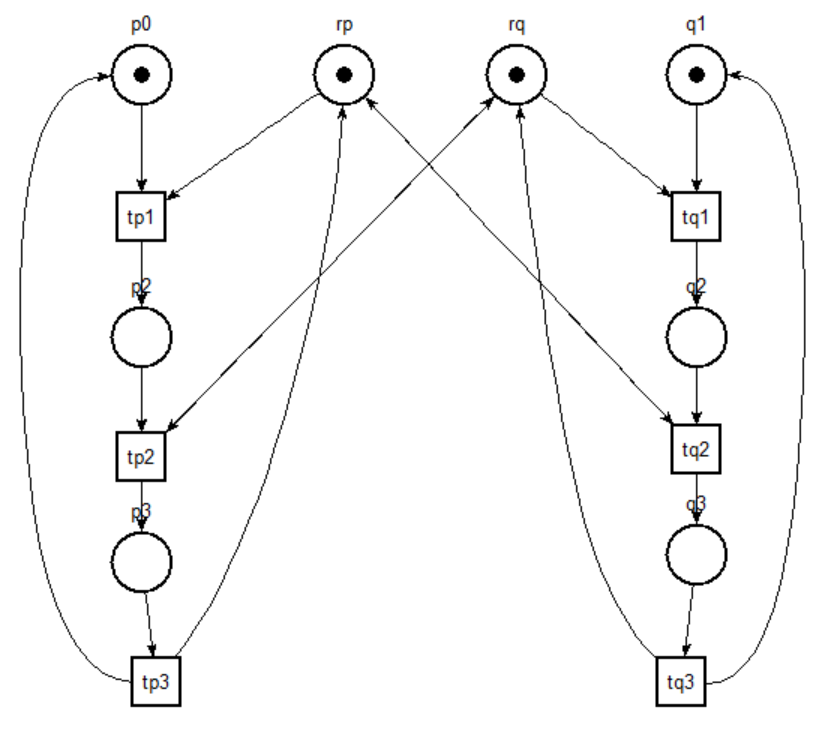
# TP2 SCV TINA ToolBox

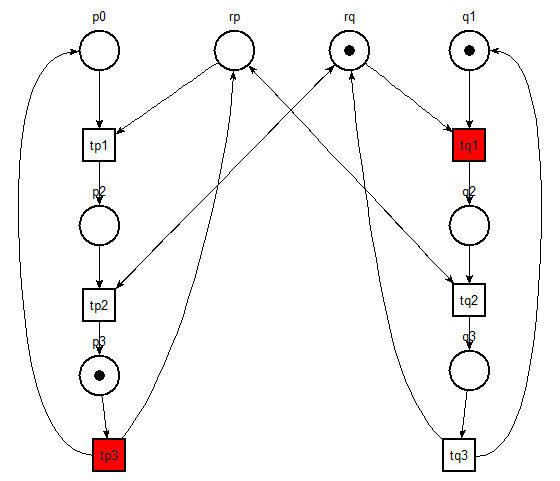
## Exercice 1 : Mutual Exclusion Algorithm

1)



