Rapport Outils Preuve et Vérification

Problème de l'exclusion mutuelle entre processus

Korlan Colas Lisa Aubry

25 octobre 2015

Table des matières

Chapitre 1

Situations d'interblocage

1.1 Algorithme 2.1

1.1.1 Description générale

L'algorithme présenté à la figure 2.1 décrit le comportement de deux processus, pour lesquels l'accès à la section critique est régi par l'existence d'une seule variable partagée. En l'occurence, cette variable nommée TURN observe un comportement booléen, en cela que sa valeur (entière) est soit un, soit deux. Afin d'apporter d'avantage de lisibilité au raisonnement qui va suivre, nous allons réécrire en partie l'algorithme proposé, en associant à chacune des instructions un numéro de ligne.

```
1   TURN := 1;
2   wait until TURN = 2;
3   Critical Section
```

1.1.2 Représentation du problème

En vue d'une modélisation de cet algorithme avec l'outil Véritaf, nous définissons deux sous-systèmes. L'un sera dédié à la représentation des processus et de leur avancée dans l'exécution du code, tandis que le second sous-système indiquera la valeur de TURN.

Le sous-système Process

$|S| = 5$

$S = \{0, 1, 2, 3, 4\}$

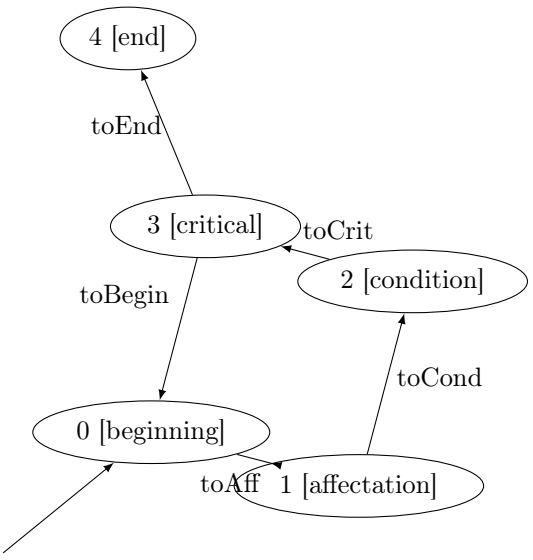
$S_0 = \{0\}$

$AP = \{beginning, affectation, condition, critical, end\}$

$$\lambda = \begin{cases} 0 \rightarrow & \text{beginning} \\ 1 \rightarrow & \text{affectation} \\ 2 \rightarrow & \text{condition} \\ 3 \rightarrow & \text{critical} \\ 4 \rightarrow & \text{end} \end{cases}$$

$\rightarrow = \{toBegin, toCond, toCrit, toAff, toEnd\}$

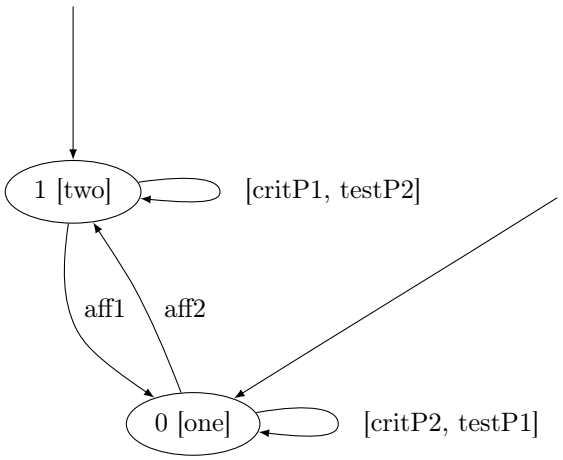
$Process =< S, S_0, \rightarrow, \lambda, AP >$



Ici, pour les états 1, 2 et 3, il y a correspondance entre le numéro de l'état et le numéro de la ligne que le processus est en train d'exécuter. On ajoute en plus l'état 0 qui indique que le processus va entrer dans la section de code (donc qu'il se trouve avant la ligne 1), et l'état 4 qui indique que le processus est sorti de manière définitive de la section de code (donc qu'il se situe après la ligne 3, sans effectuer de boucle).

Le sous-système Shared

$|S| = 2$
$S = \{0, 1\}$
$S_0 = \{0\}$
$AP = \{one, two\}$
$\lambda = \begin{cases} 0 \rightarrow & \text{one} \\ 1 \rightarrow & \text{two} \end{cases}$
$\rightarrow = \{aff2, aff1, testP1, critP2, testP2, critP1\}$
$Process =< S, S_0, \rightarrow, \lambda, AP >$



Comme dit précédemment, les états de ce sous-système donnent la valeur de la variable partagée, et ce de manière explicite.

