Modélisation des processus communicants - LOTOS/CADP
Exemple d'étude : Contôle d'accès - avril 2007

1 Exercice - Contrôle d'accès à une salle

Soit un système composé d'un utilisateur et d'un gestionnaire de cartes d'accès.
Dans ce système l'utilisateur interagit avec le gestionnaire de façon à obtenir ou non l'accès à une salle.
C'est l'utilisateur qui demande l'accès en glissant sa carte dans le lecteur de carte associé au gestionnaire.
Le gestionnaire demande le code d'accès de l'utilisateur puis, soit il autorise l'accès soit il le refuse.
Lorsque l'utilisateur a obtenu l'autorisation d'accès, il va occuper la salle pendant un moment puis en ressort par une porte de sortie en appuyant simplement sur un bouton poussoir.
Lorsque l'utilisateur n'a pas obtenu l'autorisation d'accès, il peut soit recommencer la demande d'accès soit abandonner.
On veut spécifier ce système en LOTOS et l'analyser formellement avec CADP.

2 Etude avec Lotos et CADP

2.1 Spécification LOTOS

```
(*------------------------------
  Protocole d'acces a une salle
    CA, fevrier 2004 - TD/TP LOTOS
 ------------------------------*)
specification sas [lca,lco,rca,access,refus,danspiece,bouton] : noexit
behaviour

(* on decrit ici la composition parallele qui specifie l'interaction
   entre un usager et le gestionnaire d'acces *)

     user[lca,lco,rca,access,refus,danspiece,bouton]
   |[lca,lco,access,refus]|
     cardmgr[lca, lco, access, refus]

(* les actions abstraites sont:
  lca   : lecture  carte
  lco   : lecture code
  rca   : retrait carte
  acces : acces autorise
  refus : acces refuse
  danspiece : user dans piece
  bouton    : user sort de la piece par un bouton
*)

where
  (* processus USER : demande a acceder *)
  process user [lca, lco, rca, access, refus, danspiece, bouton] : exit :=
    lca; lco; rca;
     (access ; danspiece; bouton;
            (i; exit   (* il s'en va *)
```

```
             []        (* ou il revient *)
             user[lca, lco, rca, access, refus, danspiece, bouton]
            )
      []
      refus; user[lca, lco, rca, access, refus, danspiece, bouton]
      []
      refus; exit
     )
  endproc

 (* processus CARGMGR : gestionnaire de carte
        lit carte, lit code, donne acces ou refuse
 *)
  process cardmgr [lca, lco, access, refus] : noexit :=
    lca;lco;
    ( access;cardmgr[lca, lco, access, refus]
     []
      refus;cardmgr[lca, lco, access, refus]
    )
  endproc
endspec
\end{document}verbatim}
}
\vspace{-0.3cm}
\subsection{Compilation avec \textsc{caesar}}
On a ci-dessous un exemple de la trace fournie par caesar pendant la
compilation d'un fichier .lotos\\
  Attention, le prompt de la machine est \texttt{commande\$}\\
{\small
\begin{verbatim}
commande$ caesar sas.lotos
-- caesar 7.0 -- Hubert Garavel (INRIA Rhone-Alpes) --

caesar : analyse syntaxique de ``sas''
caesar : analyse semantique de ``sas''
caesar :    - liaison des processus
caesar :    - liaison des portes
caesar :    - liaison des types
caesar :    - calcul des signatures
caesar :    - liaison des sortes
caesar :    - liaison des variables
caesar :    - liaison des operations
caesar :    - calcul des fonctionnalites
caesar : restriction de ``sas''
caesar : expansion de ``sas''
caesar : examen des types de ``sas''
caesar : generation de ``sas''
#125 avertissement pendant la generation :
     detection de rendez-vous impossibles
caesar : optimisation de ``sas''
caesar : simulation de ``sas''
caesar :    - production du simulateur
caesar :    - compilation du simulateur
caesar :    - execution du simulateur
```

```
caesar : (pour plus d'informations, voir le fichier ``.err'')

Dans la specification SAS [6]

   - un blocage existe pour le rendez-vous :
        exit
     synchronise par l'operateur parallele ``|[exit, LCA, LCO, ACCESS, REFUS]|''
     cette combinaison porte/offre figure dans le comportement a gauche:
        USER [exit, LCA, LCO, RCA, ACCESS, REFUS, DANSPIECE, BOUTON]
     mais pas dans le comportement a droite :
        CARDMGR [exit, LCA, LCO, ACCESS, REFUS]

   - un blocage existe pour le rendez-vous :
        exit
     synchronise par l'operateur parallele ``|[exit, LCA, LCO, ACCESS, REFUS]|''
     cette combinaison porte/offre figure dans le comportement a gauche:
        USER [exit, LCA, LCO, RCA, ACCESS, REFUS, DANSPIECE, BOUTON]
     mais pas dans le comportement a droite :
        CARDMGR [exit, LCA, LCO, ACCESS, REFUS]
caesar :    - impression du graphe de ``sas'' pour ``bcg''
commande$
```

