

Rapport Outils Preuve et Vérification
Problème de l'exclusion mutuelle entre processus

Korlan Colas

Lisa Aubry

25 octobre 2015

Table des matières

1	Situations d'interblocage	2
1.1	Algorithme 2.1	2
1.1.1	Description générale	2
1.1.2	Représentation du problème	2
1.1.3	Système	3
1.1.4	Résultat	3
1.2	Algorithme 2.2	3
1.2.1	Description générale	3
1.2.2	Représentation du problème	4
1.2.3	Système	4
1.2.4	Résultat	4
2	Mutex pour 2 processus	6
2.1	Solution simple	6
2.1.1	Description générale	6
2.1.2	Représentation du problème	6
2.1.3	Système	7
2.1.4	Résultat	7
2.2	Solution simple inversé	7
2.2.1	Description générale	7
2.2.2	Représentation du problème	8
2.2.3	Système	9
2.2.4	Résultat	9
3	Généralisation a N processus	10
3.1	Description générale	10
3.2	Représentation du problème	10
3.3	Système	12
3.4	Résultat	12

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'occurrence, 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.

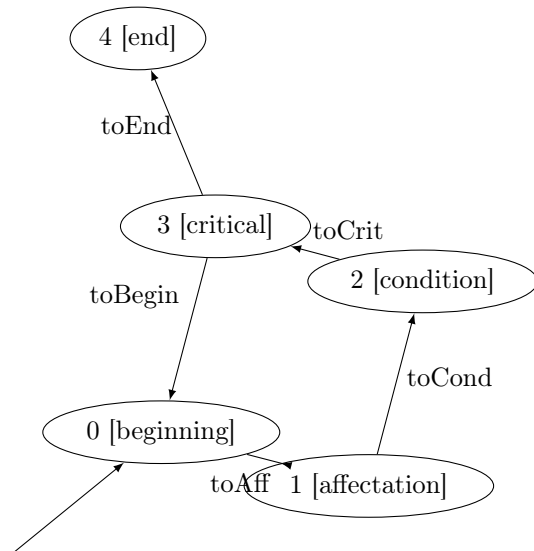
```
1  TURN := 1;
2  wait until TURN = 2;
3  Critical Section
```

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

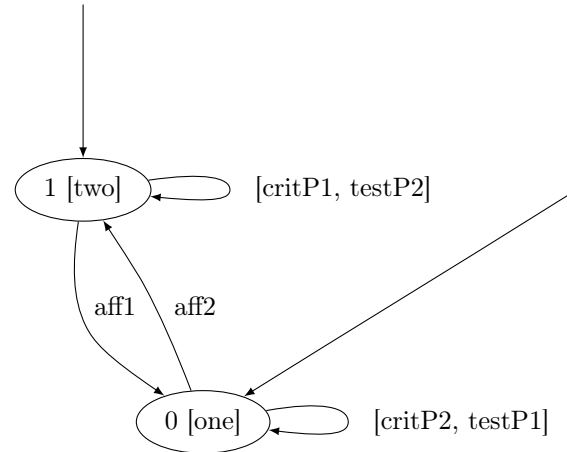
$|S| = 5$
 $S = \{0, 1, 2, 3, 4\}$
 $S_0 = \{0\}$
 $AP = \{beginning, affectation, condition, critical, end\}$
 $\lambda = \begin{cases} 0 \rightarrow beginning \\ 1 \rightarrow affectation \\ 2 \rightarrow condition \\ 3 \rightarrow critical \\ 4 \rightarrow end \end{cases}$
 $\rightarrow = \{toBegin, toCond, toCrit, toAff, toEnd\}$
 $Process = \langle S, S_0, \rightarrow, \lambda, AP \rangle$



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

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



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-systèmes 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 indéfiniment 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 partagées 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;

