Introduction à l'autostabilisation

Stéphane Devismes

Université Joseph Fourier, Grenoble I

25 septembre 2019

Plan

Plan

Définition

≪ Un algorithme distribué est dit autostabilisant dans un système donné si à partir d'une configuration quelconque du système, toute exécution de l'algorithme atteint en un temps fini (et sans intervention extérieure) une configuration, dite **légitime**, à partir de laquelle tous les suffixes d'exécution possibles sont **corrects**, c'est-à-dire qu'ils vérifient tous la spécification du problème pour lequel l'algorithme a été conçu. ≫

Dijkstra, 1974

Configurations (1/2)

Soit P un algorithme distribué déployé sur un réseau de processus R (représenté par un graphe).

Configurations (1/2)

Soit P un algorithme distribué déployé sur un réseau de processus R (représenté par un graphe).

On peut définir l'ensemble $\mathcal{C}$ des configurations possibles de P dans R.

Configurations (1/2)

Soit P un algorithme distribué déployé sur un réseau de processus R (représenté par un graphe).

On peut définir l'ensemble $\mathcal{C}$ des configurations possibles de P dans R.

En modèle à états [1] :

Etat d'un processus : ensemble des valeurs des variables d'un processus.

Configuration : vecteur d'états, un par processus.

Configuration quelconque : chaque variable de processus a une valeur quelconque prise dans son domaine de définition (*e.g.*, une variable booléenne est affectée à vrai ou faux).

1. Abstraction du modèle à messages : il n'y a pas de canaux !

Configurations (2/2)

En modèle à passage de message :

Etat d'un processus : idem

Etat des canaux : liste de messages.

Configuration : vecteur d'états, un par processus et un par canal.

Configuration quelconque :

- état quelconque des processus et
- état quelconque des canaux
  (*i.e.*, les canaux de communication contiennent un nombre fini de messages transportant des valeurs prises dans leur domaine de définition).

Système de transitions

On peut aussi définir les transitions (pas de calculs) possibles entre les différentes configurations de $\mathcal{C}$:

la relation $\mapsto \subseteq \mathcal{C} \times \mathcal{C}$.

Système de transitions

On peut aussi définir les transitions (pas de calculs) possibles entre les différentes configurations de $\mathcal{C}$:

$$\text{la relation } \mapsto \subseteq \mathcal{C} \times \mathcal{C}.$$

Parmi l'ensemble des configurations possibles, certaines sont caractérisées comme initiales : $\mathcal{I} \subseteq \mathcal{C}$.

Système de transitions

On peut aussi définir les transitions (pas de calculs) possibles entre les différentes configurations de $\mathcal{C}$:

$$\text{la relation } \mapsto \subseteq \mathcal{C} \times \mathcal{C}.$$

Parmi l'ensemble des configurations possibles, certaines sont caractérisées comme initiales : $\mathcal{I} \subseteq \mathcal{C}$.

Le déploiement de *P* sur *R* est entièrement défini par le système de transitions $\mathcal{S} = (\mathcal{I}, \mathcal{C}, \mapsto)$.

Système de transitions

On peut aussi définir les transitions (pas de calculs) possibles entre les différentes configurations de $\mathcal{C}$:

$$\text{la relation } \mapsto \subseteq \mathcal{C} \times \mathcal{C}.$$

Parmi l'ensemble des configurations possibles, certaines sont caractérisées comme initiales : $\mathcal{I} \subseteq \mathcal{C}$.

Le déploiement de P sur R est entièrement défini par le système de transitions $\mathcal{S} = (\mathcal{I}, \mathcal{C}, \mapsto)$.

Une exécution de P sur R est une suite de configurations $\gamma_0, \ldots$ vérifiant les deux conditions suivantes :

- $\gamma_0 \in \mathcal{I}$ et
- $\forall i > 0, \gamma_{i-1} \mapsto \gamma_i$.

Système de transitions

On peut aussi définir les transitions (pas de calculs) possibles entre les différentes configurations de $\mathcal{C}$:

$$\text{la relation } \mapsto \subseteq \mathcal{C} \times \mathcal{C}.$$

Parmi l'ensemble des configurations possibles, certaines sont caractérisées comme initiales : $\mathcal{I} \subseteq \mathcal{C}$.

Le déploiement de P sur R est entièrement défini par le système de transitions $\mathcal{S} = (\mathcal{I}, \mathcal{C}, \mapsto)$.

Une exécution de P sur R est une suite de configurations $\gamma_0, \ldots$ vérifiant les deux conditions suivantes :

- $\gamma_0 \in \mathcal{I}$ et
- $\forall i > 0, \gamma_{i-1} \mapsto \gamma_i$.

Si P est autostabilisant, alors on considère que $\mathcal{I} = \mathcal{C}$.

Partition : légitime/illégititme

L'ensemble des configurations possibles est divisé en deux sous-ensembles :

- Les configurations légitimes, à partir desquelles le système exécute correctement la tâche pour laquelle il a été conçu.
- Les configurations illégitimes, où le système ne vérifie pas nécessairement la spécification de la tâche pour laquelle il a été conçu.

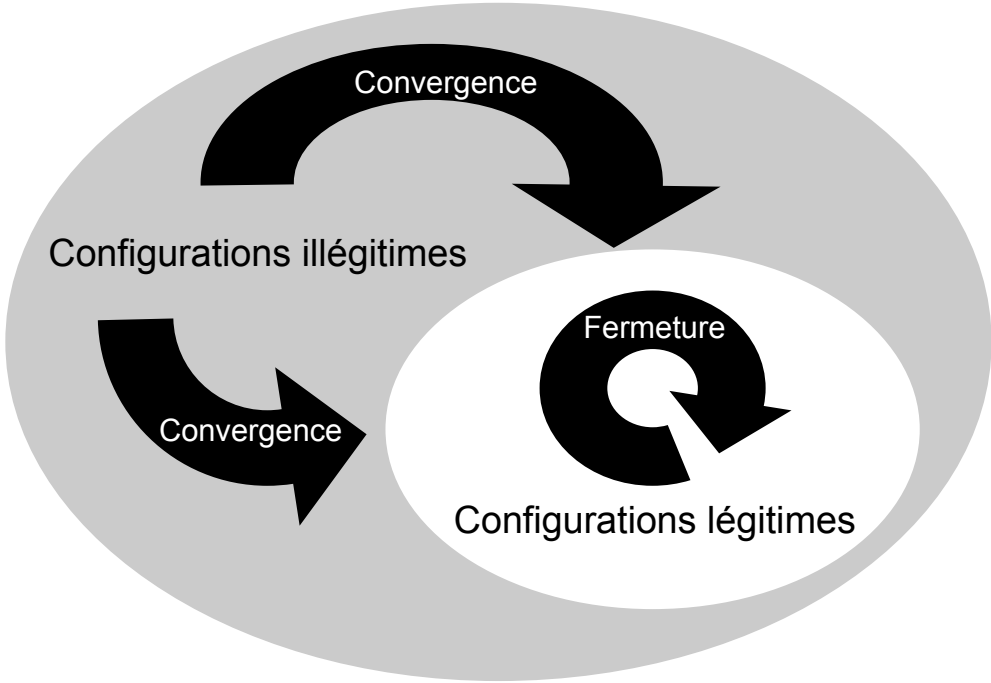
Idée intuitive

Un système peut être initialement (ou plutôt suite à une faute) dans une configuration illégitime.

Idée intuitive

Un système peut être initialement (ou plutôt suite à une faute) dans une configuration illégitime.

Le but d'un algorithme autostabilisant est alors de faire retrouver (au plus vite) au système une configuration légitime où la spécification du problème à résoudre est vérifiée.



Définition formelle

Soit $\mathcal{A}$ un algorithme distribué déployé sur un réseau R.

Soit $\mathcal{C}$ l'ensemble des configurations possibles défini en déployant l'algorithme $\mathcal{A}$ sur R.

$\mathcal{A}$ est autostabilisant pour la spécification SP dans R s'il existe un sous-ensemble non-vide $\mathcal{L}$ de $\mathcal{C}$, appelé ensemble des configurations légitimes, qui vérifie les trois propriétés suivantes :

Définition formelle

Soit $\mathcal{A}$ un algorithme distribué déployé sur un réseau R.

Soit $\mathcal{C}$ l'ensemble des configurations possibles défini en déployant l'algorithme $\mathcal{A}$ sur R.

$\mathcal{A}$ est autostabilisant pour la spécification SP dans R s'il existe un sous-ensemble non-vide $\mathcal{L}$ de $\mathcal{C}$, appelé ensemble des configurations légitimes, qui vérifie les trois propriétés suivantes :

Fermeture : Toute configuration accessible à partir d'une configuration de $\mathcal{L}$ en appliquant $\mathcal{A}$ est aussi une configuration de $\mathcal{L}$.

Définition formelle

Soit $\mathcal{A}$ un algorithme distribué déployé sur un réseau R.

Soit $\mathcal{C}$ l'ensemble des configurations possibles défini en déployant l'algorithme $\mathcal{A}$ sur R.

$\mathcal{A}$ est autostabilisant pour la spécification SP dans R s'il existe un sous-ensemble non-vide $\mathcal{L}$ de $\mathcal{C}$, appelé ensemble des configurations légitimes, qui vérifie les trois propriétés suivantes :

Fermeture : Toute configuration accessible à partir d'une configuration de $\mathcal{L}$ en appliquant $\mathcal{A}$ est aussi une configuration de $\mathcal{L}$.

Convergence : Toute exécution de $\mathcal{A}$ à partir d'une configuration quelconque de $\mathcal{C}$ atteint en un temps fini une configuration de $\mathcal{L}$. Cette phase de convergence est aussi appelée phase de stabilisation.

Définition formelle

Soit $\mathcal{A}$ un algorithme distribué déployé sur un réseau R.

Soit $\mathcal{C}$ l'ensemble des configurations possibles défini en déployant l'algorithme $\mathcal{A}$ sur R.

$\mathcal{A}$ est autostabilisant pour la spécification SP dans R s'il existe un sous-ensemble non-vide $\mathcal{L}$ de $\mathcal{C}$, appelé ensemble des configurations légitimes, qui vérifie les trois propriétés suivantes :

Fermeture : Toute configuration accessible à partir d'une configuration de $\mathcal{L}$ en appliquant $\mathcal{A}$ est aussi une configuration de $\mathcal{L}$.

Convergence : Toute exécution de $\mathcal{A}$ à partir d'une configuration quelconque de $\mathcal{C}$ atteint en un temps fini une configuration de $\mathcal{L}$. Cette phase de convergence est aussi appelée phase de stabilisation.

Correction : Toute exécution $\mathcal{A}$ à partir d'une configuration de $\mathcal{L}$ vérifie SP.

Plan

Modèle pour et par l'autostabilisation

Créé dans l'article originel de Dijkstra.

Uniquement utilisé dans le cadre de l'autostabilisation !

Modèle pour et par l'autostabilisation

Créé dans l'article originel de Dijkstra.

Uniquement utilisé dans le cadre de l'autostabilisation !

BUT : Simplifier les preuves.

Abstraction du modèle à passage de messages

Il existe des méthodes de transformation pour passer du modèle à états au modèle à passage de messages sans perdre la propriété d'autostabilisation.

Abstraction du modèle à passage de messages

Il existe des méthodes de transformation pour passer du modèle à états au modèle à passage de messages sans perdre la propriété d'autostabilisation.

Nous verrons une technique, qui fonctionne pour beaucoup d'algorithmes, en TP et en projet.

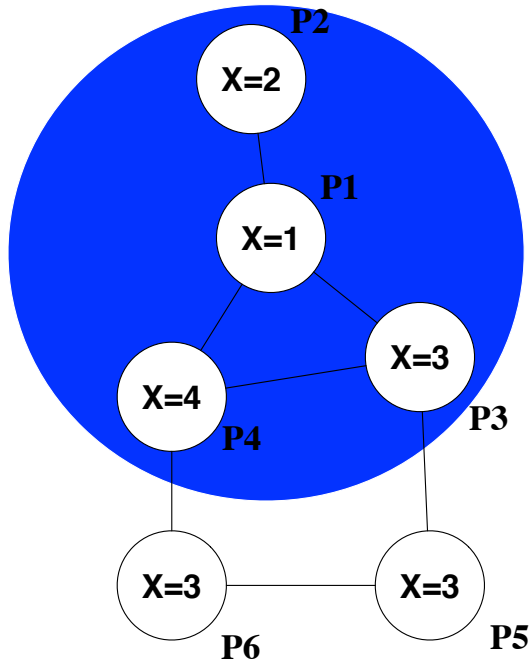
Abstraction du modèle à passage de messages

Les **canaux de communications** sont remplacés par de la mémoire localement partagée.

Abstraction du modèle à passage de messages

Les **canaux de communications** sont remplacés par de la mémoire localement partagée.

Ainsi, les liens du graphe de communications représente la possibilité pour deux voisins de lire directement la mémoire (partagée) de l'autre.

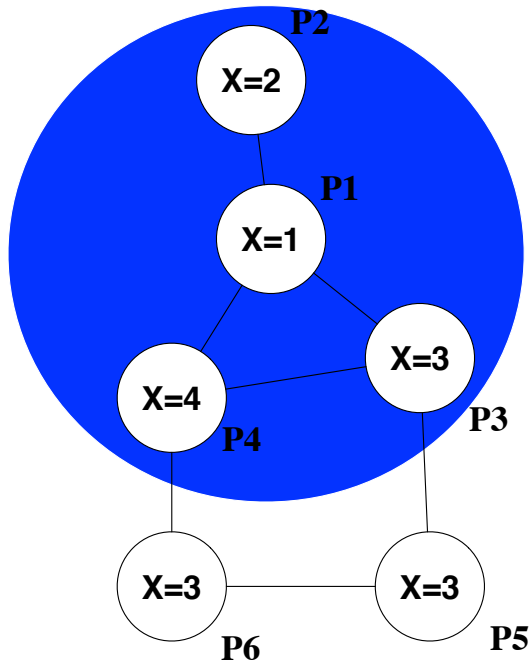


Par exemple, $P1$ peut lire directement la mémoire de $P2$, $P3$ et $P4$. Donc, il peut lire que $X_{P2} = 2$, $X_{P3} = 3$ et $X_{P4} = 4$.

Abstraction du modèle à passage de messages

Les **canaux de communications** sont remplacés par de la mémoire localement partagée.

Ainsi, les liens du graphe de communications représente la possibilité pour deux voisins de lire directement la mémoire (partagée) de l'autre.

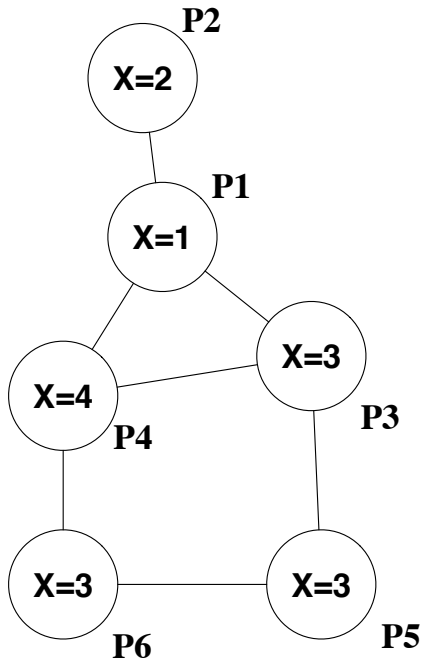


Par exemple, $P1$ peut lire directement la mémoire de $P2$, $P3$ et $P4$. Donc, il peut lire que $X_{P2} = 2$, $X_{P3} = 3$ et $X_{P4} = 4$.

Mais, $P1$ NE peut PAS lire directement la mémoire de $P5$ et $P6$.