1.1.3 Système

Le système correspondant à notre problème sera constitué de deux éléments P1 et P2, affiliés au sous-système Process, ainsi que d'un élément nommé TURN qui correspond au sous-système Shared. Afin que le présent rapport reste concis, les transitions définies pour chacun des sous-sytèmes sont plus amplement commentées dans le programme Véritaf.

1.1.4 Résultat

Conformément à notre définition du système complet, le non respect de la propriété d'exclusion mutuelle peut-être formalisée en logique CTL par :

$$(1) P1.critical \wedge P2.critical$$

De même, une situation d'interblocage sera formulée de la manière suivante :

$$(2) !EX(true) \wedge (P1.condition \vee P2.condition)$$

D'après nos résultats, l'exclusion mutuelle des processus au regard de la section critique est préservée dans le cas présent. En effet, le réduit qui vérifie la formule (1) est vide.
Cependant, le réduit vérifiant la formule (2) est non vide. On en conclut que cet algorithme mène à une situation d'interblocage qui survient lorsqu'un processus quitte cette partie de code (état *end*), tandis que le deuxième processus est en attente d'une modification de la valeur de TURN. Autrement dit, un processus est susceptible d'attendre indéfiniement sans que l'accès à la section critique ne lui soit jamais accordé.

1.2 Algorithme 2.2

1.2.1 Description générale

L'algorithme présenté à la figure 2.2 délaisse la variable partagée TURN au profit de deux variables partagée appelées Q1 et Q2, lesquelles vérifient aussi des valeurs booléennes. Une fois encore, voici la partie de code proposée pour le premier processus.

```
1   Q1 := true;
2   wait until not Q2;
3   Critical Section
4   Q1 := false;
```

1.2.2 Représentation du problème

Nous définissons également deux sous-systèmes sur le même modèle que précédemment, à savoir l'un pour décrire l'avancée des deux processus dans le code, et l'autre aui aura vocation de modéliser les variables partagées.

Le sous-système Process

$|S| = 5$

$S = \{0, 1, 2, 3, 4\}$

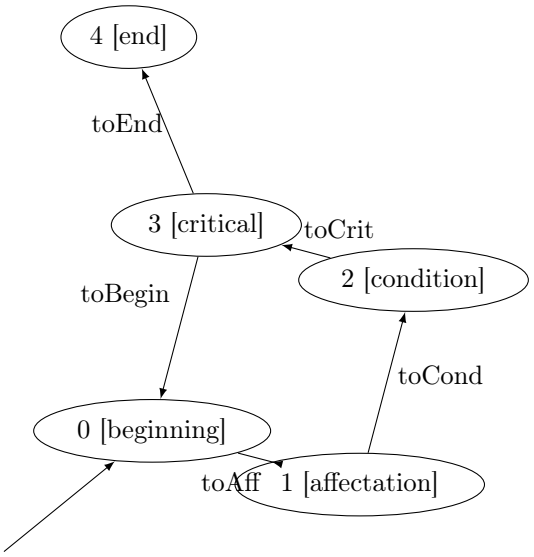
$S_0 = \{0\}$

$AP = \{beginning, affectation, condition, critical, end\}$

$$\lambda = \begin{cases} 0 \rightarrow & \text{beginning} \\ 1 \rightarrow & \text{affectation} \\ 2 \rightarrow & \text{condition} \\ 3 \rightarrow & \text{critical} \\ 4 \rightarrow & \text{end} \end{cases}$$

$\rightarrow = \{toBegin, toCond, toCrit, toAff, toEnd\}$

$Process =< S, S_0, \rightarrow, \lambda, AP >$



Pour les états 1, 2, 3 et 4, il y a correspondance entre le numéro de l'état et le numéro de la ligne que le processus est en train d'exécuter. On ajoute en plus l'état 0 qui indique que le processus va entrer dans la section de code (donc qu'il se situe avant la ligne 1).

Le sous-système Shared

$|S| = 2$

$S = \{0, 1\}$

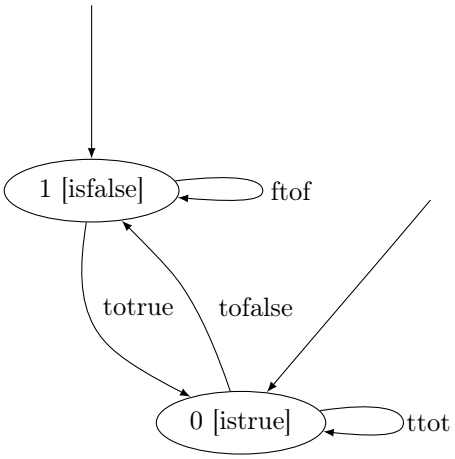
$S_0 = \{0\}$

$AP = \{isTrue, isFalse\}$

$$\lambda = \begin{cases} 0 \rightarrow & \text{isTrue} \\ 1 \rightarrow & \text{isFalse} \end{cases}$$

$\rightarrow = \{toFalse, toTrue, ftof, ttot\}$

$Process =< S, S_0, \rightarrow, \lambda, AP >$



De manière analogue, ce sous-sytème et ses états permettront d'indiquer quelle est la valeur de Qi.

