Genie Logiciel Avancé : langage LOTOS et Model checking

Bizot Louis
Blain Hugo

16 Octobre 2021

Table des matières

1 Introduction

Au cours de ces séances de TP nous avons pu mettre en pratique plusieurs notions vues en cours en utilisant l'outil LotoStem. Nous avons pu décrire différentes applications et effectuer des vérifications sur ces modèles grâce au langage Basic LOTOS.

2 Présentation de LotoStem

LotoStem est un outil permettant la description des applications sous LOTOS, ainsi que leur vérification formelle en utilisant la technique du model checking. LotoStem permet également la génération des systèmes de transitions étiquetées selon deux sémantiques : la sémantique d'entrelacement et la sémantique de maximalité. Une représentation graphique des systèmes de transitions étiquetées est possible grâce à son éditeur graphique.

3 Exemple

Dans LotoStem, une application est décrite sous forme d'un système comme suit :

```
1  System exemple[a,b]:=
2    exp1[a,b]>>exp2[b,a]
3  where
4    process exp1[x,y]:=
5      x;y;exit
6    endproc
7    process exp2[x,y]:=
8      x;exit|||y;exit
9    endproc
10 endsys
11
```

Figure 1 – Exemple de spécification LOTOS

On peut ensuite générer et afficher une représentation graphique des systèmes de transitions étiquetées obtenus selon la sémantique d'entrelacement ou celle de maximalité.

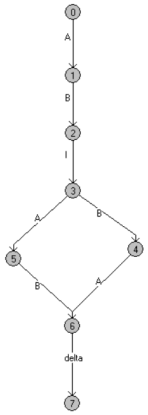


FIGURE 2 – Représentation graphique avec la sémantique d'entrelacement.

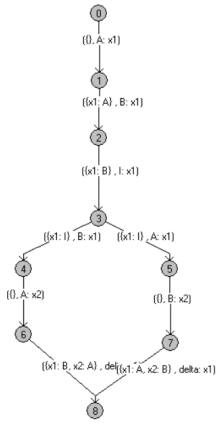


FIGURE 3 – Représentation graphique avec la sémantique de maximalité.

4 Exercice n°1 : Prise en main de l'outil

4.1 Question 1

Soit deux processus P1 et P2 dont les spécifications LOTOS sont les suivantes :

```
System Question_1 [a,b,c,d] :=
  P1[a,b,c,d]
  where
    process P1[x,y,z,t] :=
       z;t;exit [] x;y;P1[x,y,z,t]
    endproc
endsys
```

FIGURE 4 – Spécification LOTOS de P1.

```
System P2[a,b,c,d] :=
  a;b;P[c,d]
where
  process P[u,v] :=
    u;v;P[u,v]
  endproc
endsys
```

FIGURE 5 – Spécification LOTOS de P2.

4.2 Question 2

Dans LotoStem, à partir de ces deux spécifications LOTOS, on peut générer les représentation graphique des systèmes de transitions étiquetées selon les sémantiques d'entrelacement ou de maximalité.

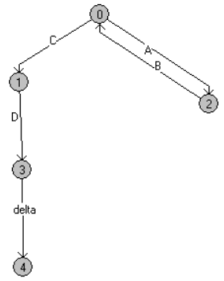


FIGURE 6 – Représentation graphique du système de transitions étiquetées de P1 selon la sémantique d'entrelacement.

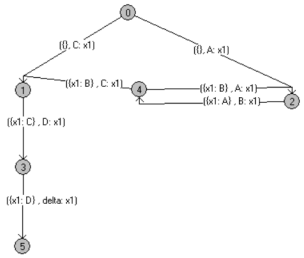


FIGURE 7 – Représentation graphique du système de transitions étiquetées de P1 selon la sémantique de maximalité.

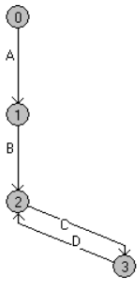


FIGURE 8 – Représentation graphique du système de transitions étiquetées de P2 selon la sémantique d'entrelacement.