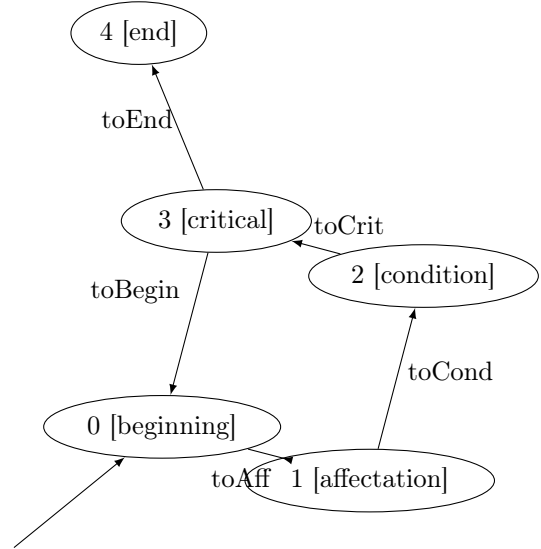
```

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 qui aura vocation de modéliser les variables partagées.

Le sous-système Process

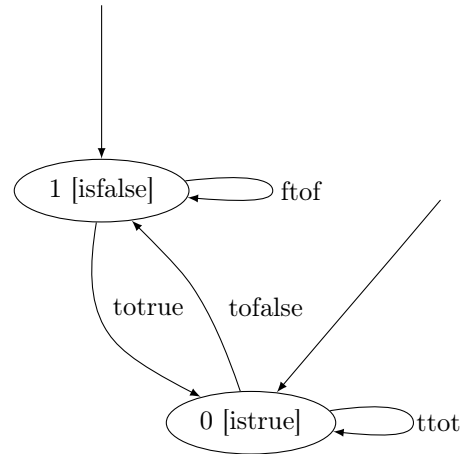
$|S| = 5$
 $S = \{0, 1, 2, 3, 4\}$
 $S_0 = \{0\}$
 $AP = \{beginning, affectation, condition, critical, end\}$
 $\lambda = \begin{cases} 0 \rightarrow \text{beginning} \\ 1 \rightarrow \text{affectation} \\ 2 \rightarrow \text{condition} \\ 3 \rightarrow \text{critical} \\ 4 \rightarrow \text{end} \end{cases}$
 $\rightarrow = \{toBegin, toCond, toCrit, toAff, toEnd\}$
 $Process = \langle S, S_0, \rightarrow, \lambda, AP \rangle$



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

$|S| = 2$
 $S = \{0, 1\}$
 $S_0 = \{0\}$
 $AP = \{isTrue, isFalse\}$
 $\lambda = \begin{cases} 0 \rightarrow \text{isTrue} \\ 1 \rightarrow \text{isFalse} \end{cases}$
 $\rightarrow = \{toFalse, toTrue, ftof, ttot\}$
 $Process = \langle S, S_0, \rightarrow, \lambda, AP \rangle$



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

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 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` modifiée 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.

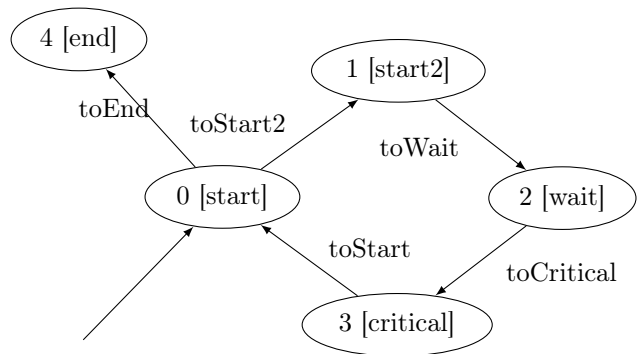
```
1  /* protocol for P1 */
2  Q1 := true;
3  TURN := 1
4  wait until not Q2 or TURN = 2;
5  Critical Section
6  Q1 := false;
```