1.2.3 Système

On définit ensuite P1 et P2, régis par le sous-système Process, ainsi que Q1 et Q2 associés au sous-système Shared.

1.2.4 Résultat

Conformément à notre définition du système complet, le non respect de la propriété d'exclusion mutuelle peut-être formalisée en logique CTL par :

$$(1) P1.critical \wedge P2.critical$$

De même, une situation d'interblocage sera formulée de la manière suivante :

$$(2)!EX(true) \wedge (P1.test \vee P2.test)$$

Tout comme pour l'algorithme précédent, les accès concurrents concurrents à la section critique sont gérés convenablement par cet algorithme. Cela se confirme par le fait que le réduit vérifiant la formule (1) est vide.
Une fois encore, le réduit vérifiant la formule (2) est non vide. Ici, et contrairement aux cas précédents, la situation d'interblocage a lieu alors que les deux processus tentent d'accéder à la section critique. En effet, on peut constater que dans ce cas, $Q1 = Q2 = true$.

Chapitre 2

Mutex pour 2 processus

2.1 Solution simple

2.1.1 Description générale

L'algorithme présenté à la figure 1 est une solution pour que deux processus puissent partager une section critique commune. Cette solution a besoin d'une variable partagée TURN modifiées par les deux processus ainsi que Q1 et Q2 appartenant respectivement au processus 1 et 2. Le processus 1 accède en lecture à la variable Q2 et inversement pour le second processus. Par ailleurs, seulement le premier processus pourra modifier Q1 (et inversement pour le deuxième processus.

```
1  /* protocol for P1 */
2  Q1 := true;
3  TURN := 1
4  wait until not Q2 or TURN = 2;
5  Critical Section
6  Q1 := false;
```

```
1  /* protocol for P2 */
2  Q2 := true;
3  TURN := 2
4  wait until not Q1 or TURN = 1;
5  Critical Section
6  Q2 := false;
```

2.1.2 Représentation du problème

Le sous-système Processus

$|S| = 5$

$S = \{0, 1, 2, 3, 4\}$

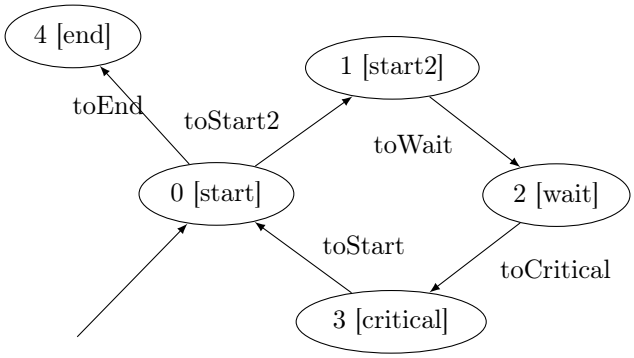
$S_0 = \{0\}$

$AP = \{start, start2, wait, critical, end\}$

$$\lambda = \begin{cases} 0 \rightarrow & \text{start} \\ 1 \rightarrow & \text{start2} \\ 2 \rightarrow & \text{wait} \\ 3 \rightarrow & \text{critical} \\ 4 \rightarrow & \text{end} \end{cases}$$

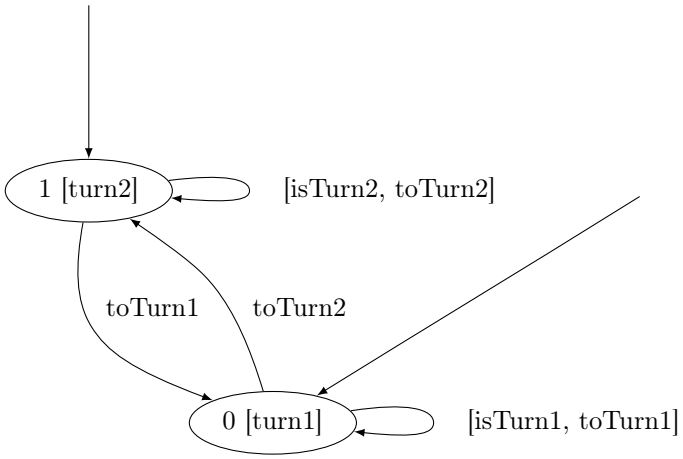
$\rightarrow = \{toStart, toStart2, toWait, toCritical, toEnd\}$

