Auteur : Thibaut Seys

Date : 19/02/2018

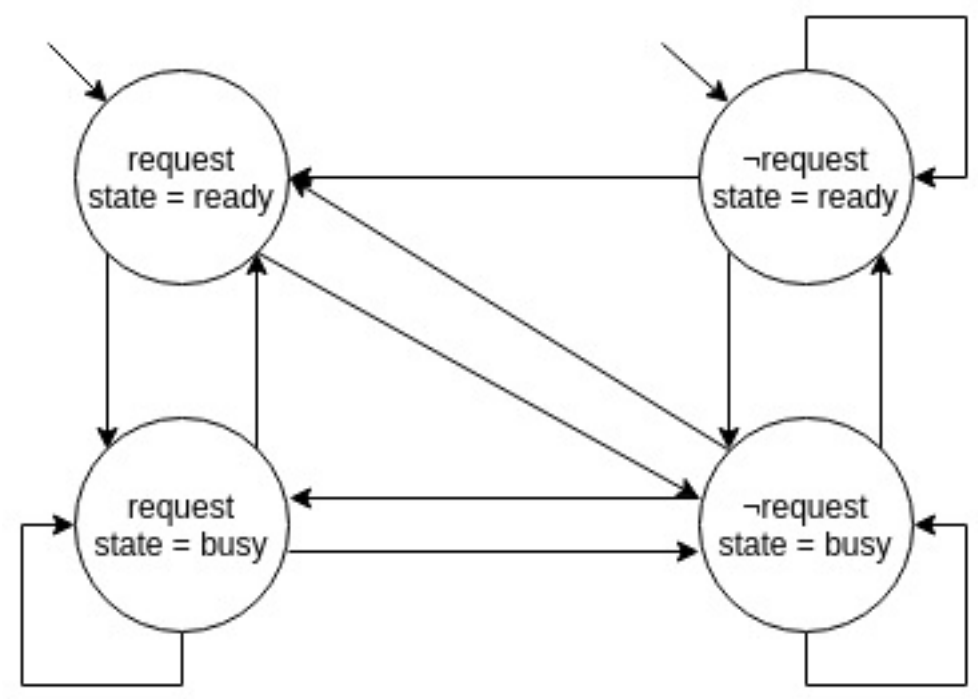
Lien Github : https://github.com/SeysT/TP-NUSMV

TP-NUSMV

Exercice 1

Question 1 :

Voici l'automate décrit par le modèle SMV :



La propriété de logique temporelle donnée dans le modèle SMV `SPEC AG(request -> AF state = busy)` signifie que quelque soit le chemin que l'on emprunte, lorsque que la variable request sera à TRUE, alors state vaudra busy au bout d'un moment, et ce quelque soit le chemin emprunté.

Question 2 :

Voici la trace obtenue lorsque l'on utilise NuSMV sur le fichier `exercice1/exercice1-1.smv`, qui correspond au modèle SMV donné en énoncé.

```
NuSMV exercice1/exercice1-1.smv

-- specification AG (request -> AF state = busy)  is true
```

Question 3

Comme nouvelles formules CTL, on peut proposer `CTLSPEC AG(request)` et `CTLSPEC AG(state = busy | state = ready)`.

La première est bien évidemment fausse puisqu'elle vérifie que request est vrai tout le temps pour tout les chemins possibles. La secondes est vraie puisqu'elle vérifie que state est soit à busy soit à ready tout le temps et quelque soit le chemin emprunté. On peut vérifier ces affirmations en utilisant NuSMV sur le fichier `exercice2/exercice1-2.smv` qui est le modèle donné par l'énoncé avec les formules CTL utilisées ci-dessus.

```
NuSMV exercice1/exercice1-2.smv

-- specification AG request  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    request = FALSE
    state = ready
-- specification AG (state = busy | state = ready)  is true
```

Exercice 2

Question 1 :

Les trois modèles SMV modélisant le problème sont présents dans le dossier `exercice2`.

Question 2 :

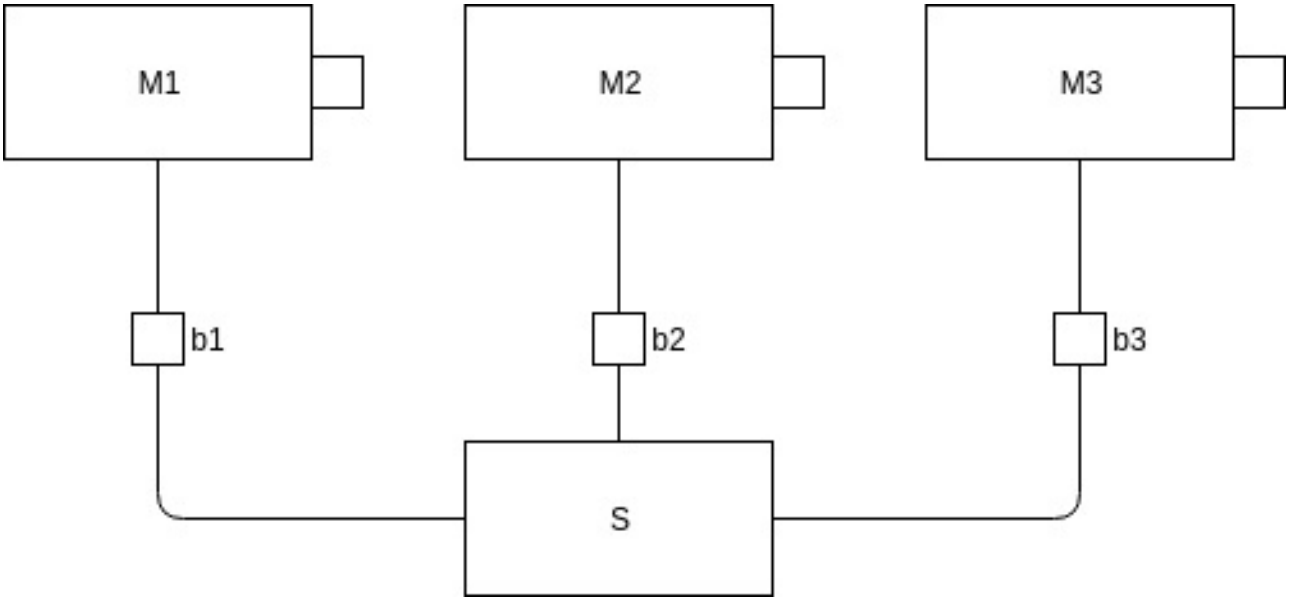
En utilisant NuSMV, on obtient bien que le résultat des formules CTL et LTL ne dépendent pas de l'implémentation choisie. Voici le résultat obtenu sur la première implémentation contenue dans le fichier `exercice2/exercice2-1.smv`.

```
NuSMV exercice2/exercice2-1.smv

-- specification  G a  is false
-- specification (a U b)  is false
-- specification (a U ( X (a & !b)))  is false
-- specification ( X !b &  G (!a | !b))  is false
-- specification ( X (a & b) &  F (!a & !b))  is false
-- specification EF (EG a)  is true
-- specification EF (EG b)  is true
-- specification  F ( G a |  G b)  is true
```

Exercice 3

Question 1 :

Voici le schéma du système :



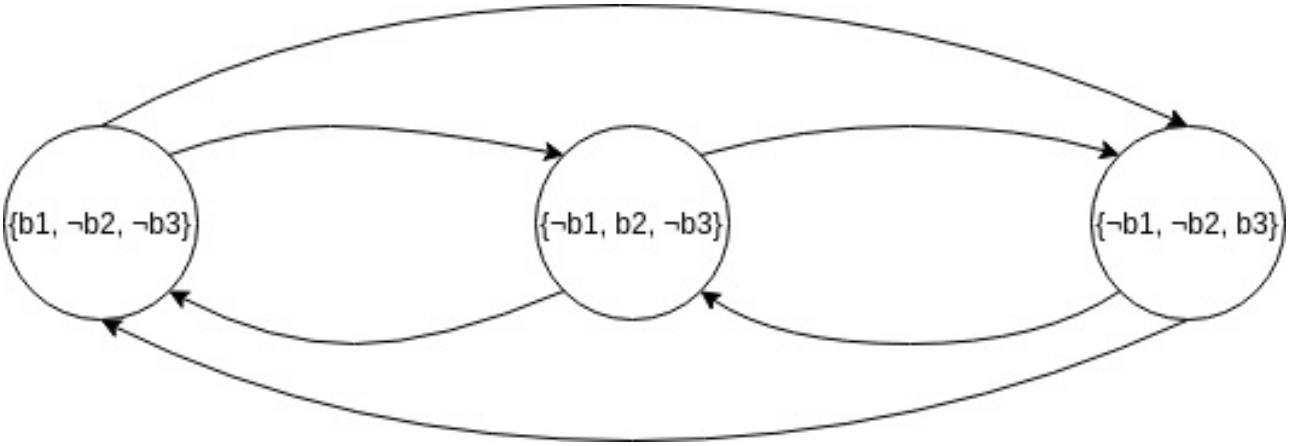
Question 2 :