Abstraction du modèle à passage de messages

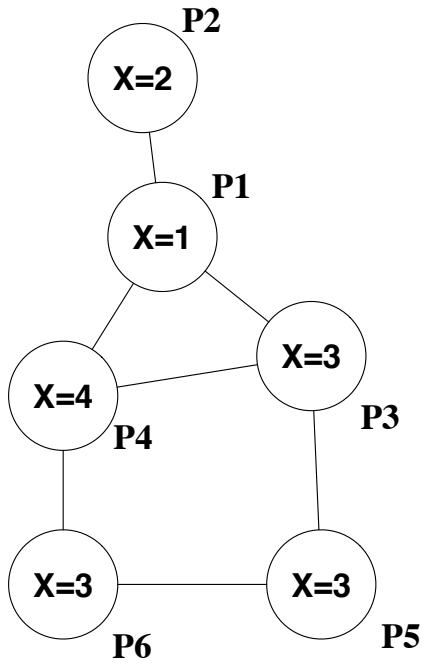
La configuration du réseau est définie par l'état de chacun des processus du réseau.



Rappel : l'état d'un processus est défini par la valeur de chacune de ses variables.

Abstraction du modèle à passage de messages

La configuration du réseau est définie par l'état de chacun des processus du réseau.

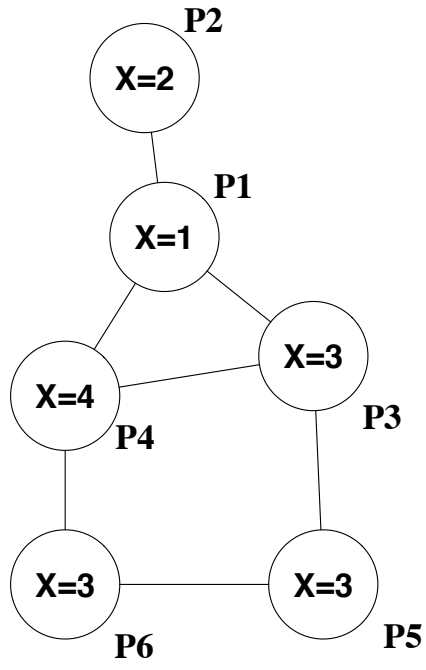


Rappel : l'état d'un processus est défini par la valeur de chacune de ses variables.

Par exemple, l'état de $P1$ est $X = 1$.

Abstraction du modèle à passage de messages

La configuration du réseau est définie par l'état de chacun des processus du réseau.



Rappel : l'état d'un processus est défini par la valeur de chacune de ses variables.

Par exemple, l'état de $P1$ est $X = 1$.

Dans notre exemple, la configuration du système est :

$$\langle X_{P1} = 1, X_{P2} = 2, X_{P3} = 3, X_{P4} = 3, X_{P5} = 3, X_{P6} = 3 \rangle$$

Règles gardées

Le programme de chaque processus est représenté par un **ensemble fini de règles** (ou actions ou commandes) gardées de la forme suivante :

$$(\langle \textit{étiquette} \rangle \ ::) \ \langle \textit{prédicat} \rangle \ \mapsto \ \langle \textit{affectations} \rangle$$

Règles gardées

Le programme de chaque processus est représenté par un **ensemble fini de règles** (ou actions ou commandes) gardées de la forme suivante :

$$(\langle \textit{étiquette} \rangle \ ::) \ \langle \textit{prédicat} \rangle \ \mapsto \ \langle \textit{affectations} \rangle$$

- L'étiquette n'est pas obligatoire, elle permet juste de nommer les actions dans les preuves.

Règles gardées

Le programme de chaque processus est représenté par un **ensemble fini de règles** (ou actions ou commandes) gardées de la forme suivante :

$$(\langle \textit{étiquette} \rangle \ ::) \ \langle \textit{prédicat} \rangle \ \mapsto \ \langle \textit{affectations} \rangle$$

- L'étiquette n'est pas obligatoire, elle permet juste de nommer les actions dans les preuves.
- Le prédicat d'une règle est une expression booléenne sur les variables du processus et de ses voisins.

Règles gardées

Le programme de chaque processus est représenté par un **ensemble fini de règles** (ou actions ou commandes) gardées de la forme suivante :

$$(\langle \textit{étiquette} \rangle \ ::) \ \langle \textit{prédicat} \rangle \ \mapsto \ \langle \textit{affectations} \rangle$$

- L'étiquette n'est pas obligatoire, elle permet juste de nommer les actions dans les preuves.
- Le prédicat d'une règle est une expression booléenne sur les variables du processus et de ses voisins.
- Les affectations permettent de modifier les variables du processus.

Règles gardées

Le programme de chaque processus est représenté par un **ensemble fini de règles** (ou actions ou commandes) gardées de la forme suivante :

$$(\langle \textit{étiquette} \rangle \ ::) \ \langle \textit{prédicat} \rangle \ \mapsto \ \langle \textit{affectations} \rangle$$

- L'étiquette n'est pas obligatoire, elle permet juste de nommer les actions dans les preuves.
- Le prédicat d'une règle est une expression booléenne sur les variables du processus et de ses voisins.
- Les affectations permettent de modifier les variables du processus.

Ainsi, chaque processus peut lire les variables de ses voisins, et ensuite modifier son état en fonction de cette lecture.

Règles gardées

Le programme de chaque processus est représenté par un **ensemble fini de règles** (ou actions ou commandes) gardées de la forme suivante :

$$(\langle \textit{étiquette} \rangle \ ::) \ \langle \textit{prédicat} \rangle \ \mapsto \ \langle \textit{affectations} \rangle$$

- L'étiquette n'est pas obligatoire, elle permet juste de nommer les actions dans les preuves.
- Le prédicat d'une règle est une expression booléenne sur les variables du processus et de ses voisins.
- Les affectations permettent de modifier les variables du processus.

Ainsi, chaque processus peut lire les variables de ses voisins, et ensuite modifier son état en fonction de cette lecture.

L'ensemble des programmes des processus définit un algorithme distribué.

Exemple de règles gardées : 1er algorithme de Dijkstra

Tous les processus ont une unique variable v dont le domaine est $\{0, \ldots, K-1\}$ avec $K \geq n$.

Un processus p_i détient un jeton si et seulement si il satisfait le prédicat $\mathit{Token}(p_i)$.

Le programme de chaque processus consiste en une unique règle :

Programme de la racine p_0 :

$$\mathit{Token}(p_0) \mapsto v_{p_0} \leftarrow (v_{p_0} + 1) \bmod K, \text{ où } \mathit{Token}(p_0) \equiv (v_{p_0} = v_{p_{n-1}})$$

Programme de chaque processus p_i nonracine :

$$\mathit{Token}(p_i) \mapsto v_{p_i} \leftarrow v_{p_{i-1}}, \text{ où } \mathit{Token}(p_i) \equiv (v_{p_i} \neq v_{p_{i-1}})$$

Activation

Une règle est activable (dans une configuration donnée) si son prédicat est **vrai**.

Activation

Une règle est activable (dans une configuration donnée) si son prédicat est **vrai**.

Un processus est activable si au moins **une de ses règles est activable**.

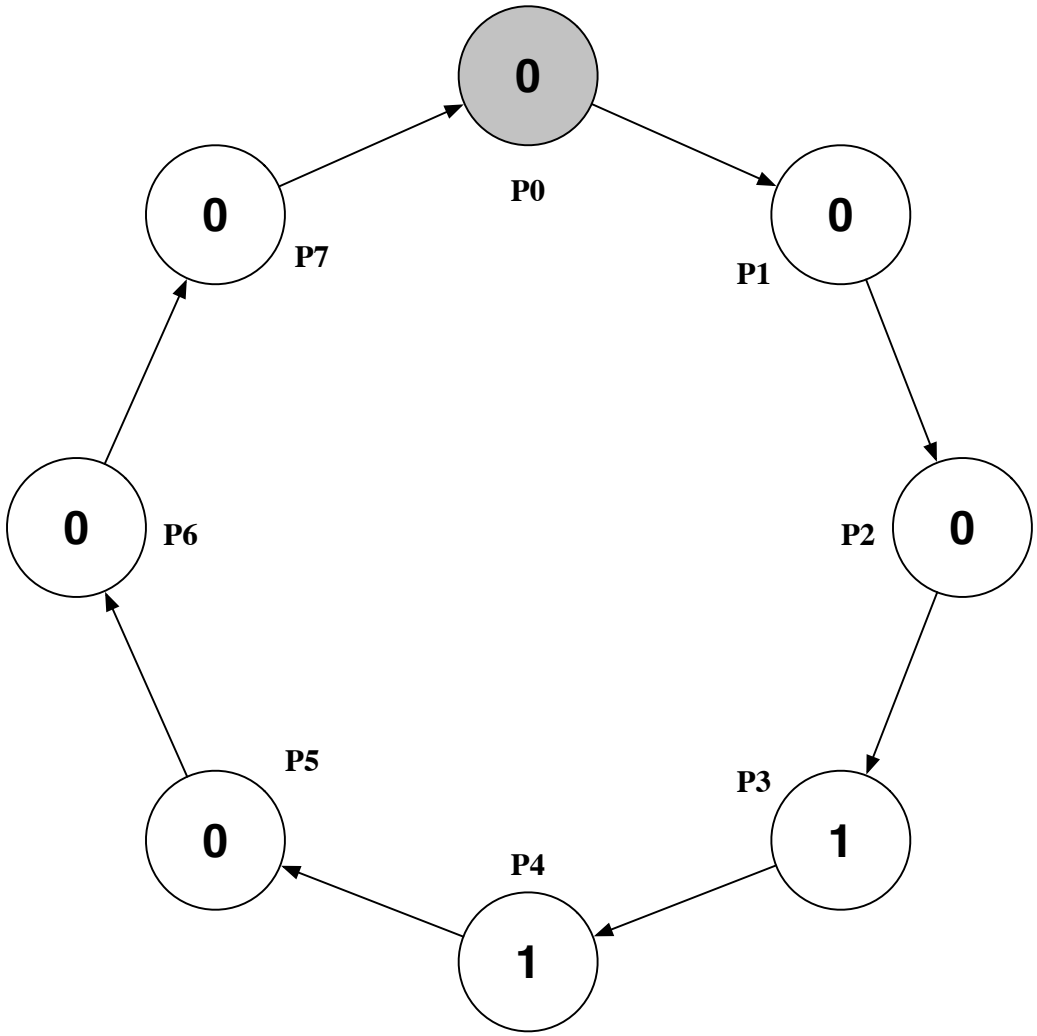
Activation

Une règle est activable (dans une configuration donnée) si son prédicat est **vrai**.

Un processus est activable si au moins **une de ses règles est activable**.

Un processus peut exécuter une de ses règles seulement si elle est activable.

Exemple d'activation : 1er algorithme de Dijkstra



*P*0, *P*3 et *P*5 sont activables.

Atomicité

Le modèle à états suppose une atomicité forte : à chaque étape, un sous-ensemble non-vide de processus activable dans la configuration courante γ est sélectionné (s'il existe au moins un processus activable).

Atomicité

Le modèle à états suppose une atomicité forte : à chaque étape, un sous-ensemble non-vide de processus activable dans la configuration courante γ est sélectionné (s'il existe au moins un processus activable).

Chaque processus de l'ensemble exécute alors $\ll$ simultanément $\gg$ une de ses actions activables dans γ ; on obtient alors la configuration suivante, *etc*.

Atomicité

Le modèle à états suppose une atomicité forte : à chaque étape, un sous-ensemble non-vide de processus activable dans la configuration courante γ est sélectionné (s'il existe au moins un processus activable).

Chaque processus de l'ensemble exécute alors $\ll$ simultanément $\gg$ une de ses actions activables dans γ ; on obtient alors la configuration suivante, *etc*.

Si aucun processus n'est activable dans une configuration donnée, alors cette configuration est dite terminale.

Atomicité

Le modèle à états suppose une atomicité forte : à chaque étape, un sous-ensemble non-vide de processus activable dans la configuration courante γ est sélectionné (s'il existe au moins un processus activable).

Chaque processus de l'ensemble exécute alors $\ll$ simultanément $\gg$ une de ses actions activables dans γ ; on obtient alors la configuration suivante, *etc*.

Si aucun processus n'est activable dans une configuration donnée, alors cette configuration est dite terminale.

Dans l'exemple précédent, la sélection s'opère parmi : $\{P0\}$, $\{P3\}$, $\{P5\}$, $\{P0, P3\}$, $\{P0, P5\}$, $\{P3, P5\}$ et $\{P0, P3, P5\}$

Démons

Le sous-ensemble de processus activés à chaque étape est choisi par un démon (ou ordonnanceur). Il peut être :

Démons

Le sous-ensemble de processus activés à chaque étape est choisi par un démon (ou ordonnanceur). Il peut être :

Central Tant que la configuration courante n'est pas terminale, il en active exactement un par étape.

Démons

Le sous-ensemble de processus activés à chaque étape est choisi par un démon (ou ordonnanceur). Il peut être :

Central Tant que la configuration courante n'est pas terminale, il en active exactement un par étape.

Localement Central Tant que la configuration courante n'est pas terminale, il en active au moins un par étape. Cependant, il n'y a pas de processus voisins activés simultanément.

Démons

Le sous-ensemble de processus activés à chaque étape est choisi par un démon (ou ordonnanceur). Il peut être :

Central Tant que la configuration courante n'est pas terminale, il en active exactement un par étape.

Localement Central Tant que la configuration courante n'est pas terminale, il en active au moins un par étape. Cependant, il n'y a pas de processus voisins activés simultanément.

Distribué Tant que la configuration courante n'est pas terminale il en active au moins un par étape.

Démons

Le sous-ensemble de processus activés à chaque étape est choisi par un démon (ou ordonnanceur). Il peut être :

Central Tant que la configuration courante n'est pas terminale, il en active exactement un par étape.

Localement Central Tant que la configuration courante n'est pas terminale, il en active au moins un par étape. Cependant, il n'y a pas de processus voisins activés simultanément.

Distribué Tant que la configuration courante n'est pas terminale il en active au moins un par étape.

Synchrone Tant que la configuration courante n'est pas terminale, il active tous les processus activables à chaque étape.

Démons

Les démons central, localement central et distribué se déclinent aussi en fonction de leur équité :

Démons

Les démons central, localement central et distribué se déclinent aussi en fonction de leur équité :

> Un démon est fortement équitable si pour toute exécution de l'algorithme, il n'existe pas de suffixe infini où un processus est activable infiniment souvent et n'exécute jamais aucune action.

Démons

Les démons central, localement central et distribué se déclinent aussi en fonction de leur équité :

> Un démon est fortement équitable si pour toute exécution de l'algorithme, il n'existe pas de suffixe infini où un processus est activable infiniment souvent et n'exécute jamais aucune action.
>
> Un démon est faiblement équitable si pour toute exécution de l'algorithme, il n'existe pas de suffixe infini où un processus est toujours activable et n'exécute jamais aucune action.

Démons

Les démons central, localement central et distribué se déclinent aussi en fonction de leur équité :

Un démon est fortement équitable si pour toute exécution de l'algorithme, il n'existe pas de suffixe infini où un processus est activable infiniment souvent et n'exécute jamais aucune action.

Un démon est faiblement équitable si pour toute exécution de l'algorithme, il n'existe pas de suffixe infini où un processus est toujours activable et n'exécute jamais aucune action.

Un démon inéquitable n'a pas de restriction si ce n'est qu'il doit au moins activer un processus tant que le système n'est pas dans une configuration terminale.