```
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

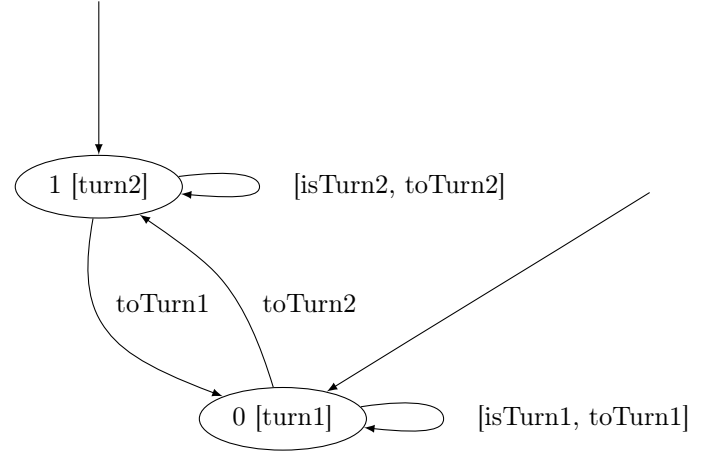
Le sous-système Processus

$|S| = 5$
 $S = \{0, 1, 2, 3, 4\}$
 $S_0 = \{0\}$
 $AP = \{start, start2, wait, critical, end\}$
 $\lambda = \begin{cases} 0 \rightarrow start \\ 1 \rightarrow start2 \\ 2 \rightarrow wait \\ 3 \rightarrow critical \\ 4 \rightarrow end \end{cases}$
 $\rightarrow = \{toStart, toStart2, toWait, toCritical, toEnd\}$
 $Process = \langle S, S_0, \rightarrow, \lambda, AP \rangle$



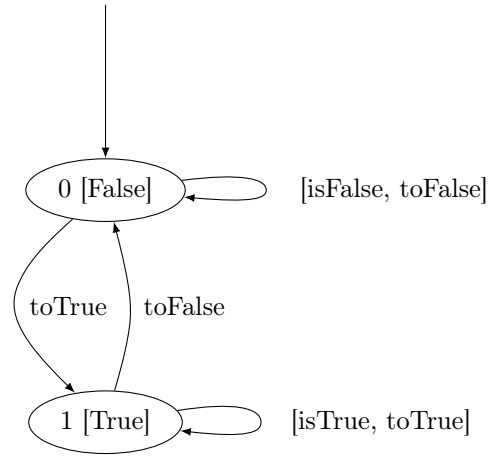
Le sous-système sharedT (TURN)

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



Le sous-système sharedQ (Q)

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



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 $systeme = \langle processP1, booleanQ1, processP2, booleanQ2, sharedTURN \rangle$

Note : les vecteurs de synchronisation sont disponibles dans le fichier *simple.mso*.

2.1.4 Résultat

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

2.2 Solution simple inversé

2.2.1 Description générale

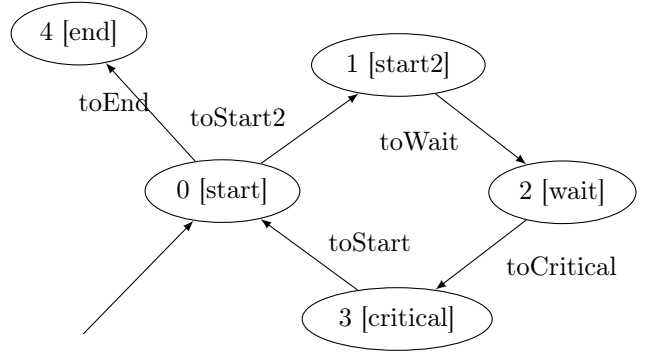
La solution modifiée ne fonctionne pas (Elle ne respecte en aucun cas les 3 principes du mutex vus en introduction. Cette solution a pour but de mettre en évidence l'importance de l'ordre d'exécution des tests et des affectations. Nous inversons donc les deux premières affectations de la version proposée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

1	/* protocol for P1 */	1	/* protocol for P2 */
2	TURN := 1	2	TURN := 2
3	Q1 := true;	3	Q2 := true;
4	wait until not Q2 or TURN = 2;	4	wait until not Q1 or TURN = 1;
5	Critical Section	5	Critical Section
6	Q1 := false;	6	Q2 := false;