On note bi lorsque l'interrupteur i est fermé et ¬bi lorsque celui-ci est ouvert. On peut alors exprimer les propriétés de l'énoncé de la manière suivante :

- Pas de court-circuit : `LTLSPEC G((b1 & !b2 & !b3) | (b2 & !b1 & !b3) | (b3 & !b1 & !b2))`
- Continuité de l'alimentation : `LTLSPEC G(b1 | b2 | b3)`
- Changement de batterie d'un état à l'état suivant : `LTLSPEC G((b1 -> X !b1) & (b2 -> X !b2) & (b3 -> X !b3))`

Question 3 :

Voici un système de transition qui respecte les propriétés énoncées ci-dessus :



Ce système a été implémenté dans le fichier `exercice3/exercice3.smv` et lorsque l'on éxécute NuSMV, on obtient bien que les 3 propriétés sont respéctées :

```
NuSMV exercice3/exercice3.smv

-- specification  G ((((b1 & !b2) & !b3) | ((b2 & !b1) & !b3)) | ((b3 & !b1) &
!b2))  is true
-- specification  G ((b1 | b2) | b3)  is true
-- specification  G (((b1 ->  X !b1) & (b2 ->  X !b2)) & (b3 ->  X !b3))  is
true
```

Exercice 4

Les propriétés LTL de l'énoncé peuvent être traduites de la manière suivante :

1. Il y a toujours p : `LTLSPEC G(p)`
2. Il y a p une infinité de fois : `LTLSPEC G(F(p))`
3. Il n'y a jamais p et q simultanément : `LTLSPEC G(!(p & q))`
4. Après chaque occurrence de p il y a au moins une occurrence de q : `LTLSPEC G(p -> F(q))`
5. S'il y a une infinité de p1 et une infinité de p2, alors toute occurrence de q1 est suivie d'une occurrence de q2 : `LTLSPEC G(F(p1)) & G(F(p2)) -> G(q1 -> F(q2))`
6. Avant chaque occurrence de p il y a au moins une occurrence de q : `LTLSPEC !q U (p & !q)`
7. Entre chaque pair d'occurrence de p il y a au moins une occurrence de q : `LTLSPEC G((p & X F p) -> X (! p U q))`

Pour vérifier ces formules, on a choisi deux modèles. Le premier `exercice4/exercice4-1.smv` valide les formules 2, 3, 4, 6, 7. Il s'agit d'un modèle où p = ¬q à chaque étape et où p passe de TRUE à FALSE à chaque transition. Les variables p1, p2, q1 et q2 ne sont pas assignées ici. Le second modèle `exercice4/exercice4-2.smv` valide les formules 1 et 5. Il s'agit d'un modèle où p est initialisée à TRUE et ne change pas. Les variables p1, p2 et q1 sont initialisées à TRUE, q2 à FALSE et ces quatre variables changent d'état à chaque transition. On ne s'occuppe pas ici de la variable q. En utilisant NuSMV sur ces deux modèles ont obtient :

```
NuSMV exercice4/exercice4-1.smv

-- specification  G ( F p)  is true
-- specification  G !(p & q)  is true
-- specification  G (p ->  F q)  is true
-- specification (!q U (p & !q))  is true
-- specification  G ((p &  X ( F p)) ->  X (!p U q))  is true

============================================================

NuSMV exercice4/exercice4-2.smv

-- specification  G p  is true
-- specification (( G ( F p1) &  G ( F p2)) ->  G (q1 ->  F q2))  is true
```

Exercice 5

Pour modéliser le problème, nous allons utiliser une variable booléenne pour la feuille, le lombric, le millepatte, et la sauterelle. Si la variable est à FALSE alors l'objet et à gauche de la rivière, sinon il est à droite. Pour modéliser les transitions, nous allons lister les configurations accessibles à partir d'un état quelqueconque. Il y a donc 16 états à parcourir (pour des raisons de lisibilité, les variables seront notées f, l, m, s dans le tableau suivant) :

Configuration de départ	**Configurations accessibles**
{f, l, m, s}	{¬f, ¬l, m, s} ; {¬f, l, ¬m, s} ; {¬f, l, m, ¬s} ; {¬f, l, ¬m, ¬s}
{f, l, m, ¬s}	{¬f, ¬l, m, ¬s} ; {¬f, l, ¬m, ¬s}
{f, l, ¬m, s}	{¬f, ¬l, ¬m, s} ; {¬f, l, ¬m, ¬s}
{f, l, ¬m, ¬s}	{¬f, ¬l, ¬m, ¬s}
{f, ¬l, m, s}	{¬f, ¬l, ¬m, s} ; {¬f, ¬l, m, ¬s} ; {¬f, l, ¬m, ¬s}
{f, ¬l, ¬m, ¬s}	Configuration impossible, la feuille ne peut pas avoir traversé seule
{f, ¬l, ¬m, s}	{¬f, ¬l, ¬m, ¬s}
{f, ¬l, m, ¬s}	{¬f, ¬l, ¬m, ¬s}
{¬f, l, m, s}	Configuration impossible, la feuille ne peut pas avoir traversé seule
{¬f, l, m, ¬s}	{f, l, m, s}
{¬f, l, ¬m, s}	{f, l, m, s}
{¬f, l, ¬m, ¬s}	{f, l, m, ¬s} ; {f, l, ¬m, s} ; {f, l, ¬m, ¬s}
{¬f, ¬l, m, s}	{f, l, m, s}
{¬f, ¬l, ¬m, ¬s}	{f, l, ¬m, ¬s} ; {f, ¬l, m, ¬s} ; {f, ¬l, ¬m, s} ; {f, ¬l, m, s}
{¬f, ¬l, m, ¬s}	{f, l, m, ¬s} ; {f, ¬l, m, s}
{¬f, ¬l, ¬m, s}	{f, l, ¬m, s} ; {f, ¬l, m, s}

Pour notre modèle nous allons exprimer trois propriétés différentes :

- Le lombric, le millepatte et la sauterelle arrivent à traverser : `SPEC EF(lombric & millepatte & sauterelle)`.
- La feuille ne peut être toute seule à droite : `AG !(feuille & !lombric & !millepatte & !sauterelle)`.
- La feuille ne peut être toute seule à gauche : `AG !(!feuille & lombric & millepatte & sauterelle)`.

L'implémentation du problème a été réalisé dans le fichier `exercice5/exercice5.smv`. En utilisant NuSMV dessus on obtient :

```
NuSMV exercice5/exercice5.smv

-- specification EF ((lombric & millepatte) & sauterelle)  is true
-- specification AG !(((feuille & !lombric) & !millepatte) & !sauterelle)  is
true
-- specification AG !(((!feuille & lombric) & millepatte) & sauterelle)  is
true
```

Exercice 6

Parmis les formules données par l'énoncé, on a les équivalence suivante :

- `AF phi | AF psi` et `AF (phi | psi)`
- `AF ¬psi` et `¬EG psi`

- `A[psi U A[phi U ro]]` et `A[A[psi U phi] U ro]`
- `TRUE` et `AG phi -> EG phi`