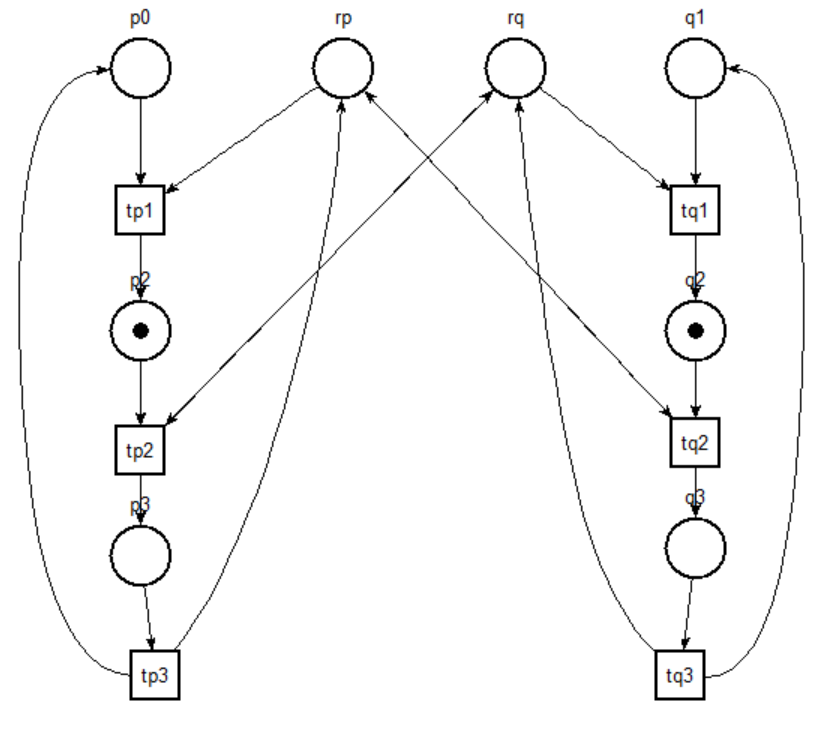
Exclusion mutuelle Petri Net

2)



Simulation étaoe par étape

3)



Situation de deadlock

On peut voir que dans ce cas de figure, les deux processus, on acquis une resource et chacun bloqué un mutex. Les deux mutex ayant été pris, aucun des deux processus ne peut plus relâcher sa resource.

En analysant le graphe de marquage, on peut voir que à l'état 4, il n'y a aucune transition possible.

state 4

props p2 q2

trans

5) selt TP2.1.4.ktz -f "- <> dead"

selt TP2.1.4.ktz -f "- <> dead"

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 8 states, 12 transitions

0.000s

FALSE

state 0: p0 q1 rp rq

-tp1 ... (preserving T)->

state 5: L.dead p2 q2

-L.deadlock ... (preserving dead)->

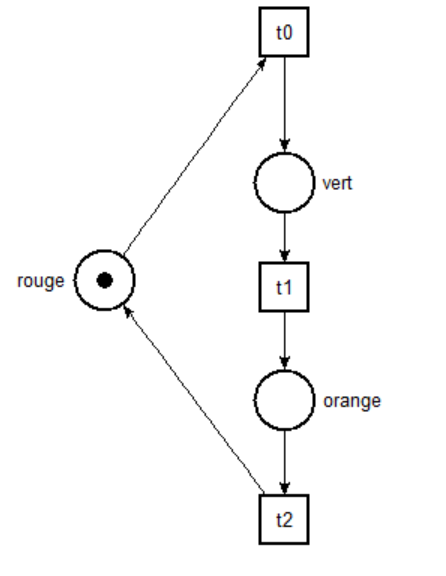
state 6: L.dead p2 q2

[accepting all]

0.016s

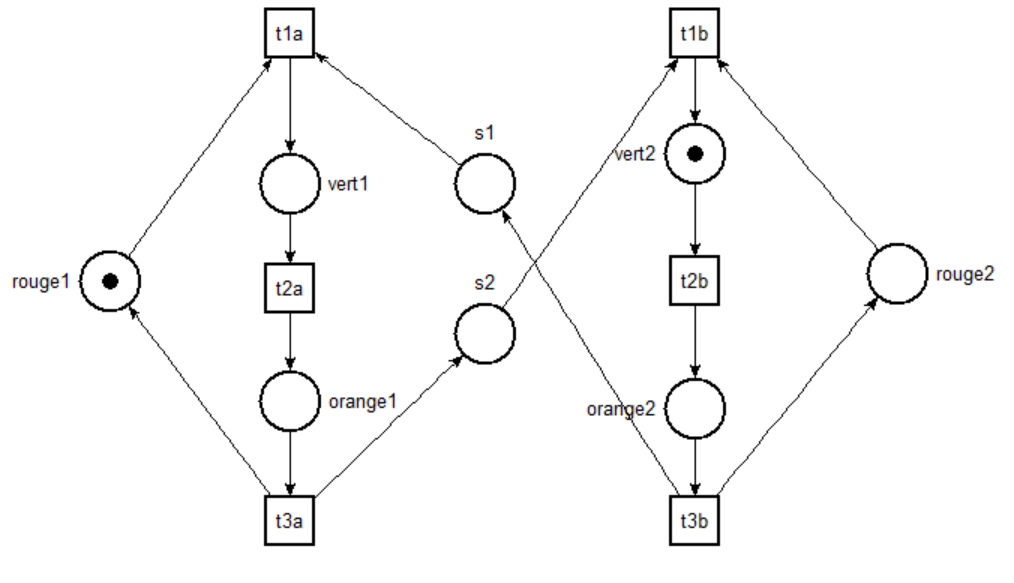
## Exercice 2 : Traffic light

1)



Petri Net pour un feu tricolore

2)



Petri Net pour deux feux tricolores

3)

state 0

props rouge1 vert2

trans t2b/1

state 1

props orange2 rouge1

trans t3b/2

state 2

props rouge1 rouge2 s1

trans t1a/3

state 3

props rouge2 vert1

trans t2a/4

state 4

props orange1 rouge2

trans t3a/5

4)

a) The two lights can never be both green at the same time.