$Process =< S, S_0, \rightarrow, \lambda, AP >$



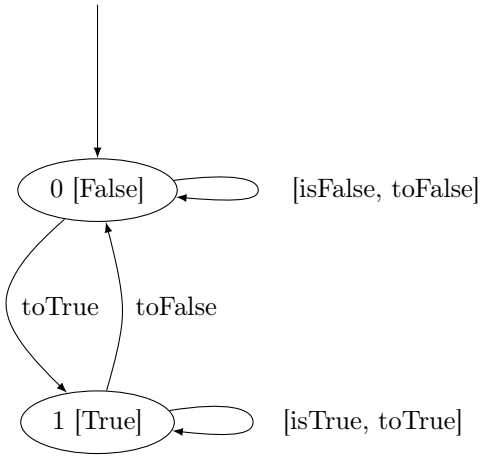
Le sous-système sharedT (TURN)

$|S'| = 2$
$S' = \{0, 1\}$
$S'_0 = \{0, 1\}$
$AP = \{turn1, turn2\}$
$\lambda = \begin{cases} 0 \rightarrow & \text{turn1} \\ 1 \rightarrow & \text{turn2} \end{cases}$
$\rightarrow = \{isTurn1, isTurn2, toTurn1, toTurn2\}$
$Shared =< S', S'_0, \rightarrow, \lambda, AP >$



Le sous-système sharedQ (Q)

$|S''| = 2$
$S'' = \{0, 1\}$
$S''_0 = \{0, 1\}$
$AP = \{False, True\}$
$\lambda = \begin{cases} 0 \rightarrow & \text{False} \\ 1 \rightarrow & \text{True} \end{cases}$
$\rightarrow = \{isFalse, isTrue, toFalse, toTrue\}$
$Shared =< S'', S''_0, \rightarrow, \lambda, AP >$



2.1.3 Système

On définit ensuite P1 et P2, régis par le sous-système Process, Q1 et Q2 associés au sous-système SharedQ. Pour finir la variable TURN appartenant au sous-système sharedT
Le système s'écrit donc $systeme =< processP1, booleanQ1, processP2, booleanQ2, sharedTURN >$
Note : les vecteurs de synchronisation sont disponibles dans le fichier ***simple.mso***.

2.1.4 Résultat

Les résultats sont très satisfaisant : L'exclusion mutuelle est respecté, il n'y a pas de deadlock et l'équité est aussi validé. Les trois conditions sont réunis pour qualifier cet algorithme de viable pour jouer le rôle d'un mutex.

2.2 Solution simple inversé

2.2.1 Description générale

La solution modifié ne fonctionne pas (Elle ne respecte en aucun cas les 3 principes du mutex vu en introduction. Cette solution a pour but de mettre en evidence l'importance de l'ordre d'exécution des tests et des affectations. Nous inversons donc les deux premières affectations de la version proposé précédement.

```
1  /* protocol for P1 */
2  TURN := 1
3  Q1 := true;
4  wait until not Q2 or TURN = 2;
5  Critical Section
6  Q1 := false;
```

```
1  /* protocol for P2 */
2  TURN := 2
3  Q2 := true;
4  wait until not Q1 or TURN = 1;
5  Critical Section
6  Q2 := false;
```

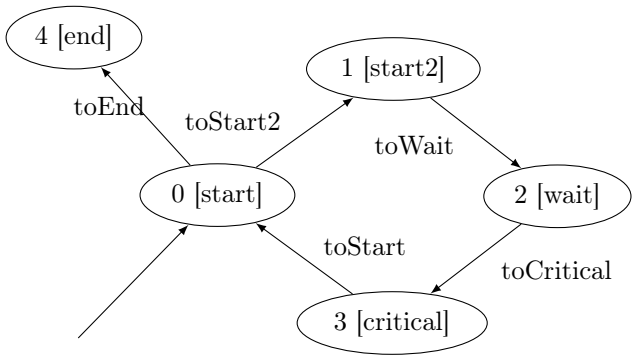
2.2.2 Représentation du problème

Le sous-système Processus

$|S| = 5$
$S = \{0, 1, 2, 3, 4\}$
$S_0 = \{0\}$
$AP = \{start, start2, wait, critical, end\}$

$$\lambda = \begin{cases} 0 \rightarrow & \text{start} \\ 1 \rightarrow & \text{start2} \\ 2 \rightarrow & \text{wait} \\ 3 \rightarrow & \text{critical} \\ 4 \rightarrow & \text{end} \end{cases}$$

$\rightarrow = \{toStart, toStart2, toWait, toCritical, toEnd\}$
$Process =< S, S_0, \rightarrow, \lambda, AP >$

