

TD LOTOS

November 18, 2008

1 Basic LOTOS

Note: avant chaque question il peut etre bon de chercher la reponse sur feuille avant d'utiliser xeuca.

1.1 Exercice 1

Decrire un ascenseur qui peut faire deux actions: `up` et `down`. On supposera que le nombre d'etages est de 3. Engendrer le modele du processus puis proceder a une animation. Expliquer comment permettre de rendre observable a tout moment l'etage courant du processus.

1.2 Exercice 2

Comparer $a;b;stop \parallel a;c;stop$ et $a;(b;stop \parallel c;stop)$ (cherchez deux equivalences, une qui ne les distingue pas et une les distingant).

1.3 Exercice 3

Comparer $a;b;c;stop \parallel d;b;stop$ et $a;b;c;stop \parallel [b] d;b;stop$.

1.4 Exercice 4

Donner un comportement equivalent a:

- $a;b;exit \parallel (a;c;exit \parallel c;b;exit)$
- $(a;b;stop \parallel c;d;stop) \parallel [a,b] (a;b;stop \parallel d;f;stop)$

et expliquer comment prouver l'equivalence.

1.5 Exercice 5

Ecrire un buffer a une place (operations `put` et `get`). Ecrire un buffer a deux places (memes operations). Comment realiser un buffer a deux places avec deux buffers a une place en parallele ? Comparer.

2 Full LOTOS

2.1 Exercice 5

Reprenre l'exercice 1 avec maintenant N etages (une donnee du processus). Essayer avec N=3 puis comparer avec le processus de l'exercice 1. Rappel : il est possible d'importer la librairie des naturels avec `library Boolean, Natural endlib`.

2.2 Exercice 6

Ecrire un processus modelisant le jeu du nombre cache. Le processus commence par permettre de choisir un nombre avec `choix` (se limiter aux entiers inferieurs a 10, ie 5+5). On entre un nombre et on recoit **trop petit** s'il est trop petit, **tropgrand** s'il est trop grand et **gagne** si c'est le bon. On s'arrete en choisissant une action `arret`. Ce processus passe alors le nombre d'essais necessaire (ou realises avant arret) a un second processus qui l'affiche (utiliser `>>`).

2.3 Exercice 7

Modeliser un processus de machine a cafe. Les boissons sont: cafe (.35 EUR) et the (.5 EUR). Les pieces acceptees sont: .5, .10, .20, .5 et 1 EUR. Vous ecrivez une sorte pour les boissons avec trois constantes (une par boisson). Les actions de la machine sont: `paie`, `choix` et `monnaie`. Le nombre de doses de cafe (resp. de the) est initialement de 5. La machine ne rend pas la monnaie (extension possible).

2.4 Exercice 8

Soit l'algorithme d'exclusion mutuelle suivant (naif):

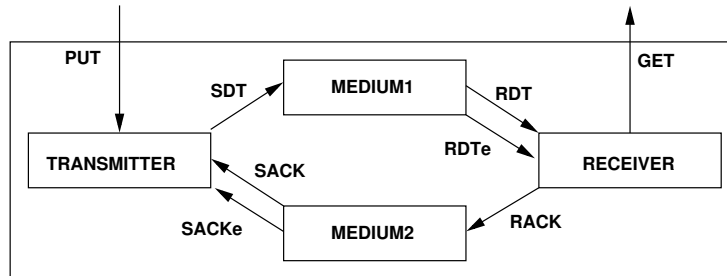
```
flag[0] = 0; flag[1] = 0;
```

```
Process 0 :                               Process 1 :
A: flag[0] := 1                             A: flag[1] := 0
B: while(flag[1])                           B: while(flag[0])
    ; // rien                                ; // rien
C: // section critique                       C: // section critique
D: flag[0] := 0                             D: flag[1] := 0
```

Cet algorithme verifie-t-il l'exclusion mutuelle ? L'absence d'interblocage ? L'absence de famine ? Faire aussi les questions en suspens du cours.

3 Protocole du bit alterne

Notes additionnelles aux informations donnees en TD.



<i>processus</i>	<i>signification</i>
TRANSMITTER	entite emettrice T
RECEIVER	entite receptrice R
MEDIUM1	transmission des messages de T vers R
MEDIUM2	transmission des messages de R vers T

Table 1: Liste des entites

<i>signal</i>	<i>orgine</i>	<i>destination</i>	<i>signification</i>
PUT?M	service	TRANSMITTER	emission d'un message
SDT!M!B	TRANSMITTER	MEDIUM1	envoi du message
RDT!m!B	MEDIUM1	RECEIVER	transmission message
RDTe	MEDIUM1	RECEIVER	perte du message
GET!M	RECEIVER	service	reception d'un message
RACK!B	RECEIVER	MEDIUM2	renvoi d'un acquitement
SACK!B	MEDIUM2	TRANSMITTER	transmission de l'acquitement
SACKe	MEDIUM2	TRANSMITTER	perte de l'acquitement

Table 2: Liste des signaux

On utilise un bit de controle avec les messages (voir plus bas pour eux).

```

type BIT is
  sorts BIT
  opns
    0 (*! constructor *) : -> BIT
    1 (*! constructor *) : -> BIT
    not : BIT -> BIT
  eqns
  forall b:BIT
  ofsort BIT
    not (0) = 1;
    not (1) = 0;
endtype
  
```

MEDIUM1. En recevant un message M avec un bit de controle egal a B, le MEDIUM1 peut reagir de trois facons differentes: transmettre correctement le message et son bit de controle (en aucun cas le medium ne peut changer ce bit), perdre le message et envoyer une indication de perte ou perdre silencieusement le message.

MEDIUM2. Fonctionnement analogue a la difference des noms de signaux et le fait que les acquitements, contrairement aux messages ne portent pas d'autre information que le bit de controle.

TRANSMITTER. Un message est acquis via PUT et est transmis au medium 1 apres lui avoir ajoute la valeur courante B du bit de controle (B est ensuite change). Si un acquitement est reçu avec le meme bit alors ok, sinon il faut remettre. Il y a trois causes de reemission: acquitement avec $\neg B$ comme bit de controle, indication de perte d'acquitement ou temporisation a echeance (modele par i en LOTOS).

RECEIVER. Symetrique a l'entite paire. Lorsqu'elle recoit un message avec un bit de controle B correct, le message est delivre via GET et un acquitement avec bit de controle B est renvoye (B est ensuite change). Un bit de controle $\neg B$ est renvoye dans les autres cas: message reçu avec bit de controle $\neg B$, indication de perte de message reçue ou temporisation a echeance (voir ci-dessus).

Ecrire les quatres composants et l'architecture puis comparer a la specification du protocole:

```

specification SERVICE[PUT,GET] : noexit
behavior
    SERVICE[PUT,GET]
where
    type MESSAGE is
        sorts MSG
        opns
            MSG0 (*! constructor *) : -> MSG
            MSG1 (*! constructor *) : -> MSG
            MSG2 (*! constructor *) : -> MSG
            MSG3 (*! constructor *) : -> MSG
    endtype
    process SERVICE[PUT,GET] : noexit :=
        PUT?M:MSG; GET!M; SERVICE[PUT,GET]
    endproc
endspec

```