**[] - (vert1 /\ vert2)**

selt TP2.2.3.ktz -f "[] - (vert1 /\ vert2)"

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 6 states, 6 transitions

0.000s

TRUE

0.016s

b) When a light is red, then it will become green in the future.

**[] (rouge1 => <> vert1) \/ [] (rouge2 => <> vert2)**

λ selt TP2.2.3.ktz -f "[] (rouge1 => <> vert1) \/ [] (rouge2 => <> vert2)"

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 6 states, 6 transitions

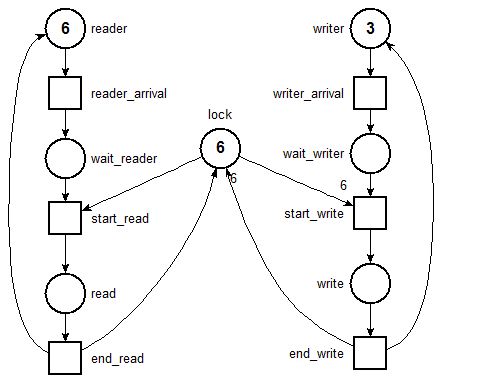
0.000s

TRUE

0.000s

## Exercice 3 : Readers Writers

1)



Readers Writers Petri Net

2)

state 0

props lock\*6 reader\*6 writer\*3

trans reader\_arrival/1 writer\_arrival/2

state 1

props lock\*6 reader\*5 wait\_reader writer\*3

trans reader\_arrival/3 start\_read/4 writer\_arrival/5

state 2

props lock\*6 reader\*6 wait\_writer writer\*2

trans reader\_arrival/5 start\_write/6 writer\_arrival/7

state 4

props lock\*5 read reader\*5 writer\*3

trans end\_read/0 reader\_arrival/9 writer\_arrival/11

...

Il y a 133 places et 16 arrêtes dans ce graphe.

3)

(a) Place lock is 5-bounded.

**[] (lock <= 5)**

selt petri.ktz -f "[] (lock <= 5)"

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 133 states, 410 transitions

0.000s

FALSE

state 0: lock\*6 reader\*6 writer\*3

-reader\_arrival ... (preserving T)->

state 72: lock\*6 reader\*6 wait\_writer\*3

-reader\_arrival ... (preserving - (5 >= lock))->

state 80: lock\*6 reader\*5 wait\_reader wait\_writer\*3

[accepting all]

0.000s

Cette expression est fausse car plusieurs états (0,72,80) contiennent plus de 5 jetons

(b) Reading and writing in the database are mutually exclusive.

**[] - (read /\ write)**

selt petri.ktz -f "[] - (read /\ write)"

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 133 states, 410 transitions

0.000s

TRUE

0.000s

(c) At most, 5 five reader can read simultaneously in the database.

**[] (read <= 5)**

selt petri.ktz -f "[] (read <= 5)"

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 133 states, 410 transitions

0.000s

FALSE

state 0: lock\*6 reader\*6 writer\*3

-reader\_arrival ... (preserving T)->

state 88: read\*6 wait\_writer\*3

-end\_read ... (preserving - (5 >= read))->

state 89: lock read\*5 reader wait\_writer\*3

[accepting all]

0.000s

Cette expression est fausse car on peut avoir à plusieurs reprises 6 lecteurs en même temps

(d) Any reader arrival will be followed a read.

**[] (reader\_arrival => <> read)**

selt petri.ktz -f "[] (reader\_arrival => <> read)"

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 133 states, 410 transitions

0.000s

FALSE

state 0: lock\*6 reader\*6 writer\*3

-writer\_arrival ... (preserving T)->

state 29: lock\*6 reader wait\_reader\*5 wait\_writer\*3

-reader\_arrival ... (preserving - read /\ reader\_arrival)->

\* [accepting] state 30: lock\*6 wait\_reader\*6 wait\_writer\*3

-start\_write ... (preserving - read)->

state 30: lock\*6 wait\_reader\*6 wait\_writer\*3

0.000s

Cette expression est fausse. En effet, au moment où un lecteur arrive, il peut y avoir pénurie, par exemple si un écrivain est en train de faire des modifications.