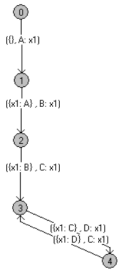


FIGURE 9 – Représentation graphique du système de transitions étiquetées de P2 selon la sémantique de maximalité.

Pour chacun des deux processus, on remarque des différences entre les systèmes de transitions étiquetées obtenus selon la sémantique d'entrelacement et ceux obtenus selon la sémantique de maximalité. Dans un premier temps, on peut voir que les graphiques obtenus selon la sémantique d'entrelacement sont les mêmes que ceux dans l'énoncé. Par contre, avec la sémantique de maximalité, les résultats sont différents. En effet, avec la sémantiques de maximalité, on garde un "historique" des étapes passées. Ceci est bien illustré dans la figure 7, il y a cette fois deux manières d'arriver à l'état '1'. Ces deux chemins marquent bien la distinction entre le fait le faire 'C' sans rien n'avoir fait avant (départ de l'état '0'), ou bien de faire C en ayant déjà fait 'A' et 'B' (départ de l'état '4').

4.3 Question 3

Maintenant si on met les deux processus P1 et P2 en parallèle et synchronisés sur 'B', on obtient la spécification LOTOS suivante.

```
1  system Question3 [a,b,c,d]:=
2        P1[a,b,c,d] |[b]| P2[a,b,c,d]
3        where
4              process P1[x,y,z,t]:=
5                z;t;exit[]x;y;p1[x,y,z,t]
6              endproc
7              process P2[x,y,z,t] :=
8                x;y;p[z,t]
9                where
10                 process p[u,v]:=
11                   u;v;p[u,v]
12                 endproc
13             endproc
14 endsys
```

FIGURE 10 – Spécification LOTOS de P1 $\otimes_{\{b\}} P2$

4.4 Question 4

Grâce à l'éditeur graphique on peut facilement obtenir la représentation graphique du système de transitions étiquetées selon la sémantique de maximalité.

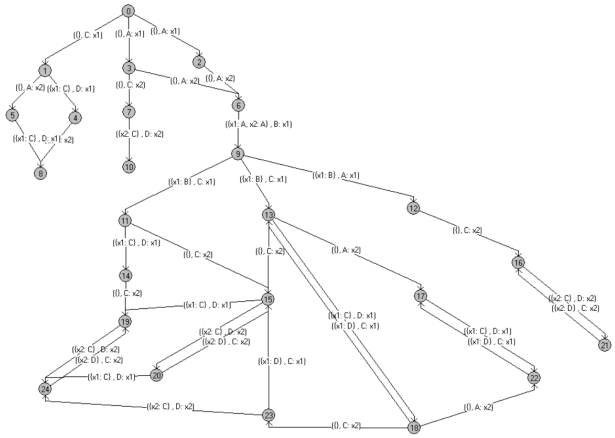


FIGURE 11 – Système de transitions étiquetées selon la sémantique de maximalité de P1 $\otimes_{\{b\}} P2$

4.5 Question 5

A partir des spécification des applications sous LOTOS, l'outil LotoStem permet de lancer des vérifications formelles. Pour cette question, nous devions vérifier quelques propriétés en donnant la liste des états qui les infirmaient ou les affirmaient.

1. "Dans chaque état du système, il est possible d'exécuter l'action a ou l'action b"
   Formule : A G(a or b)
   États où la formule est vérifiée : 5, 7, 8, 10, 12, 16, 17, 21 et 22
   États où la formule n'est pas vérifiée : 0, 1, 2, 3, 4, 6, 9, 11, 13, 14, 15, 18, 19, 20, 23 et 24
   La propriété n'est donc pas vérifiée.

2. "L'exécution de l'action c est immédiatement suivie par l'exécution de l'action d"
   Formule : A G(c =>A X d)
   États où la formule est vérifiée : 4, 5, 7, 8, 10, 12, 14, 16, 17, 19, 21, 22 et 24
   États où la formule n'est pas vérifiée : 0, 1, 2, 3, 6, 9, 11, 13, 15, 18, 20 et 23
   La propriété n'est donc pas vérifiée.

