Réseau de Petri : modélisation et propriétés
Corrigé

Exercice 1 On considère le réseau de Petri donné à la figure 1.

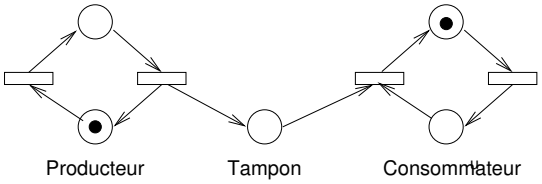


FIGURE 1 – Un exemple de réseau de Petri

1.1. Lister les différents éléments qui constituent ce réseau de Petri et rappeler leur signification.
Solution : D'après la figure 1, on identifie au moins :

- place
- transition
- arc
- nb de jetons d'une place et marquage (nombre de jetons des différentes places)
- nom (pour les places et les transitions)

En lisant l'annexe, on constate qu'il y a d'autres concepts :

- poids : on peut préciser un nombre de jetons sur les arcs
- `read_arc` : les arcs en lecture seule
- le temps : un temps min (0 par défaut) et un temps max (infini) par défaut indique dans quel intervalle de temps la transition peut être tirée.

Quand on a dit ça, nous n'avons pas tout dit pour autant. Par exemple, parler d'arc n'est pas suffisant. On a déjà précisé la notion de poids. Il faut aussi signaler qu'un arc relie une place et une transition.

1.2. Faire évoluer le réseau de Petri en précisant à chaque étape les transitions sensibilisées.
Solution : Il s'agit d'identifier les transitions franchissables. Il faut que pour chaque place d'entrée de la transition contiennent un nombre de jetons au moins égal au poids de l'arc qui la relie à cette transition.

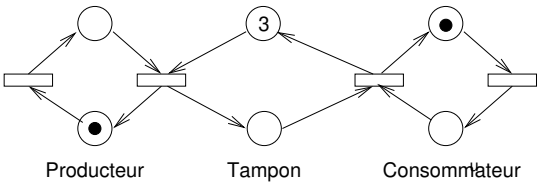
On peut alors en franchir une (toujours une seule transition à la fois). On constate qu'un nombre de jetons égal au poids des arcs sont enlevés des places d'entrées et un nombre de jeton égale au poids de l'arc est ajouté dans les places sortantes.
Remarque : Dans le cas d'un *read arc* (arc en lecture), on n'enlève pas de jeton de la place d'entrée. Il doit bien sûr y avoir un nombre suffisant de jetons dans cette place pour que la transition soit sensibilisée.

1.3. Expliquer pourquoi la place nommée *Tampon* est non bornée (elle peut contenir un nombre quelconque de jetons).

Proposer une évolution du réseau de Petri qui garantit que la place *Tampon* n'aura jamais plus de 3 jetons.

Solution : Il est certainement intéressant de voir comment cette place *Tampon* peut être interprétée. Il s'agit de la quantité de pièces (?) que peut produire le producteur sans qu'elles soient consommées par le consommateur.

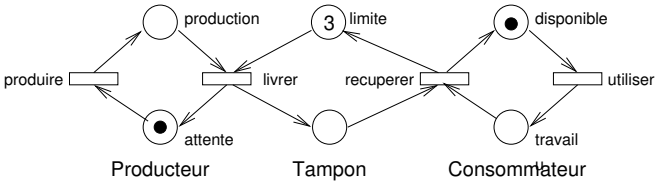
Voici une solution qui permet de borner le réseau : on limite le nombre de pièces que l'on pourra produire sans qu'elles soient livrées.



1.4. Construire le graphe de marquage pour ce réseau de Petri.

Solution : Partant du marquage initial (que l'on peut nommer M_0), il s'agit de regarder tous les transitions franchissables et de construire les états atteints après franchissement de ces transitions. On recommence pour les nouveaux états.