Démons

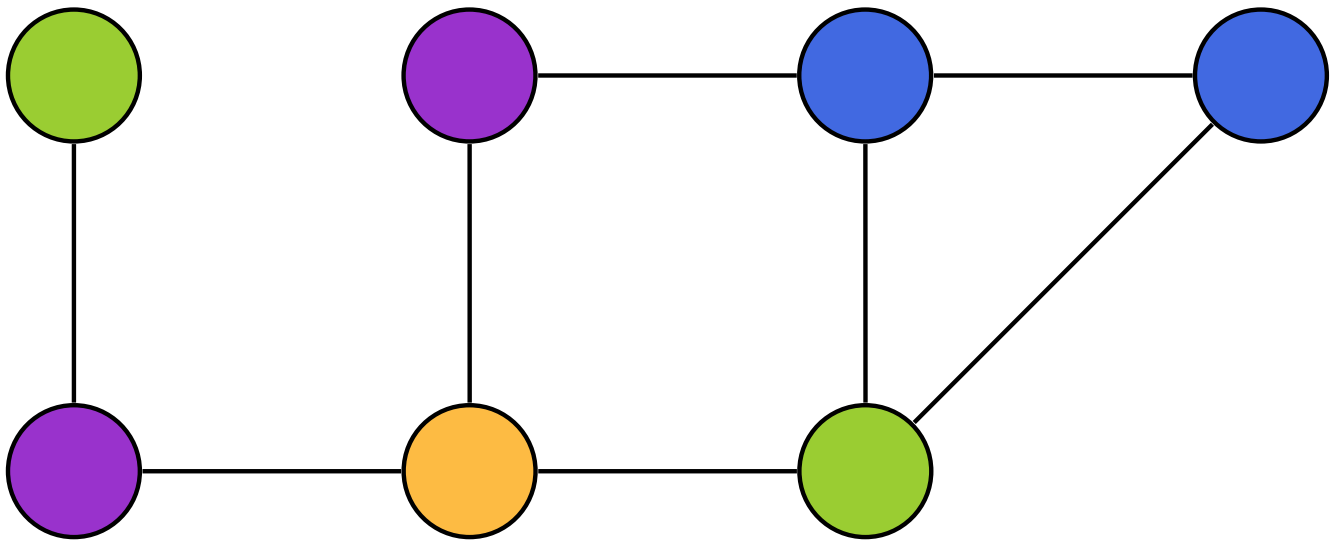
Les démons central, localement central et distribué se déclinent aussi en fonction de leur équité :

> Un démon est fortement équitable si pour toute exécution de l'algorithme, il n'existe pas de suffixe infini où un processus est activable infiniment souvent et n'exécute jamais aucune action.
>
> Un démon est faiblement équitable si pour toute exécution de l'algorithme, il n'existe pas de suffixe infini où un processus est toujours activable et n'exécute jamais aucune action.
>
> Un démon inéquitable n'a pas de restriction si ce n'est qu'il doit au moins activer un processus tant que le système n'est pas dans une configuration terminale.

Le démon **distribué inéquitable** est le démon **le plus général** (plus faible) du modèle à états.

Modèle à états : résumé

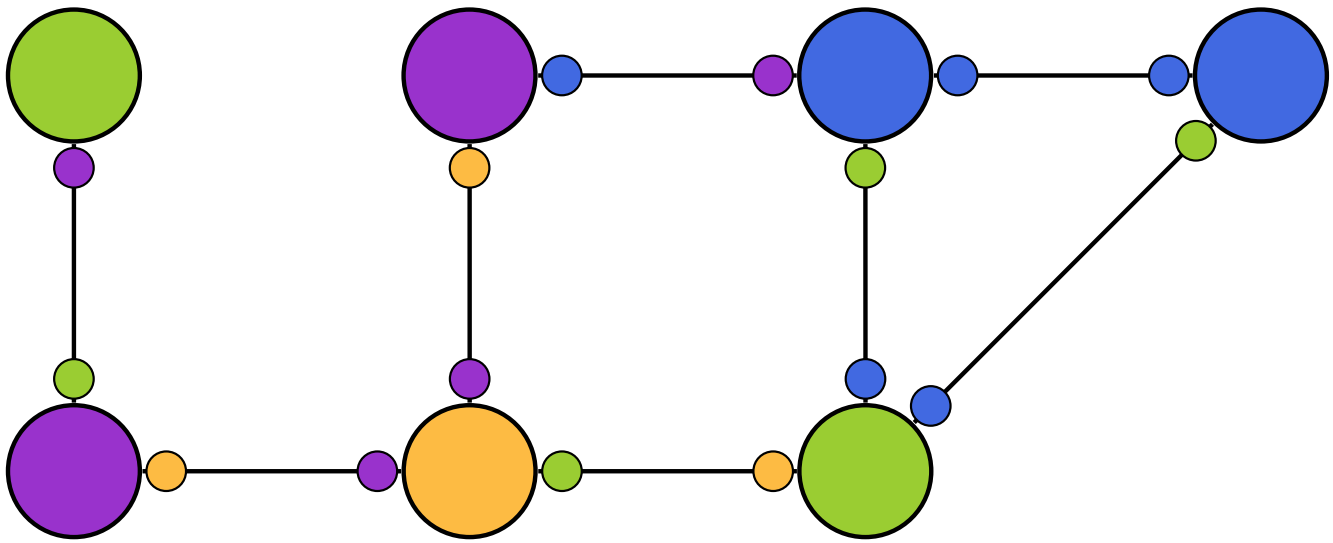
Configuration



Modèle à états : résumé

Etape atomique

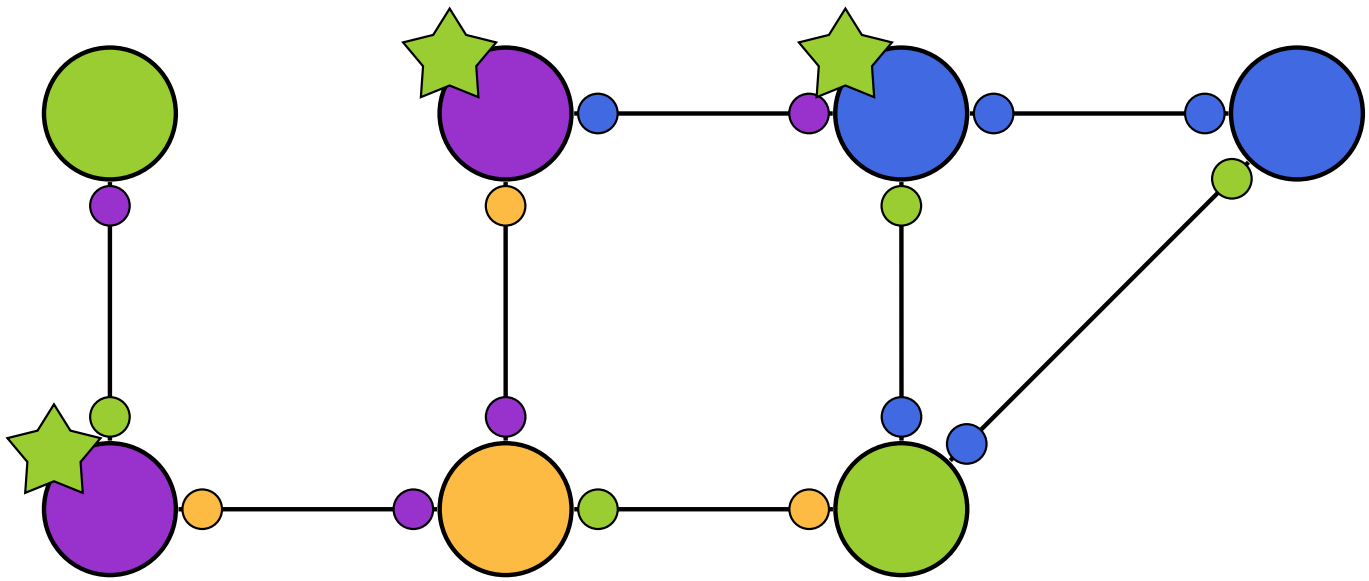
- Lecture des variables des voisins



Modèle à états : résumé

Etape atomique

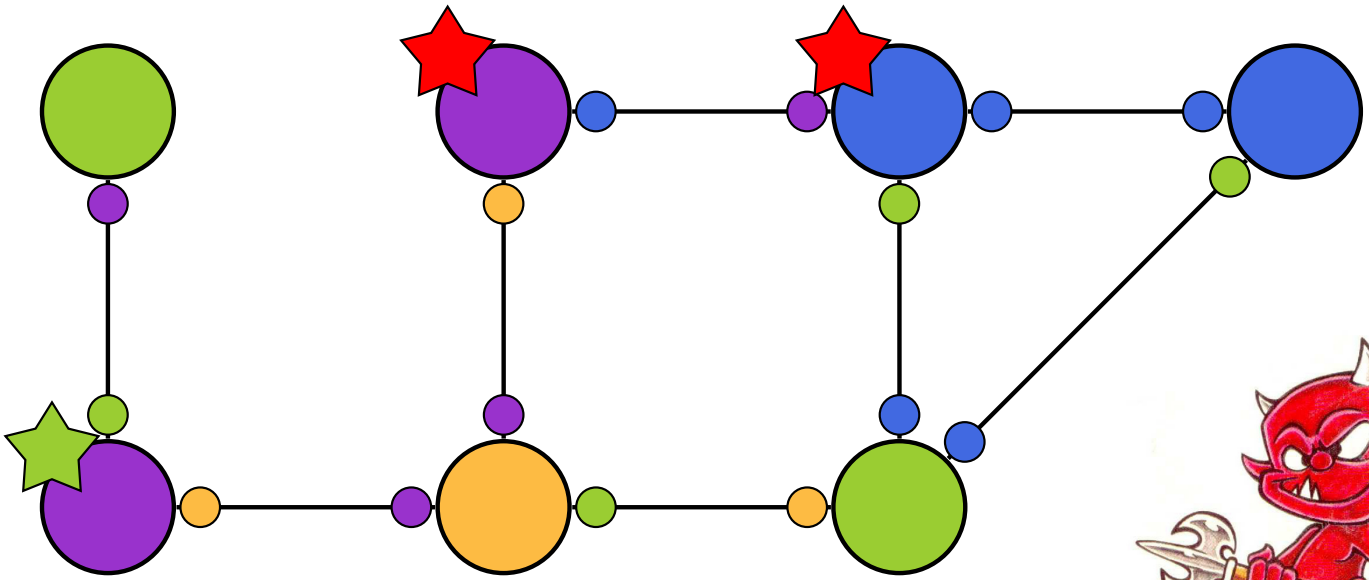
- Lecture des variables des voisins
- Processus activables



Modèle à états : résumé

Etape atomique

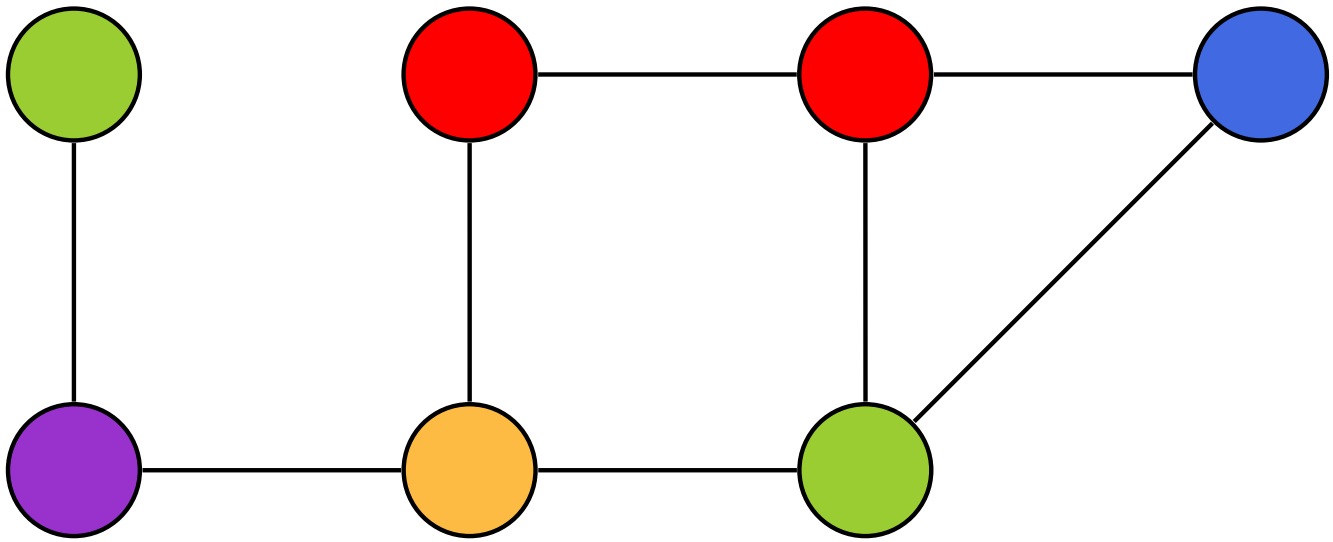
- Lecture des variables des voisins
- Processus activables
- Sélection du démon : modélisation de l'asynchrononisme



Modèle à états : résumé

Etape atomique

- Lecture des variables des voisins
- Processus activables
- Sélection du démon : modélisation de l'asynchrononisme
- Mise à jour des états locaux et on recommence ...



Complexité en espace

Dans le modèle à états, on considère l'occupation mémoire de la même manière que dans le modèle à passage de messages.

Cependant, dans le modèle à états, cette mesure revêt un intérêt particulier car, elle représente généralement le volume de communication.

Complexité en temps

Le temps s'évalue en nombre d'étapes (ou pas) de calcul ou en nombre de rondes.

Ronde : idée intuitive

La ronde permet d'évaluer le temps d'exécution par rapport aux processus les plus lents.

Définition formelle

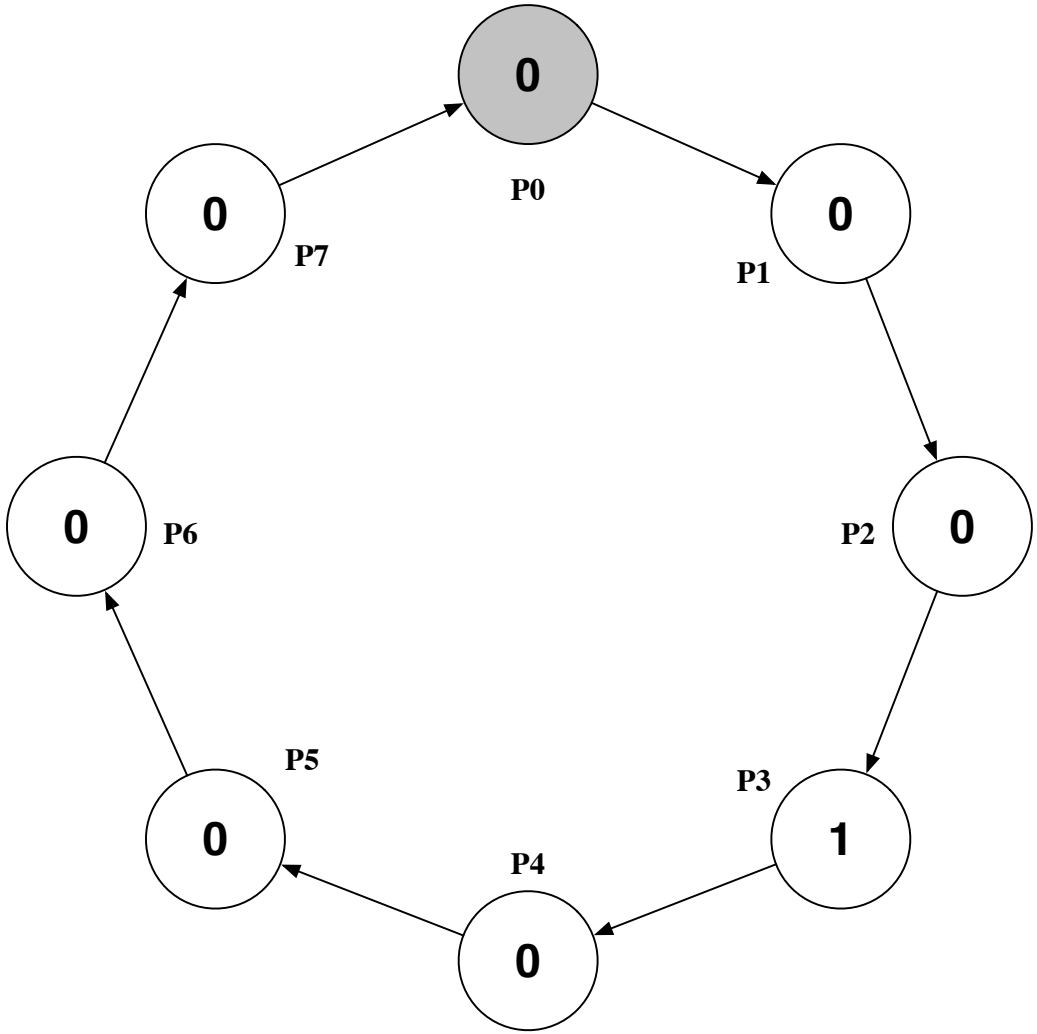
Un processus subit une neutralisation s'il passe d'activable à non activable sans exécuter la moindre action.

