Chapitre 2.

Modèles non-temporisés de systèmes à événements discrets

2.1 Introduction

Développer des modèles appropriés pour décrire le comportement des SED.

Séquence des états visités (ni *quand ?* ni *pendant combien de temps ?*)

Hypothèse : SED décrit par une séquence d'événements $(e_1, e_2, \dots)$.

Les modèles non temporisés spécifient l'ordre des événements mais pas les instants d'occurrence.

Les modèles non temporisés sont pour nous une étape vers la construction de modèles temporisés.

Dans ce chapitre, description de deux classes de modèles non temporisés : ***automates à états*** et ***réseaux de Petri***.

Les automates à états permettent de décrire une bonne part des SED. Ces modèles reviennent à spécifier des ensembles d'états et des transitions entre ces états.

Les réseaux de Petri (RdP) sont plus généraux, ce qui a un coût.

Note : la vérification du logiciel est une des motivations pour le développement de modèles non temporisés des SED.

Nous verrons aussi dans ce chapitre quelques problèmes fondamentaux liés au comportement logique des SED (analyse qualitative) et des techniques pour les résoudre basées sur le formalisme des RdP.

2.2 Langages et automates

Chaque SED a un ensemble d'événements ***E*** qui lui est associé, ces événements font évoluer le SED.
L'ensemble ***E*** peut être vu comme un alphabet d'un langage et les séquences d'événements sont des mots dans ce langage.

Pour motiver la discussion des langages considérons un exemple simple :
On veut concevoir un système simple pour un photocopieur.

Quand le photocopieur est mis en marche, le système indiquerait « en marche », puis reporterait « pas de problème » ou « manque papier » ou « manque toner », et conclurait avec « rapport terminé ». Chaque message définit un événement, et tous les messages possibles définissent un alphabet.

Le système est un SED. Ce SED doit reconnaître des événements et interpréter correctement chaque séquence d'événements reçue. La séquence d'événements (« en marche », « pas de problème », « rapport terminé ») est interprétée comme normale. De même pour la séquence (« en marche », « manque papier », « rapport terminé »).
Par contre la séquence (« en marche », « rapport terminé ») n'est pas normale; la procédure de rapport sur l'état du photocopieur n'a pas fonctionné normalement.

Les combinaisons normales de messages du photocopieur (événements) sont des mots appartenant au langage du SED.
Le langage contient seulement des mots de trois événements, tous commençant par « en marche » et se terminant par « rapport terminé ».
Quand le SED voit un tel mot, il sait que la tâche est accomplie.
Quand il voit un autre mot, il sait que quelque chose est anormal.

2.2.1 Langages

Définition : Un langage ***L,*** défini sur un alphabet ***E***, est un ensemble de mots (ou chaînes) formés à partir des (étiquettes des) événements dans ***E***.

Exemples :
$\boldsymbol{E} = \{a, b, c\}$ est l'alphabet

$\boldsymbol{L_1} = \{\varepsilon, a, abb\}$ où le mot vide, de longueur 0, est noté ε
$\boldsymbol{L_2}$ = { chaînes de longueur 3 commençant par a } comporte 9 mots
$\boldsymbol{L_3}$ = { chaînes de longueur finie commençant par a } comporte un nombre infini de mots et ne peut donc pas être totalement décrit par une énumération.

Toutes les opérations communes sur les ensembles peuvent être effectuées sur les langages. Soit $\boldsymbol{E}$ un alphabet et $\boldsymbol{A}$ et $\boldsymbol{B}$ deux langages,

- $\boldsymbol{A} \cup \boldsymbol{B}$ et $\boldsymbol{A} \cap \boldsymbol{B}$ sont respectivement l'union et l'intersection de $\boldsymbol{A}$ et $\boldsymbol{B}$
- $\boldsymbol{A}^c$ est le complément de $\boldsymbol{A}$ par rapport à $\boldsymbol{E}^*$, l'ensemble de toutes les chaînes sur l'alphabet $\boldsymbol{E}$.

Seule l'opération d'union sera utile dans la suite.

Soient deux chaînes, u et v, la ***concaténation*** de ces chaînes, notée $u.v$ ou uv , est la chaîne de tous les événements dans u immédiatement suivie de la chaîne de tous les événements dans v.

Exemples :

- si $u = abb$ et $v = cb$ alors $uv = abbcb$.
- pour toute chaîne u, $u\varepsilon = \varepsilon u = u$.

La concaténation est associative, elle n'est pas commutative et elle admet le mot vide pour élément neutre.

Expressions régulières (ou rationnelles)

Munis de l'opération de concaténation sur les chaînes, on peut définir une opération de ***concaténation sur les langages*** (c.-à-d. ensembles de chaînes). Soit $\boldsymbol{E}$ un alphabet et $\boldsymbol{A}$ et $\boldsymbol{B}$ deux langages la concaténation de $\boldsymbol{A}$ et $\boldsymbol{B}$ est

$$\boldsymbol{AB} = \{w\colon\ w = uv, u \in \boldsymbol{A}, v \in \boldsymbol{B}\}$$

Maintenant que la concaténation de deux langages est définie, on peut définir une autre opération sur un langage, sa ***fermeture itérative (étoile de Kleene)***

$$\boldsymbol{A}^* = \bigcup_{j=0}^{\infty} \boldsymbol{A}^i, \text{ où } \boldsymbol{A}^0 = \{\varepsilon\} \text{ et } \boldsymbol{A}^i = \boldsymbol{A}\boldsymbol{A}^{i-1} \text{ pour tout } i \geq 1.$$

Exemple :

$$\boldsymbol{E} = \{a, b, c\},\ \boldsymbol{L}_1 = \{\varepsilon, a, abb\},\ \boldsymbol{L}_2 = \{c\},\ \boldsymbol{L}_3 = \emptyset.$$

$$\boldsymbol{L}_1 \cup \boldsymbol{L}_2 = \{\varepsilon, a, abb, c\},$$
$$\boldsymbol{L}_1\boldsymbol{L}_2 = \{c, ac, abbc\},$$
$$\boldsymbol{L}_2^* = \{\varepsilon, c, cc, ccc, \ldots\},$$
$$\boldsymbol{L}_1^* = \{\varepsilon, a, abb, aa, aabb, abba, abbabb, \ldots\},$$
$$\boldsymbol{L}_3^* = \{\varepsilon\}.$$

Exemple de calcul de $\boldsymbol{L}^*$, pour $\boldsymbol{L} = \{abc, cba\}$:

$$\boldsymbol{L}^0 = \{\varepsilon\},$$
$$\boldsymbol{L}^1 = \boldsymbol{L}\boldsymbol{L}^0 = \{abc, cba\},$$
$$\boldsymbol{L}^2 = \boldsymbol{L}\boldsymbol{L}^1 = \{abc, cba\}\{abc, cba\}$$
$$= \{abcabc, abccba, cbaabc, cbacba\},$$
$$\boldsymbol{L}^3 = \boldsymbol{L}\boldsymbol{L}^2, \text{ etc.}$$

ce qui conduit à

$$\boldsymbol{L}^* = \{\varepsilon, abc, cba, abcabc, abccba, cbaabc, cbacba, \ldots\}$$

Conventions (pour alléger la notation des expressions régulières) :

- utiliser le symbole ∨ (« ou ») à la place de l'union ∪,
- remplacer $\{u\}^*$ par $\boldsymbol{u}^*$,
- écrire simplement $\boldsymbol{uv}$ pour la concaténation $\{uv\}$ de $\{u\}$ et de $\{v\}$,
- écrire $\boldsymbol{u} \vee \boldsymbol{v}$ pour l'union $\{u\} \cup \{v\}$ de $\{u\}$ et de $\{v\}$.