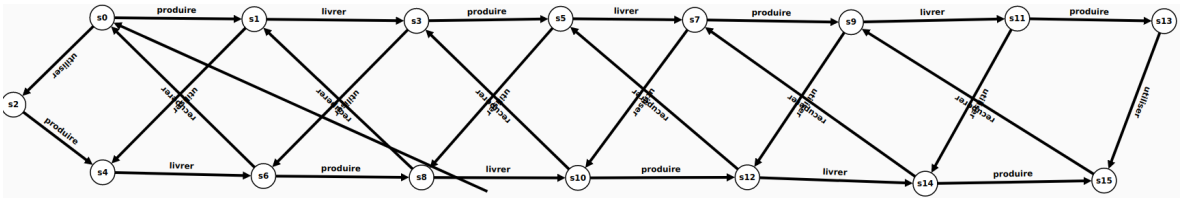
On peut commencer par nommer les transitions (produire et livrer côté Producteur et récupérer et utiliser côté Consommateur). On utilisera les abréviations p, l, r et u.



Pour représenter un état, on note le marquage (le nombre de jetons dans chaque place). Par exemple, ici les 6 places étant représentées sous la forme d'une matrice 2 lignes 3 colonnes, on pourrait représenter l'état initial par $\begin{pmatrix} 0 & 3 & 1 \\ 1 & 0 & 0 \end{pmatrix}$. La notation classique en réseau de Petri serait d'utiliser le nom des places. En utilisant l'initial en majuscule du nom de la place, on écrirait $A + L * 3 + D$.

Ici, on peut mettre en ligne la partie producteur (alternance transition p et l). On aura 4 p et 3 l. De chacun de ces états, on peut faire u (nouvel état sur une ligne en dessous, un peu à gauche de l'état source. Depuis cet état on peut faire r, on remonte alors sur l'état de la première ligne, à gauche. Sur la deuxième ligne on retrouve l'alternance p et l.

On arrive alors à un graphe de marquage qui a l'allure suivante (produit via APO) où le grand trait en travers est là pour pointer l'état initial $s0$:



Exercice 2 : Modèle de processus

Nous nous intéressons à des modèles de processus composés d'activités et de dépendances entre ces activités. Nous souhaitons pouvoir vérifier la cohérence d'un tel modèle de processus, en particulier savoir si le processus décrit peut se terminer ou non.

Le modèle de procédé utilisé est inspiré de SPEM [1], standard de l'OMG. Nous donnons entre

1. `http://www.omg.org/spec/SPEM/2.0/`

parenthèses le vocabulaire utilisé par cette norme. Dans un premier temps, nous nous intéressons à des processus simples composés seulement d'activités (WorkDefinition) et de dépendances (WorkSequence). La figure 2 donne un exemple de processus qui comprend quatre activités : Conception, RédactionDoc, Programmation et RédactionTests. Les activités sont représentées par des ovales. Les arcs entre activités représentent les dépendances. Une étiquette permet de préciser la nature de la dépendance sous la forme « étatToAction » qui précise l'état qui doit être atteint par l'activité source pour réaliser l'action sur l'activité cible. Par exemple, on ne peut commencer RédactionTests que si Conception est commencée. On ne peut commencer Programmation que si Conception est terminée. On ne peut terminer RédactionTests que si Programmation est terminé.

