M2Pro : Génie Logiciel

Winter Trimester 2009/2010

Mounir Lallali, Prof. Burkhart Wolff
Parc Orsay Université
4, rue Jacques Monod
Building H / Room 012

TD 3

Date : 06.10.2009, Durée : 3 heures

Exercice 1

Soit les processus :

```
process P[g] : exit :=
 (g?x:nat [x>5 and x<9]; P[g]
 []
 g !12; exit )
endproc (*P*)

process Q[g] : noexit :=
 g? x:nat [x mod 3 = 0]; Q[g]
endproc (*Q*)

process R[g] : noexit :=
 g? x:nat [x mod 2 = 0]; R[g]
endproc (*R*)
```

Questions

- Le comportement `P[gate]` ‖ `(Q[gate]` [> `exit)` ‖ `(R[gate]` [> `exit)` termine-t-il avec succès toujours, parfois, ou jamais ?
- Construire l'automate correspondant à cette composition parallèle.

Exercice 2
Soit le type abstrait :

```
type PaquetMessage is NaturalNumber, Message
    sorts Paquet
    opns
        vide : Paquet
        ajouter : Paquet, Message -> Paquet
        taille : Paquet -> Nat
    eqns
        forall x : Message, z : Paquet
        ofsort Nat
            taille (vide) = 0
            taille (ajouter(z, x)) =  taille(z) + taille(x)
endtype (*PaquetMessage*)
```

Le type Message n'est pas donné. On suppose qu'il y a une opération taille sur la sorte Message.

Dans ce problème, on vous demande de spécifier des processus qui reçoivent des messages sur la porte `input` et renvoient des paquets sur la porte `output`. Une propriété importante requise de tous ces processus est qu'ils respectent l'ordre de réception des messages, dans les paquets, et dans les envois de paquets.

1. Donner la spécification en LOTOS d'un processus `Pack2Stupide`, paramétré par un entier naturel `Max`, qui répète le traitement suivant. Il lit deux messages. Si la somme de leurs tailles est inférieure ou égale à `Max`, il envoie un paquet fait de ces deux messages. Sinon il envoie ces messages dans deux paquets séparés.
   Montrer par un exemple que `Pack2Stupide` ne met pas toujours dans un paquet deux messages consécutifs de taille totale inférieure ou égale à `Max`. C'est pour cela qu'il est stupide.
2. Spécifier un processus `Pack2`, moins stupide, qui met toujours dans un paquet deux messages consécutifs de taille totale inférieure ou égale à `Max`.
   Conseil : un processus auxiliaire, un peu différent de `Pack2`, peut servir. Il existe une solution simple.
3. Comment généraliser `Pack2` à `Pack`, qui construit des paquets d'un nombre quelconque de messages, de taille la plus grande possible mais toujours inférieure ou égale à `Max`, et toujours en respectant l'ordre de réception ?

Exercice 3

a) Soit les spécifications en LOTOS de base

```
Emitter[send] ::= send; Emitter [send]
Receiver[receive] ::= receive; Receiver[receive]
```

– Donner les traces de ces processus et le début de l'arbre des traces de :

```
Emitter[send] ||| Receiver[receive]
```

(il suffit de se limiter à 3 niveaux de transitions)

– Donner les traces de :

```
Emitter[send] || Receiver[receive]
Emitter[send] |[send]| Receiver[receive]
Emitter[send] |[receive]| Receiver[receive]
```

b) La spécification LOTOS ci-dessous décrit un processus transmetteur qui reçoit des messages avec priorité, les stocke dans une file d'attente avec priorité, les émet selon leur ancienneté et leur priorité.

```
process Transmitter[inGate, outGate](Q:Queue):noexit :=
    inGate?M:Message; Transmitter[inGate, outGate] (add(M,Q))
    []
    [isEmpty(Q)= false] -> outGate !get(Q); Transmitter[inGate,
       outGate] (remove(Q))
endproc
```

On a également les spécifications :

```
process Emitter2[inGate]: exit
    := inGate !mess(1,'HELLO'); inGate !mess(2,'WORLD');
  exit
endproc
process Receiver2[outGate] : exit
    := outGate?M:Message; outGate ?M': Message; exit
endproc
```

Donner les traces de l'expression de comportement :

```
behaviour
    (Emitter2 [inGate] ||| Receiver2[outGate])
    ||
    Transmitter[inGate, outGate](emptyq)
```