Les expressions régulières sont définies récursivement selon

1. $\emptyset$ (l'ensemble vide), $\boldsymbol{\varepsilon}$ (l'ensemble $\{\varepsilon\}$), et $\boldsymbol{e}$ (l'ensemble $\{e\}$, où $e \in \boldsymbol{E}$), sont des expressions régulières,
2. si $\boldsymbol{r}$ et $\boldsymbol{s}$ sont des expressions régulières, alors $\boldsymbol{rs}$, $(\boldsymbol{r} \vee \boldsymbol{s})$, $\boldsymbol{r}^*$ et $\boldsymbol{s}^*$ sont des expressions régulières,
3. il n'y a pas d'autres expressions régulières en dehors de celles obtenues en appliquant les règles 1 et 2 un nombre fini de fois.

Définition : tout langage qui peut être décrit par une expression régulière est un ***langage régulier***.

Pour ces langages on peut ainsi remplacer des opérations sur des ensembles par des opérations algébriques sur des expressions régulières, ce qui donne souvent des expressions bien plus compactes.

Exemples :

l'expression régulière $(\boldsymbol{a} \vee \boldsymbol{b})\boldsymbol{c}^*$ représente le langage

$\boldsymbol{L} = \{a, b, ac, bc, acc, bcc, accc, bccc, \ldots\}$,

l'expression régulière $(\boldsymbol{ab})^* \vee \boldsymbol{c}$ représente le langage

$\boldsymbol{L} = \{\varepsilon, c, ab, abab, ababab, \ldots\}$.

2.2.2 Automates (à (nombre d')états) finis

Un automate est capable de reconnaître un langage selon des règles bien définies. Il est utilisé pour observer des séquences d'événements formées selon un alphabet $\boldsymbol{E}$.
Il est doté d'un ensemble d'états $\boldsymbol{X}$, un état étant désigné comme état initial x_0 . L'état courant x_k résume l'information sur le passé. Cette information est suffisante pour déterminer l'évolution future, connaissant les entrées appliquées appartenant à l'ensemble $\boldsymbol{E}$.

Définition : un ***automate fini*** est un quintuplet $(\boldsymbol{E}, \boldsymbol{X}, f, x_0, \boldsymbol{F})$
où

$\boldsymbol{E}$ est un alphabet fini,
$\boldsymbol{X}$ est un ensemble fini d'états, c'est *l'espace des états*, il est discret,
f est une *fonction de transition* (partielle si non définie pour certaines valeurs) d'état $\boldsymbol{X} \times \boldsymbol{E} \rightarrow \boldsymbol{X}$,
x_0 est un état initial, $x_0 \in \boldsymbol{X}$,
$\boldsymbol{F}$ est un ensemble d'états finaux, $\boldsymbol{F} \subseteq \boldsymbol{X}$.

Diagramme des (transitions d') états

Graphe orienté, nœuds = états, arcs = événements.
Un arc étiqueté e reliant le nœud x au nœud x' représente la transition de l'état courant x au nouvel état x' lorsque l'événement e se produit.

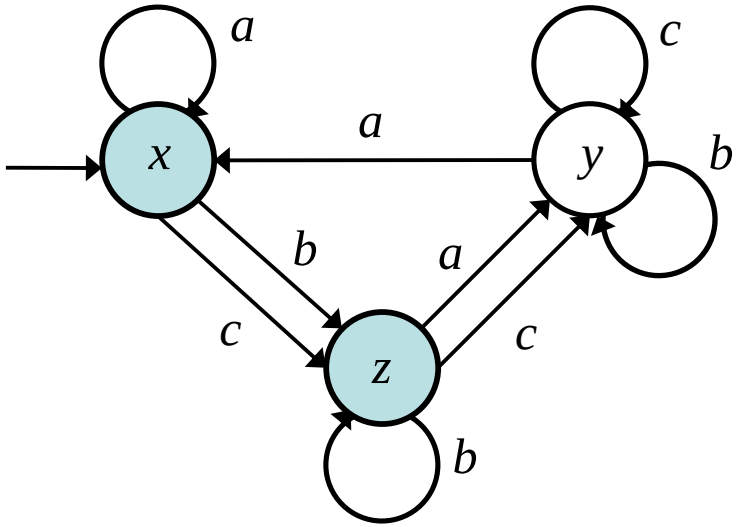
Exemple : $\boldsymbol{E} = \{a, b, c\}$, $\boldsymbol{X} = \{x, y, z\}$,

$$f(x,a) = x, \quad f(x,b) = f(x,c) = z,$$
$$f(y,a) = x, \quad f(y,b) = f(y,c) = y,$$
$$f(z,b) = z, \quad f(z,a) = f(z,c) = y,$$

$x_0 = x$, $\boldsymbol{F} = \{x, z\}$,

l'état initial est repéré par une flèche entrante

les états finaux sont appelés états marqués

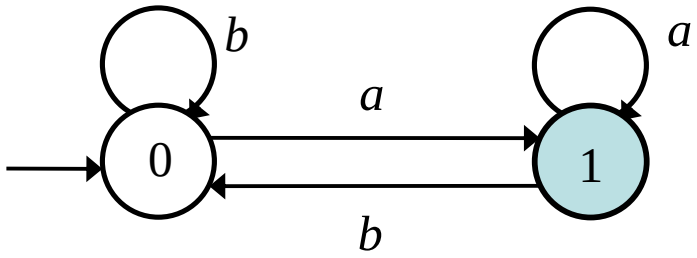


Exemple : $\boldsymbol{E} = \{a, b\}$,
l'expression régulière $(\boldsymbol{a} \vee \boldsymbol{b})^* \boldsymbol{a}$ représente le langage
$\boldsymbol{L} = \{a, aa, ba, aaa, aba, baa, bba, \ldots\}$

Ce langage peut être reconnu par l'automate fini $(\boldsymbol{E}, \boldsymbol{X}, f, x_0, \boldsymbol{F})$
où

$$\boldsymbol{E} = \{a, b\}, \quad \boldsymbol{X} = \{0, 1\},$$
$$f(0, a) = 1, \quad f(0, b) = 0,$$
$$f(1, a) = 1, \quad f(1, b) = 0,$$
$$x_0 = 0, \qquad F = \{1\}$$

En effet avec 0 comme état initial, l'état final 1 ne peut être atteint que si l'événement a se produit. Alors soit l'état reste inchangé, soit un événement b se produit et l'état retourne à 0. Dans ce dernier cas, on est revenu au point de départ et le processus se répète.
Le diagramme des transitions d'état est donc



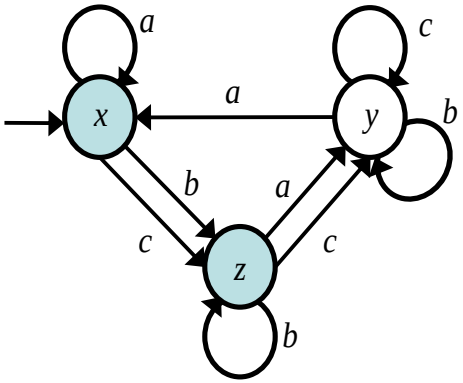
Automates pour la reconnaissance de langages

Un automate peut être vu comme un procédé permettant de reconnaître un langage.