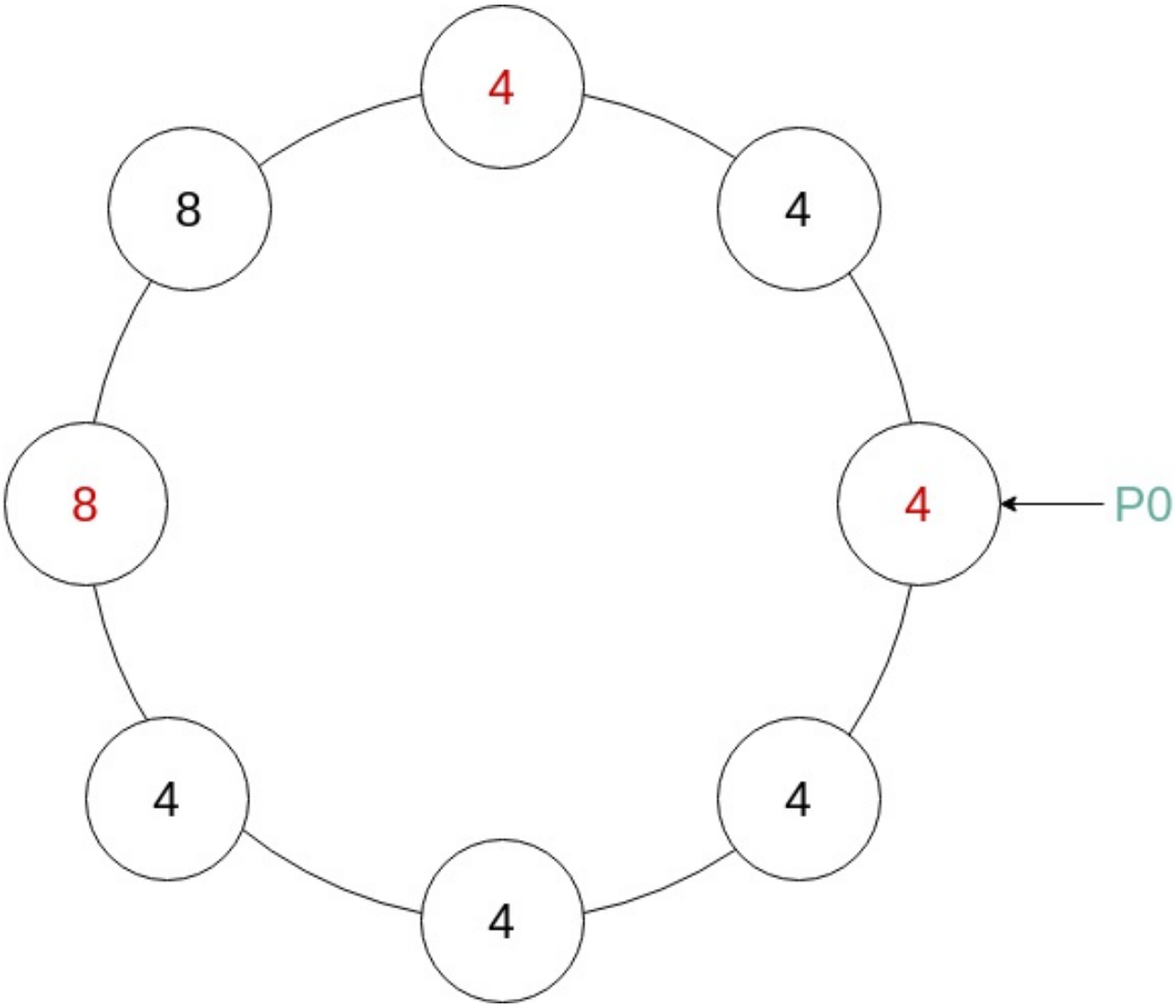
On peut attester de cela en utilisant NuSMV sur le fichier `exercice6/exercice6.smv` qui contient trois variables booléennes phi, psi et ro, et qui regarde l'équivalence entre les formules de l'énoncé :

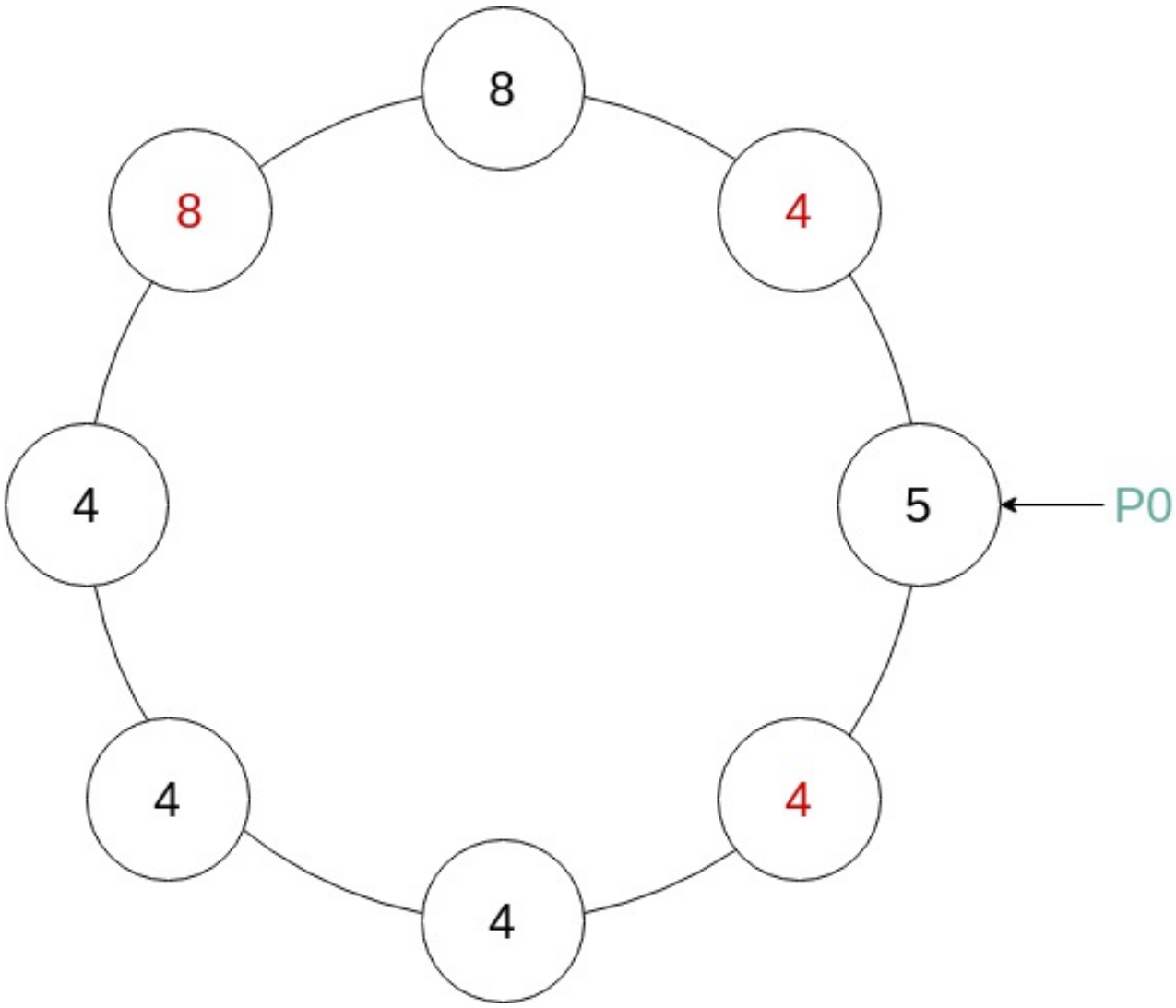
```
NuSMV exercice6/exercice6.smv

-- specification (EF phi <-> EG phi)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    phi = FALSE
    psi = FALSE
    ro = FALSE
  -> State: 1.2 <-
    phi = TRUE
-- specification (AF !psi <-> !(EG psi))  is true
-- specification (EF !psi <-> !(AF psi))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 2.1 <-
    phi = FALSE
    psi = TRUE
    ro = FALSE
  -> State: 2.2 <-
    psi = FALSE
-- specification (TRUE <-> (AG phi -> EG phi))  is true
-- specification (TRUE <-> (EG phi -> AG phi))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 3.1 <-
    phi = TRUE
    psi = FALSE
    ro = FALSE
  -> State: 3.2 <-
-- specification ((EF phi | EF psi) <-> EG (phi | psi))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 4.1 <-
    phi = FALSE
    psi = FALSE
    ro = FALSE
  -> State: 4.2 <-
    phi = TRUE
-- specification ((AF phi | AF psi) <-> AF (phi | psi))  is true
-- specification (A [ psi U A [ phi U ro ]  ]  <-> A [ A [ psi U phi ]  U ro ]
)  is true
```