2.2 Impression du graphe de comportement

```
commande$ bcg_draw sas.bcg
```

ou bien

```
commande$ bcg_edit sas.bcg
```

pour avoir la possibilité de repositionner les composants du graphe(ce qui est fait pour la figure 1).

2.3 Impression des informations sur le graphe

```
commande$ bcg_info sas.bcg
./sas.bcg:
created by caesar
        9 states
        11 transitions
        8 labels
        initial state: 0
        list of deadlock state(s): 4
        branching factor: average = 1.22, minimal = 0, maximal = 3
        1 transition(s) with a hidden label ("i")
        non-deterministic behavior for:
           label "REFUS" at state(s): 3
commande$
```

2.4 Verification des propriétés

Détection de Deadlock

```
aldebaran -dead sas.bcg
list of deadlock state(s):
4
commande$
```

L'état 4 est un état de *deadlock*. Il s'agit effectivement dans notre cas d'étude d'un état à partir duquel on ne peut plus rien faire ; l'interaction est terminée.

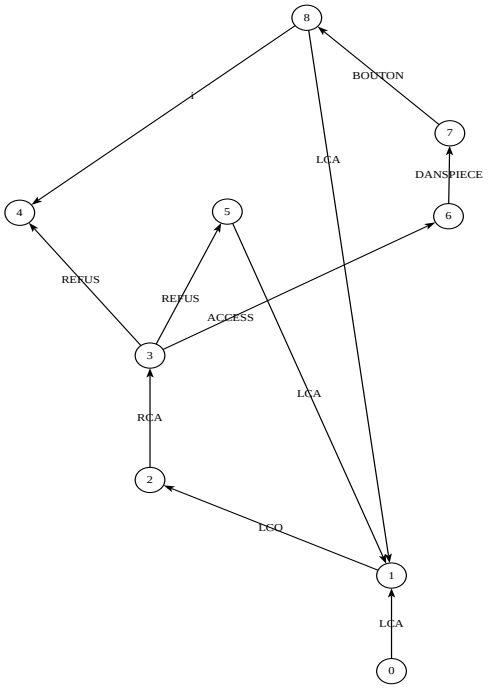

FIG. 1 – Graphe de comportement du système

Vivacité

Prenons par exemple propriété :

on ne peut rentrer dans la pièce si on n'a pas donné son code

. La propriété s'écrit

```
"[(not LCO.*) . DANSPIECE.*"
```

elle est enregistrée dans un fichier `prop1.xtl`. Ensuite on effectue la commande :

```
xtl -verbose -french  sas.bcg prop1.xtl
```

La trace de l'exécution est la suivante :

```
commande$ xtl -french -verbose prop1.xtl sas.bcg
xtl : expansion de ‘‘prop1’’
xtl : analyse syntaxique de ‘‘prop1’’
xtl : analyse semantique de ‘‘prop1’’
xtl :    - reification
xtl :    - liaison des types
xtl :    - liaison des variables
xtl :    - liaison des fonctions
xtl :    - typage des expressions
xtl : traduction en C de ‘‘prop1’’
xtl : creation de la bibliotheque dynamique de ‘‘sas’’
xtl : compilation C de ‘‘prop1’’
xtl : execution de ‘‘prop1’’
commande$
```