \ n}$) immédiat. Pour chaque $P_i$, le privilège d'être en section critique est matérialisé par une configuration de ses variables propres et de celles de son voisin de gauche.

Chaque processus possède une variable *drapeau* qu'il peut lire et ecrire, et il peut également lire le contenu de *drapeau* pour le processus à sa droite.

Le protocole proposé n'est pas symétrique, et $P_0$ y joue un role particulier.

| Protocole pour $P_0$ : |
|---|
| attendre $drapeau == P_{n-1}.drapeau$<br>< Section critique><br>drapeau = drapeau +1 mod k |

| Protocole pour $P_i, i \neq 0$ : |
|---|
| attendre $drapeau \neq P_{i-1}.drapeau$<br>< Section Critique ><br>$drapeau = P_{i-1}.drapeau$ |

Initialement chaque *drapeau* est mis à 0 et $k > n$.

1. Réaliser un programme Spin qui implémente ce protocole.
2. Après simulation, on peut observer quelques particularités de ce protocole. Donner en langue naturelle un ensemble de specifications significatives.
3. Essayer de faire croitre la taille de l'anneau pour trouver les limites de l'outil.

4. Cet algorithme est dit autostabilisant car si on n'impose plus que chaque drapeau soit mis à zéro initialement, on parvient au bout d'un nombre fini de transitions dans un état où de toutes facons un seul processus possède le privilège d'être en section critique. Cette propriété permet d'envisager par exemple un problème électrique temporaire, et une correction automatique. Pouvez vous integrer ce defaut de facon realiste dans votre modele ? Observer la nouvelle limite de N.