2.2.2 Représentation du problème

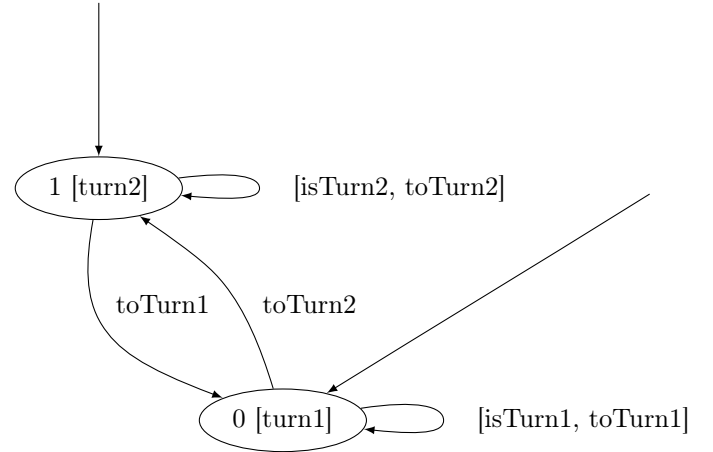
Le sous-système Processus

$|S| = 5$
 $S = \{0, 1, 2, 3, 4\}$
 $S_0 = \{0\}$
 $AP = \{start, start2, wait, critical, end\}$
 $\lambda = \begin{cases} 0 \rightarrow start \\ 1 \rightarrow start2 \\ 2 \rightarrow wait \\ 3 \rightarrow critical \\ 4 \rightarrow end \end{cases}$
 $\rightarrow = \{toStart, toStart2, toWait, toCritical, toEnd\}$
 $Process = \langle S, S_0, \rightarrow, \lambda, AP \rangle$



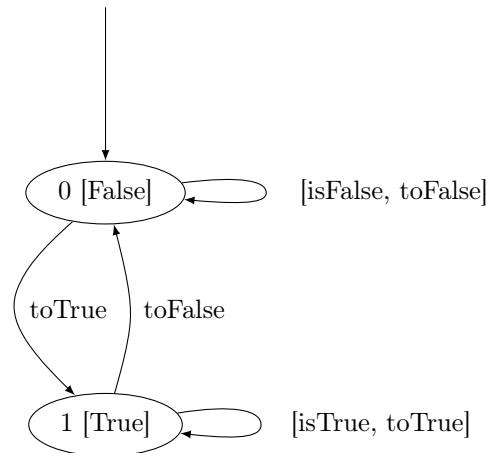
Le sous-système sharedT (TURN)

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



Le sous-système sharedQ (Q)