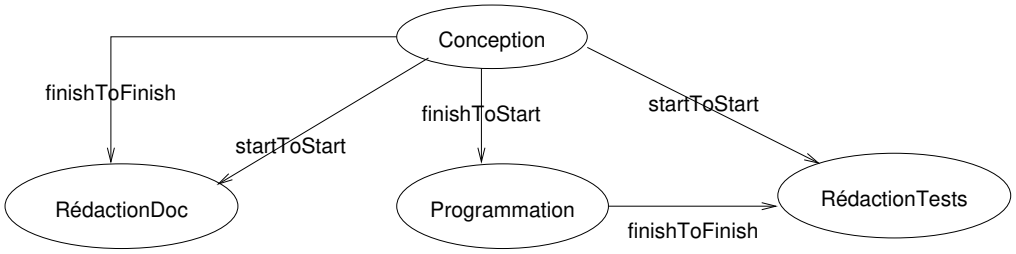


FIGURE 2 – Exemple de modèle de procédé

2.1. On considère les trois (modèle de) processus suivants :

1. `A1 --f2s--> A2`
2. `A1 --s2f--> A2`
3. `A1 --s2f--> A2 et A0 --f2f--> A2`

2.1.1. Proposer, pour chacun de ces modèles, plusieurs scénarios qui montrent que le processus peut se terminer (et que donc toutes ses activités se terminent). Il faut bien sûr que les contraintes de dépendance soient respectées.

Solution : Il faut savoir ce qui nous intéresse concernant les activités. C'est un problème d'abstraction : on ne veut pas connaître tous leur détails.

Ici, d'après la description du problème, en particulier les dépendances, on a besoin de savoir si une activité est commencée (started) ou terminée (finished). Ceci conditionnera ce qu'on pourra faire sur une activité, la commencer (start) ou la terminer (finish) en fonction de ses dépendances vis à vis des autres activités. Ces informations sont bien présentes dans la forme d'une dépendance *étatToAction*.

Notre scénario sera donc constitué de ces actions qui auront pour cibles les activités de notre processus. Le but est qu'à la fin du processus toutes les activités soient terminées.

`A1 --f2s--> A2` Ici la dépendance dit qu'il faut avoir terminé A1 pour commencer A2. Un seul scénario permet donc de terminer ce processus :

```
Commencer A1
Terminer A1
Commencer A2
Terminer A2
```

`A1 --s2f--> A2` Pour ce processus, la contrainte est plus faible. Il suffit d'avoir commencé A1 pour finir A2. Voici quelques scénarios possibles.

```
Commencer A1        Commencer A1        Commencer A1        Commencer A1
Commencer A2        Commencer A2        Commencer A2        Terminer A1
Terminer A2         Terminer A1         Terminer A2         Commencer A2
Terminer A1         Terminer A2         Terminer A1         Terminer A2
```

A1 --s2f--> A2 et A0 --f2f--> A2 Voici quelques scénarios possibles :

```
Commencer A0        Commencer A0        Commencer A1        Commencer A1
Terminer A0         Terminer A0         Terminer A1         Commencer A0
Commencer A1        Commencer A1        Commencer A0        Terminer A0
Commencer A2        Terminer A1         Terminer A0         Commencer A2
Terminer A1         Commencer A2        Commencer A2        Terminer A2
Terminer A2         Terminer A2         Terminer A2         Terminer A1
```

2.1.2. Donner également des scénarios d'erreur qui montrent soit un blocage, soit un non respect des contraintes de dépendance.

Solution :

Dans cette version simple des modèles de processus, on ne peut pas avoir de blocage en tant que tel. On pourrait en avoir si on ajoutait des ressources nécessaires pour réaliser une activité. L'utilisation des ressources par une première activité pourraient empêcher une autre activité de commencer et donc bloquer le processus.

A1 --f2s--> A2 Dans la suite, on mettra « erreur » sur la première ligne du scénario qui fait que le scénario est invalide.

```
Commencer A2  -- erreur     Commencer A1                Commencer A1
Terminer A2                 Commencer A2  -- erreur     Commencer A2  -- erreur
Commencer A1                Terminer A2                 Terminer A1
Terminer A1                 Terminer A1                 Terminer A2
```

A1 --s2f--> A2 Pour ce processus, la contrainte est plus faible. Il suffit d'avoir commencé A1 pour finir A2. Voici quelques scénarios possibles.

```
Commencer A2  -- erreur     Commencer A2  -- erreur     Terminer A1   -- erreur
Commencer A1                Terminer A2                 Commencer A2
Terminer A2                 Commencer A1                Terminer A2
Terminer A1                 Terminer A1                 Commencer A1
```