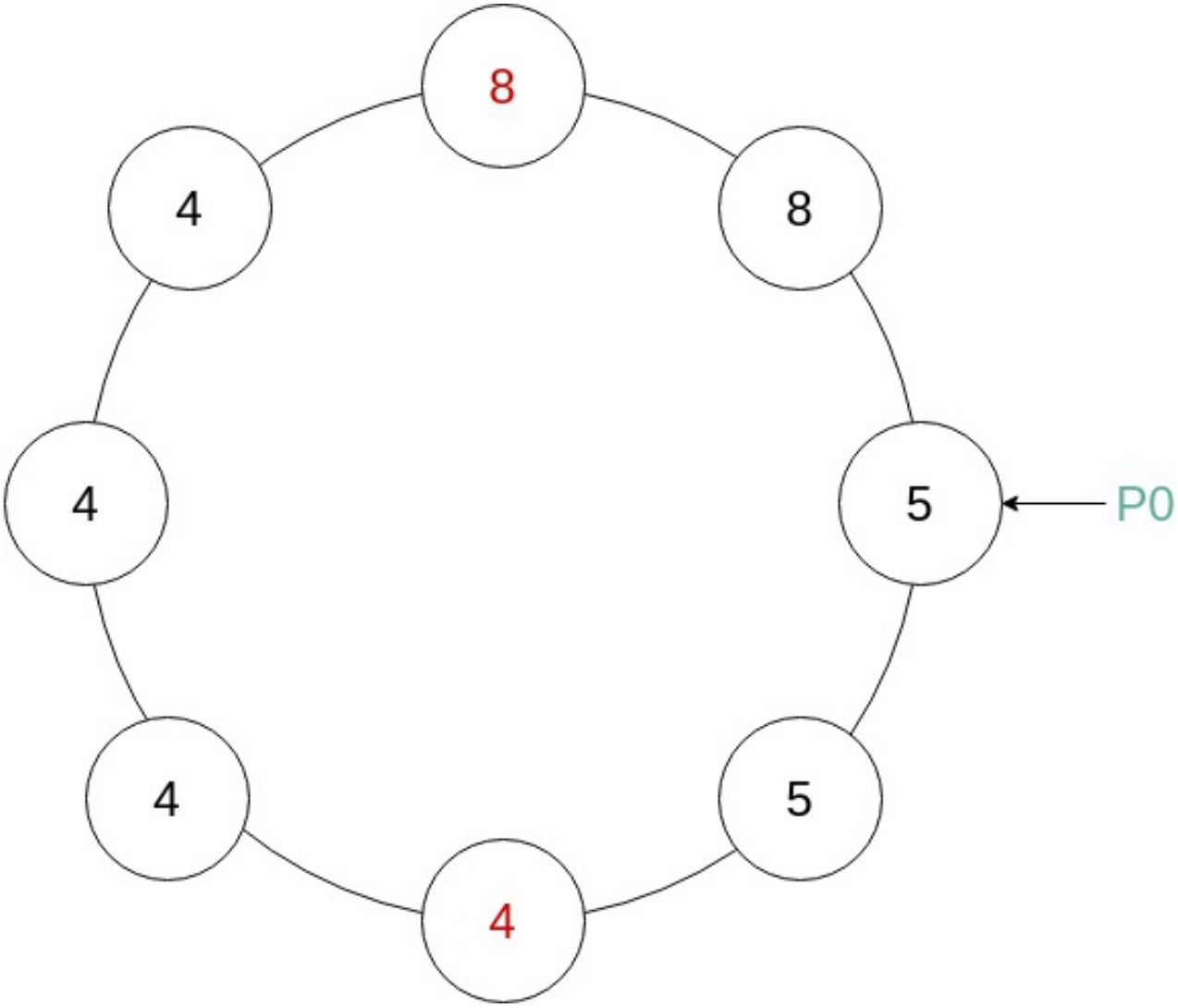
Exercice 7

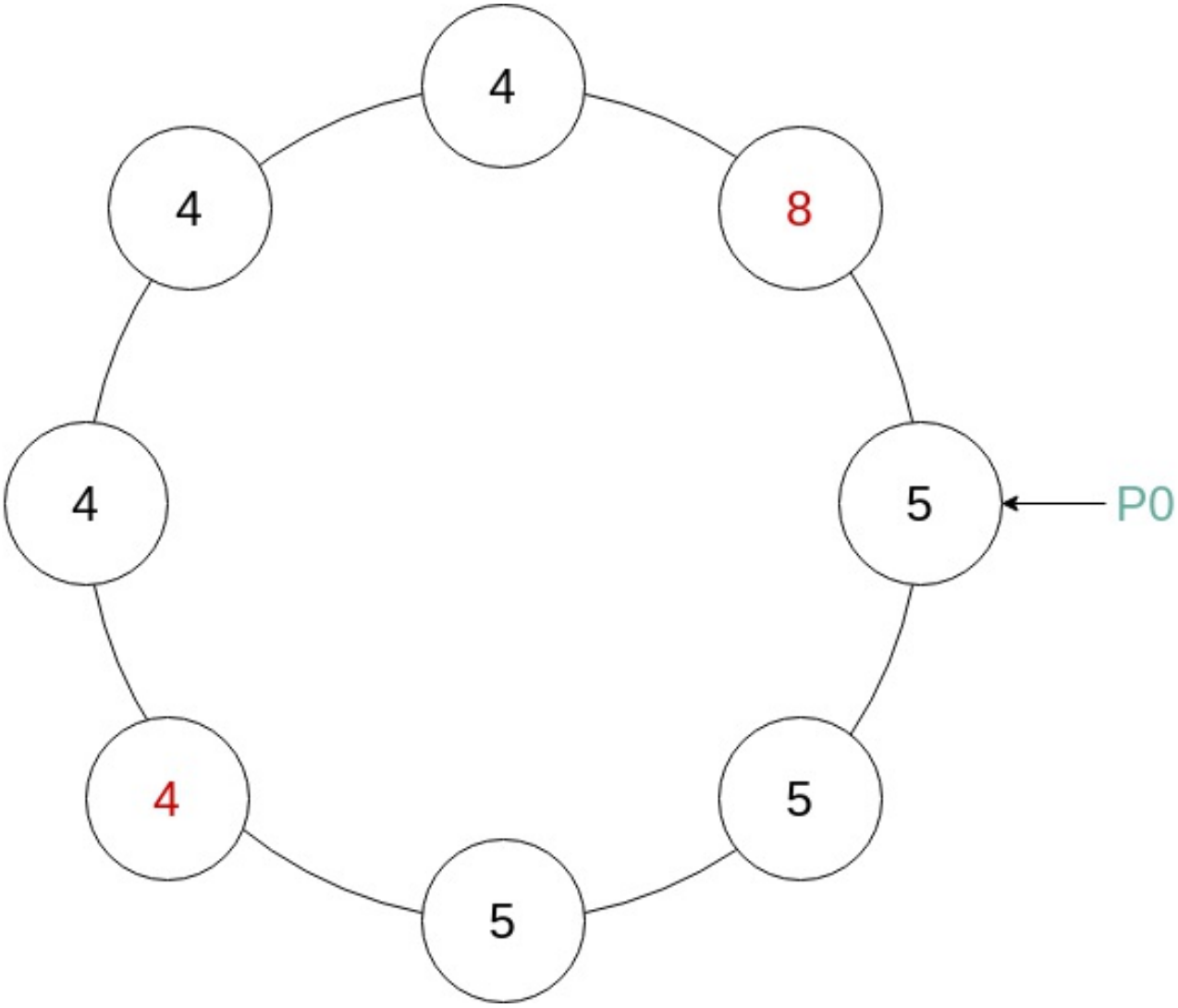
Question 1 :

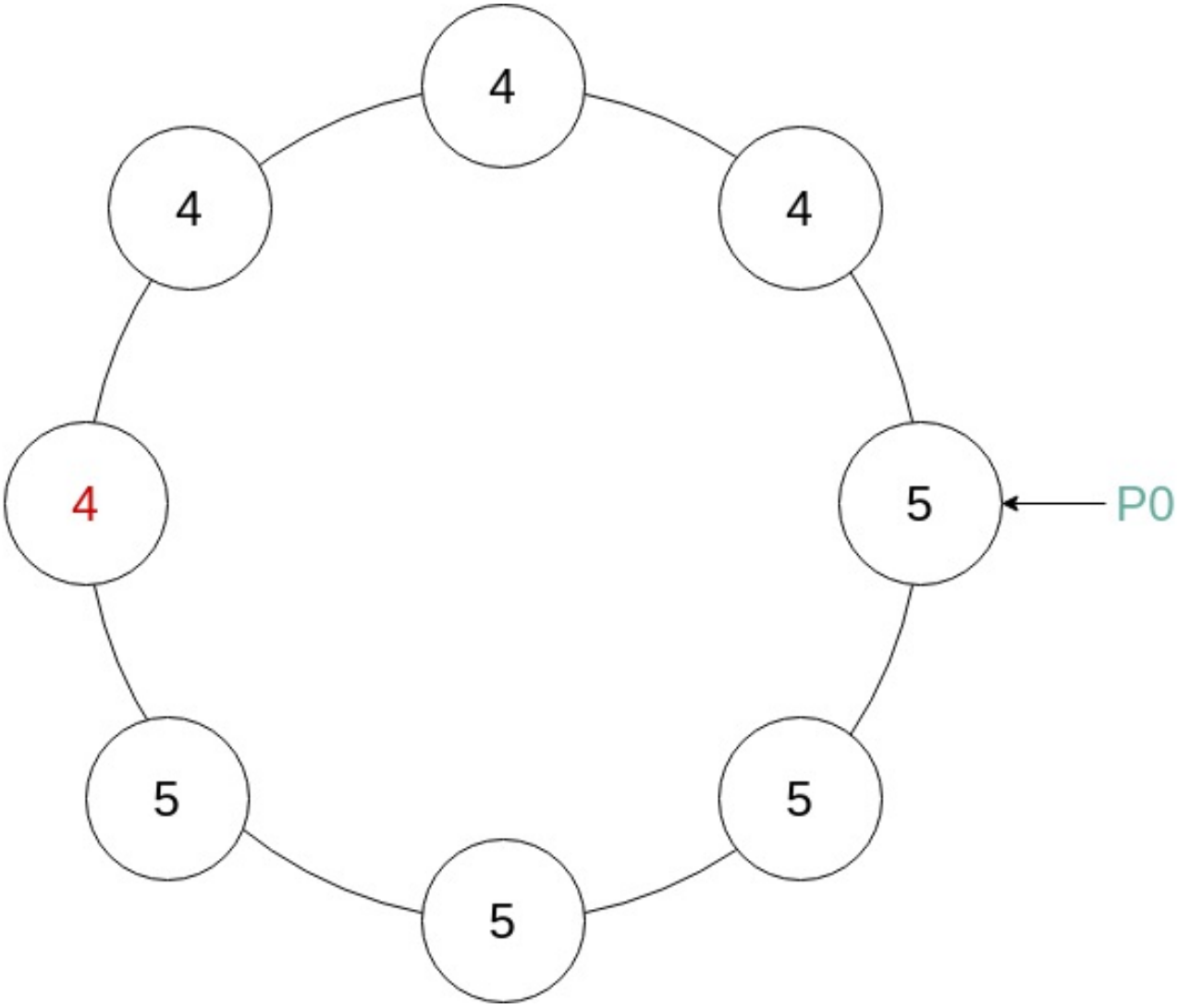
La première étape de l'algorithme va être de se stabiliser pour arriver à un état à partir duquel seul un processus entre en état critique à la fois. Voici sur un exemple les étapes à réaliser pour arriver à cet état :