Que signifie reconnaître, ou accepter, une chaîne ?
On généralise la fonction de transition f pour l'appliquer à une chaîne u.
On a alors $f : \boldsymbol{X} \times \boldsymbol{E}^* \to \boldsymbol{X}$, et $x' = f(x,u)$ est l'état atteint à partir de l'état x quand la chaîne u est appliquée en entrée.
Plus précisément, pour toute chaîne u et tout événement e,

$$f(x,ue) = f(f(x,u),e)$$

Exemple:



$$\begin{aligned} f(x,bba) &= f(f(x,bb),a) \\ &= f(f(f(x,b),b),a) \\ &= f(f(z,b),a) \\ &= f(z,a) \\ &= y \end{aligned}$$

Définition : une chaîne u est reconnue par un automate fini $(\boldsymbol{E}, \boldsymbol{X}, f, x_0, \boldsymbol{F})$ si $f(x_0, u) = x$, où $x \in \boldsymbol{F}$.

Définition : le langage reconnu par un automate fini $(\boldsymbol{E}, \boldsymbol{X}, f, x_0, \boldsymbol{F})$ est l'ensemble des chaînes $\{u : f(x_0, u) \in \boldsymbol{F}\}$.

Le langage reconnu par un automate A est noté $\boldsymbol{L}(A)$.

Comment utiliser les langages et les automates en pratique ?

Un problème fondamental pour les SED est, étant donné un ensemble d'événements, de concevoir un SED qui implémente certaines ***tâches***, définies par certaines séquences d'événements, dans le sens où il détecte chacune de ces séquences d'événements.
On peut procéder en deux étapes, d'abord traduire les tâches en un langage et ensuite construire un automate fini qui reconnaisse le langage.

Exemple :

Soit $\boldsymbol{E} = \{a_1, a_2, b\}$ un alphabet. Une tâche est définie comme une séquence de trois événements, commençant par b, suivi par a_1 ou a_2, puis par b, suivi par une séquence arbitraire d'événements. Ainsi on veut concevoir un automate qui lit toute séquence sur l'alphabet $\boldsymbol{E}$ mais reconnait uniquement les séquences de longueur 3 ou plus; chaque chaîne doit commencer par b et inclure a_1 ou a_2, suivi par b en troisième position.

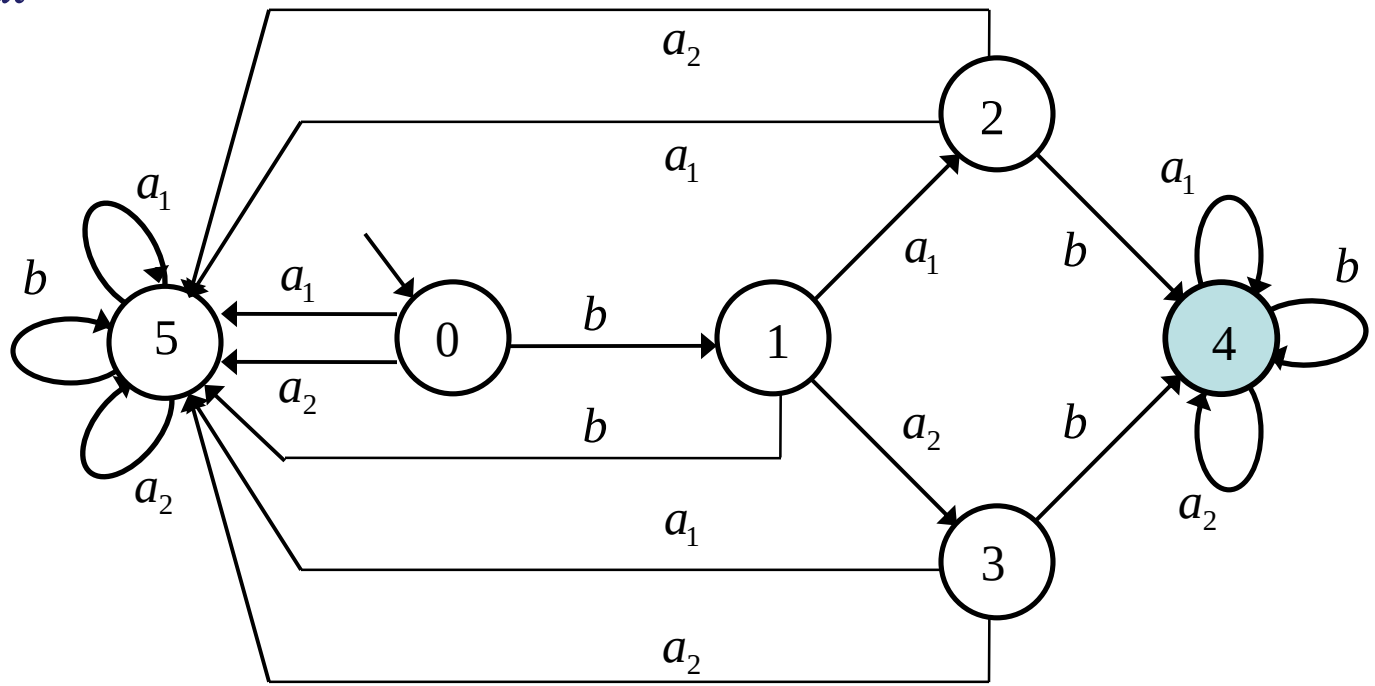
Un langage possible pour cette tâche est défini par l'expression régulière

$$\boldsymbol{b(a_1 \vee a_2)b(a_1 \vee a_2 \vee b)^*}$$

Ce langage correspond à celui du SED que l'on voulait adjoindre au photocopieur pour rapporter sur son état et indiquer si le rapport est normal.

Les événements « en marche » et « rapport terminé » sont du même type, $\boldsymbol{b}$. L'événement $\boldsymbol{a_1}$ indique qu'il n'y a « pas de problème » tandis que les événements « manque papier » et « manque toner » sont du même type, $\boldsymbol{a_2}$.

Un automate reconnaissant ce langage est décrit par le diagramme d'état



$\boldsymbol{X} = \{0, 1, 2, 3, 4, 5\}$, x_0=0, $\boldsymbol{F}$={4}

Toute chaîne avec moins de trois événements termine dans l'un des états $1, 2, 3$ ou 5 et n'est donc pas reconnue.

Les seules chaînes avec trois événements ou plus qui soient reconnues commencent avec b, continuent avec a_1 ou a_2, et atteignent avec un événement b l'état 4 où elles demeurent.