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



2.2.3 Système

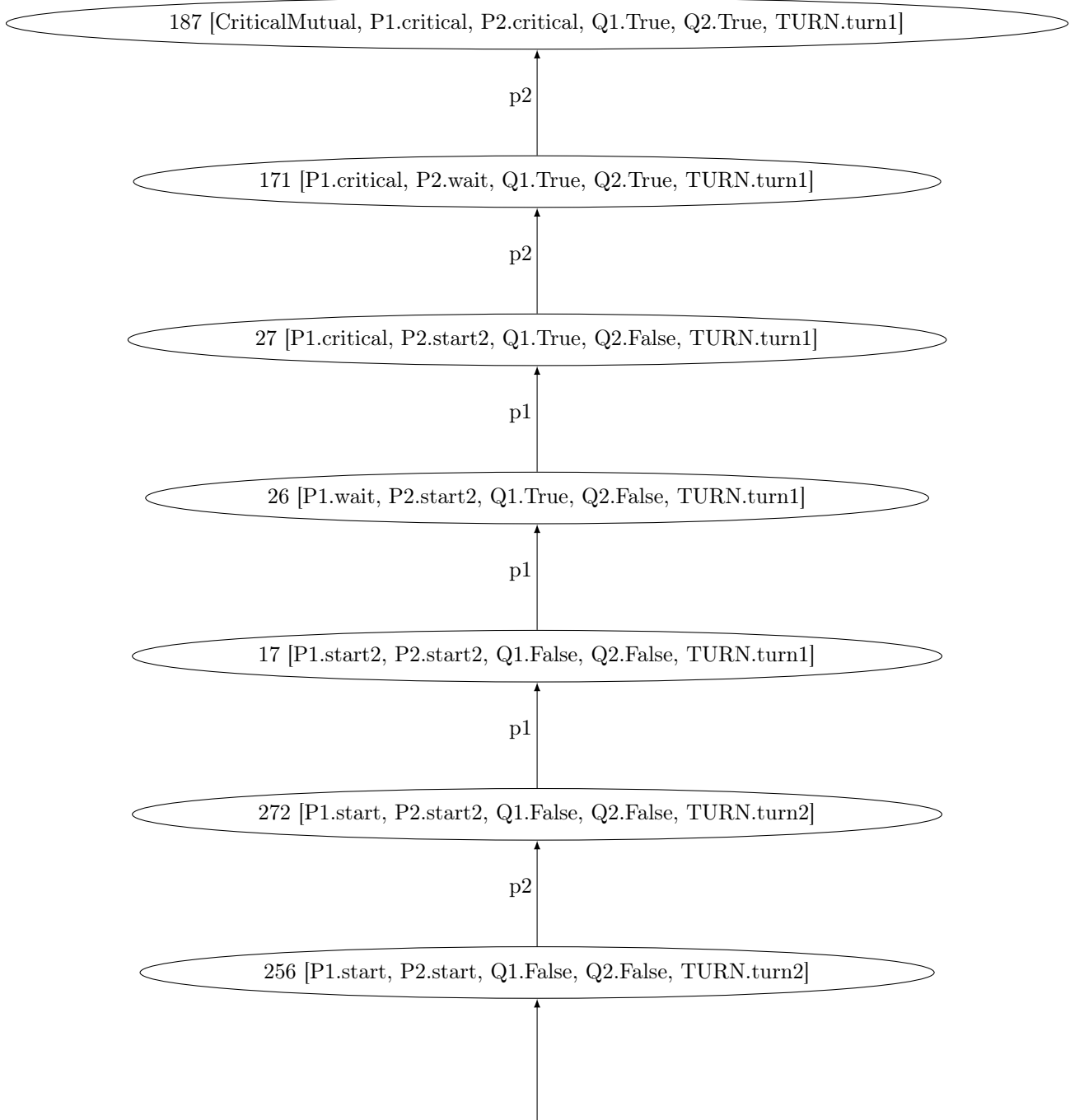
On définit ensuite P1 et P2, régis par le sous-système Process, Q1 et Q2 associés au sous-système SharedQ. Pour finir la variable TURN appartenant au sous-système sharedT

Le système s'écrit donc $systeme = \langle processP1, booleanQ1, processP2, booleanQ2, sharedTURN \rangle$

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 premières instructions a fortement impacté la qualité de notre algorithme. En effet celui-ci ne garantit 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 parmi plusieurs qui mène a une execution de la section critique.