A1 --s2f--> A2 et A0 --f2f--> A2 Voici quelques scénarios d'erreur possibles :

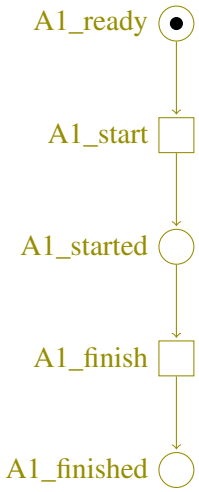
```
Commencer A0                Commencer A0                Commencer A0
Commencer A1                Terminer A0                 Terminer A0
Terminer A1                 Commencer A2  -- erreur     Commencer A2  -- erreur
Commencer A2  -- erreur     Commencer A1                Terminer A2
Terminer A0                 Terminer A1                 Commencer A1
Terminer A2                 Terminer A2                 Terminer A1
```

2.2. Proposer une traduction en réseau de Petri des trois modèles de processus précédents.

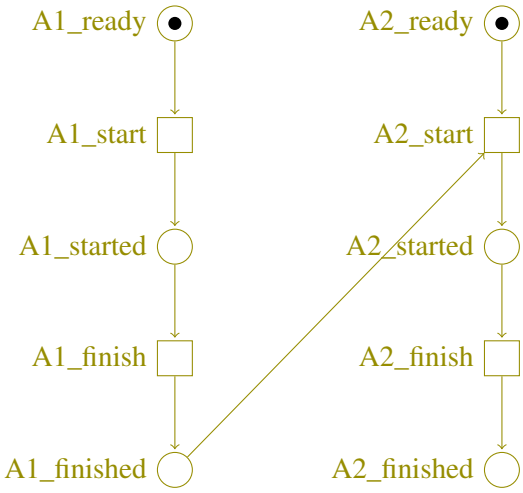
On essaiera de proposer une traduction systématique des éléments d'un processus (activité et dépendance) pour être capable de traduire tout modèle de processus en un réseau de Petri.

Solution : On remarque qu'il y a deux actions que l'on peut faire sur un activité : commencer (start) et terminer (finish). On peut en faire des transitions. Il faut des places entre ces transitions qui correspondront à des états des activités. On pourrait appeler ces états prête (ready), commencée (started) et terminée (finished). Bien entendu ces transitions et ces places doivent être définies pour chaque activité. On peut donc les préfixer (ou suffixer) par le nom de l'activité (qui devrait être unique pour assurer l'unicité des places et des transitions).

On obtient alors, pour A1, le réseau de Petri suivant :



Pour représenter le premier exemple de processus, il suffit donc de construire le motif d'une activité pour A1 et pour A2. L'activité A2 ne peut commencer que quand l'activité A1 est terminée. On pourrait donc mettre un arc entre la place finished de A1 et la transition start de A2.

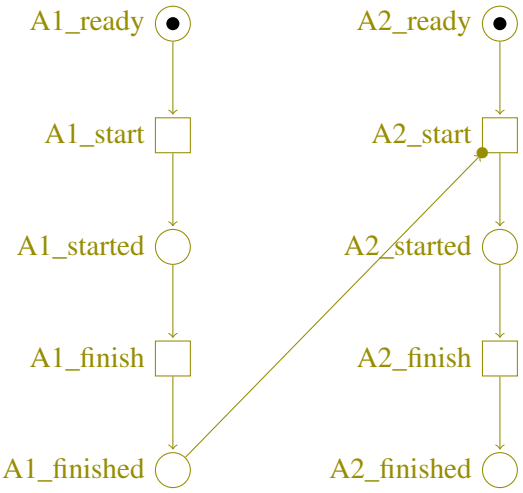


Ceci pose un problème si on considère le processus suivant : `A1 --f2s--> A2 et A0 --f2f--> A3`. En effet, pour commencer A1 (ou finir A3), on va enlever le jeton dans la place finished de A1. L'autre activité ne pourra alors plus évoluer : on ne pourra pas finir A3 (ou commencer A1).