L'automate peut être utilisé comme modèle d'un système de vérification de la condition d'une machine quand elle est mise en marche, lorsque l'interprétation suivante est utilisée pour ses états : 0 = « je m'initialise », 1 = « je suis en marche », 2 = « ma condition est bonne », 3 = « j'ai un problème », 4 = « le rapport est fait », 5 = « erreur ».

Equivalence entre automates finis et expressions régulières

Peut-on toujours construire un automate fini pour reconnaître un langage donné ?
La réponse est « oui » pour tous les langages réguliers, c.-à-d. ceux décrits par des expressions régulières.

Théorème (Kleene) : Si un langage est régulier, alors il peut être reconnu par un automate fini; réciproquement, si un langage est reconnu par un automate fini, alors il est régulier.

Comment savoir si un langage donné peut être représenté par une expression régulière ?

Exemple :

$$\boldsymbol{E} = \{a, b\},$$
$$\boldsymbol{L} = \{ab, aab, abb, aaab, abbb, aaaab, \dots\}$$

Notant a^m la concaténation m fois de a avec lui-même,

$$\boldsymbol{L} = \{a^m b^n \mid m \geq 1, n \geq 1\}$$
$$= \{a\}\{a\}^*\{b\}\{b\}^*$$

Exemple :

$$\boldsymbol{E} = \{a, b\},$$
$$\boldsymbol{L} = \{\varepsilon, ab, aabb, aaabbb, \dots\}$$

Soit

$$\boldsymbol{L} = \{a^n b^n | n \geq 0\}$$

Il est facile de trouver des automates qui reconnaissent ce langage, mais tous ces automates reconnaissent un nombre infini d'autres chaînes, comme par exemple l'automate associé à l'expression régulière $\boldsymbol{a}^*\boldsymbol{b}^*$, qui contient aussi aa et aab, qui ne sont pas dans $\boldsymbol{L}$. En effet il n'est pas possible de trouver un automate fini qui compte n fois, car un nombre infini d'états est nécessaire pour compter jusqu'à un nombre qui n'est pas borné.

2.2.3 Fusion d'états dans les automates

Problème : construire un modèle sous forme d'automate à états d'un SED peut devenir très lourd : explosion combinatoire du nombre d'états.

Solution : repérer les groupes d'états redondants, chaque groupe pouvant être fusionné en un seul état, afin de réduire la taille de l'espace d'état.

Formellement : notion d'états équivalents.

Définition : Soit $(\boldsymbol{E}, \boldsymbol{X}, f, x_0, \boldsymbol{F})$ un automate fini et $\boldsymbol{R} \subseteq \boldsymbol{X}$. L'ensemble $\boldsymbol{R}$ consiste en des états équivalents par rapport à $\boldsymbol{F}$ si pour toute paire $x, y \in \boldsymbol{R}$, $x \neq y$, et pour toute chaîne u, $f(x,u) \in \boldsymbol{F}$ si et seulement si $f(y,u) \in \boldsymbol{F}$.

Deux états peuvent être fusionnés en un seul état si, lorsqu'une chaîne appliquée à l'un des états mène à un état final, elle le fait aussi lorsqu'elle est appliquée à l'autre état.

Exemple : détecteur de séquence de chiffres

$\boldsymbol{E} = \{1, 2, 3\}$, où n signifie « le chiffre n est arrivé »,
détection de la chaîne $u = 123$.

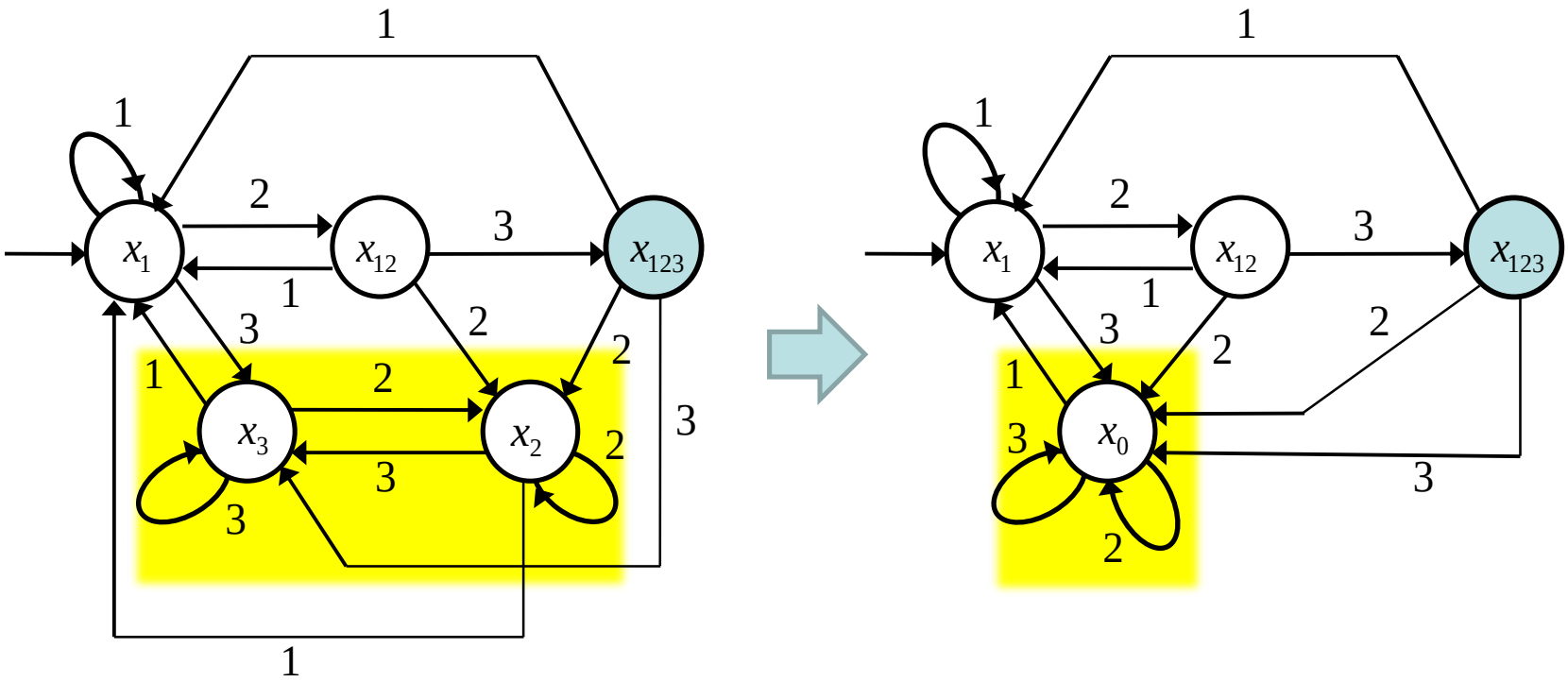
Pour construire un espace d'état $\boldsymbol{X}$, on commence par l'état x_1, représentant le fait que l'événement le plus récent était 1, et on considère les trois cas possibles :

- arrivée de 1 : on pose $f(x_1, 1) = x_1$;
- arrivée de 2 : on pose $f(x_1, 2) = x_{12}$, l'état x_{12} indiquant que la chaîne 12 est arrivée;
- arrivée de 3 : on pose $f(x_1, 3) = x_3$.