Chapitre 3

Généralisation a N processus

3.1 Description générale

La dernière partie de notre analyse consiste a exploiter les résultats issus 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 celles-ci sont maintenant des tableaux (TURN[1..N-1] et Q[1..N] respectivement initialisés 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(  $\forall k \neq i, Q[k] < j$  ) or  $TURN[j] \neq i$ 
7   end;
8   Critical Section;
9   Q[i] := 0
```

3.2 Représentation du problème

Le sous-système Processus

$$|S| = 8$$

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

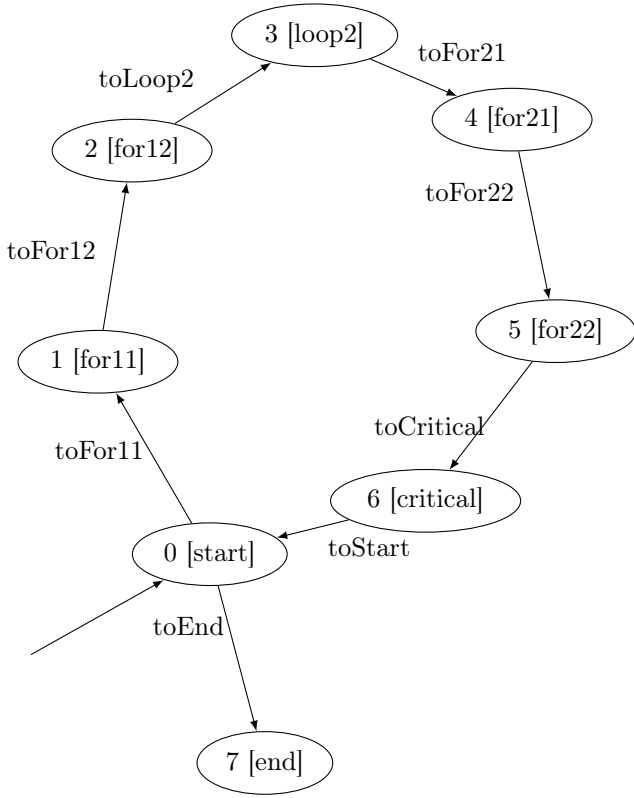
$$S_0 = \{0\}$$

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

$$\lambda = \left\{ \begin{array}{l} 0 \rightarrow start \\ 1 \rightarrow for11 \\ 2 \rightarrow for12 \\ 3 \rightarrow loop2 \\ 4 \rightarrow for21 \\ 5 \rightarrow for22 \\ 6 \rightarrow critical \\ 7 \rightarrow end \end{array} \right.$$

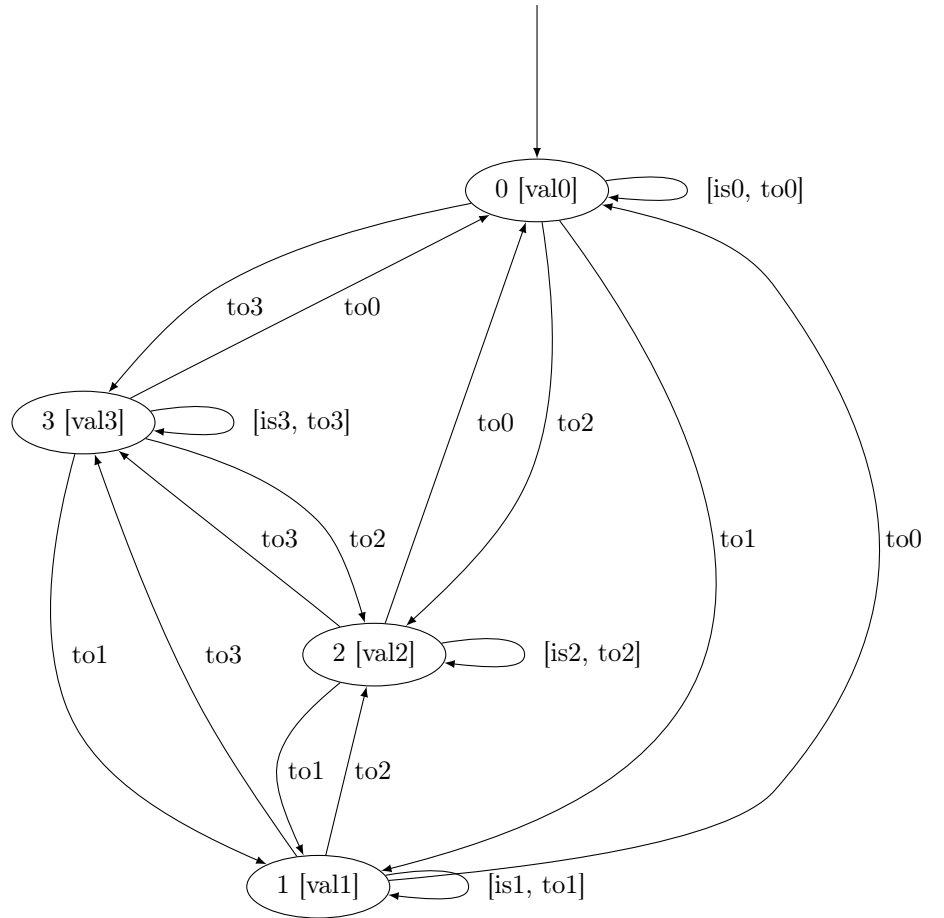
$$\rightarrow = \{toStart, toFor11, toFor12, toLoop2, toFor21, toFor22, toCritical, toEnd\}$$

$$Process = \langle S, S_0, \rightarrow, \lambda, AP \rangle$$



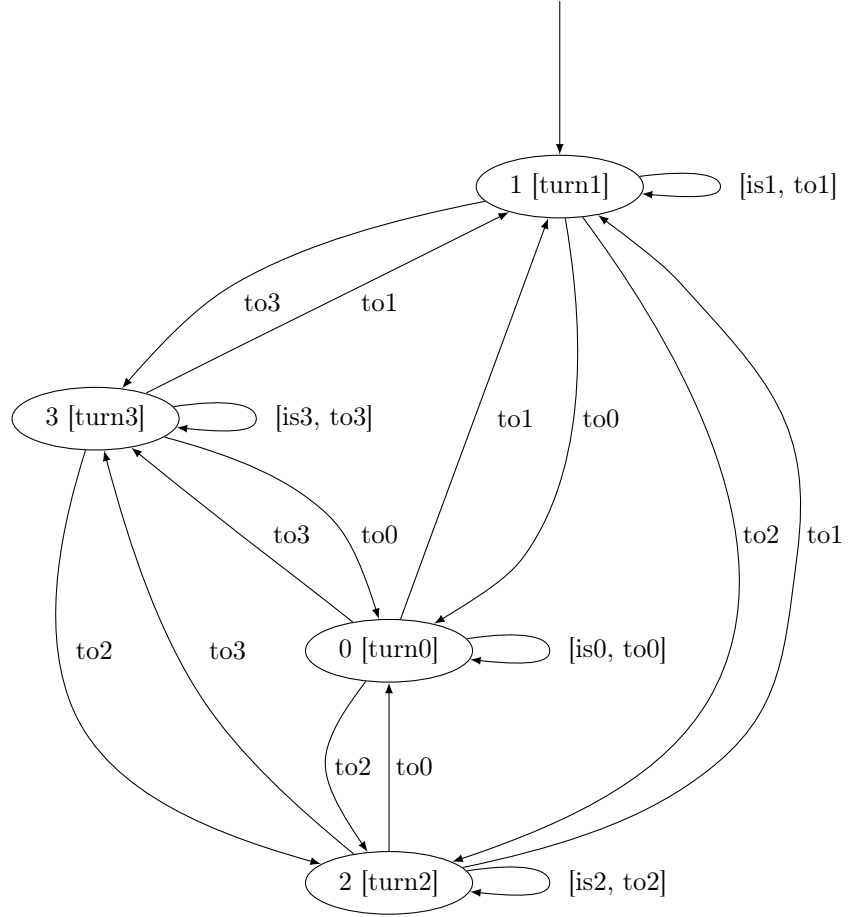
Le sous-système sharedQ (Q)

$|S'| = 4$
 $S' = \{0, 1, 2, 3\}$
 $S'_0 = \{0\}$
 $AP = \{val0, val1, val2, val3\}$