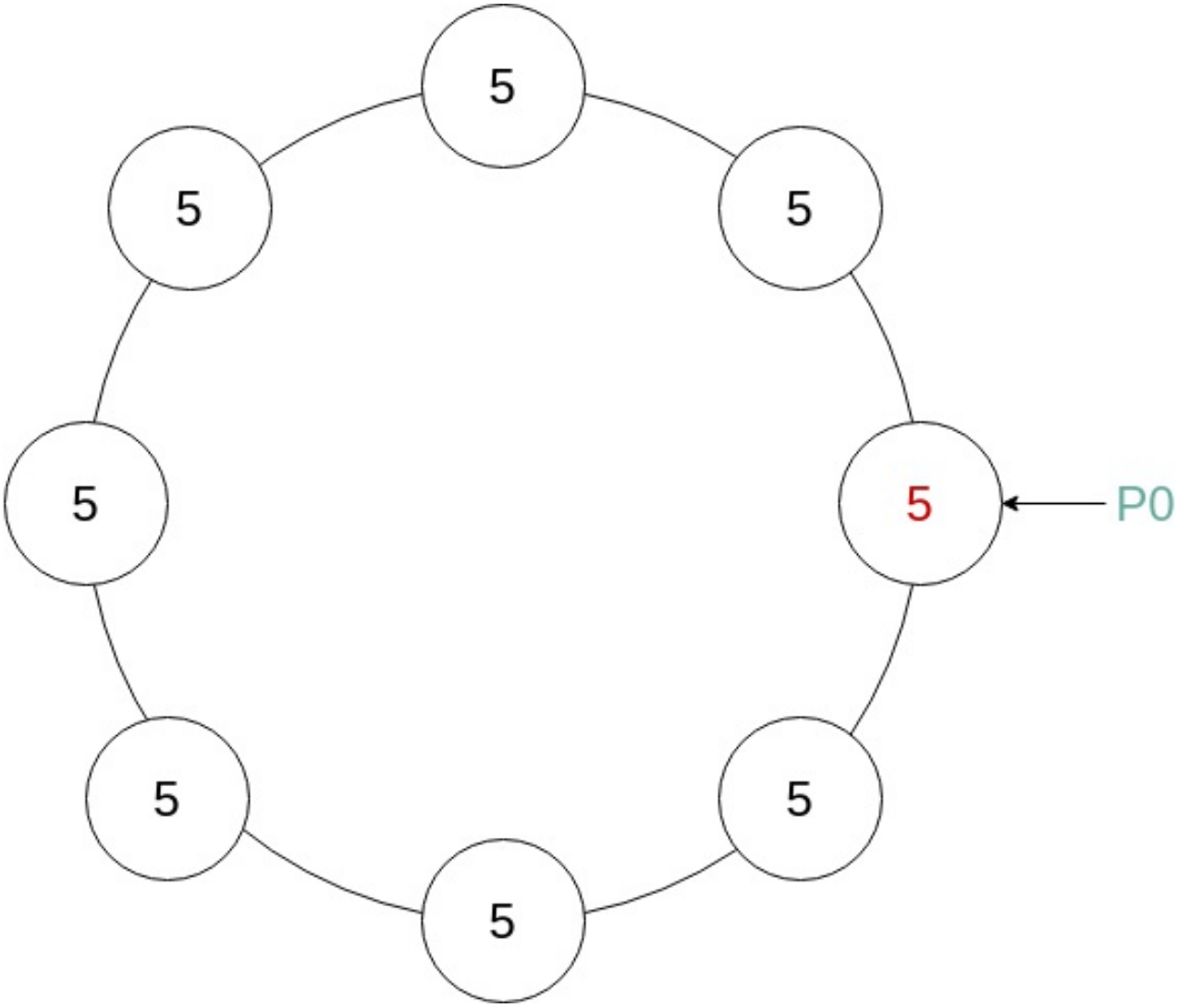

8
8
4
4
5
P0
4
4
4

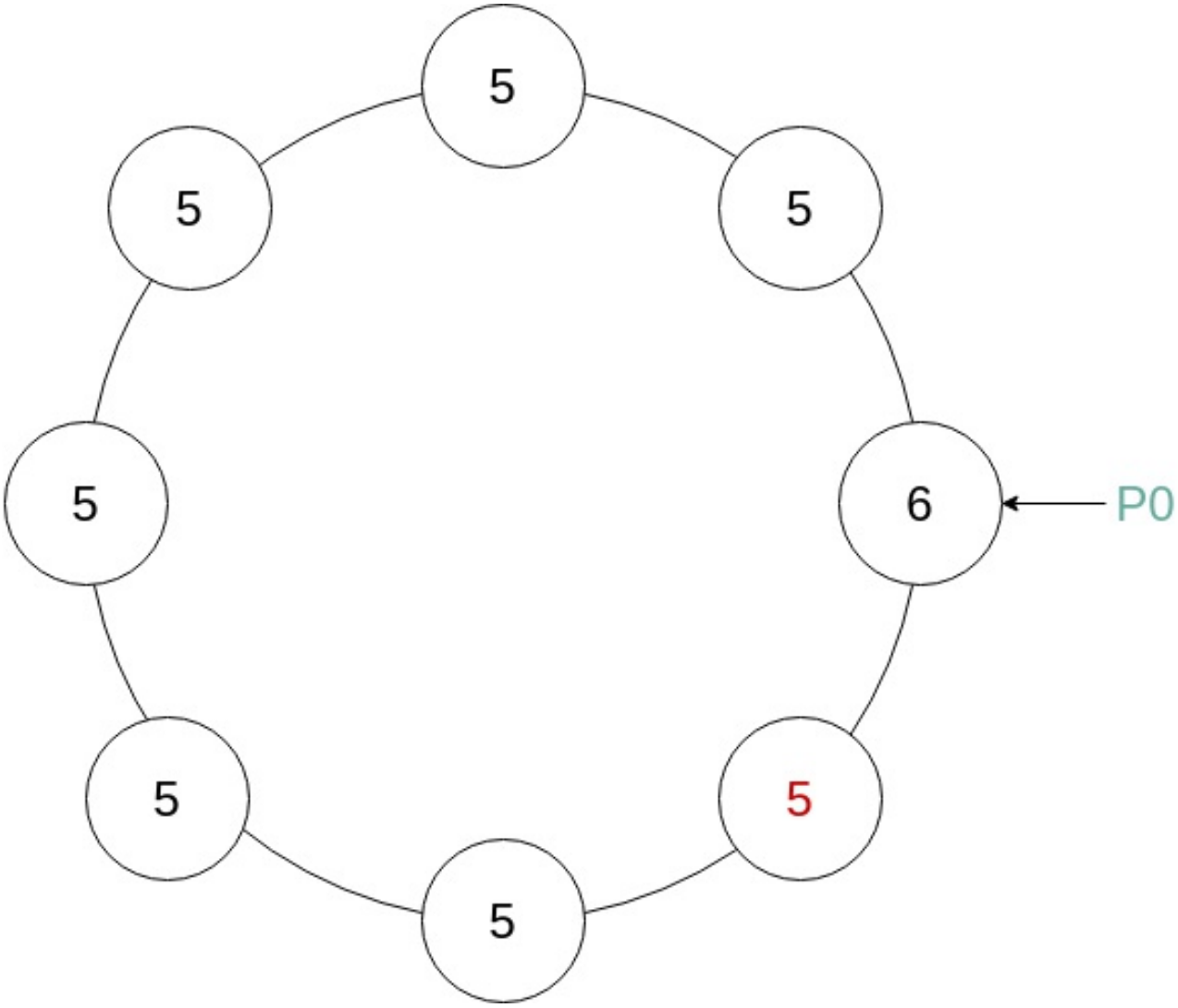

8
4
8
4
5
P0
4
5
4


4
4
8
4
5
P0
4
5
5



A partir de cette étape, seul un processus va entrer dans son état critique à la fois. On arrive alors ici à une sorte de cycle où tous les processus vont mettre leur drapeau à égalité. Ensuite le processus 0 va incrémenter son drapeau, et le cycle va recommencer à l'infini. Ceci est illustré sur les figures suivantes.