3. "Après une exécution de l'action c, il est possible d'observer une exécution de l'action a ou une exécution de l'action b"
   Formule : A G (c =>E F(a or b))
   États où la formule est vérifiée :
   États où la formule n'est pas vérifiée : 1, 4, 5, 7, 8, 10, 12, 16, 17, 21 et 22
   Il n'y a pas d'états ou la propriété est verifiée
   La propriété n'est donc pas vérifiée.

4. "Il existe un état dans lequel les deux actions a et a sont en cours d'exécution"
   Formule : E F (a and (E X (E [not b U A X a])))
   Traduction : "il existe un chemin où on emprunte a et il existe plus loin un a que l'on emprunte sans passer par b avant"
   États où la formule est vérifiée : 0, 2, 3, 6, 9, 12, 13, 16, 17, 18, 21 et 22
   États où la formule n'est pas vérifiée : 1, 4, 5, 7, 8, 10, 11, 14, 15, 19, 20, 23 et 24
   La propriété est donc vérifiée car il existe des états ou la propriété est vérifiée.

4.6 Question 6

On définie la spécification LOTOS pour la composition parallèle P1 $\otimes_{\{a,b\}} P2$.

```
1  system Question6 [a,b,c,d]:=
2        P1[a,b,c,d] |[a,b]| P2[a,b,c,d]
3        where
4             process P1[x,y,z,t]:=
5               z;t;exit[]x;y;p1[x,y,z,t]
6             endproc
7             process P2[x,y,z,t] :=
8               x;y;p[z,t]
9               where
10                process p[u,v]:=
11                  u;v;p[u,v]
12                endproc
13            endproc
14 endsys
```

FIGURE 12 – Spécification LOTOS pour la composition parallèle P1 $\otimes_{\{a,b\}} P2$.

4.7 Question 7

Grâce à la spécification LOTOS de la question précédente, on peut générer et afficher le système de transitions étiquetées maximal pour P1 $\otimes_{\{a,b\}} P2$.

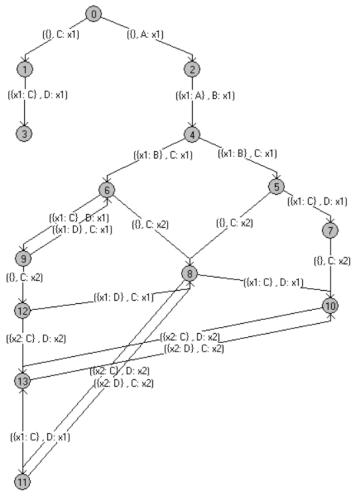


FIGURE 13 – Système de transitions étiquetées maximal pour P1 $\otimes_{\{a,b\}} P2$

5 Exercice 2 : Distributeur de boissons

Dans cette partie, nous allons refaire un exercice vu en cours en allant plus loin grâce à sa mise en application sur LotoStem. L'exercice consiste en la mise en place d'un schéma retraçant une demande de boissons dans une machine distribuant soit des cafés, soit du thé.

5.1 Question 1

Tout d'abord, avant de faire la spécification Lotos, il est utile de faire un schéma récapitulatif des possibilités.
Il est à noter qu'on se place du coté de la machine et non du côté de l'utilisateur qui est un cas qui sera abordé par la suite .

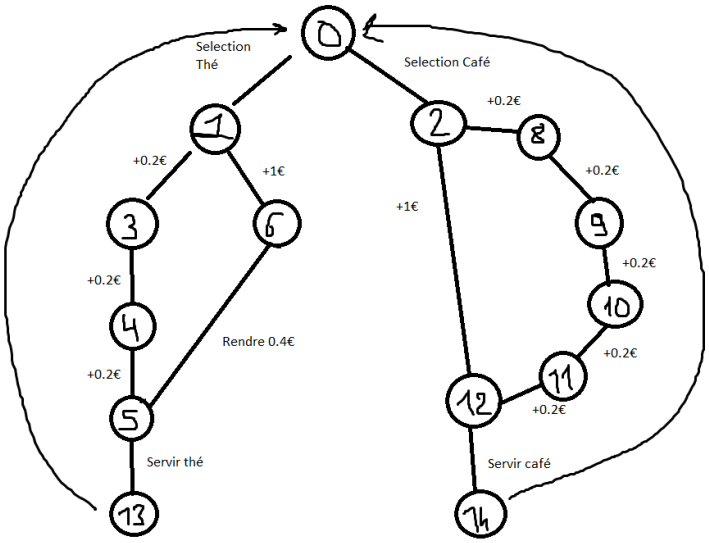


FIGURE 14 – Schéma des cas d'utilisation de la machine (côté machine)