Définition formelle

Un processus subit une neutralisation s'il passe d'activable à non activable sans exécuter la moindre action.

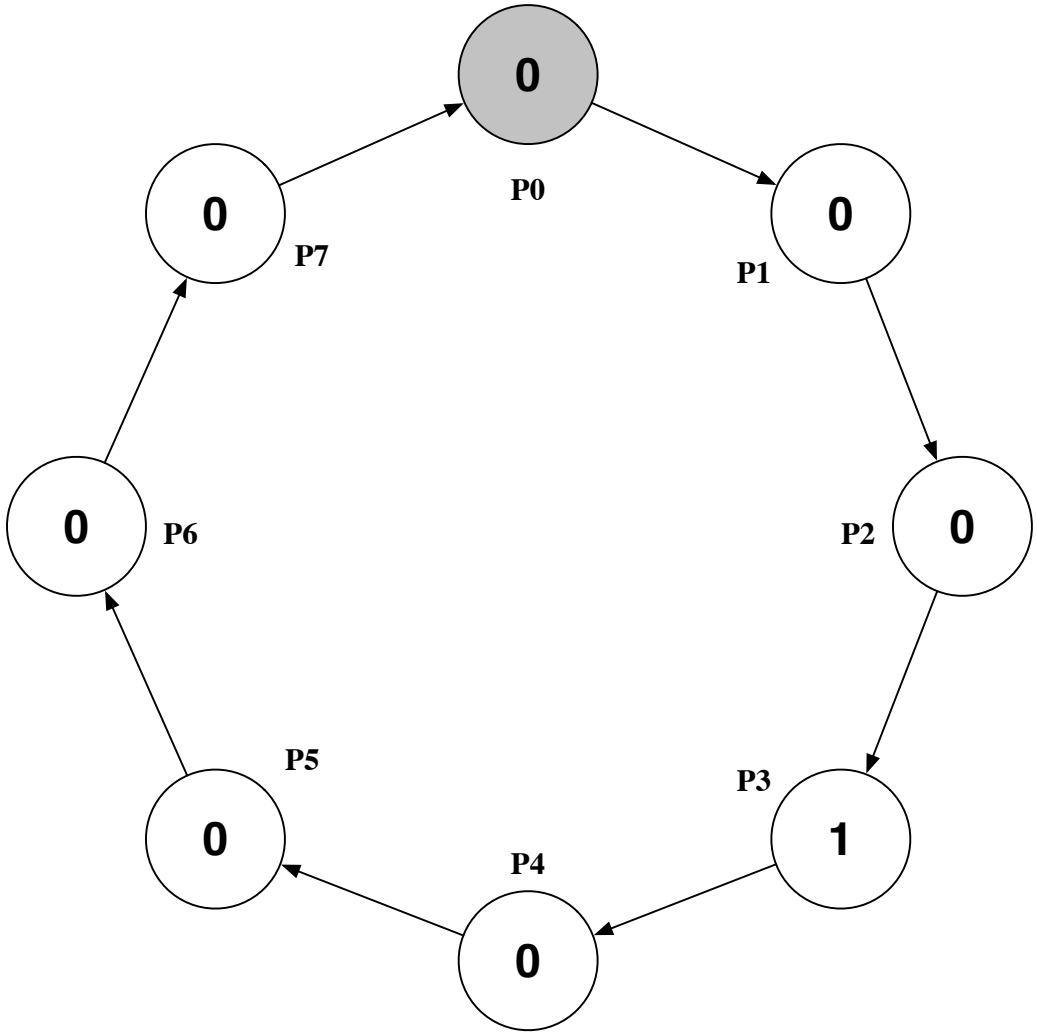
En fait, la neutralisation de p correspond à la situation suivante : un ou plusieurs voisins de p exécutent une règle et les changements occasionnés par l'exécution de ces règles rendent toutes les règles de p non activables.

Exemple neutralisation, algorithme de Dijkstra



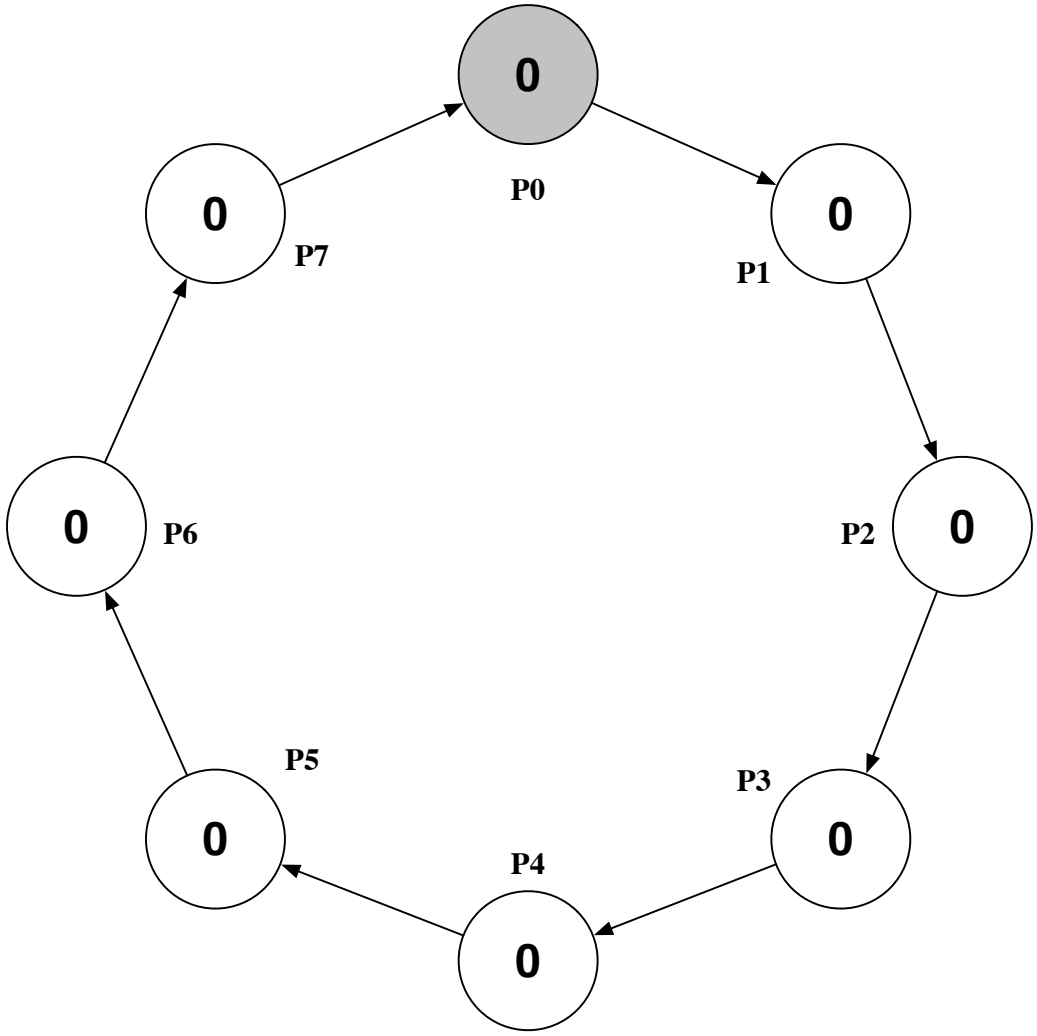
$P0$, $P3$, $P4$ sont activables

Exemple neutralisation, algorithme de Dijkstra



*P*3 est activé

Exemple neutralisation, algorithme de Dijkstra



*P*0 reste activable et *P*4 a subi une neutralisation !

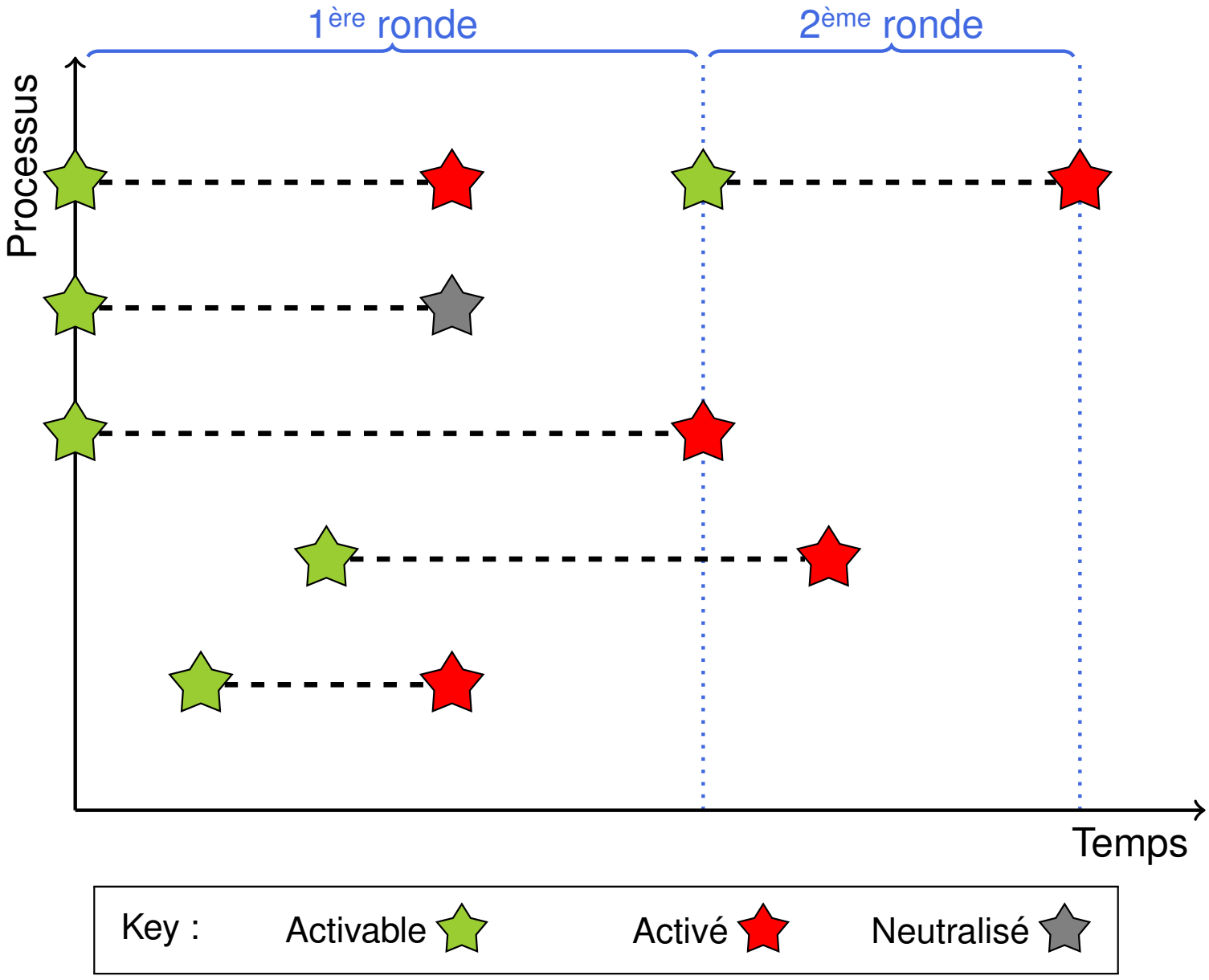
Définition formelle

Etant donnée une exécution e, la **première ronde de** e, notée e', est le préfixe minimal de e contenant, pour chaque processus activable lors de la première configuration de e,

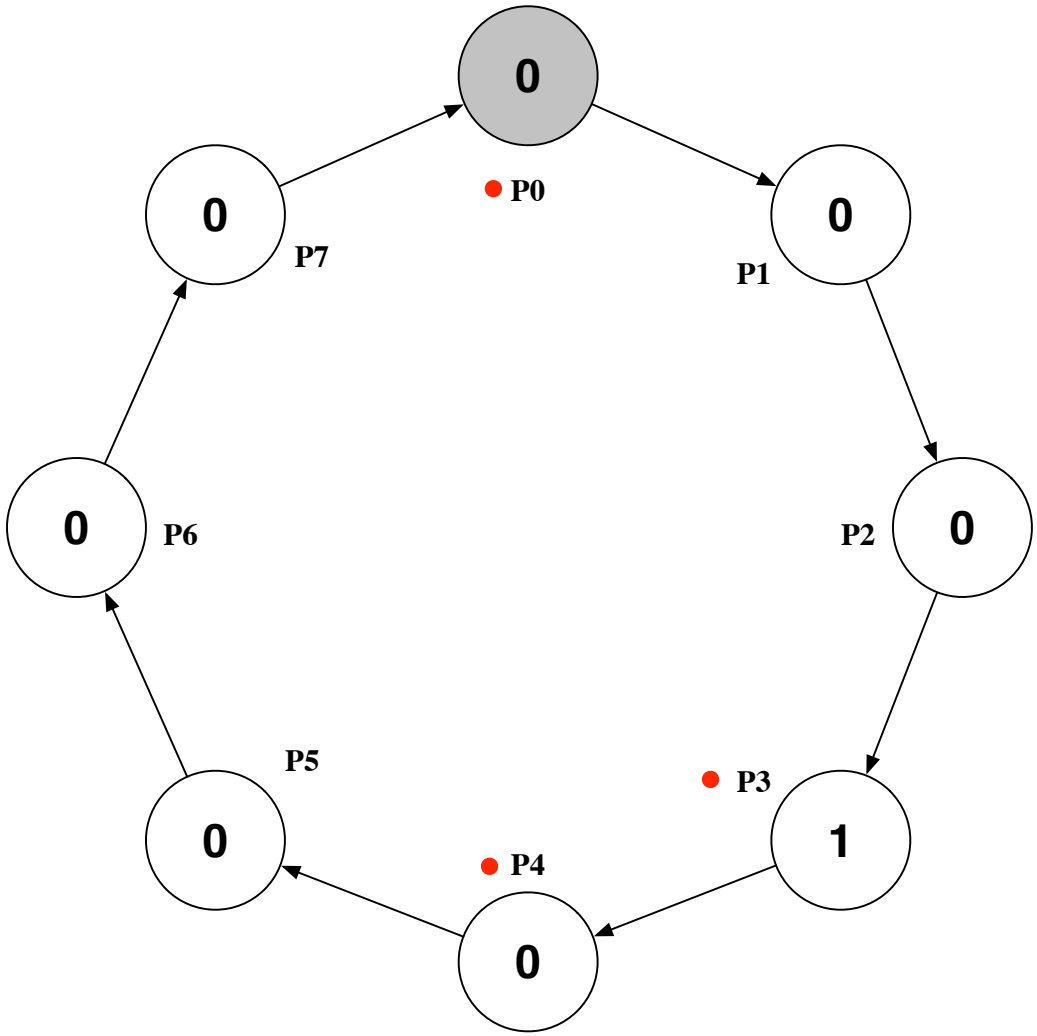
- l'exécution d'une de ses règles ou
- la neutralisation du processus.

