LINFO1104 TP9: Programmation concurrente et systèmes multi-agents (suite)

Cette séance approfondit les concepts de programmation concurrente déclarative.

Énoncés des exercices

1. *Concurrence et sémantique I.* Comme nous l'avons vu la semaine passée, les deux programmes ci-dessous ne sont pas identiques. Dessinez pour chacun de ces deux programmes l'arbre d'exécution en utilisant la machine abstraite (càd que l'arbre doit représenter tous les scénarios d'exécution du programme)

```
% Programme 1
local X Y Z in
    thread if X==1 then Y=2 else Z=2 end end
    thread if Y==1 then X=1 else Z=2 end end
    X=1
end

% Programme 2
local X Y Z in
    thread if X==1 then Y=2 else Z=2 end end
    thread if Y==1 then X=1 else Z=2 end end
    X=2
end
```

2. *Concurrence et sémantique, II.*

   Dessinez l'arbre d'exécution de ce programme en utilisant la machine abstraite.

   Donnez toutes les instructions qui peuvent s'exécuter.

   Toutes les exécution de ce programme passent-elles forcément par cet état ? Si non, donnez un contre-exemple.

```
local A B C D
    thread D = C+1 end
    thread C = B+1 end
    thread A = 1 end
    thread B = A+1 end
    {Browse D}
end
```

3. *Fabriques.* Dans les slides, on vous présente les fonctions `{AndG X Y}`, `{OrG X Y}` et `{XorG X Y}` qui représentent des portes logiques acceptant deux flux et en renvoyant un seul. Ces trois fonctions sont quasiment identiques. En bon programmeurs, vous décidez de factoriser votre code.

   - Écrivez une fonction `{MakeBinaryGate F}` qui prend en argument un operateur binaire et retourne une porte logique pour celui-ci. Par exemple, le code suivant permettrait de générer `XorG`.

     ```
     fun {Xor A B} (A + B) mod 2 end
     XorG = {MakeBinaryGate Xor}
     ```

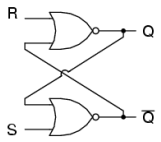
   - Quelle est la complexité de `MakeBinaryGate` ? Quelle est la complexité des fonctions retournés par `MakeBinaryGate` ?

   - À l'aide de `MakeBinaryGate`, donnez les expressions permettant de créer `{AndG X Y}`, `{OrG X Y}` et `{NorG X Y}` ?

4. *Portes logiques.* Le programme suivant implémente une bascule RS telle que dessinée sur le schéma ci dessous.

   Malheureusement, ce programme ne fonctionne pas correctement. Trouvez, expliquez et corrigez la(les) erreur(s).

```
local R=1|1|1|0|_ S=0|1|0|0_ Q NotQ
    proc {Bascule Rs Ss Qs NotQs}
        {NorG Rs NotQs Qs}
```



S	R	Q	$\overline{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

Figure 1: bascule

```
        {NorG Ss Qs NotQs}
    end
in
    {Bascule R S Q NotQ}
    {Browse Q#NotQ}
end
```

5. *J'ai rien compris aux* ***list comprehension****.* La fonction `ForCollect` est implémentée à l'aide d'une cellule, comme suit :

```
declare
proc {ForCollect Xs P Ys}
   Acc={NewCell Ys}
   proc {C X} R2 in @Acc=X|R2 Acc:=R2 end
in
   for X in Xs do {P C X} end @Acc=nil
end
{Browse {ForCollect [ 0 2 4 6 8 ]
    proc {$ Collect X} {Collect X div 2} end}} % Affiche ... ?
```

- Qu'affiche l'appel à Browse ?
- Comme personne n'aime les cellules, vous aimeriez réécrire ce code sans en utiliser. Écrivez une procédure `{ForCollectDecl Xs Proc Ys}` qui n'utilise pas de cellule.
- Expliquez ce que renvoie l'appel suivant.

```
{Browse {ForCollect [0 1 2 3 4 5] proc {$ Collect X}
   case X mod 3
   of 0 then {Collect X} {Collect X}
   [] 1 then {Collect X}
   [] 2 then skip
   end
end}} % Affiche ?
```

Votre fonction ForCollect déclarative supporte-t-elle d'être appelée avec ces arguments ? Pourquoi ?