Si on traduit ce schéma en Lotos, on obtient le code suivant :

```
system Question1[a,b,c,d,e]:=
     P[a,b,c,d,e]
          where
               process P[k,l,m,n,o]:=
                    demanderThe[k,l,m,n,o] [] demanderCafe[k,l,m,n,o]
                    where
                         process demanderThe[v,w,x,y,z]:=
                              v;v;v;y;P[v,w,x,y,z][]w;z;y;P[v,w,x,y,z]
                         endproc
                         process demanderCafe[v,w,x,y,z] :=
                              v;v;v;v;v;x;P[v,w,x,y,z][]w;x;P[v,w,x,y,z]
                         endproc
               endproc
endsys

(
a / k / v ->  Insérer 0.2€
b / l / w ->  Insérer 1€
c / m / x ->  Servir café
d / n / y ->  Servir thé
e / o / z ->  Rendre 0.4€
)
```

FIGURE 15 – code Lotos de la spécification du distributeur de boissons

Dans ce code, on crée deux processus.
Un pour le cas d'utilisation correspondant à la commande d'un thé et l'autre pour la commande d'un café. Puis, on parallélise les deux cas d'utilisation afin de retrouver le précédent schéma.

On remarque qu'une fois une boisson commandée, on peut recommencer l'opération car ce schéma décrit l'utilisation du côté de la machine. Une fois une boisson commandée, la machine est à nouveau opérationnelle. Dans le code, cela se manifeste par la non-présence d'un "exit" à la fin de chacun des processus.

Le graphe résultant de ce code est le suivant :

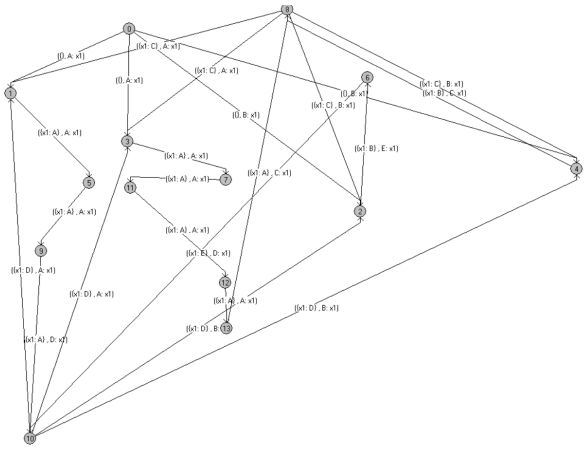


FIGURE 16 – Schéma généré par le code (spécification de la machine à boissons

5.2 Question 2

Dans cette partie, on cherche à reproduire le cas d'utilisation d'un utilisateur qui cherche à prendre uniquement un café. Ce partie est plus simple car on considère uniquement le cas d'utilisation de la prise d'un café.
Voici le code associé à cette question :

```
system Question2[a,b,c]:=
      P[a,b,c]
            where
                  process P[k,l,m]:=
                        k;k;k;k;k;m;exit[]l;m;exit
                  endproc
endsys
```

FIGURE 17 – code Lotos de la spécification de la prise d'un café par l'utilisateur

Il n'y a que deux cas possibles : soit l'utilisateur met une série de 5 pièces de 20 centimes dans la machine jusqu'à atteindre 1 euro puis sélectionne un café, soit il rentre une unique pièce de 1 euro et sélectionne son café.
Un point important à noter est que ces deux cas se terminent par un exit. Cela signifie qu'on ne peut pas répéter l'opération une seconde fois. En effet, on considère que si un utilisateur prend un café, il devra d'abord le boire puis revenir plus tard à la machine à café afin d'en prendre un nouveau.