Une solution serait de remettre le jeton. Si on faisait des réseaux temporel, ceci aurait pour effet de désensibiliser les transitions qui ont cette place en entrée et les re-sensibiliser, le temps repartant à 0.

L'autre solution, la bonne ici, est d'utiliser un arc en lecture (*read arc*). Il faut le bon nombre de jetons dans la place d'entrée mais ils ne sont pas enlevés de la place d'entrée quand la transition est exécutée.

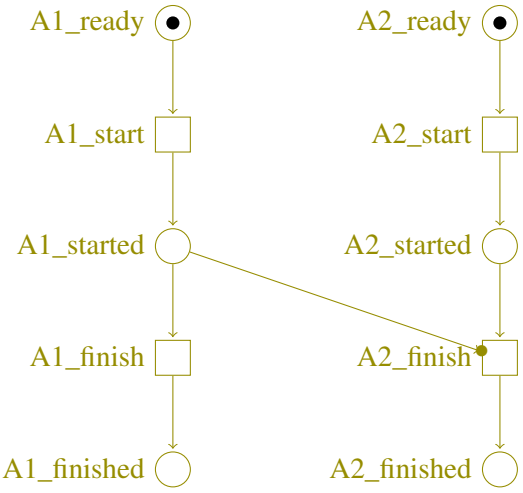
Voici donc la bonne solution :



Elle comporte cependant encore un défaut. Considérons le deuxième exemple : `A1 --s2f--> A2` et le scénario suivant :

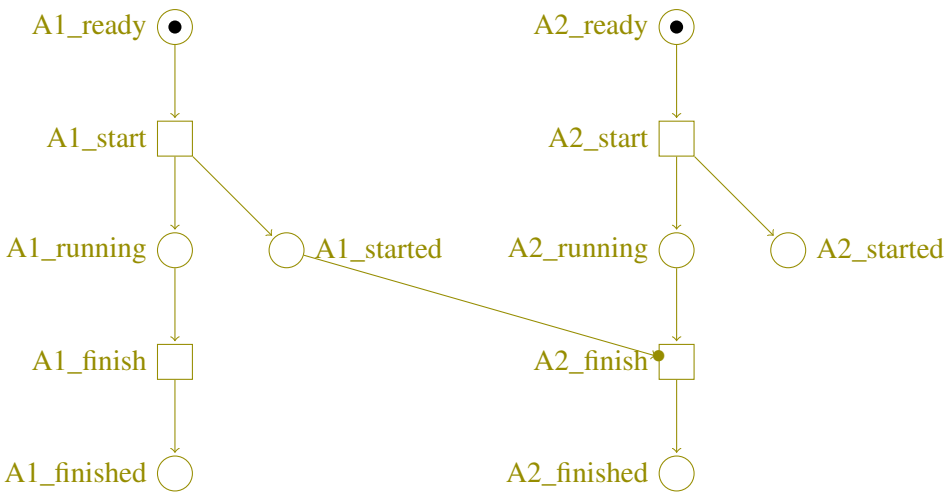
```
Commencer A1
Terminer A1
Commencer A2
Terminer A2
```

La modélisation nous donne :



Mais le scénario ne fonctionne pas puisqu'au moment de faire « Commencer A2 » le jeton n'est plus dans la place *started* mais dans la place *finished*.

En fait, le nom choisi pour la place *started* n'est pas le bon. Un meilleur nom serait *running* (l'activité est en cours). On a besoin de savoir si une activité a été commencée. On peut donc ajouter une place à côté de *running*, appelée *started* qui mémorisera que l'activité a été commencée. Voici le réseau final pour notre exemple.



2.3. L'exécution d'un réseau de Petri correspond à une séquence d'états caractérisés par le marquage du réseau. Le passage d'un état au suivant immédiat correspond à une transition unique du réseau. Donner des contraintes sur le marquage d'un réseau de Petri pour que les propriétés suivantes qui portent sur le modèle de processus de la figure 2 soient satisfaites.