5
5
5
5
5
5
5
5
P0



Question 2 :

D'après l'énoncé et la description du protocole faite ci-dessus, nous en arrivons au spécifications suivantes :

- Chaque processus rentre en état critique infiniment souvent.
- A partir d'un moment donné, au moins un des processus rentre en état critique.
- A partir d'un moment, seul un processus à la fois entre en état critique.
- Toutes les valeurs des drapeaux sont égales infiniment souvent.

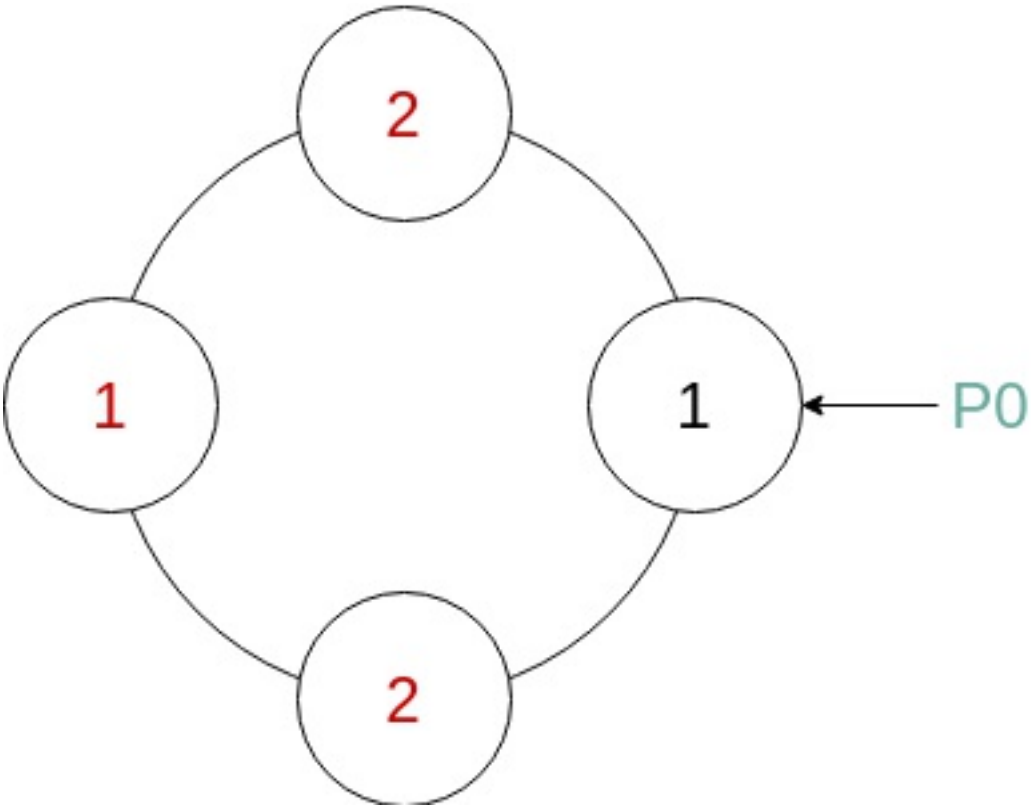
Question 3 :

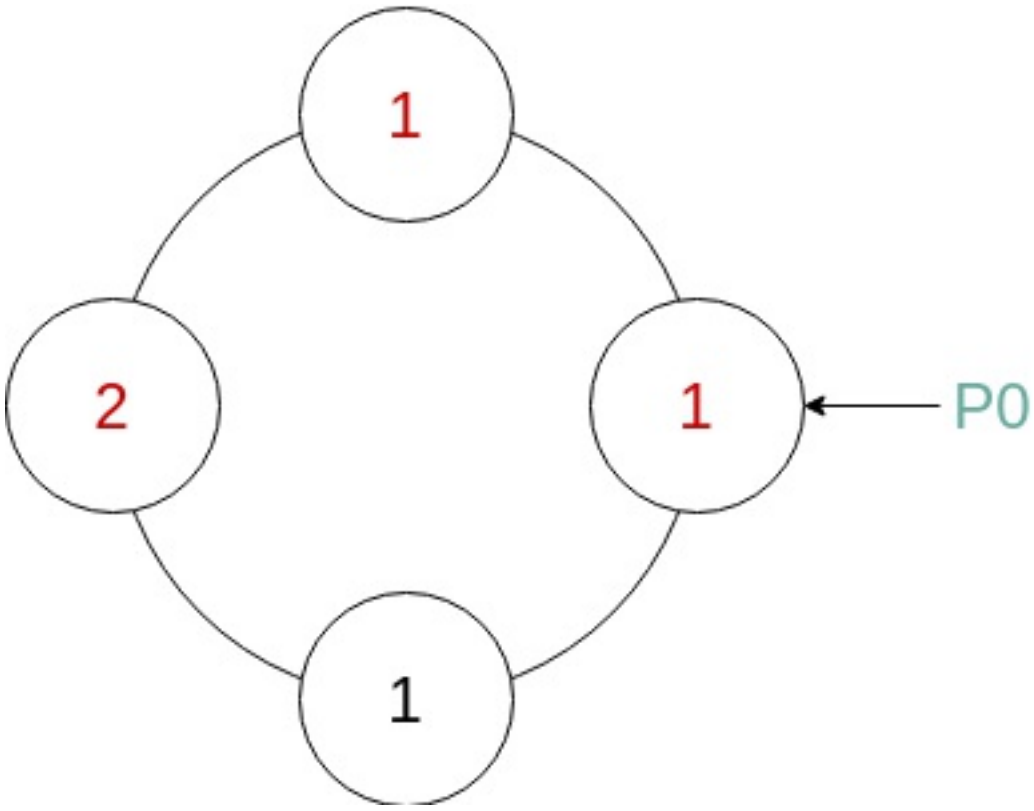
Il a été choisi d'implémenter l'algorithme de Dijkstra avec des valeurs de K = 7 et N = 3 dans le fichier `exercice7/exercice7.smv`. Voici le résultat obtenu :

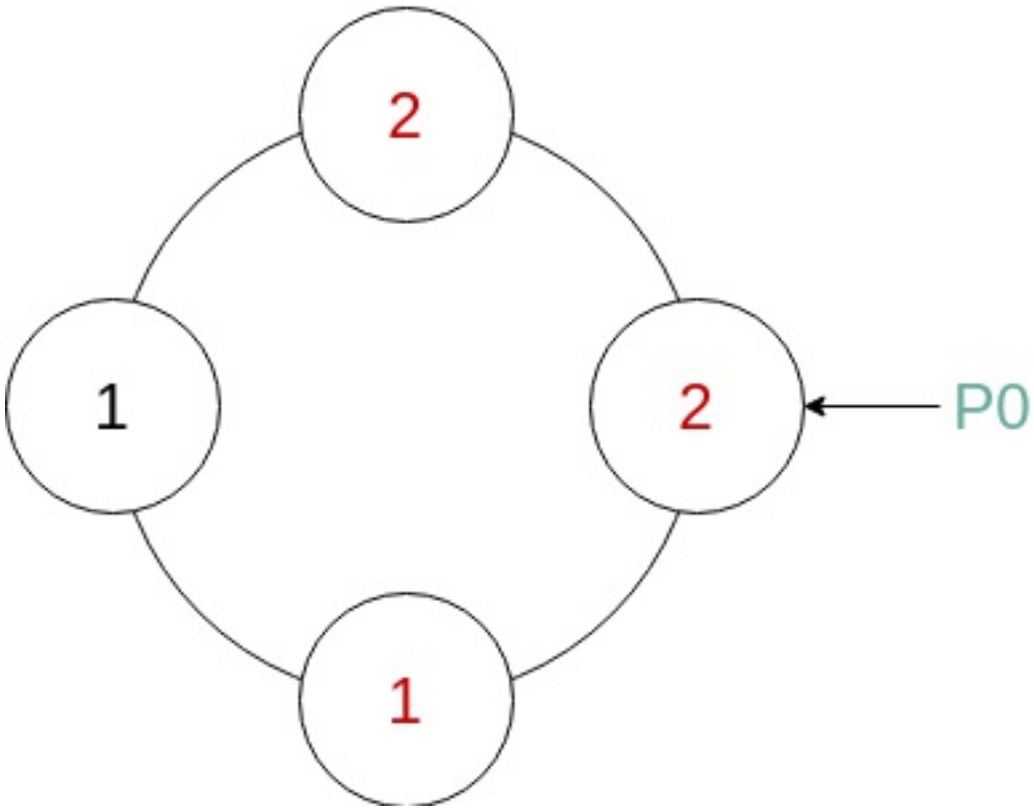
```
NuSMV exercice7/exercice7.smv

-- specification AG (AF (drapeaux[0] = drapeaux[1] & drapeaux[1] =
drapeaux[2]))  is true
-- specification AG (AF etats_critiques[0])  is true
-- specification AG (AF etats_critiques[1])  is true
-- specification AG (AF etats_critiques[2])  is true
-- specification AF (AG ((etats_critiques[0] | etats_critiques[1]) |
etats_critiques[2]))  is true
-- specification AF (AG (etats_critiques[0] -> (((!etats_critiques[1] &
!etats_critiques[2]) & (etats_critiques[1] -> (!etats_critiques[0] &
!etats_critiques[2]))) & (etats_critiques[2] -> (!etats_critiques[0] &
!etats_critiques[1])))))  is true
```