5.3 Question 3

Dans cette question, nous allons mettre en parallèle les deux premiers codes vus dans les questions 1 et 2. Le but est de donner une spécification qui reprend les cas d'utilisation de la machine à boisson ET d'un utilisateur qui souhaite commander un café.

Nous allons simplement reprendre les deux codes précédents tout en ajoutant des états sur lesquels les processus devront se synchroniser.
Voici le code associé :

```
system Question3[a,b,c,d,e]:=
     P[a,b,c,d,e]||[a,b,c,d]|Pl[a,b,c]
          where
               process Pl[k,l,m]:=
                    k;k;k;k;k;m;exit[]l;m;exit
               endproc

               process P[k,l,m,n,o]:=
                    demanderCafe[k,l,m,n,o] [] demanderThe[k,l,m,n,o]
                    where
                         process demanderThe[v,w,x,y,z]:=
                              v;v;v;y;P[v,w,x,y,z][]w;z;y;P[v,w,x,y,z]
                         endproc
                         process demanderCafe[v,w,x,y,z] :=
                              v;v;v;v;v;x;P[v,w,x,y,z][]w;x;P[v,w,x,y,z]
                         endproc
               endproc
endsys

(
a / k / v ->  Insérer 0.2€
b / l / w ->  Insérer 1€
c / m / x ->  Servir café
d / n / y ->  Servir thé
e / o / z ->  Rendre 0.4€
```

FIGURE 18 – Spécification Lotos de la parallélisation des question 1 et 2

Il est à noter que LotoStem à tendance à toujours choisir le premier ”chemin” proposé dans le code.
Par exemple, si on créer un processus qui parallèlise le processus demanderCafé et demanderThé et que le processus demanderCafé est proposé en premier, LotoStem affichera uniquement ce chemin car il sera le premier qui aura été choisi.
Pour remédier à ce problème, on peut créer deux états qui permettront de ”lancer” chacun des chemins dans les deux directions afin que LotoStem puisse par la suite afficher le second.

Afin que le code génère le bon affichage, il est nécessaire de synchroniser les processus sur chacun des états correspondant à ”mettre une pièce de 20 centimes ” (car il faut mettre toutes les pièces de 20 centimes nécessaires avant de commander). Dans la même logique, il est nécessaire de synchroniser sur ”mettre une pièce de 1 euro”.
De plus, il faut également appliquer la synchronisation sur ”servir café ” et ”servir thé” car la machine ne doit pas recommencer à servir le client tant que celui-ci n'a pas pris sa commande. La même logique d'applique à l'état ”rendre monnaie”.

5.4 Question 4

Dans cette partie, nous avons essayer de réaliser les contraintes demandées, sans pour autant être sur de leur résultat. Nous pensons néanmoins que montrer des pistes de réflexion est important.

Vérifier les propriétés suivantes :
a. Si un client demande un café, il finira par l'avoir :

A G (c =>E F(m and x))

En se référant aux indications en commentaire notés à la fin de notre code (voir image ci-dessous), le chemin correspondant au service d'un café est le chemin c puis m puis x. On écrit donc que si on rencontre un c, alors il y aura plus loin un m et un x.

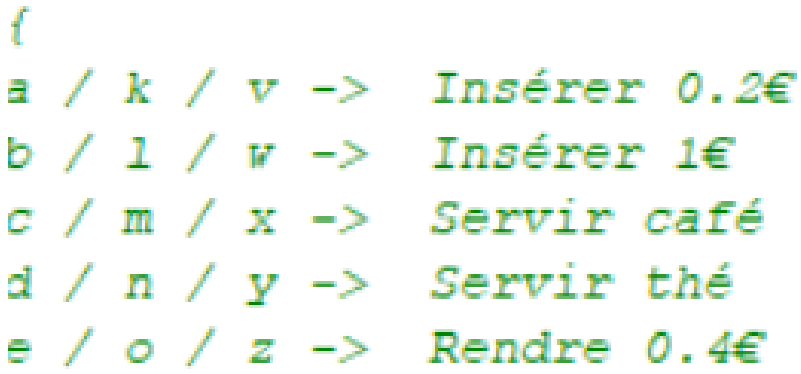


FIGURE 19 – Commentaire dans notre code indiquant les ”chemins” menant aux cas d'utilisations finaux

Malgré de nombreux essais, nous n'avons pas pu réaliser la suite des contraintes, notamment par manque de temps.