1. Le processus peut terminer.

   Solution : Le processus se termine si l'un des états du graphe de couverture est telle que toutes les activités du processus est dans l'état *finished*.

   On peut l'exprimer en LTL de la manière suivante (en définissant un opérateur op).

   ```
   op finished = A1_finished /\ A2_finished;
   <> finished;
   ```

2. Le processus ne peut pas terminer.

   Solution : C'est la négation de la propriété précédente :

   ```
   op finished = A1_finished /\ A2_finished;
   - <> finished;
   ```

   L'intérêt de cette formulation est que si l'outil de vérification de modèle (*model-checking*) trouve un état où la propriété qui est fausse, il nous le fournira comme contre-exemple ainsi que les transitions qui y ont conduit. Cette état est un état où le processus se termine et l'ordre des transitions exécutées nous donne un scénario qui permet de terminer le processus. Ce n'est pas forcément le seul bien sûr !

2.4. *Ressources.* Pour réaliser une activité, des ressources peuvent être nécessaires. Une ressource peut correspondre à un acteur humain, un outil ou tout autre élément jouant un rôle dans le déroulement de l'activité. Ici, nous nous intéressons simplement aux types de ressources nécessaires et au nombre d'occurrences d'un type de ressource. Par exemple, il peut y avoir deux développeurs, deux concepteurs, trois machines, un bloc-note, etc. Un type de ressource sera seulement caractérisé par son nom et la quantité d'occurrences de celle-ci.

Pour pouvoir être réalisée, une activité peut nécessiter plusieurs ressources (éventuellement aucune). Par exemple, l'activité *RedactionTest* nécessite un développeur (une occurrence de la ressource de type Développeur) et deux machines (deux occurrences de la ressource Machine). Les occurrences de ressources nécessaires à la réalisation d'une activité sont prises au moment de son démarrage et rendues à la fin de son exécution. Bien entendu, une même occurrence de ressource ne peut pas être

utilisée simultanément par plusieurs activités. Les types de ressource et leur nombre d'occurrences sont définis en même temps que le procédé lui-même.
Solution : Pas de solution pour cette partie. C'est le travail qu'il y aura à faire sur la partie mini-projet !
2.4.1. Modéliser la notion de ressources.
2.4.2. Les propriétés proposées à la question 2.3 doivent elles être adaptées ?
2.4.3. Donner des contraintes sur le marquage d'un réseau de Petri qui expriment que le processus peut se terminer sans utiliser une des occurrences de la ressource Développeur. Ceci signifie que l'on pourrait retirer un développeur au projet sans dommage.
2.5. On considère maintenant que chaque activité doit se dérouler dans un intervalle de temps donné précisant un temps minimal et un temps maximal. Proposer une évolution du réseau de Petri pour prendre en compte cet aspect. On utilisera les aspects temporels des réseaux de Petri.

Rappel : Réseaux de Petri

Le rappel donné ici est extrait de `http://fr.wikipedia.org/wiki/R%C3%A9seau_de_Petri`

Définition Un réseau de Petri est un tuple (S, T, F, M_0, W) où :

- S définit une ou plusieurs places.
- T définit une ou plusieurs transitions.
- F définit un ou plusieurs arcs (flèches).
- Un arc ne peut pas être connecté entre 2 places ou 2 transitions ; plus formellement : $F \subseteq (S \times T) \cup (T \times S)$.
- $M_0 : S \to \mathbb{N}$ appelé marquage initial (ou place initiale) où, pour chaque place $s \in S$, il y a $n \in \mathbb{N}$ jetons.
- $W : F \to \mathbb{N}^+$ appelé ensemble d'arcs primaires , assignés à chaque arc $f \in F$ un entier positif $n \in \mathbb{N}^+$ qui indique combien de jetons sont consommés depuis une place vers une transition, ou sinon, combien de jetons sont produis par une transition et arrivent pour chaque place.