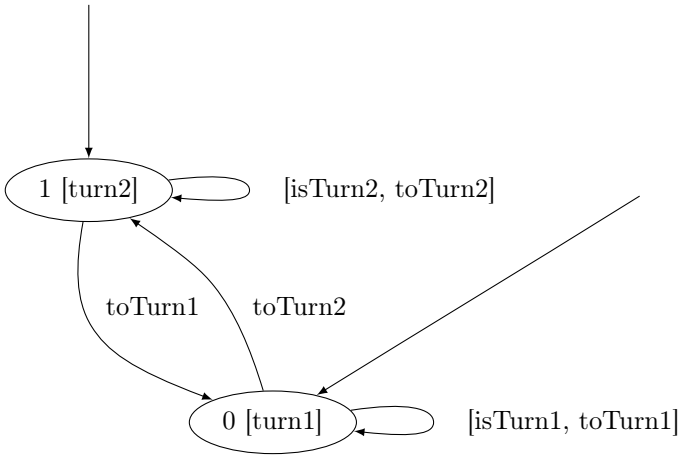


Le sous-système sharedT (TURN)

$|S'| = 2$
$S' = \{0, 1\}$
$S'_0 = \{0, 1\}$
$AP = \{turn1, turn2\}$

$$\lambda = \begin{cases} 0 \rightarrow & \text{turn1} \\ 1 \rightarrow & \text{turn2} \end{cases}$$

$\rightarrow = \{isTurn1, isTurn2, toTurn1, toTurn2\}$
$Shared =< S', S'_0, \rightarrow, \lambda, AP >$

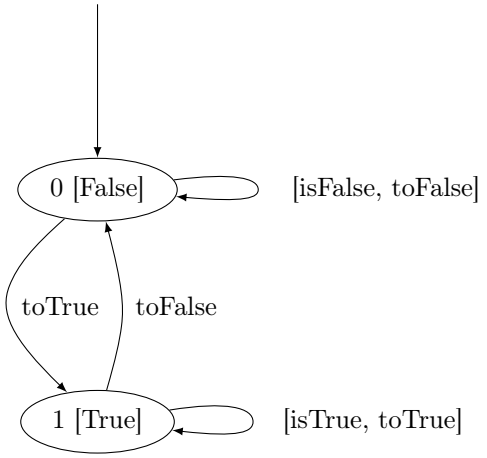


Le sous-système sharedQ (Q)

$|S''| = 2$
$S'' = \{0, 1\}$
$S''_0 = \{0, 1\}$
$AP = \{False, True\}$

$$\lambda = \begin{cases} 0 \rightarrow & \text{False} \\ 1 \rightarrow & \text{True} \end{cases}$$

$\rightarrow = \{isFalse, isTrue, toFalse, toTrue\}$
$Shared =< S'', S''_0, \rightarrow, \lambda, AP >$