6 Exercice 3 : Dîner des philosophes

Dans cet exercice nous travaillons sur un problème vu en cours, le dîner des philosophes. Pour ne pas être répétitif, nous ne rappellerons pas l'énoncé de ce problème dans ce compte rendu.

6.1 Question 1

On commence par le cas avec deux philosophes.

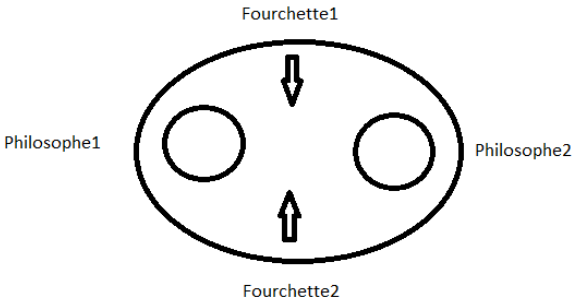
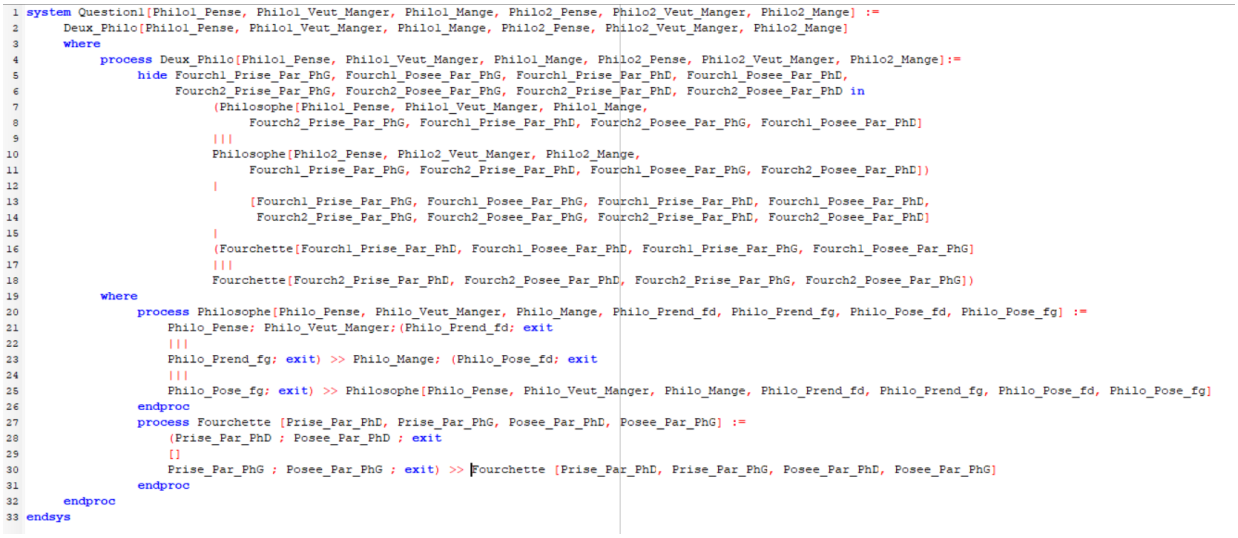