Soit e'' le suffixe de e commençant à la dernière configuration de e'. La seconde ronde de e correspond à la première ronde de e'', *etc.*

Ronde : exemple schématique

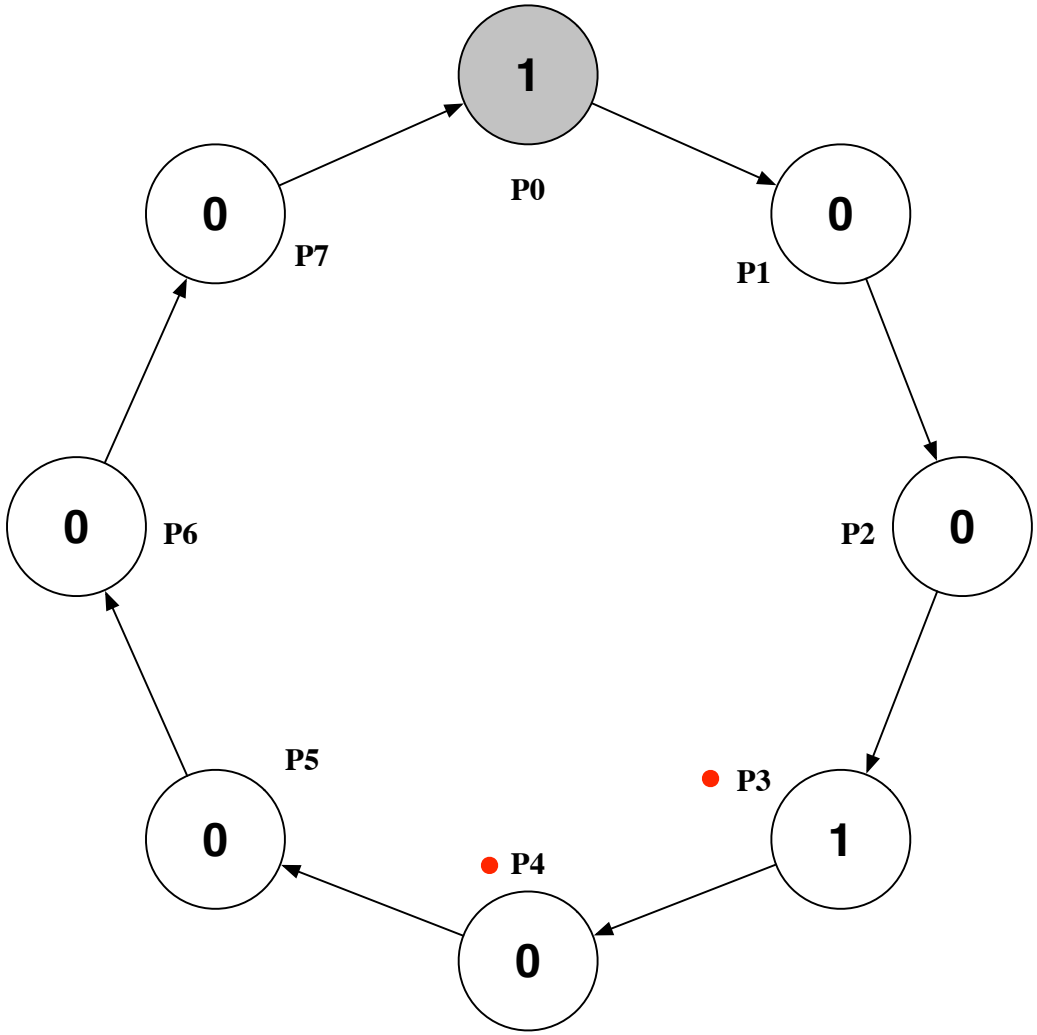


Exemple ronde, algorithme de Dijkstra



$P0$, $P3$, $P4$ sont activables

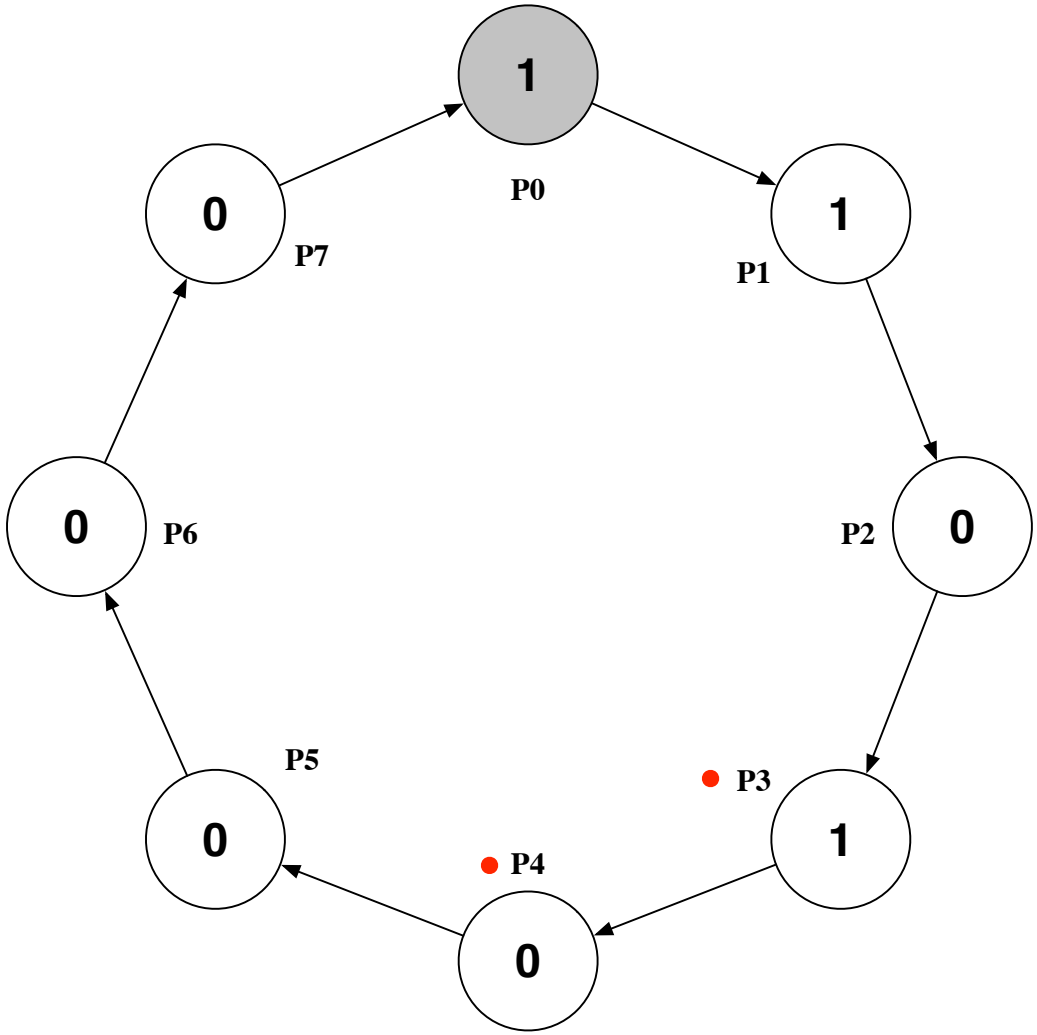
Exemple ronde, algorithme de Dijkstra



*P*0 est activé

*P*1 devient activable, *P*3 et *P*4 reste activable.

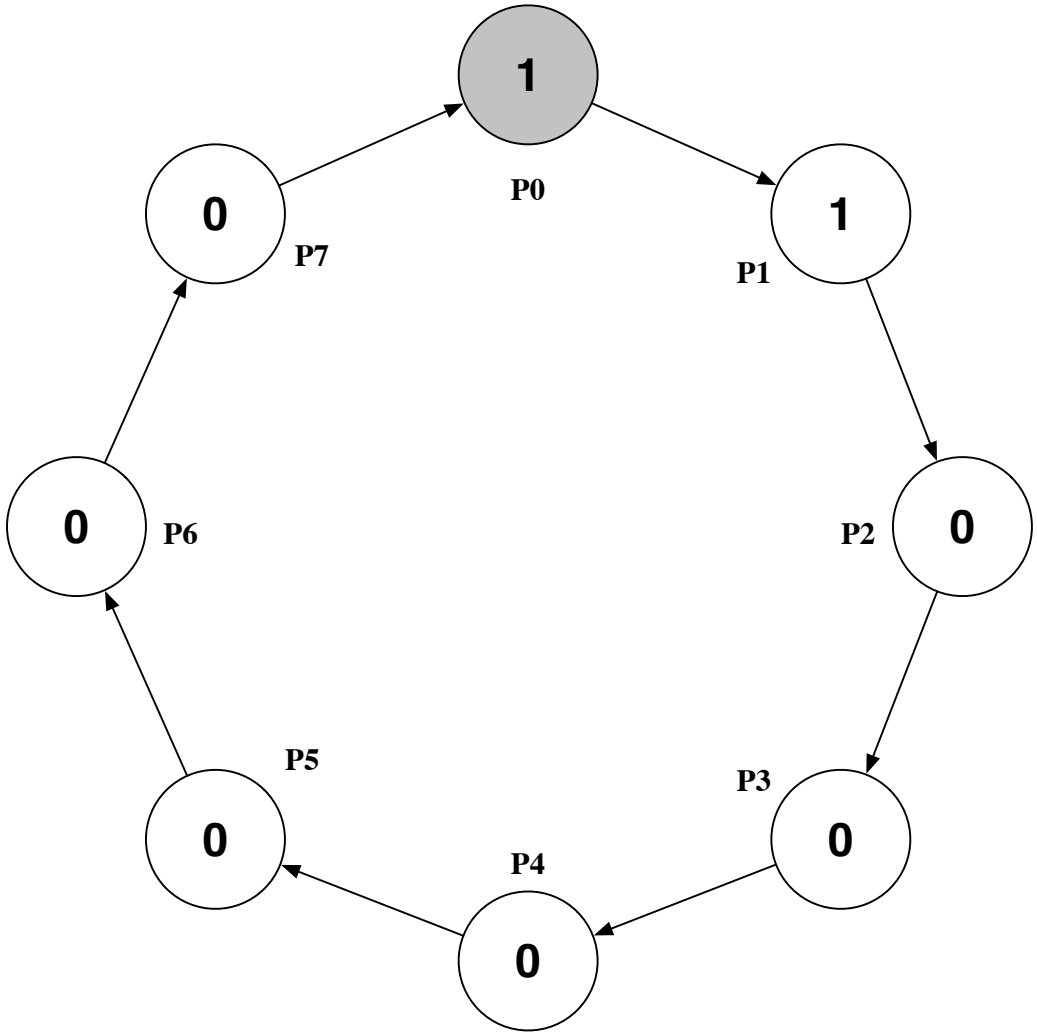
Exemple ronde, algorithme de Dijkstra



*P*1 est activé

*P*2 devient activable, *P*3 et *P*4 reste activable.

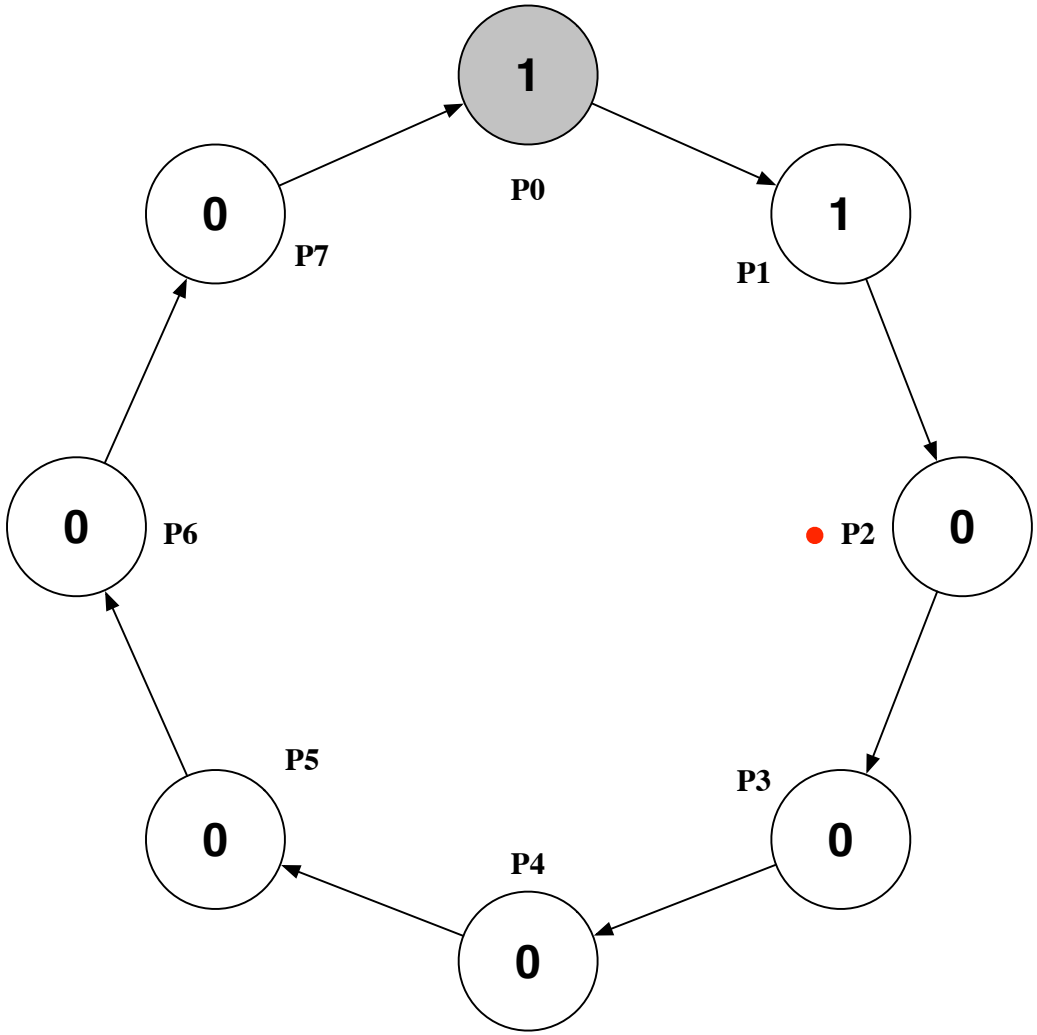
Exemple ronde, algorithme de Dijkstra



*P*3 est activé

*P*4 est neutralisé, la ronde est terminée

Exemple ronde, algorithme de Dijkstra



Une nouvelle ronde commence
Seul $P2$ est activable

Temps de stabilisation

Le temps **maximum** (en rondes et/ou en étapes) pour retrouver une configuration légitime à partir de n'importe quelle configuration illégitime.

C'est l'une des métriques les plus importantes pour juger de l'efficacité d'un algorithme autostabilisant.

Plan

Démon

Nous supposons un démon distribué inéquitable.

Propriété (1/2)

Lemme 1

Il n'existe pas de configuration sans jeton.

Propriété (1/2)

Lemme 1

Il n'existe pas de configuration sans jeton.

Preuve. Supposons, par contradiction, qu'il existe une configuration γ sans jeton.

Propriété (1/2)

Lemme 1

Il n'existe pas de configuration sans jeton.

Preuve. Supposons, par contradiction, qu'il existe une configuration γ sans jeton.

Pour tout $i \in [1 \dots n-1]$, $v_{p_i} = v_{p_{i-1}}$ par définition de $Token(p_i)$.