 $\lambda = \begin{cases} 0 \rightarrow val0 \\ 1 \rightarrow val1 \\ 2 \rightarrow val2 \\ 3 \rightarrow val3 \end{cases}$
 $\rightarrow = \{is0, is1, is2, is3, to0, to1, to2, to3\}$
 $Shared = \langle S', S'_0, \rightarrow, \lambda, AP \rangle$



Le sous-système sharedT (TURN)

$|S''| = 4$
 $S'' = \{0, 1, 2, 3\}$
 $S''_0 = \{0\}$
 $AP = \{turn0, turn1, turn2, turn3\}$
 $\lambda = \begin{cases} 0 \rightarrow turn0 \\ 1 \rightarrow turn1 \\ 2 \rightarrow turn2 \\ 3 \rightarrow turn3 \end{cases}$
 $\rightarrow = \{is0, is1, is2, is3, to0, to1, to2, to3\}$
 $Shared = \langle S'', S''_0, \rightarrow, \lambda, AP \rangle$



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 accessibles en lecture/écriture par tous les programmes. Le système s'écrit donc $\langle processP1, processP2, processP3, sharedQQ1, sharedQQ2, sharedQQ3, sharedTt1, sharedTt2 \rangle$

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 puisse pas être lu (le nombre d'états ne permet pas a un humain de deceler des dysfonctionnements) nous utilisons alors les formules CTL pour vérifier le modèle. D'un point de vue inter blocage et exclusion mutuelle les résultats sont encourageants car les réduits nous montrent que les processus ne s'inter-bloquent jamais ni n'exécutent la section critique en même temps. Par ailleurs l'équité n'est pas respectée.