FIGURE 20 – Dîner des philosophes avec 2 philosophes.



```
system Question1[Philo1_Pense, Philo1_Veut_Manger, Philo1_Mange, Philo2_Pense, Philo2_Veut_Manger, Philo2_Mange] :=
    Deux_Philo[Philo1_Pense, Philo1_Veut_Manger, Philo1_Mange, Philo2_Pense, Philo2_Veut_Manger, Philo2_Mange]
    where
        process Deux_Philo[Philo1_Pense, Philo1_Veut_Manger, Philo1_Mange, Philo2_Pense, Philo2_Veut_Manger, Philo2_Mange]:=
            hide Fourch1_Prise_Par_PhG, Fourch1_Posee_Par_PhG, Fourch1_Prise_Par_PhD, Fourch1_Posee_Par_PhD,
                 Fourch2_Prise_Par_PhG, Fourch2_Posee_Par_PhG, Fourch2_Prise_Par_PhD, Fourch2_Posee_Par_PhD in
                    (Philosophe[Philo1_Pense, Philo1_Veut_Manger, Philo1_Mange,
                         Fourch2_Prise_Par_PhG, Fourch1_Prise_Par_PhD, Fourch2_Posee_Par_PhG, Fourch1_Posee_Par_PhD]
                    |||
                    Philosophe[Philo2_Pense, Philo2_Veut_Manger, Philo2_Mange,
                         Fourch1_Prise_Par_PhG, Fourch2_Prise_Par_PhD, Fourch1_Posee_Par_PhG, Fourch2_Posee_Par_PhD])
                    |
                         [Fourch1_Prise_Par_PhG, Fourch1_Posee_Par_PhG, Fourch1_Prise_Par_PhD, Fourch1_Posee_Par_PhD,
                          Fourch2_Prise_Par_PhG, Fourch2_Posee_Par_PhG, Fourch2_Prise_Par_PhD, Fourch2_Posee_Par_PhD]
                    |
                    (Fourchette[Fourch1_Prise_Par_PhD, Fourch1_Posee_Par_PhD, Fourch1_Prise_Par_PhG, Fourch1_Posee_Par_PhG]
                    |||
                    Fourchette[Fourch2_Prise_Par_PhD, Fourch2_Posee_Par_PhD, Fourch2_Prise_Par_PhG, Fourch2_Posee_Par_PhG])
        where
            process Philosophe[Philo_Pense, Philo_Veut_Manger, Philo_Mange, Philo_Prend_fd, Philo_Prend_fg, Philo_Pose_fd, Philo_Pose_fg] :=
                Philo_Pense; Philo_Veut_Manger;(Philo_Prend_fd; exit
                |||
                Philo_Prend_fg; exit) >> Philo_Mange; (Philo_Pose_fd; exit
                |||
                Philo_Pose_fg; exit) >> Philosophe[Philo_Pense, Philo_Veut_Manger, Philo_Mange, Philo_Prend_fd, Philo_Prend_fg, Philo_Pose_fd, Philo_Pose_fg]
            endproc
            process Fourchette [Prise_Par_PhD, Prise_Par_PhG, Posee_Par_PhD, Posee_Par_PhG] :=
                (Prise_Par_PhD ; Posee_Par_PhD ; exit
                []
                Prise_Par_PhG ; Posee_Par_PhG ; exit) >> Fourchette [Prise_Par_PhD, Prise_Par_PhG, Posee_Par_PhD, Posee_Par_PhG]
            endproc
    endproc
endsys
```

FIGURE 21 – Spécification LOTOS, dîner des philosophes avec 2 philosophes.

6.2 Question 2

De même que pour la question 4 de l'exercice 2, nous n'avons pas eu le temps de réaliser ces contraintes.

6.3 Question 3

On reprend les question 1 2 dans le cas de trois philosophes.