Propriété (1/2)

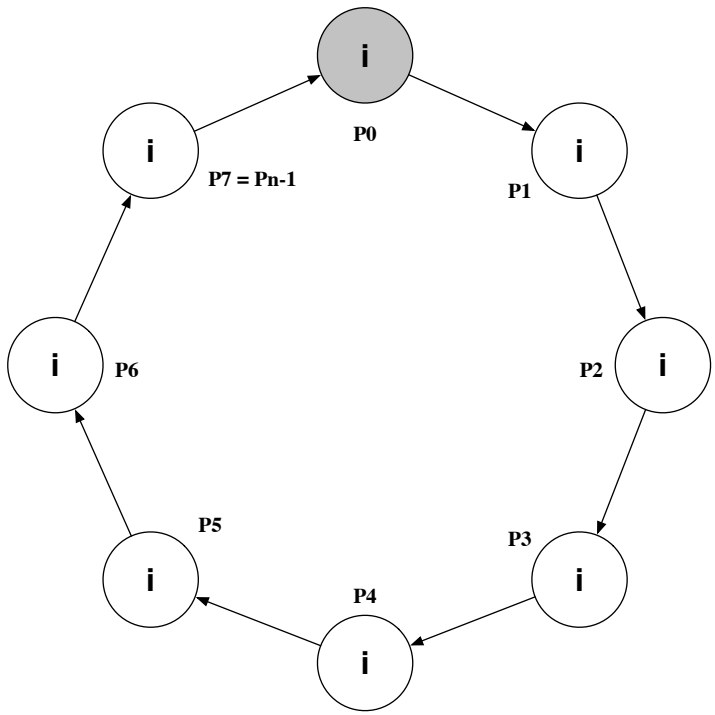
Lemme 1

Il n'existe pas de configuration sans jeton.

Preuve. Supposons, par contradiction, qu'il existe une configuration γ sans jeton.

Pour tout $i \in [1 \ldots n-1]$, $v_{p_i} = v_{p_{i-1}}$ par définition de $Token(p_i)$.

Par transitivité, $v_{p_{n-1}} = v_{p_0}$.



Propriété (1/2)

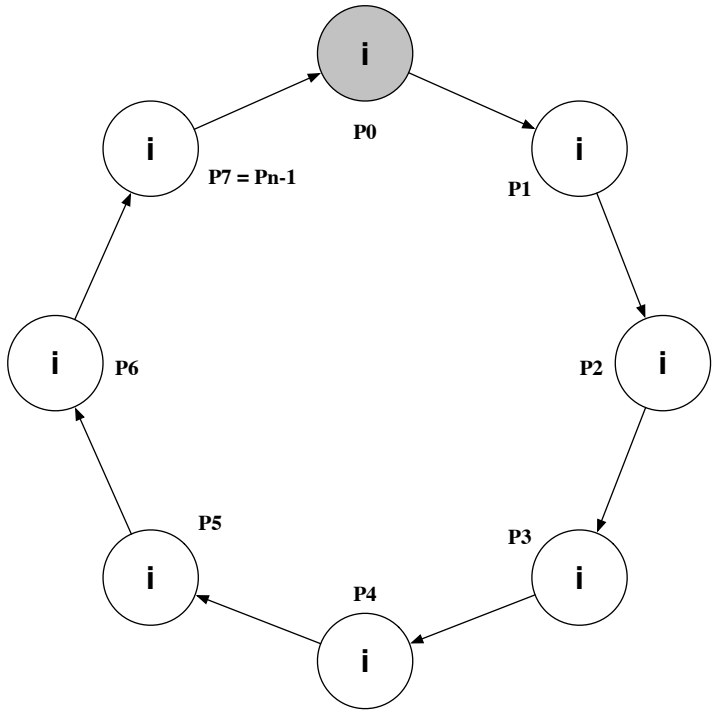
Lemme 1

Il n'existe pas de configuration sans jeton.

Preuve. Supposons, par contradiction, qu'il existe une configuration γ sans jeton.

Pour tout $i \in [1 \ldots n-1]$, $v_{p_i} = v_{p_{i-1}}$ par définition de $Token(p_i)$.

Par transitivité, $v_{p_{n-1}} = v_{p_0}$.



Puisque p_{n-1} est le prédécesseur de p_0, p_0 vérifie $Token(p_0)$, contradiction.

□

Propriété (2/2)

Corollaire 1

Il n'y a pas de configuration terminale.

Configurations légitimes

Définition 1

Soit $\mathcal{L}$ l'ensemble des configurations contenant un unique jeton.

Configurations légitimes

Définition 1

Soit $\mathcal{L}$ l'ensemble des configurations contenant un unique jeton.

Puisque, la configuration où tous les processus p_i vérifient $v_{p_i} = 0$ est légitime, on a :

Remarque 1

$\mathcal{L}$ n'est pas vide.

Fermeture et correction

Lemme 2

$\mathcal{L}$ est clos et à partir d'une configuration de $\mathcal{L}$, il existe un unique jeton qui circule parmi tous les processus.

Fermeture et correction

Lemme 2

$\mathcal{L}$ est clos et à partir d'une configuration de $\mathcal{L}$, il existe un unique jeton qui circule parmi tous les processus.

Preuve. Soit $\gamma \in \mathcal{L}$.

Par définition, il existe un unique processus p_i dans γ qui satisfait le prédicat *Token*.

D'après l'algorithme, seul p_i est activable dans γ et donc seul p_i sera activé durant le prochain pas de calcul.

Fermeture et correction

Lemme 2

$\mathcal{L}$ est clos et à partir d'une configuration de $\mathcal{L}$, il existe un unique jeton qui circule parmi tous les processus.

Preuve. Soit $\gamma \in \mathcal{L}$.

Par définition, il existe un unique processus p_i dans γ qui satisfait le prédicat *Token*.

D'après l'algorithme, seul p_i est activable dans γ et donc seul p_i sera activé durant le prochain pas de calcul.

D'après l'algorithme, le changement d'état de p_i n'affecte que p_i et son successeur dans l'anneau p_{i+1}.

Fermeture et correction

Lemme 2

$\mathcal{L}$ est clos et à partir d'une configuration de $\mathcal{L}$, il existe un unique jeton qui circule parmi tous les processus.

Preuve. Soit $\gamma \in \mathcal{L}$.

Par définition, il existe un unique processus p_i dans γ qui satisfait le prédicat *Token*.

D'après l'algorithme, seul p_i est activable dans γ et donc seul p_i sera activé durant le prochain pas de calcul.

D'après l'algorithme, le changement d'état de p_i n'affecte que p_i et son successeur dans l'anneau p_{i+1}.

En particulier, cela signifie que les autres processus ne détiennent toujours pas de jeton après le pas de calcul.

Suite de la preuve

Après avoir exécuter son action, p_i ne satisfait plus le prédicat *Token*.

Suite de la preuve

Après avoir exécuter son action, p_i ne satisfait plus le prédicat *Token*.

Cependant p_{i+1} satisfait nécessairement le prédicat *Token* d'après le lemme 1.

Suite de la preuve

Après avoir exécuter son action, p_i ne satisfait plus le prédicat *Token*.

Cependant p_{i+1} satisfait nécessairement le prédicat *Token* d'après le lemme 1.

Ainsi, le nombre de jeton est toujours de 1 et p_i a passé le jeton à son successeur.

Conclusion de la preuve

Ainsi, $\mathcal{L}$ est un ensemble clos.

Conclusion de la preuve

Ainsi, $\mathcal{L}$ est un ensemble clos.

De plus, à chaque pas de calcul, l'unique jeton passe d'un processus à son successeur.

Conclusion de la preuve

Ainsi, $\mathcal{L}$ est un ensemble clos.

De plus, à chaque pas de calcul, l'unique jeton passe d'un processus à son successeur.

Donc, il circule infiniment souvent parmi tous les processus. □

Convergence (1/2)

Lemme 3

Dans toute exécution, p_0 exécute une infinité d'actions.

Preuve.

Supposons le contraire. Considérons donc une configuration γ à partir de laquelle p_0 n'exécute plus aucune action à jamais.

Convergence (1/2)

Lemme 3

Dans toute exécution, p_0 exécute une infinité d'actions.

Preuve.

Supposons le contraire. Considérons donc une configuration γ à partir de laquelle p_0 n'exécute plus aucune action à jamais.

Soit $F = \Sigma_{i \in S}(n - i)$, où $S = \{i \mid i \neq 0 \wedge \mathit{Token}(p_i)\}$, c'est-à-dire la somme des distances à p_0 des processus nonracines détenant un jeton. Par définition, $F \geq 0$.

Convergence (1/2)

Lemme 3

Dans toute exécution, p_0 exécute une infinité d'actions.

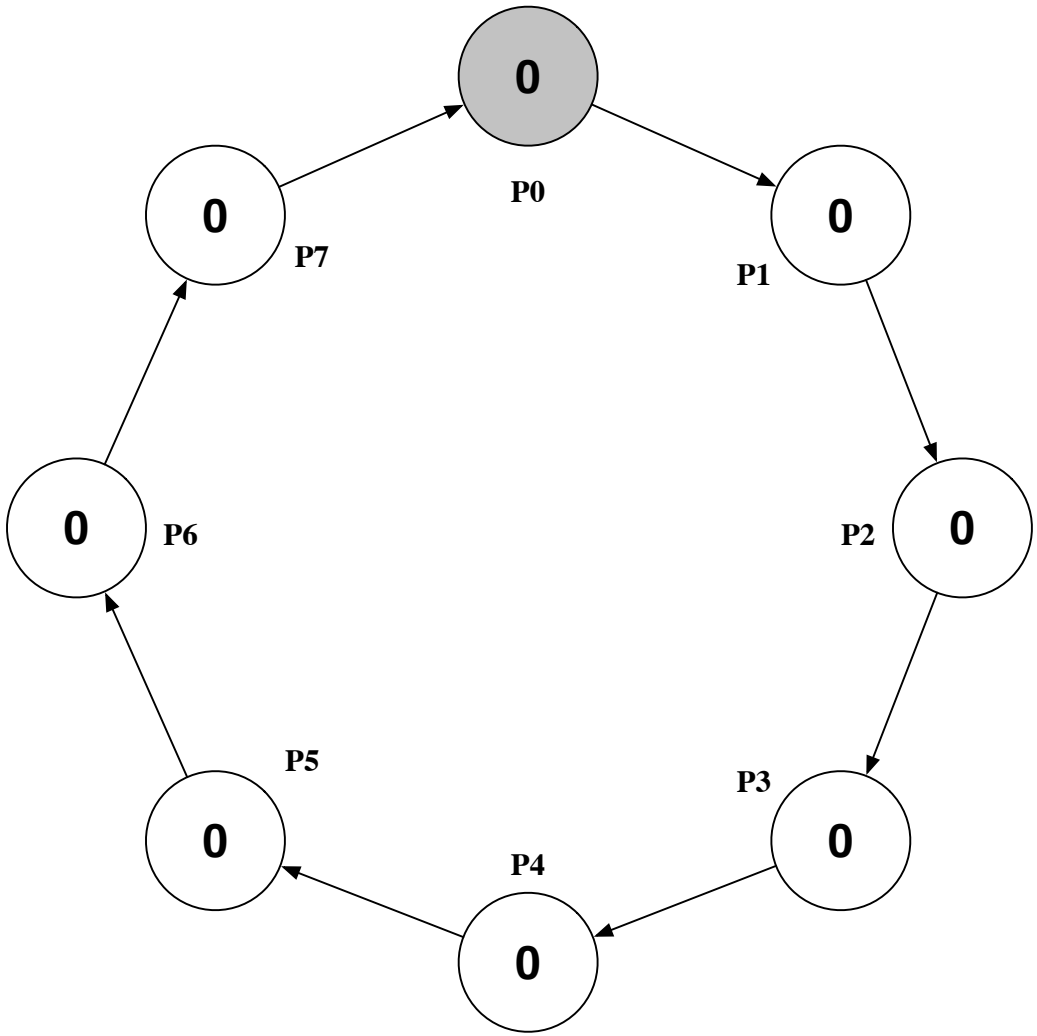
Preuve.

Supposons le contraire. Considérons donc une configuration γ à partir de laquelle p_0 n'exécute plus aucune action à jamais.

Soit $F = \Sigma_{i \in S}(n - i)$, où $S = \{i \mid i \neq 0 \wedge \mathit{Token}(p_i)\}$, c'est-à-dire la somme des distances à p_0 des processus nonracines détenant un jeton. Par définition, $F \geq 0$.

Considérons alors les deux cas suivants : $F = 0$ et $F > 0$.

Cas $F = 0$, exemple



Cas $F = 0$

Par définition, si $F = 0$, alors $S = \emptyset$, c'est-à-dire aucun processus nonracine ne détient un jeton.

Cas $F = 0$

Par définition, si $F = 0$, alors $S = \emptyset$, c'est-à-dire aucun processus nonracine ne détient un jeton.

Par conséquent, p_0 est l'unique processus détenant un jeton d'après le lemme 1.

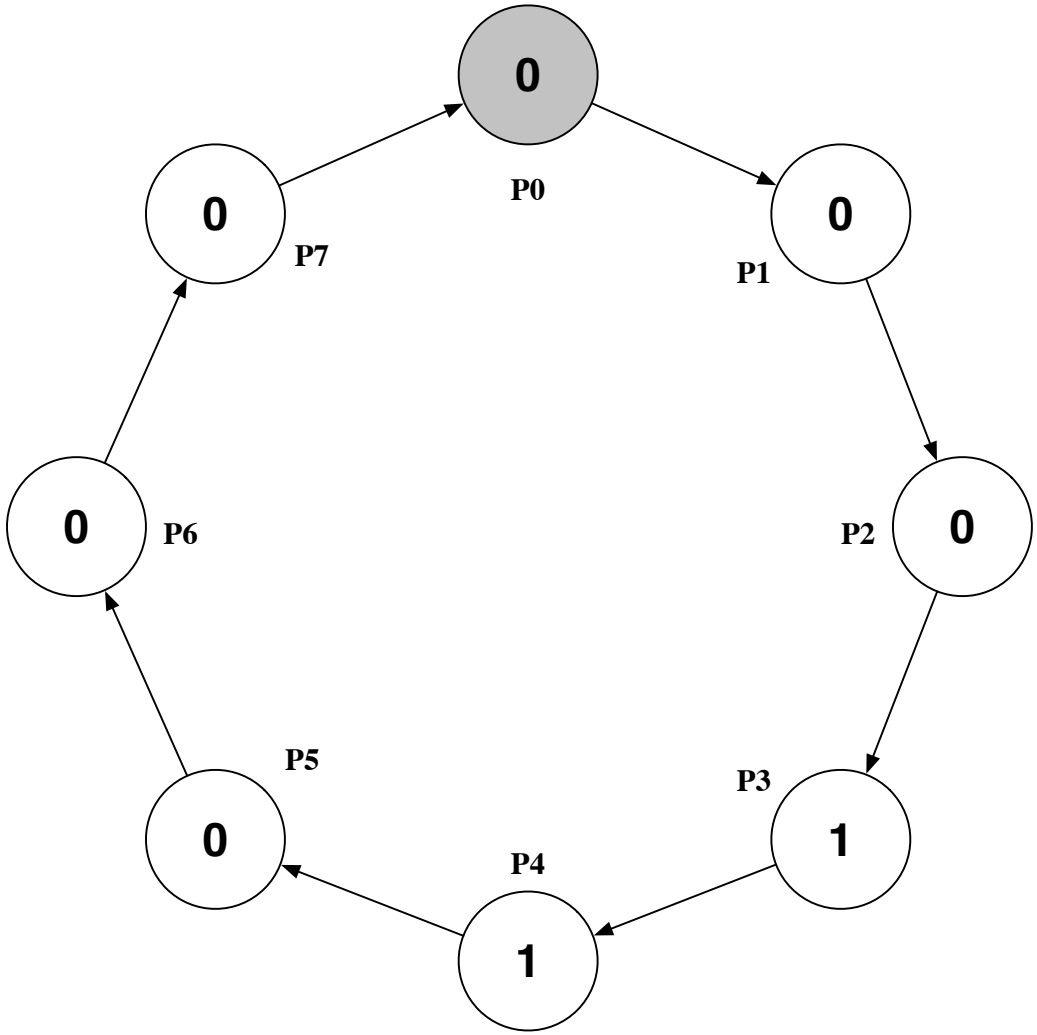
Cas $F = 0$

Par définition, si $F = 0$, alors $S = \emptyset$, c'est-à-dire aucun processus nonracine ne détient un jeton.