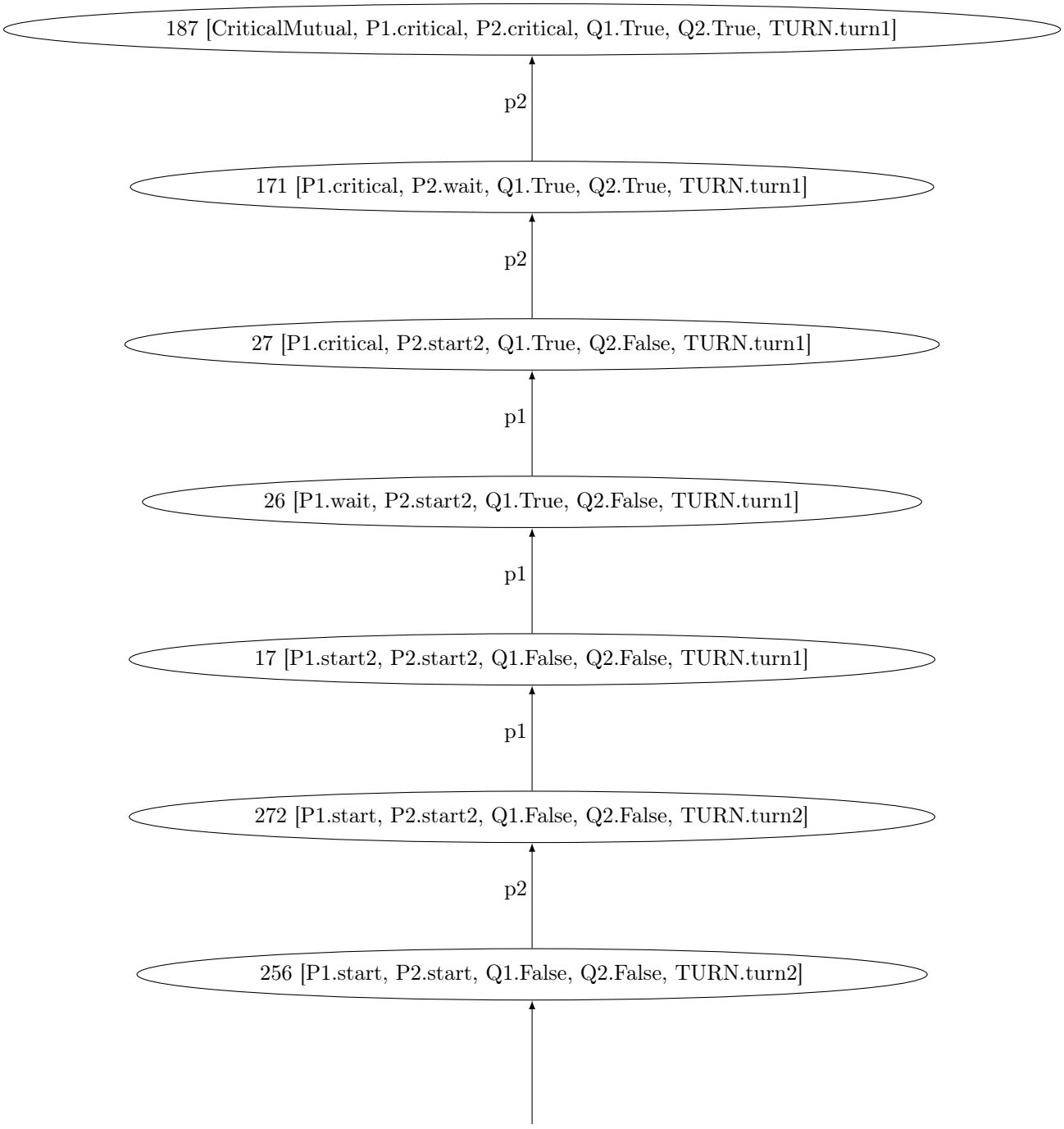


2.2.3 Système

On définit ensuite P1 et P2, régis par le sous-système Process, Q1 et Q2 associés au sous-système SharedQ. Pour finir la variable TURN appartenant au sous-système sharedT
Le système s'écrit donc $systeme =< processP1, booleanQ1, processP2, booleanQ2, sharedTURN >$
Note : les vecteurs de synchronisation sont disponibles dans le fichier ***simpleInverse.mso***.

2.2.4 Résultat

La modification de notre solution en inversant l'ordre des deux première instructions a fortement impacté la qualité de notre algorithme. En effet celui-ci ne garanti plus les 3 règles du mutex validées précédemment. L'ensemble de l'inter-blocage est toujours vide sauf que le réduit nous montre que l'exclusion mutuelle n'est plus respectée comme le montre le graphe suivant. (Par soucis de place, voici 1 chemin parmis plusieurs qui mène a une execution de la section critique simultanément)



Chapitre 3

Généralisation a N processus

3.1 Description générale

La dernière partie de notre analyse consiste a exploiter les résultats issu d'une généralisation de la solution simple pour 2 processus en l'étendant à N processus. On retrouve ainsi les variables TURN et Q sauf que ceux ci sont maintenant des tableaux (TURN[1..N-1] et Q[1..N] respectivement initialisé a 1 et 0).
Note : dans l'algorithme i = Numéro du processus.

```
1  /* protocol for Pi */
2  for j := 1 to n-1 do
3  begin
4    Q[i] :=  j;
5    TURN[j] := i;
6     wait until( ∀k ≠ i, Q[k] < j) or TURN[j] ≠ i
7  end;
8  Critical Section;
9  Q[i] := 0
```