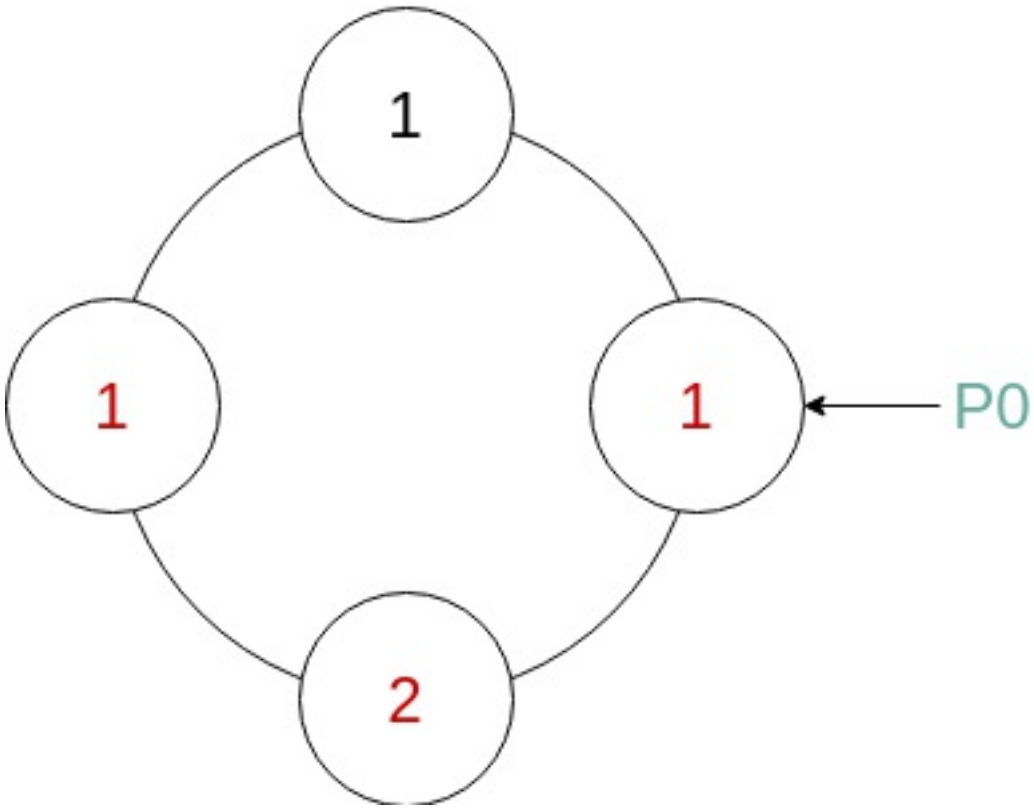
Question 4 :

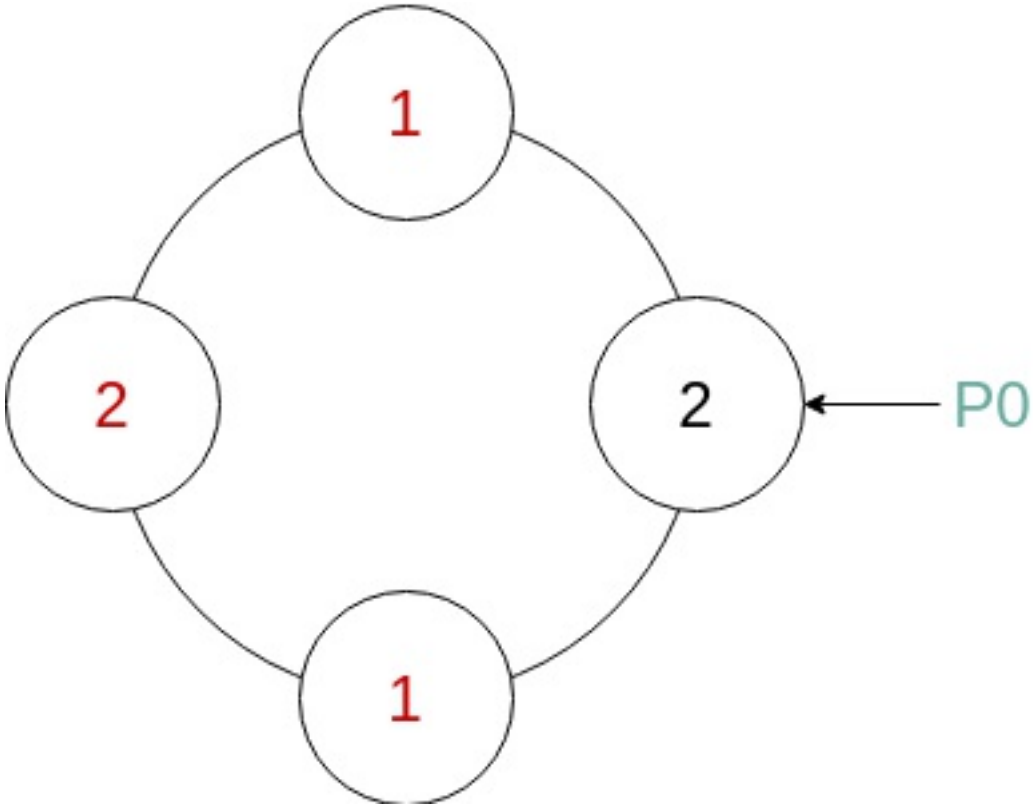
Il faut que K > N pour éviter que l'algorithme ne tombe dans un cycle et ne se stabilise pas à au plus un changement de drapeaux par étape. Ceci est illustré par le cycle ci-dessous.