On continue de même à partir de chaque nouvel état :

$$f(x_{12}, 1) = x_1, \quad f(x_{12}, 2) = x_2, \quad f(x_{12}, 3) = x_{123},$$
$$f(x_3, 1) = x_1, \quad f(x_3, 2) = x_2, \quad f(x_3, 3) = x_3,$$
$$f(x_2, 1) = x_1, \quad f(x_2, 2) = x_2, \quad f(x_2, 3) = x_3,$$
$$f(x_{123}, 1) = x_1, \quad f(x_{123}, 2) = x_2, \quad f(x_{123}, 3) = x_3.$$

Diagramme des transitions d'état



Le modèle plus simple est obtenu en appliquant la condition de la définition à $\boldsymbol{R} = \{x_2, x_3\}$. En effet $f(x_2, 1) = f(x_3, 1) = x_1$, $f(x_2, 2) = f(x_3, 2) = x_2$, $f(x_2, 3) = f(x_3, 3) = x_3$.

Le modèle original à 5 états est réduit à l'espace d'état $\boldsymbol{X} = \{x_0, x_1, x_{12}, x_{123}\}$.

Observations *au sujet de l'équivalence entre états* :

- Deux états x, y tels que $x \in \boldsymbol{F}$ et $y \notin \boldsymbol{F}$ ne peuvent pas être équivalents par rapport à $\boldsymbol{F}$. Ainsi $f(x, \varepsilon) \in \boldsymbol{F}$ alors que $f(y, \varepsilon) \notin \boldsymbol{F}$.
- Si deux états x, y sont tous deux soit dans $\boldsymbol{F}$ soit en dehors de $\boldsymbol{F}$, il est possible qu'ils soient équivalents par rapport à $\boldsymbol{F}$. Si $f(x, e) = f(y, e)$ pour tout événement $e \in \boldsymbol{E}$ (excepté ε), les états x et y sont équivalents par rapport à $\boldsymbol{F}$ car ils traversent la même séquence d'états pour toute chaîne qui leur est appliquée.
- Cette propriété est encore vraie si $f(x, e) = y$ et $f(y, e) = x$ pour au moins un événement e – puisque les rôles de x et y sont échangés après e – et si pour tous les autres événements $f(x, e) = f(y, e)$.
- Plus généralement, soit $\boldsymbol{R}$ un ensemble d'états tels que $\boldsymbol{R} \subseteq \boldsymbol{F}$ ou $\boldsymbol{R} \cap \boldsymbol{F} = \emptyset$, $\boldsymbol{R}$ regroupe des états équivalents si, pour toute paire $x, y \in \boldsymbol{R}$, $f(x, e) = z \notin \boldsymbol{R}$ implique $f(y, e) = z$, c.-à-d. toutes les transitions des états dans $\boldsymbol{R}$ vers les états hors de $\boldsymbol{R}$ doivent être causées par le même événement et amener au même état.
- Si $\boldsymbol{F} = \boldsymbol{X}$, tous les états sont équivalents par rapport à $\boldsymbol{F}$, ce qui n'est pas très intéressant. Dans ce cas cependant, la dernière observation est très utile car elle permet de fusionner tous les états dans $\boldsymbol{R}$ sans perte d'information sur le comportement de l'automate.

Le processus de remplacement de plusieurs états par un seul, appelé ***fusion d'états***, est important en théorie des systèmes car il réduit l'espace d'états et donc la complexité des calculs.

Étant donné un automate fini $(\boldsymbol{E}, \boldsymbol{X}, f, x_0, \boldsymbol{F})$, l'algorithme d'identification des ensembles d'états équivalents permet de fusionner les états de chacun de ces ensembles en un seul état et de simplifier le modèle original autant que possible. On suppose que $\boldsymbol{F} \neq \boldsymbol{X}$. L'idée de départ de l'algorithme est de *marquer* toutes les paires d'états (x, y) telles que x est un état final et y ne l'est pas. Les états dans ces paires ne peuvent pas être équivalents.

Ensuite on considère chaque paire (x, y) non marquée et on recherche s'il existe un événement $e \in \boldsymbol{E}$ conduisant à une paire d'états $(f(x, e), f(y, e))$ qui est déjà marquée. Si tel n'est pas le cas, et $f(x, e) \neq f(y, e)$, alors on crée une liste d'états associés à $(f(x, e), f(y, e))$ et on place (x, y) dans cette liste.

La poursuite du processus permet de déterminer si $(f(x, e), f(y, e))$ doit être marquée; si oui alors le marquage se propage à (x, y) et à toutes les paires de sa liste. À la fin de la procédure, chaque paire qui n'est pas marquée définit une classe d'équivalence.

Algorithme pour l'identification d'états équivalents :

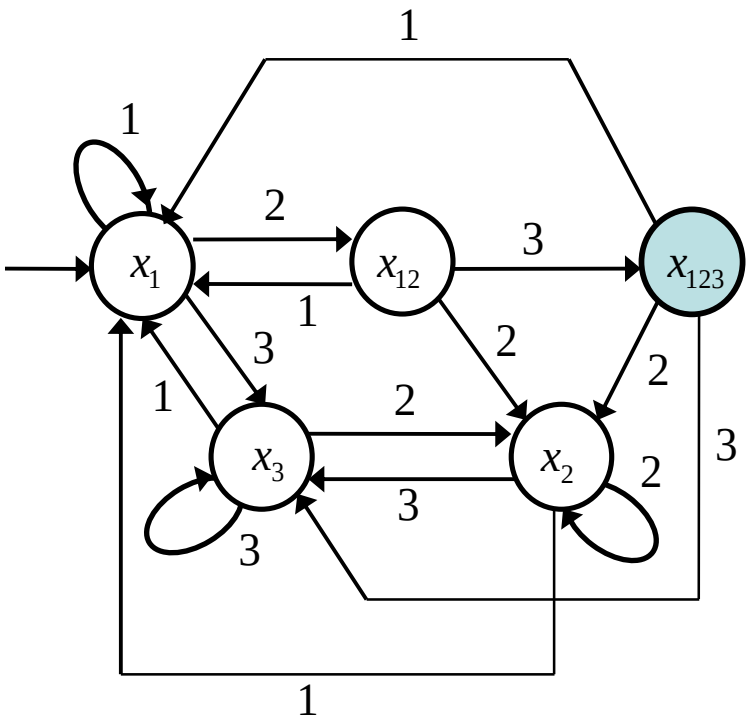
Étape 1. Marquer (x, y) pour tout $x \in \boldsymbol{F}, y \notin \boldsymbol{F}$.