3.2 Représentation du problème

Le sous-système Processus

$|S| = 8$

$S = \{0, 1, 2, 3, 4, 5, 6, 7\}$

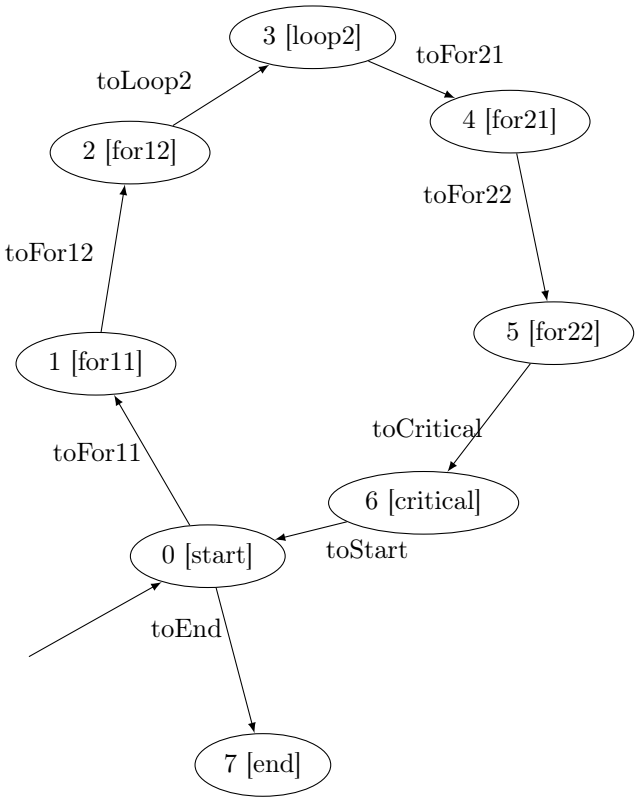
$S_0 = \{0\}$

$AP = \{start, for11, for12, loop2, for21, for22, critical, end\}$

$$\lambda = \begin{cases} 0 \rightarrow & \text{start} \\ 1 \rightarrow & \text{for11} \\ 2 \rightarrow & \text{for12} \\ 3 \rightarrow & \text{loop2} \\ 4 \rightarrow & \text{for21} \\ 5 \rightarrow & \text{for22} \\ 6 \rightarrow & \text{critical} \\ 7 \rightarrow & \text{end} \end{cases}$$

$\rightarrow = \{toStart, toFor11, toFor12, toLoop2, toFor21, toFor22, toCritical, toEnd\}$

$Process = < S, S_0, \rightarrow, \lambda, AP >$

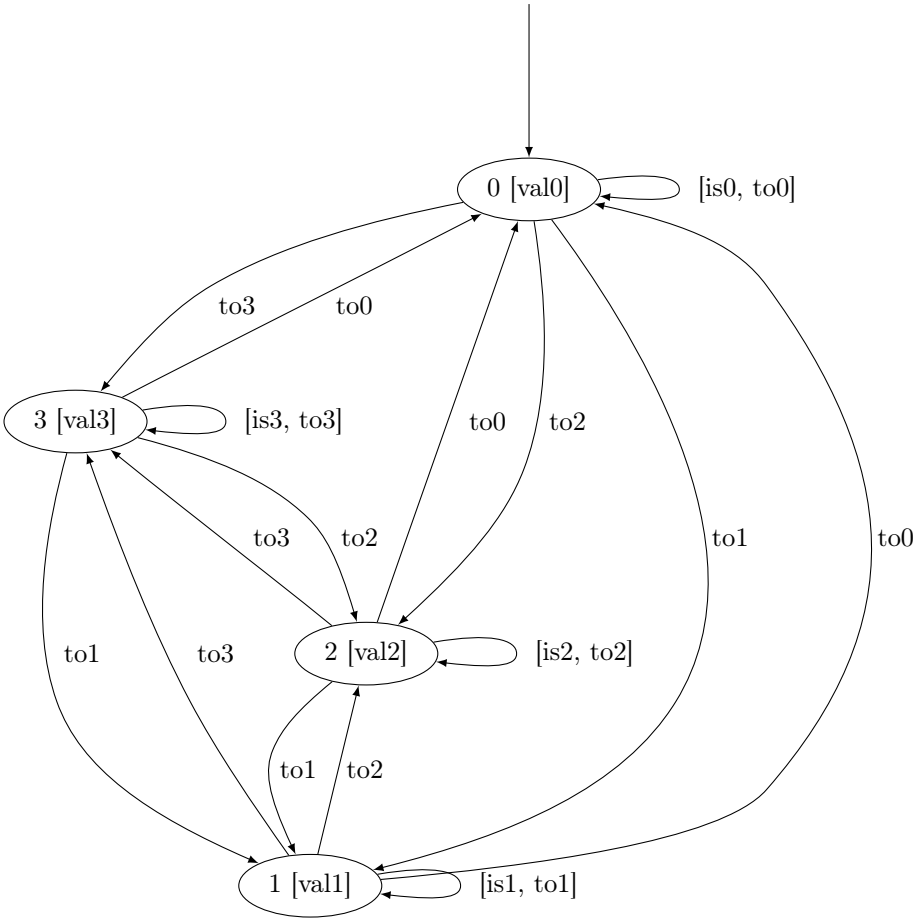


Le sous-système sharedQ (Q)

$|S'| = 4$
$S' = \{0, 1, 2, 3\}$
$S'_0 = \{0\}$
$AP = \{val0, val1, val2, val3\}$

$$\lambda = \begin{cases} 0 \rightarrow & \text{val0} \\ 1 \rightarrow & \text{val1} \\ 2 \rightarrow & \text{val2} \\ 3 \rightarrow & \text{val3} \end{cases}$$