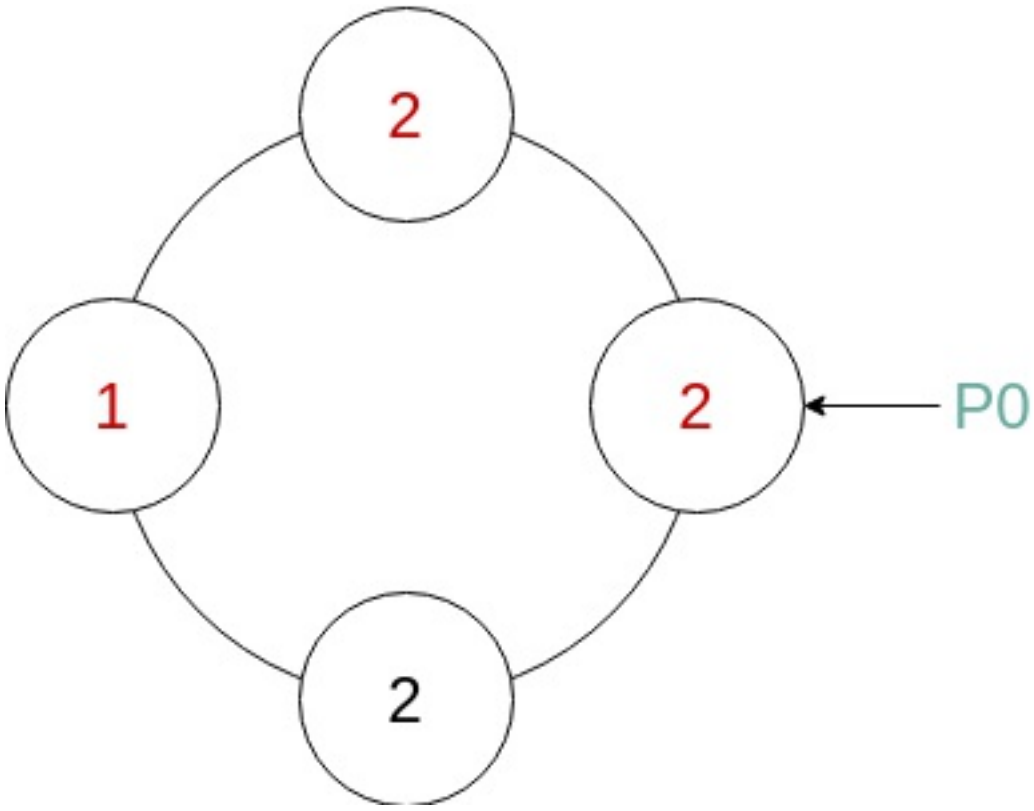


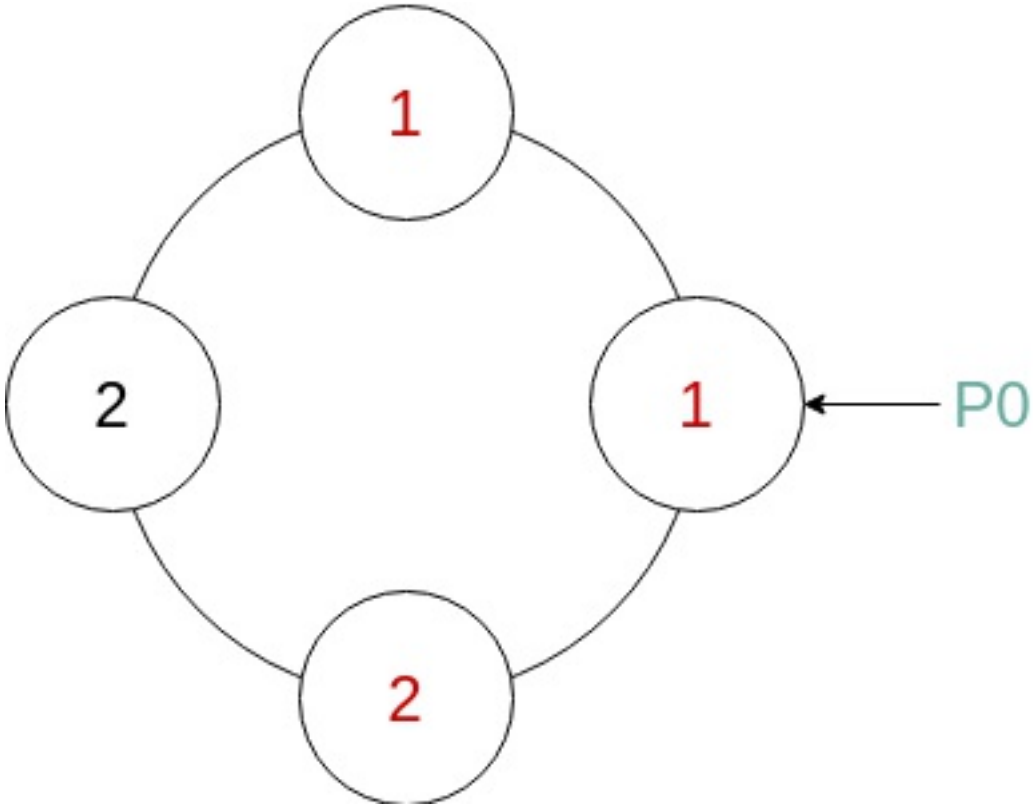

1
2
1
P0
1

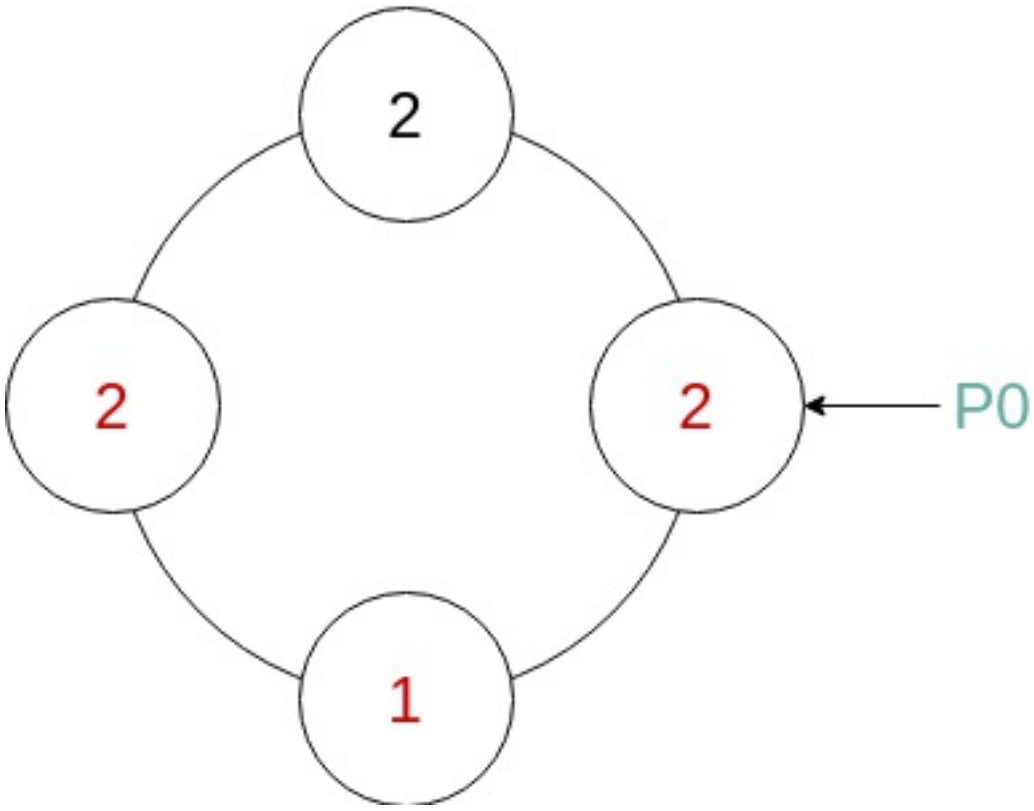

2
1
2
P0
1

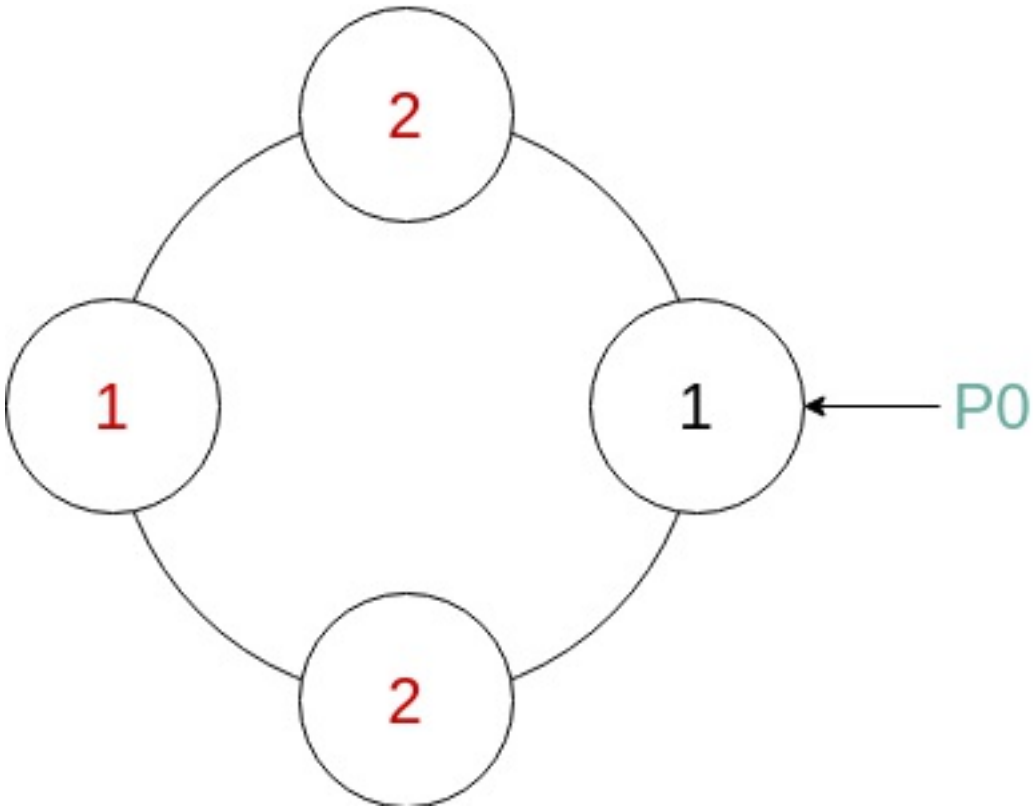

1
1
1
P0
2


1
2
2
P0
1


2
1
2
P0
2


1
2
1
P0
2


2
2
2
P0
1


2
1
1
P0
2