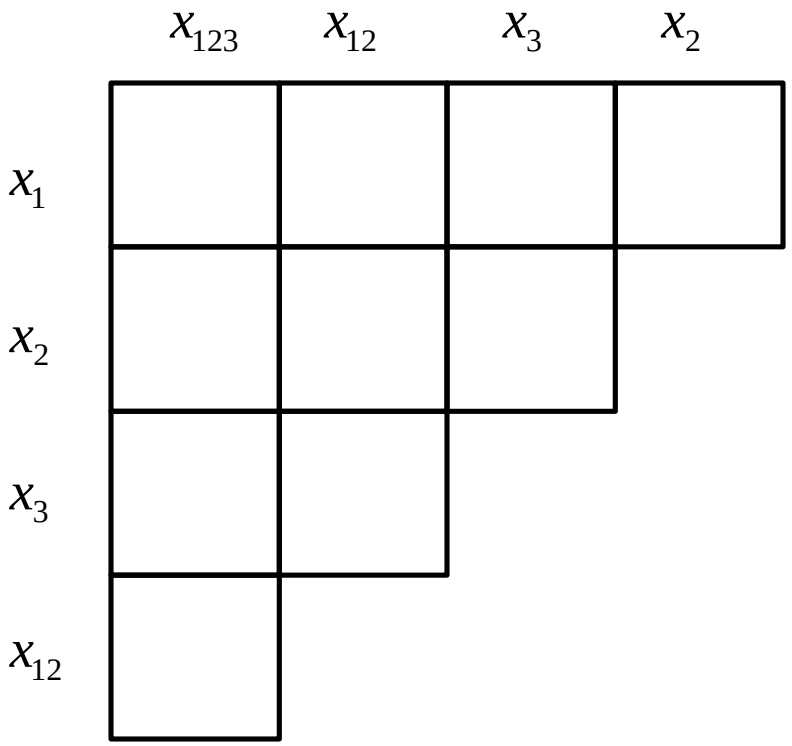
Étape 2. Pour chaque paire (x, y) non marquée à l'étape 1.
Étape 2.1. Si $(f(x, e), f(y, e))$ est marquée pour un $e \in \boldsymbol{E}$, alors :
Étape 2.1.1. Marquer (x, y),
Étape 2.1.2. Marquer toutes les paires (w, z) dans la liste de (x, y).
Répéter cette étape pour chaque (w, z) tant qu'il est possible de marquer des paires.

Étape 2.2. Si aucun $(f(x, e), f(y, e))$ n'est marqué, alors pour chaque $e \in \boldsymbol{E}$:
Étape 2.2.1. Si $f(x, e) \neq f(y, e)$, alors ajouter (x, y) à la liste de $(f(x, e), f(y, e))$

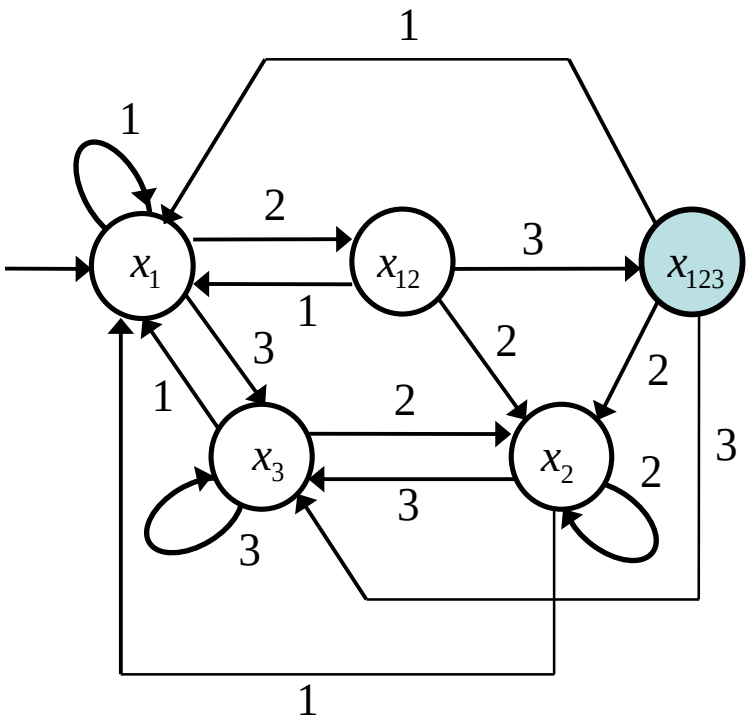
On construit donc une table contenant toutes les paires d'états possibles et l'on marque les paires (x, y) telles que x et y ne sont pas équivalents.
A la fin de l'algorithme, les paires qui n'ont pas été marquées indiquent les états qui sont équivalents.

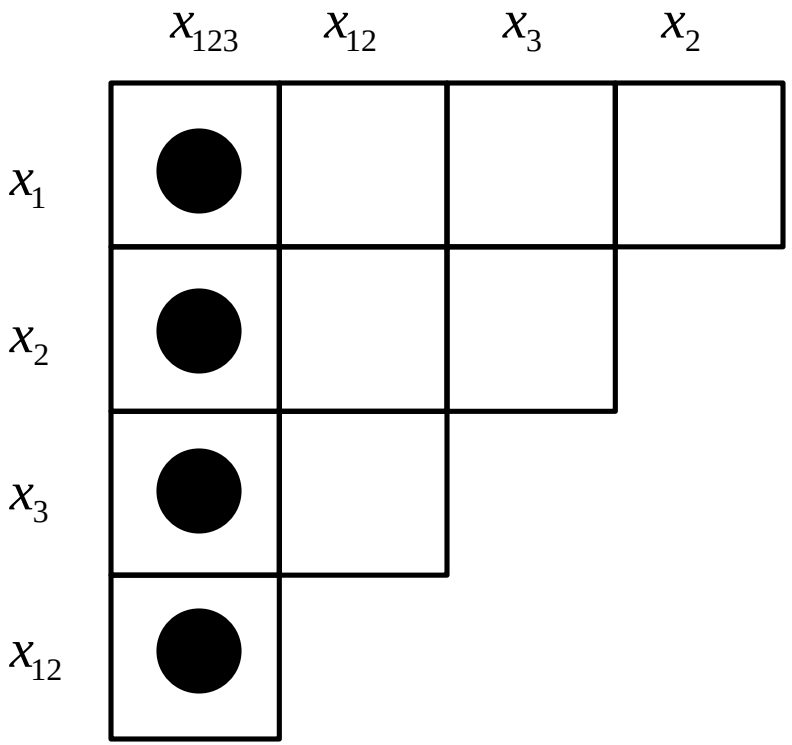
Exemple : application au détecteur de séquence de chiffres