Par conséquent, p_0 est l'unique processus détenant un jeton d'après le lemme 1.

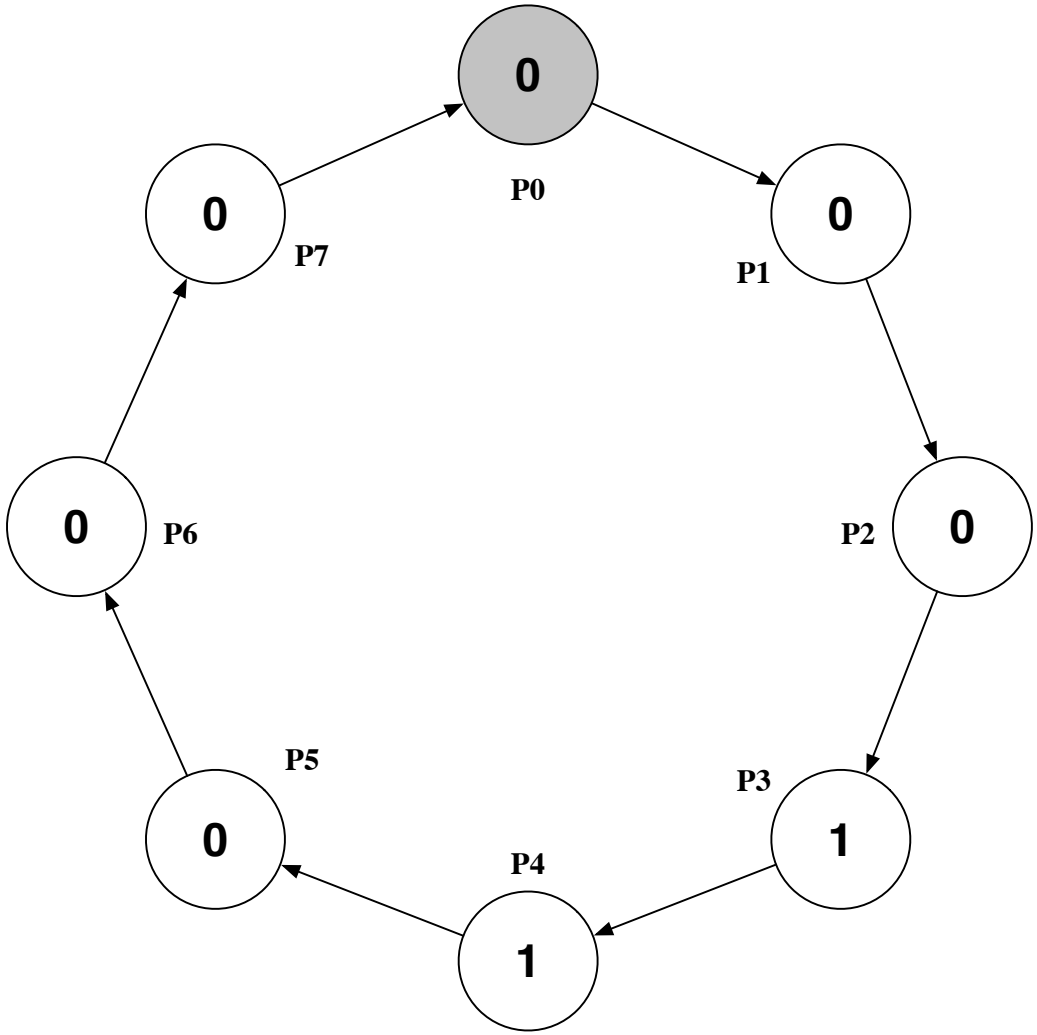
Ainsi, p_0 est le seul processus activable et par suite, il exécute une action lors du pas suivant, contradiction.

Cas $F > 0$, exemple



Valeur de F ?

Cas $F > 0$, exemple



Valeur de F ? 8

Cas $F > 0$

Dans ce cas, à chaque pas de calcul, la valeur de F diminue d'au moins 1 (voir plus dans le cas où un jeton disparaît).

Cas $F > 0$

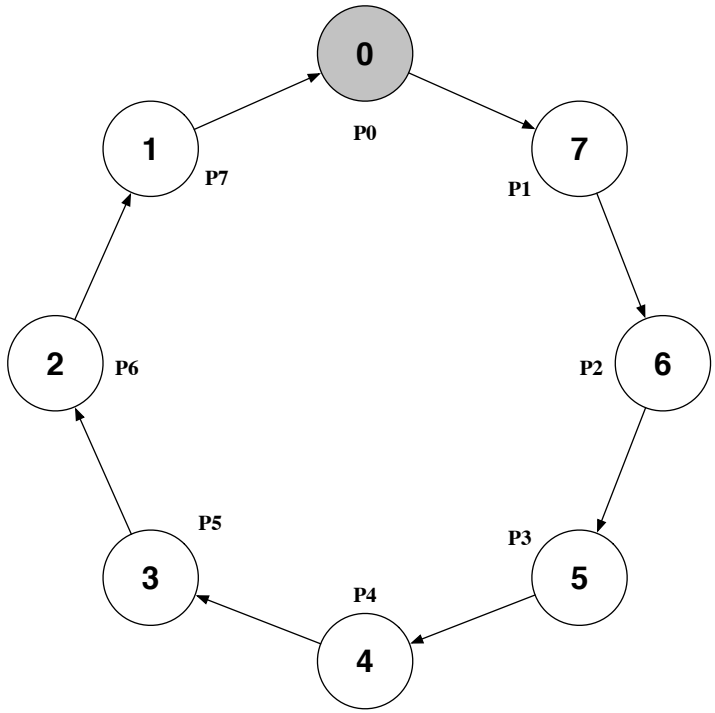
Dans ce cas, à chaque pas de calcul, la valeur de F diminue d'au moins 1 (voir plus dans le cas où un jeton disparaît).

Or, la valeur maximum de F est bornée par $\frac{n(n-1)}{2}$.

Cas $F > 0$

Dans ce cas, à chaque pas de calcul, la valeur de F diminue d'au moins 1 (voir plus dans le cas où un jeton disparaît).

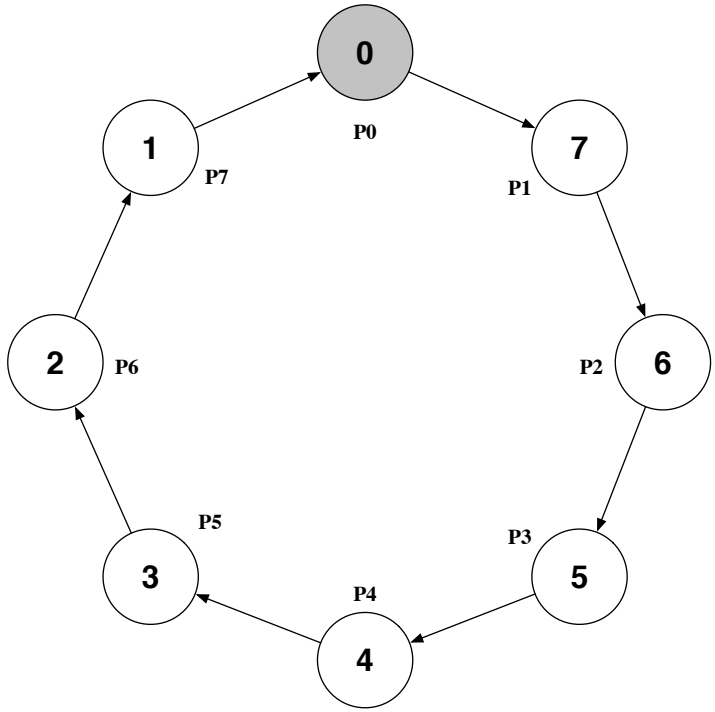
Or, la valeur maximum de F est bornée par $\frac{n(n-1)}{2}$.



Cas $F > 0$

Dans ce cas, à chaque pas de calcul, la valeur de F diminue d'au moins 1 (voir plus dans le cas où un jeton disparaît).

Or, la valeur maximum de F est bornée par $\frac{n(n-1)}{2}$.



Donc, après un nombre borné de pas, on retrouve le cas précédent, contradiction.

Convergence (2/2)

Lemme 4

Toute exécution contient une configuration de $\mathcal{L}$.

Preuve. Soit γ une configuration.

Convergence (2/2)

Lemme 4

Toute exécution contient une configuration de $\mathcal{L}$.

Preuve. Soit γ une configuration.

Soit $VNR = \{v_{p_i}^{\gamma} \mid i > 0\}$, où $v_{p_i}^{\gamma}$ est la valeur de v_{p_i} dans la configuration γ.

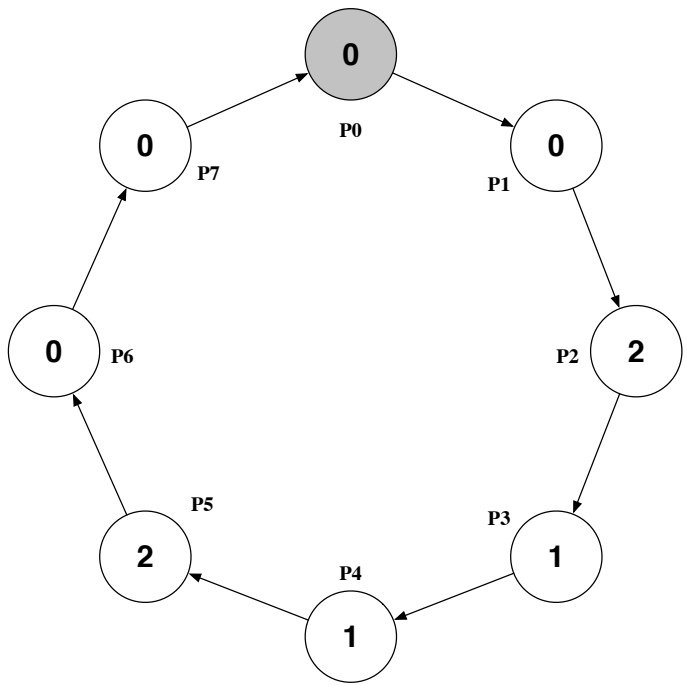
Convergence (2/2)

Lemme 4

Toute exécution contient une configuration de $\mathcal{L}$.

Preuve. Soit γ une configuration.

Soit $VNR = \{v_{p_i}^{\gamma} \mid i > 0\}$, où $v_{p_i}^{\gamma}$ est la valeur de v_{p_i} dans la configuration γ.



Valeur de VNR ?

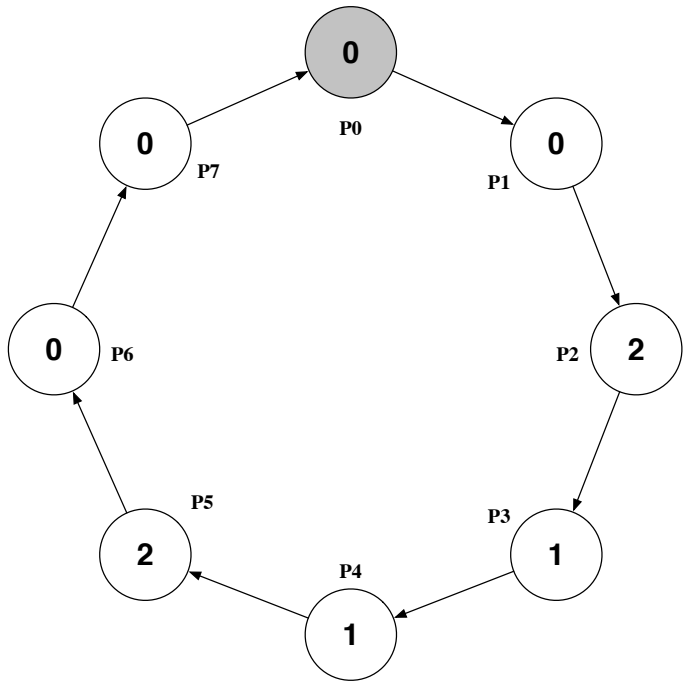
Convergence (2/2)

Lemme 4

Toute exécution contient une configuration de $\mathcal{L}$.

Preuve. Soit γ une configuration.

Soit $VNR = \{v_{p_i}^{\gamma} \mid i > 0\}$, où $v_{p_i}^{\gamma}$ est la valeur de v_{p_i} dans la configuration γ.



Valeur de VNR ? $\{0, 1, 2\}$

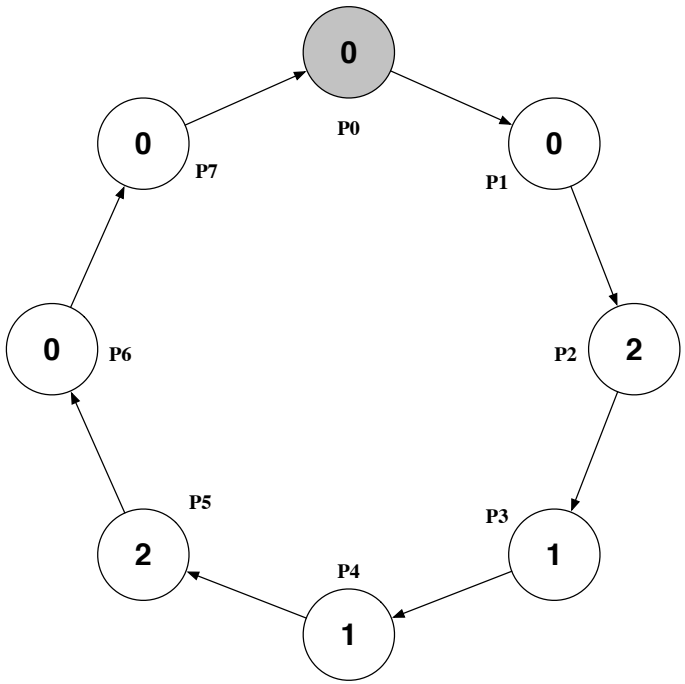
Convergence (2/2)

Lemme 4

Toute exécution contient une configuration de $\mathcal{L}$.

Preuve. Soit γ une configuration.

Soit $VNR = \{v_{p_i}^{\gamma} \mid i > 0\}$, où $v_{p_i}^{\gamma}$ est la valeur de v_{p_i} dans la configuration γ.



Valeur de VNR ? $\{0, 1, 2\}$

$|VNR| < n$.

Convergence (2/2)

Lemme 4

Toute exécution contient une configuration de $\mathcal{L}$.

Preuve. Soit γ une configuration.

Soit $VNR = \{v_{p_i}^{\gamma} \mid i > 0\}$, où $v_{p_i}^{\gamma}$ est la valeur de v_{p_i} dans la configuration γ.

$|VNR| < n$.

Puisque p_0 exécute une infinité d'actions (lemme 3) où il incrémente sa valeur modulo K (par définition de l'algorithme) et que le nombre d'éléments du domaine de v (au moins $K \geq n$) est supérieur à $|VNR|$, l'exécution atteint nécessairement une configuration où $v_{p_0} \notin VNR$.

Convergence (2/2)

Lemme 4

Toute exécution contient une configuration de $\mathcal{L}$.

Preuve. Soit γ une configuration.

Soit $VNR = \{v_{p_i}^{\gamma} \mid i > 0\}$, où $v_{p_i}^{\gamma}$ est la valeur de v_{p_i} dans la configuration γ.

$|VNR| < n$.

Puisque p_0 exécute une infinité d'actions (lemme 3) où il incrémente sa valeur modulo K (par définition de l'algorithme) et que le nombre d'éléments du domaine de v (au moins $K \geq n$) est supérieur à $|VNR|$, l'exécution atteint nécessairement une configuration où $v_{p_0} \notin VNR$.