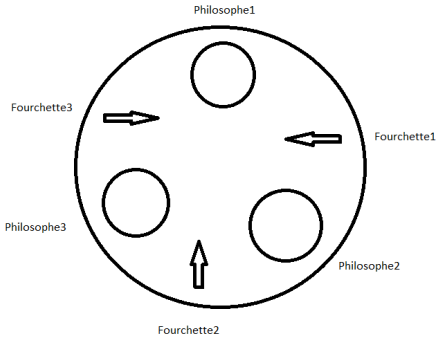


FIGURE 22 – Dîner des philosophes avec 3 philosophes.

```
system Question3[Philo1_Pense, Philo1_Veut_Manger, Philo1_Mange, Philo2_Pense, Philo2_Veut_Manger, Philo2_Mange, Philo3_Pense, Philo3_Veut_Manger, Philo3_Mange] :=
    Trois_Philo[Philo1_Pense, Philo1_Veut_Manger, Philo1_Mange, Philo2_Pense, Philo2_Veut_Manger, Philo2_Mange, Philo3_Pense, Philo3_Veut_Manger, Philo3_Mange]
    where
        process Trois_Philo[Philo1_Pense, Philo1_Veut_Manger, Philo1_Mange, Philo2_Pense, Philo2_Veut_Manger, Philo2_Mange, Philo3_Pense, Philo3_Veut_Manger, Philo3_Mange]:=
            hide Fourch1_Prise_Par_PhG, Fourch1_Posee_Par_PhG, Fourch1_Prise_Par_PhD, Fourch1_Posee_Par_PhD,
                 Fourch2_Prise_Par_PhG, Fourch2_Posee_Par_PhG, Fourch2_Prise_Par_PhD, Fourch2_Posee_Par_PhD,
                 Fourch3_Prise_Par_PhG, Fourch3_Posee_Par_PhG, Fourch3_Prise_Par_PhD, Fourch3_Posee_Par_PhD in
                    (Philosophe[Philo1_Pense, Philo1_Veut_Manger, Philo1_Mange,
                                Fourch2_Prise_Par_PhG, Fourch1_Prise_Par_PhD, Fourch2_Posee_Par_PhG, Fourch1_Posee_Par_PhD]
                        |||
                        Philosophe[Philo2_Pense, Philo2_Veut_Manger, Philo2_Mange,
                            Fourch3_Prise_Par_PhG, Fourch2_Prise_Par_PhD, Fourch3_Posee_Par_PhG, Fourch2_Posee_Par_PhD]
                        |||
                        Philosophe[Philo3_Pense, Philo3_Veut_Manger, Philo3_Mange,
                              Fourch1_Prise_Par_PhG, Fourch3_Prise_Par_PhD, Fourch1_Posee_Par_PhG, Fourch3_Posee_Par_PhD])
                    |
                        [Fourch1_Prise_Par_PhG, Fourch1_Posee_Par_PhG, Fourch1_Prise_Par_PhD, Fourch1_Posee_Par_PhD,
                         Fourch2_Prise_Par_PhG, Fourch2_Posee_Par_PhG, Fourch2_Prise_Par_PhD, Fourch2_Posee_Par_PhD,
                         Fourch3_Prise_Par_PhG, Fourch3_Posee_Par_PhG, Fourch3_Prise_Par_PhD, Fourch3_Posee_Par_PhD]
                    |
                    (Fourchette[Fourch1_Prise_Par_PhD, Fourch1_Posee_Par_PhD, Fourch1_Prise_Par_PhG, Fourch1_Posee_Par_PhG]
                        |||
                        Fourchette[Fourch2_Prise_Par_PhD, Fourch2_Posee_Par_PhD, Fourch2_Prise_Par_PhG, Fourch2_Posee_Par_PhG]
                        |||
                        Fourchette[Fourch3_Prise_Par_PhD, Fourch3_Posee_Par_PhD, Fourch3_Prise_Par_PhG, Fourch3_Posee_Par_PhG])
        where
            process Philosophe[Philo_Pense, Philo_Veut_Manger, Philo_Mange, Philo_Prend_fd, Philo_Prend_fg, Philo_Pose_fd, Philo_Pose_fg] :=
                Philo_Pense; Philo_Veut_Manger;(Philo_Prend_fd; exit
                |||
                Philo_Prend_fg; exit) >> Philo_Mange; (Philo_Pose_fd; exit
                |||
                Philo_Pose_fg; exit) >> Philosophe[Philo_Pense, Philo_Veut_Manger, Philo_Mange, Philo_Prend_fd, Philo_Prend_fg, Philo_Pose_fd, Philo_Pose_fg]
            endproc
            process Fourchette [Prise_Par_PhD, Prise_Par_PhG, Posee_Par_PhD, Posee_Par_PhG] :=
                (Prise_Par_PhD ; Posee_Par_PhD ; exit [] Prise_Par_PhG ; Posee_Par_PhG ; exit) >> Fourchette [Prise_Par_PhD, Prise_Par_PhG, Posee_Par_PhD, Posee_Par_PhG]
            endproc
    endproc
endsys
```

FIGURE 23 – Spécification LOTOS, dîner des philosophes avec 3 philosophes.