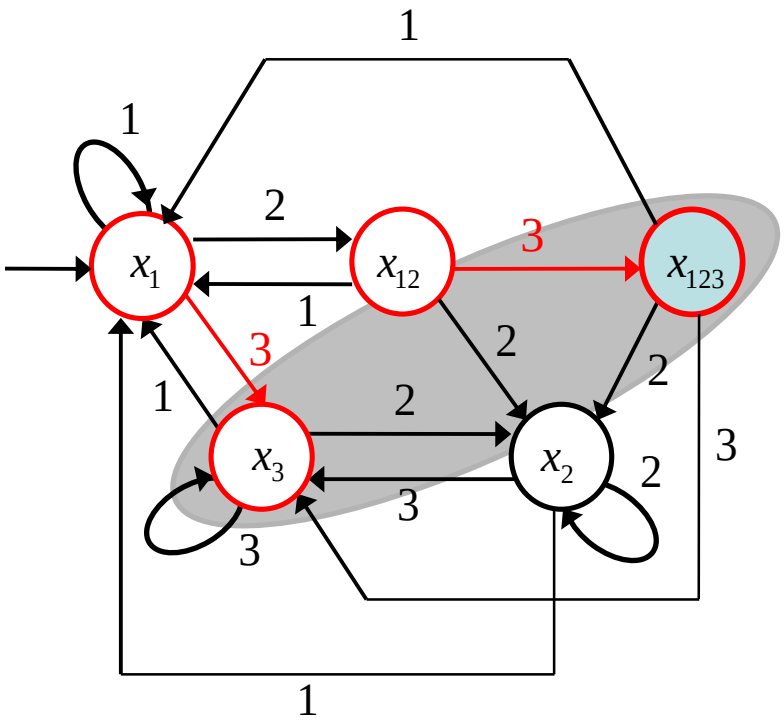


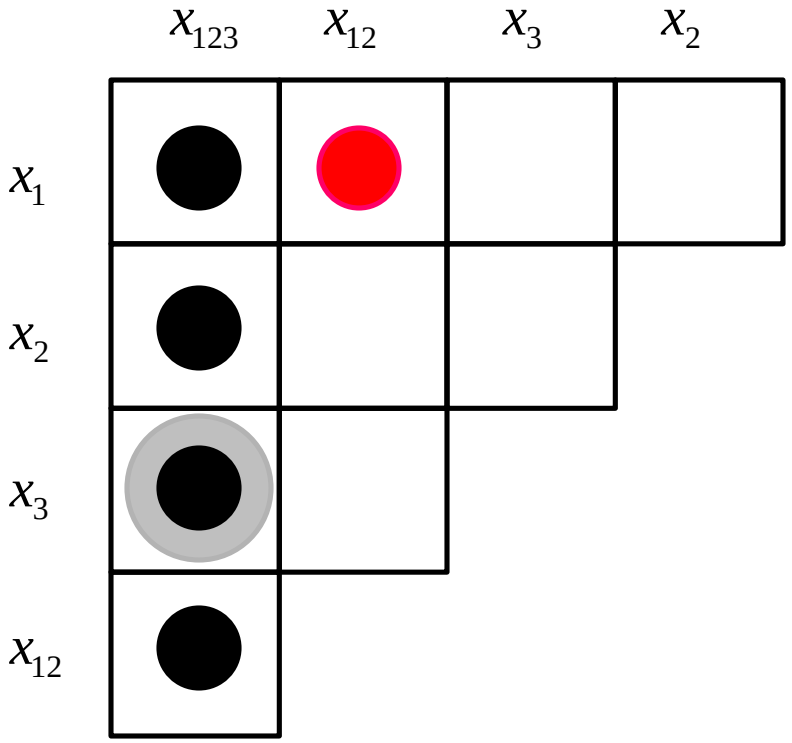
Exemple : application au détecteur de séquence de chiffres



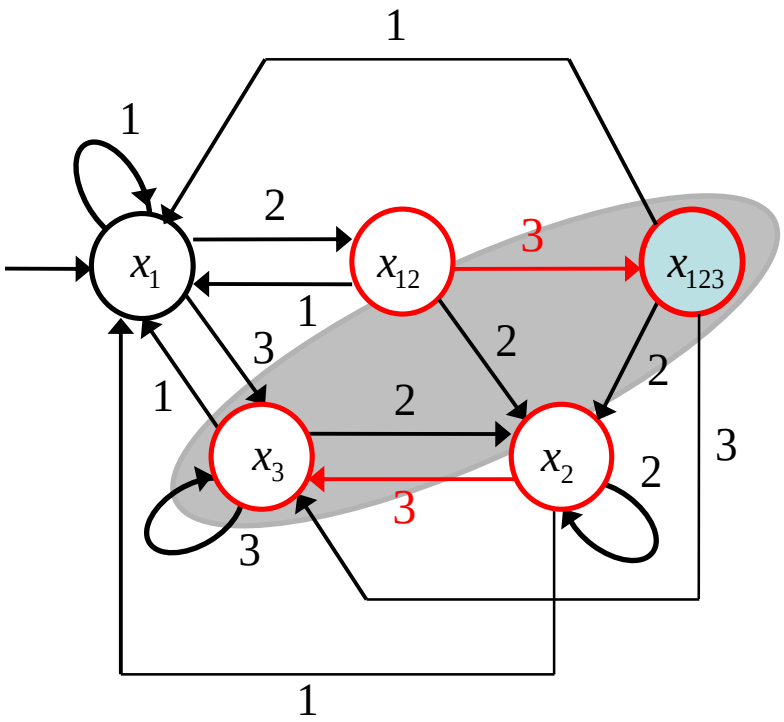


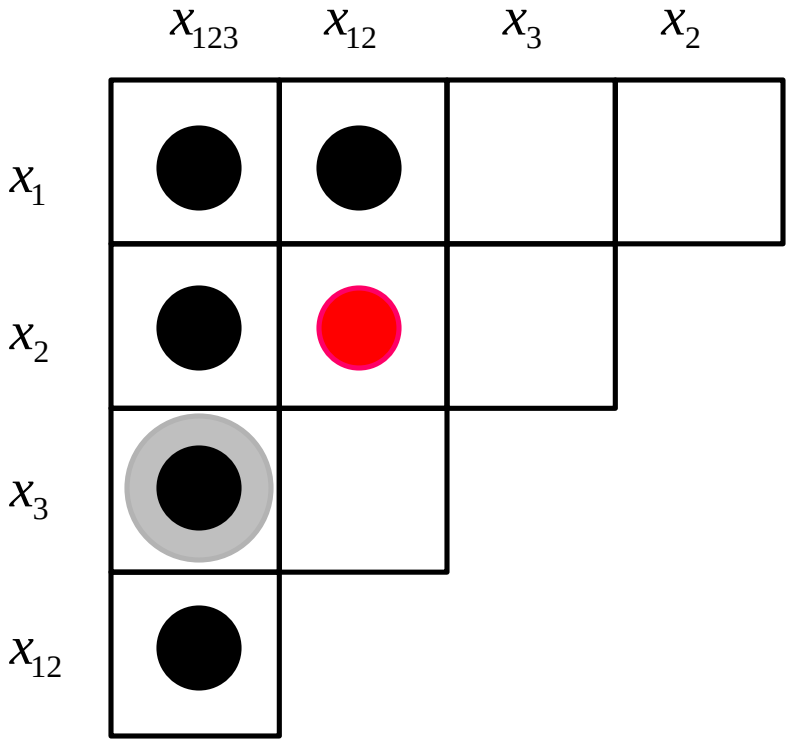
Exemple : application au détecteur de séquence de chiffres