$\rightarrow = \{is0, is1, is2, is3, to0, to1, to2, to3\}$
$Shared =< S', S'_0, \rightarrow, \lambda, AP >$

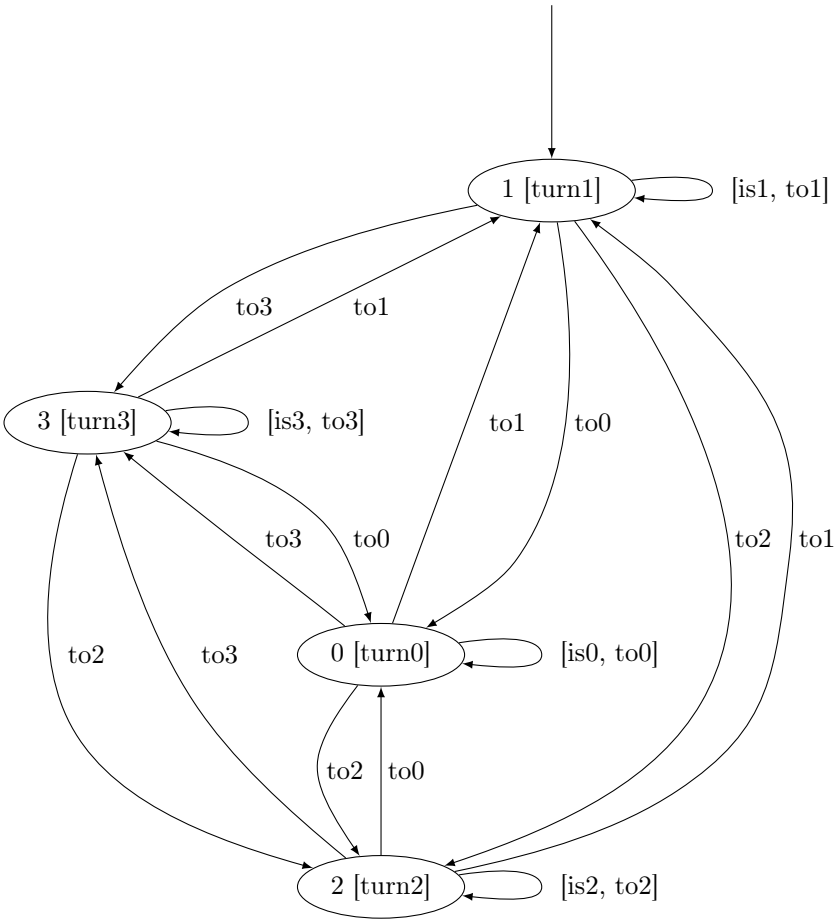


Le sous-système sharedT (TURN)

$|S'| = 4$
$S' = \{0, 1, 2, 3\}$
$S'_0 = \{0\}$
$AP = \{turn0, turn1, turn2, turn3\}$

$$\lambda = \begin{cases} 0 \rightarrow & \text{turn0} \\ 1 \rightarrow & \text{turn1} \\ 2 \rightarrow & \text{turn2} \\ 3 \rightarrow & \text{turn3} \end{cases}$$

$\rightarrow = \{is0, is1, is2, is3, to0, to1, to2, to3\}$
$Shared =< S', S'_0, \rightarrow, \lambda, AP >$



3.3 Système

On définit ensuite P1, P2 et P3, régis par le sous-système Process. Nous rapellons que dans notre cas N=3 nous avons besoin de deux tableaux Q[3] et TURN[2]. Nous simulons donc ces tableaux par Q1,Q2,Q3 et TURN1, TURN2 respectivement de type sharedQ et sharedT.
Ces tableaux sont accessible en lecture/écriture par tous les programmes. (je sais pas si on l'écrit ça comme c'est pas trop ça mais un peu...) Le système s'écrit donc
$< processP1, processP2, processP3, sharedQQ1, sharedQQ2, sharedQQ3, sharedTt1, sharedTt2 >$
Note : les vecteurs de synchronisation sont disponibles dans le fichier ***nProcesses.mso***.

3.4 Résultat

Bien que le fichier au format DOT montrant le fonctionnement du système (*nProcesses/systeme.dot*) ne peux pas être lu (le nombre d'état ne permet pas a un humain de déceler des dysfonctionnement) nous utilisons alors les formules CTL pour vérifier le modèle. D'un point de vu inter blocage et exclusion mutuelle les résultat sont encourageant car les réduits nous montre que les processus ne s'inter-bloque jamais ni n'exécutent la section critique en même temps. Par ailleurs l'équité n'est pas respecté.