À partir de cette configuration, la prochaine fois que p_0 sera activable (cela arrivera d'après le lemme 3), le système se retrouvera dans une configuration où tous les processus auront la même valeur dans leur variable v.

Convergence (2/2)

Lemme 4

Toute exécution contient une configuration de $\mathcal{L}$.

Preuve. Soit γ une configuration.

Soit $VNR = \{v_{p_i}^{\gamma} \mid i > 0\}$, où $v_{p_i}^{\gamma}$ est la valeur de v_{p_i} dans la configuration γ.

$|VNR| < n$.

Puisque p_0 exécute une infinité d'actions (lemme 3) où il incrémente sa valeur modulo K (par définition de l'algorithme) et que le nombre d'éléments du domaine de v (au moins $K \geq n$) est supérieur à $|VNR|$, l'exécution atteint nécessairement une configuration où $v_{p_0} \notin VNR$.

À partir de cette configuration, la prochaine fois que p_0 sera activable (cela arrivera d'après le lemme 3), le système se retrouvera dans une configuration où tous les processus auront la même valeur dans leur variable v.

Par définition, une telle configuration appartient à $\mathcal{L}$. □

Conclusion

D'après, les lemmes 2, 4 et la remarque 1, nous avons :

Théorème 1

L'algorithme de Dijkstra est autostabilisant pour la circulation de jeton dans un anneau orienté enraciné sous l'hypothèse d'un démon distribué inéquitable.

Complexité (1/4)

$F = \Sigma_{i \in S}(n - i)$, où $S = \{i \mid i \neq 0 \wedge \mathit{Token}(p_i)\}$

Complexité (1/4)

$F = \Sigma_{i \in S}(n - i)$, où $S = \{i \mid i \neq 0 \wedge \mathit{Token}(p_i)\}$

- Lorsque $F = 0$, p_0 est le seul processus activable
- Si $F > 0$, alors F décroît à chaque étape et F est bornée par $\frac{n(n-1)}{2}$

Complexité (1/4)

$F = \Sigma_{i \in S}(n - i)$, où $S = \{i \mid i \neq 0 \wedge \mathit{Token}(p_i)\}$

- Lorsque $F = 0$, p_0 est le seul processus activable
- Si $F > 0$, alors F décroît à chaque étape et F est bornée par $\frac{n(n-1)}{2}$

p_0 incrémente v_{p_0} après au plus $\frac{n(n-1)}{2}$ étapes des autres processus

Complexité (2/4)

Après $n-1$ incrémentations, v_{p_0} a pris n valeurs différentes.

Complexité (2/4)

Après $n-1$ incrémentations, v_{p_0} a pris n valeurs différentes.

Donc, au moins une valeur différente de toutes celles présentes ailleur dans le système.

Complexité (2/4)

Après $n-1$ incrémentations, v_{p_0} a pris n valeurs différentes.

Donc, au moins une valeur différente de toutes celles présentes ailleur dans le système.

Donc, lorsque p_0 devient activable pour effectuer sa $n^{\text{ème}}$ incrémentation, le système est nécessairement dans une configuration légitime.

Complexité (2/4)

Après $n-1$ incrémentations, v_{p_0} a pris n valeurs différentes.

Donc, au moins une valeur différente de toutes celles présentes ailleur dans le système.

Donc, lorsque p_0 devient activable pour effectuer sa $n^{\text{ème}}$ incrémentation, le système est nécessairement dans une configuration légitime.

D'où, le temps de stabilisation est inférieur ou égal à $n \times \frac{n(n-1)}{2} + n - 1 = O(n^3)$ étapes de calcul.

Complexité (2/4)

Après $n-1$ incrémentations, v_{p_0} a pris n valeurs différentes.

Donc, au moins une valeur différente de toutes celles présentes ailleur dans le système.

Donc, lorsque p_0 devient activable pour effectuer sa $n^{\text{ème}}$ incrémentation, le système est nécessairement dans une configuration légitime.

D'où, le temps de stabilisation est inférieur ou égal à $n \times \frac{n(n-1)}{2} + n - 1 = O(n^3)$ étapes de calcul.

Cette borne peut être réduite à $O(n^2)$ étapes de calcul. D'où un pire des cas en $\Theta(n^2)$ étapes de calcul (*cf.*, TD pour un pire cas).

Complexité (3/4)

Temps de stabilisation en rondes :

Complexité (3/4)

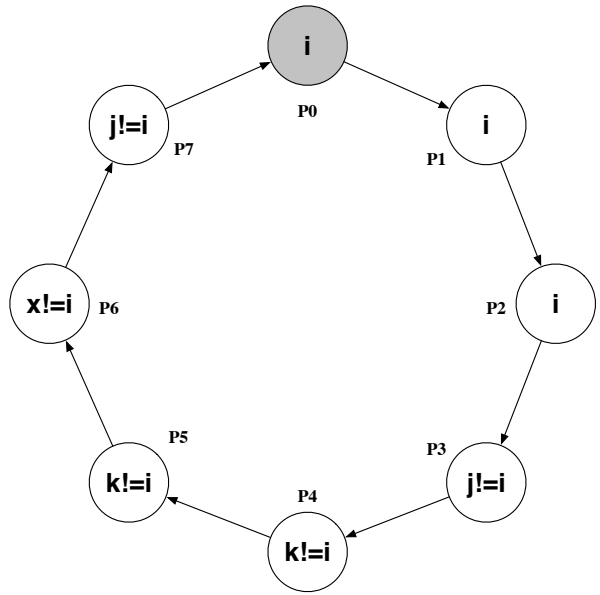
Temps de stabilisation en rondes :

- A chaque ronde, chaque jeton avance ou disparaît (cependant, au moins un ne disparaît jamais).

Complexité (3/4)

Temps de stabilisation en rondes :

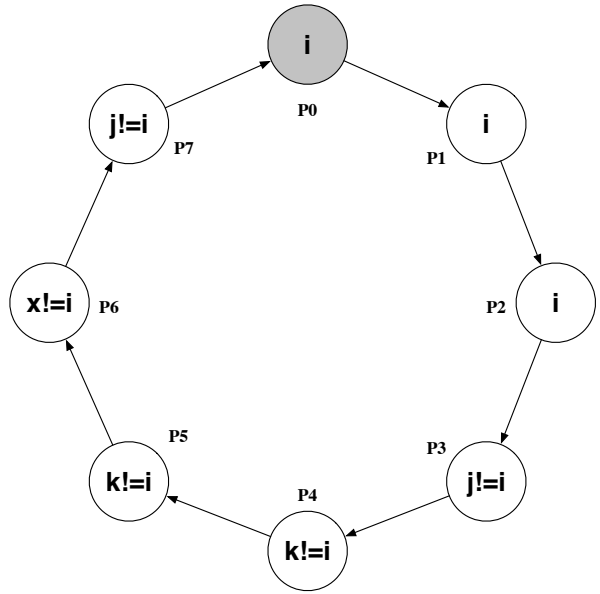
- A chaque ronde, chaque jeton avance ou disparaît (cependant, au moins un ne disparaît jamais).
- En au plus $n - 1$ rondes, le système atteint une configuration ≪ convexe ≫.



Complexité (3/4)

Temps de stabilisation en rondes :

- A chaque ronde, chaque jeton avance ou disparaît (cependant, au moins un ne disparaît jamais).
- En au plus $n - 1$ rondes, le système atteint une configuration ≪ convexe ≫.

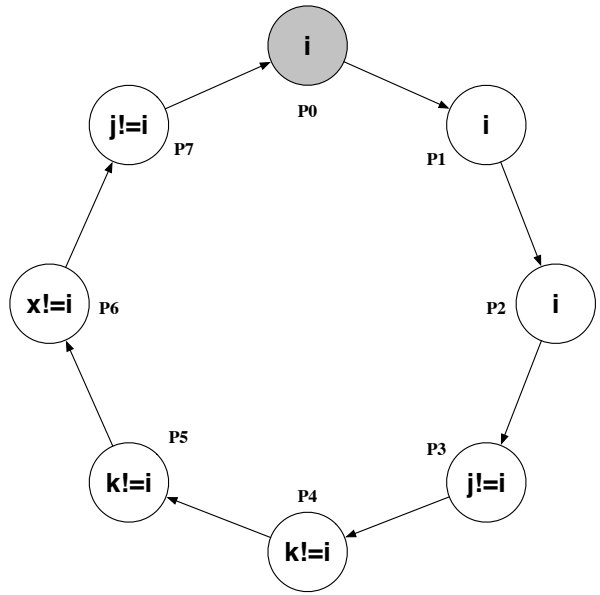


- Au plus $n - 2$ rondes plus tard (le temps nécessaire pour propager la valeur de la racine dans tout le système, sauf son prédécesseur), le système a stabilisé.

Complexité (3/4)

Temps de stabilisation en rondes :

- A chaque ronde, chaque jeton avance ou disparaît (cependant, au moins un ne disparaît jamais).
- En au plus $n - 1$ rondes, le système atteint une configuration « convexe ».



- Au plus $n - 2$ rondes plus tard (le temps nécessaire pour propager la valeur de la racine dans tout le système, sauf son prédécesseur), le système a stabilisé.

D'où, un temps de stabilisation en au plus $2n - 3$ rondes.

Complexité (4/4)

	$p_0.v$	$p_1.v$	$p_2.v$	$p_3.v$	$p_4.v$
1 :	0	3	2	1	0
2 :	1	0	3	2	1
3 :	2	1	0	3	2
4 :	3	2	1	0	3
5 :	4	3	2	1	0
6 :	4	4	3	2	1
7 :	4	4	4	3	2
8 :	4	4	4	4	3

FIGURE: Pire des cas (synchrone) pour $K \geq n = 5$

Plan

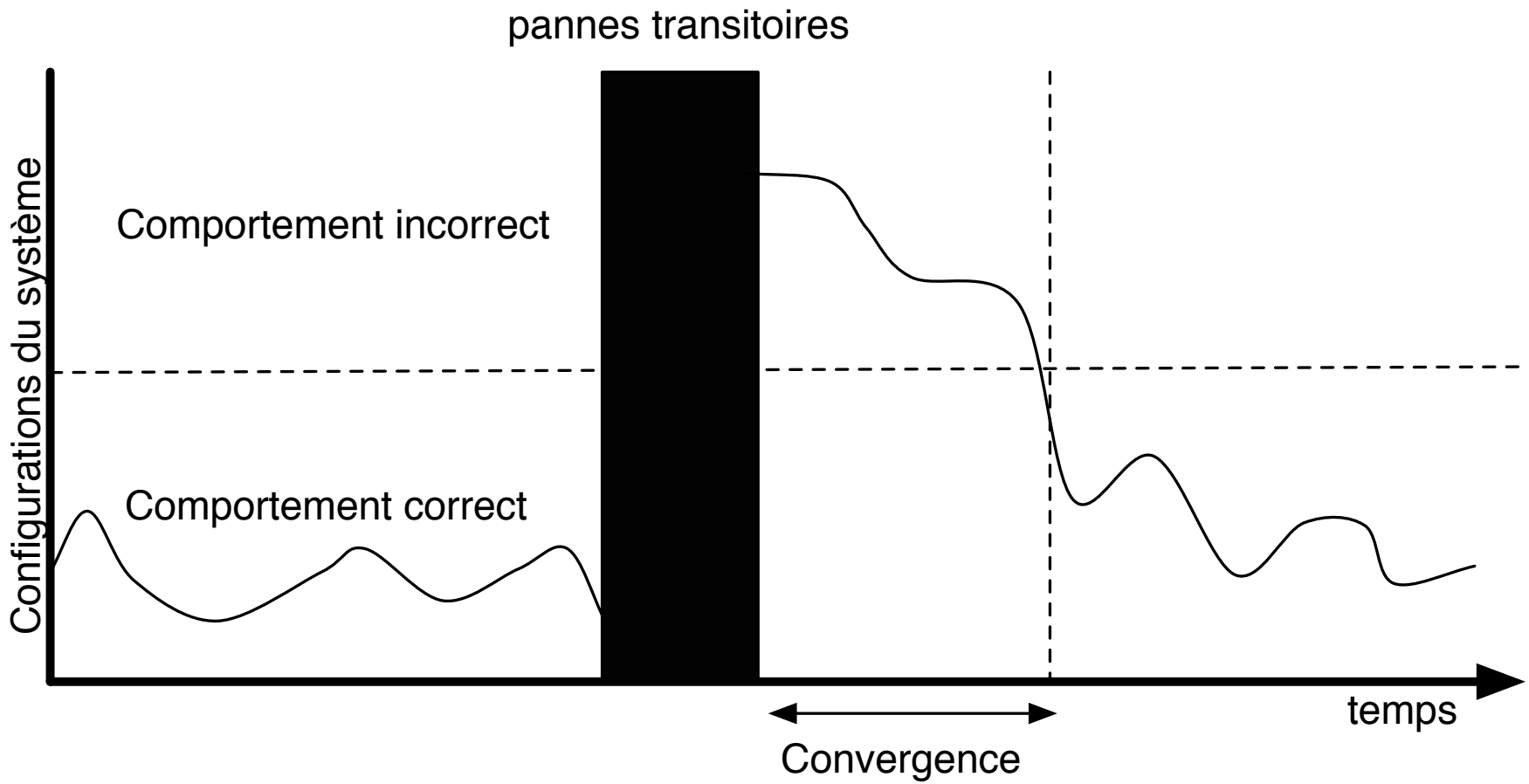
Intérêt fondamental : tolérance aux fautes transitoires

Une faute transitoire est une faute non définitive qui altère le contenu du composant du réseau (processus ou canal de communication) où elle se produit.

Une autre caractéristique importante des fautes transitoires est qu'elles sont supposées **rares** par opposition aux fautes intermittentes qui sont aussi non définitives mais supposées fréquentes.

Par exemple, une perte de message non-fréquente ou la corruption d'une partie de la mémoire locale d'un processus peuvent être considérées comme des fautes transitoires.

Intérêt fondamental : tolérance aux fautes transitoires



Important : On suppose que les fautes transitoires n'altèrent pas le code de l'algorithme.

Autres avantages

Pas besoin d'initialisation : Dans un système distribué, cette phase est critique, en particulier lorsque le réseau est un système à large-échelle où des milliers de nœuds peuvent être géographiquement distants.

Autres avantages

Pas besoin d'initialisation : Dans un système distribué, cette phase est critique, en particulier lorsque le réseau est un système à large-échelle où des milliers de nœuds peuvent être géographiquement distants.

Tolérance à la dynamicité : De nombreux algorithmes autostabilisants, notamment les algorithmes de calcul de tables de routages ou d'arbres couvrants, tolèrent une certaine dynamicité du réseau au cours de l'exécution, **à condition que cette dynamicité soit non silencieuse et de fréquence peu élevée.**

La dynamicité se définit en termes d'ajouts et/ou suppressions de nœuds et/ou de canaux de communications.

Plan

Approche non masquante

Les algorithmes autostabilisants ne masquent pas l'effet des fautes qu'ils subissent.

Ainsi, les fautes transitoires provoquent une perte de sûreté temporaire : aucune garantie n'est *a priori* donnée sur les calculs effectués durant la phase de stabilisation.

Ainsi, l'objectif est de concevoir des algorithmes offrant un temps de stabilisation le plus petit possible.

Type de fautes

Tout algorithme autostabilisant tolère les fautes transitoires.

Type de fautes

Tout algorithme autostabilisant tolère les fautes transitoires.

En revanche, la plupart des algorithmes autostabilisants ne sont pas conçus pour tolérer d'autres types de fautes, notamment lorsque les fautes sont définitives (arrêt, comportement byzantin) ou intermittentes.

De ce fait, la plupart des algorithmes autostabilisants deviennent totalement inopérants en présence de telles fautes.

Non détection de la stabilisation

À part dans des cas très simples, un processus ne peut pas localement décider si le système est dans une configuration légitime.

Conclusion

Pour plus d'info :