(e) Any reading start will be followed by a reading end.

**[] (start\_read => <> end\_read)**

selt petri.ktz -f "[] (start\_read => <> end\_read)"

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 133 states, 410 transitions

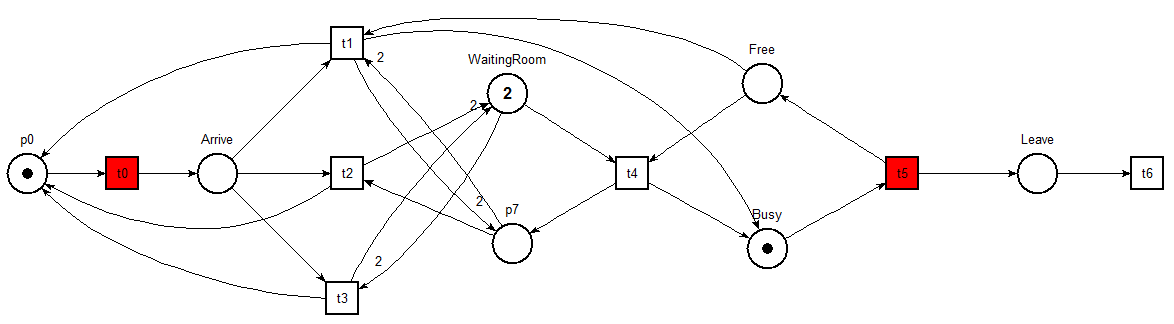
0.000s

TRUE

0.000s

## Exercice 4 : The barber problem

1)

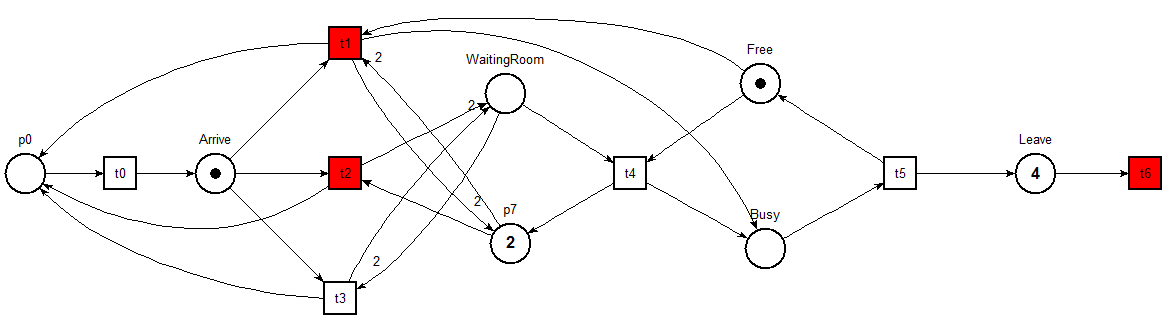


Ce Petri Net répond bien aux contraintes définies dans le sujet

2) Un Petri Net est borné si et seulement si son graphe des marquages est fini, c’est-à-dire si l’ensemble de ses états sont bornés. Dans notre cas, le graphe des marquages n'est pas fini et ne peut donc pas être représenté. De plus, en générant le graphe de couverture, on voit qu'il retourne "bounded: No", donc ce Petri Net n'est pas borné.

F:\EFREI\M1\S8\EFREI-S8-Specification-Design-and-Verification\TP2\E4\TP2.4.2.png

Ce Petri Net n’est pas borné



Sur la simulation ci-dessus, on peut voir que l’état « Leave » n’est pas borné car il possède déjà 4 jetons alors que n vaut 2. Sa valeur peut augmenter, il n’est donc pas borné, ce petri net n’est donc pas borné