De nombreuses définitions formelles existent. Cette définition concerne un réseau place-transition (ou P-T).

Représentation Un réseau de Petri se représente par un graphe orienté composé d'arcs reliant des places et des transitions. Deux places ne peuvent pas être reliées entre elles, ni deux transitions.

Les places peuvent contenir des jetons. La distribution des jetons dans les places est appelée le marquage du réseau de Petri.

Les entrées d'une transition sont les places desquelles part une flèche pointant vers cette transition, et les sorties d'une transition sont les places pointées par une flèche ayant pour origine cette transition.

La figure 3 propose quelques exemples de réseaux de Petri.

Dynamique d'exécution Un réseau de Petri évolue lorsqu'on exécute [2] une transition : des jetons sont pris dans les places en entrée de cette transition et envoyés dans les places en sortie de cette transition. Le nombre de jetons pris dans les places d'entrée et mis dans les places de sortie est celui

2. On emploie aussi les verbes franchir ou tirer.

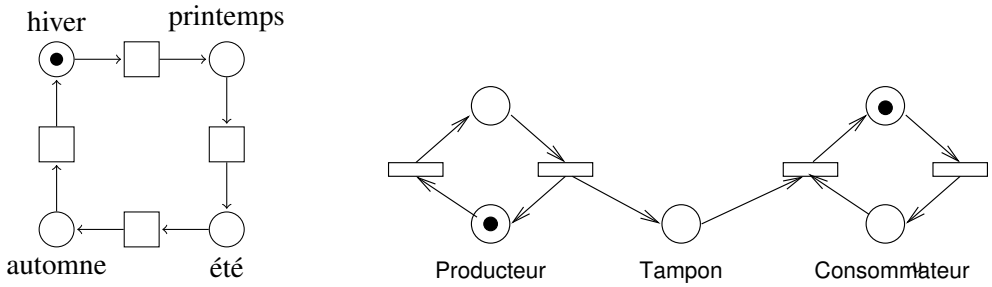


FIGURE 3 – Exemples de réseaux de Petri

indiqué par l'arc correspondant. Une transition ne peut être exécutée que si elle est exécutable [3], c'est-à-dire qu'il y a dans chaque place d'entrée un nombre de jetons au moins égal au poids de l'arc.

L'exécution d'une transition est une opération indivisible qui est déterminée par la présence de jetons sur les places d'entrée.

L'exécution d'un réseau de Petri n'est pas déterministe, car il peut y avoir plusieurs possibilités d'évolution à un instant donné.

Si chaque transition dans un réseau de Petri a exactement une entrée et une sortie alors ce réseau est un automate fini.

Extensions On définit deux extensions sur ces réseaux de Petri. Tout d'abord nous définissons une notion de temps sur les transitions sous la forme d'un intervalle $[temps_{min}, temps_{max}]$. Le temps commence à s'écouler à partir du moment où la transition est exécutable. Elle doit alors s'exécuter dans l'intervalle de temps indiqué. On parle alors de réseau de Petri temporel.

On ajoute aussi un nouveau type d'arc appelé *read-arc*. Il s'agit d'un arc qui relie nécessairement une place d'entrée à une transition. Il consiste à vérifier que la place a bien au moins le nombre de jetons indiqué sur cet arc. Si c'est le cas, la transition est exécutable. Lors de l'exécution de la transition, les jetons ne sont pas enlevés de la place d'entrée du *read-arc*. Cette notion de *read-arc* n'a de sens que dans le cas de réseau de Petri temporel. Sinon, elle pourrait être simulée par un arc remettant les jetons consommés par la transition dans la place concernée. Dans le cas d'un réseau de Petri temporel, le temps serait remis à 0 pour les transitions qui ont cette place pour entrée.

La figure 4 présente un exemple de réseau de Petri temporel avec read-arc.

FIGURE 4 – Le même réseau de Petri avec un read-arc et sans

3. On dit aussi franchissable, sensibilisée, validée ou tirable.