3)

state 0

props Free p0 p7\*2

trans t0/1

state 1

props Arrive Free p7\*2

trans t1/2 t2/3

state 2

props Busy p0 p7\*2

trans t0/4 t5/5

state 3

props Free WaitingRoom p0 p7

trans t4/2 t0/6

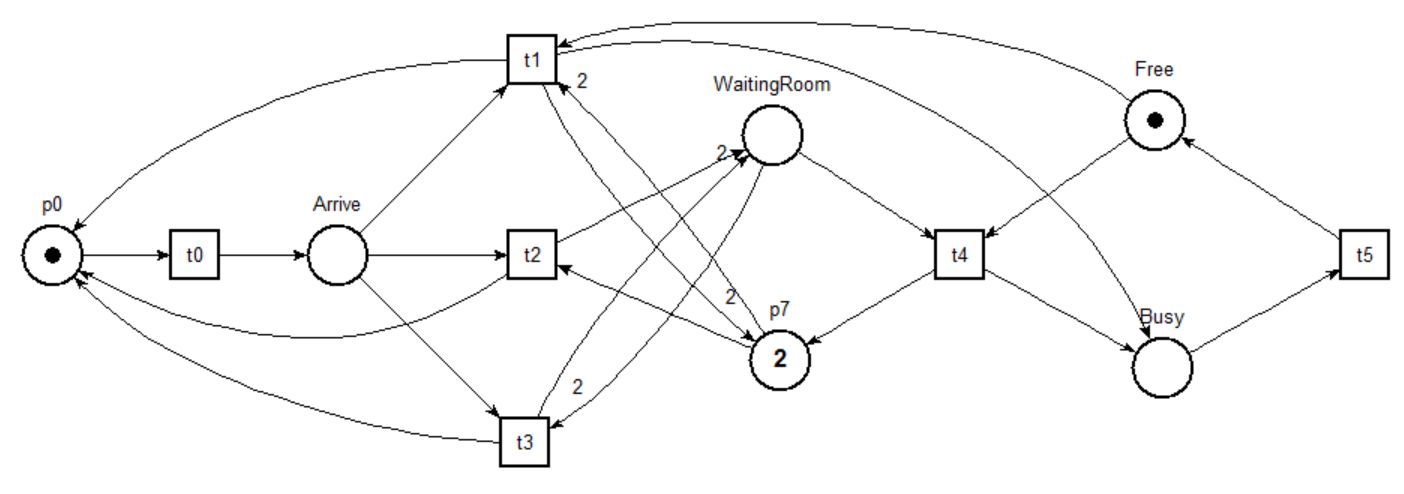
Le symbole w représente ici le nombre de clients dont s'est occupé le barbier

state 27

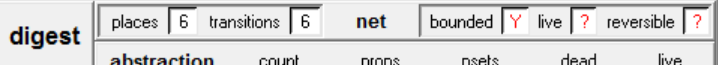
props Arrive Busy Leave\*w WaitingRoom\*2

trans t5/25 t3/26 t6/27

4)



Petri Net borné



Vérification avec le graphe de couverture

5)

state 0

props Free p0 p7\*2

trans t0/1

state 1

props Arrive Free p7\*2

trans t1/2 t2/3

state 2

props Busy p0 p7\*2

trans t5/0 t0/4

state 3

props Free WaitingRoom p0 p7

trans t4/2 t0/5

state 4

props Arrive Busy p7\*2

trans t5/1 t2/6

state 5

props Arrive Free WaitingRoom p7

trans t4/4 t2/7

state 6

props Busy WaitingRoom p0 p7

trans t5/3 t0/8

state 7

props Free WaitingRoom\*2 p0

trans t4/6 t0/9

state 8

props Arrive Busy WaitingRoom p7

trans t5/5 t2/10

state 9

props Arrive Free WaitingRoom\*2

trans t3/7 t4/8

state 10

props Busy WaitingRoom\*2 p0

trans t5/7 t0/11

state 11

props Arrive Busy WaitingRoom\*2

trans t5/9 t3/10

6)

a) Si le barbier est libre et la salle d'attente vide, il peut être servi immédiatement

**<>((Arrive /\ p7\*2 /\ Free) => <>(t1))**

selt TP2.4.5.ktz -f "<>((Arrive /\ p7\*2 /\ Free) => <>(t1))" -p

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 12 states, 23 transitions

0.000s

TRUE

0.016s

b) Si le barbier est occupé, il n'accepte pas de client avant d'être à nouveau libre

**[](Busy => <>((-Free) U Busy))**

selt TP2.4.5.ktz -f "[](Busy => <>((-Free) U Busy))"

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 12 states, 23 transitions

0.000s

TRUE

0.016s

selt TP2.4.5.ktz -f "[](Busy <= 1)

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 12 states, 23 transitions

0.000s

TRUE

0.000s

4) Si la salle d'attente est pleine, le client part

**[]((-p7) => <>p0)**

selt TP2.4.5.ktz -f "[]((-p7) => <>p0)"

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS

ktz loaded, 12 states, 23 transitions

0.000s

TRUE

0.016s