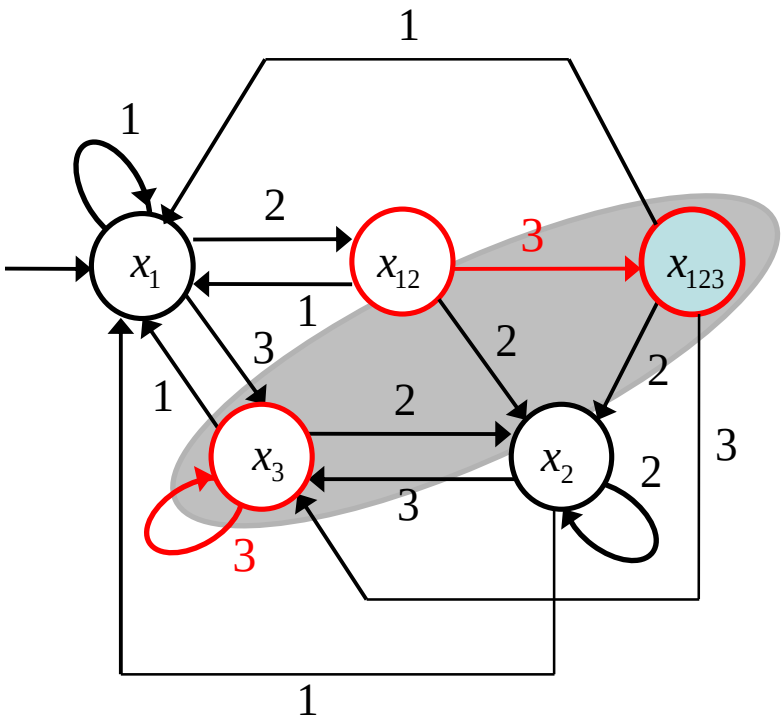


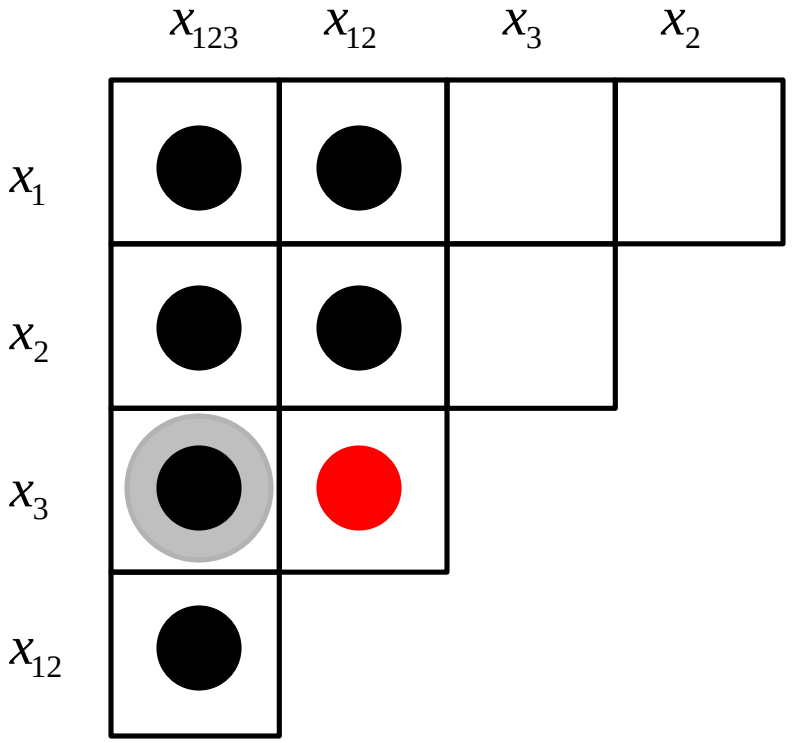
Exemple : application au détecteur de séquence de chiffres



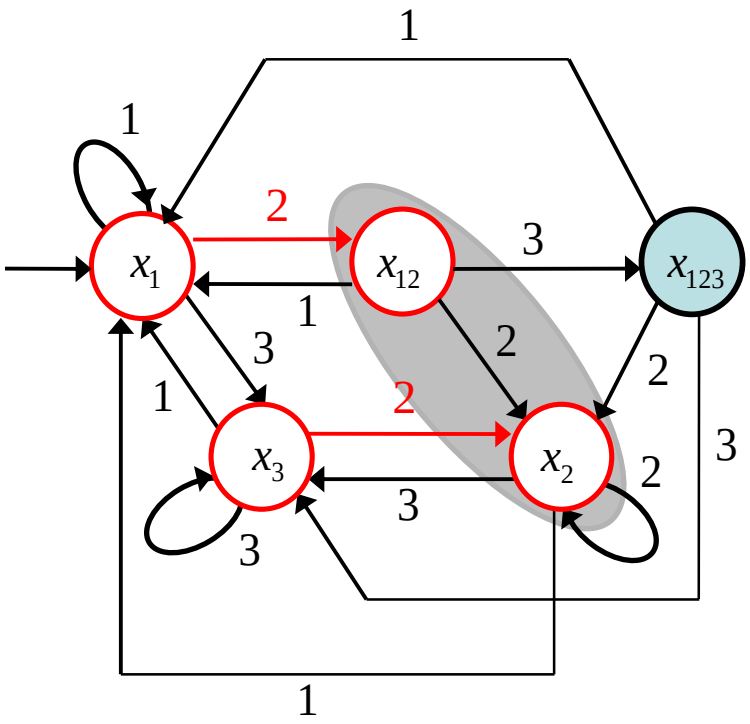


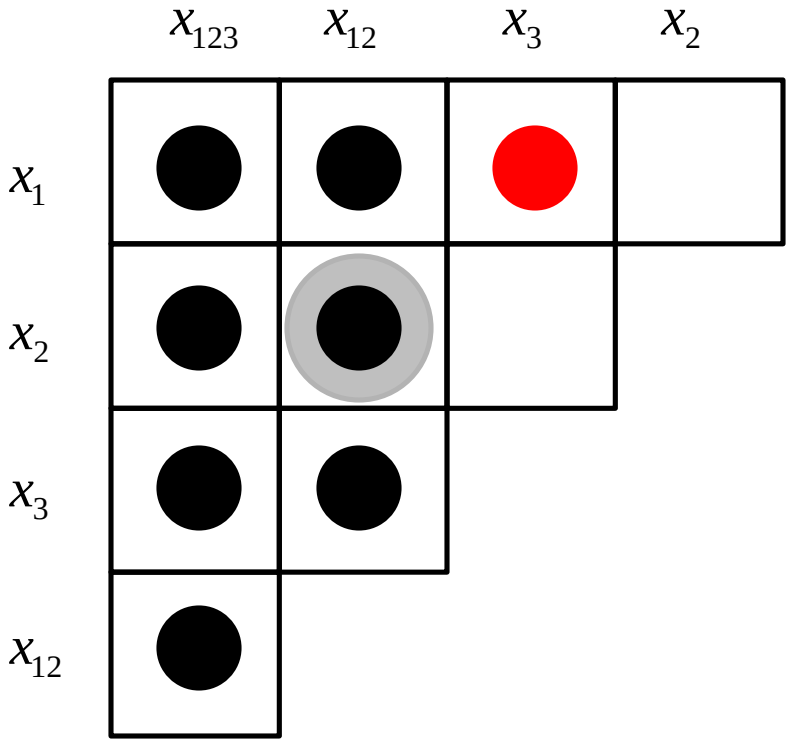
Exemple : application au détecteur de séquence de chiffres



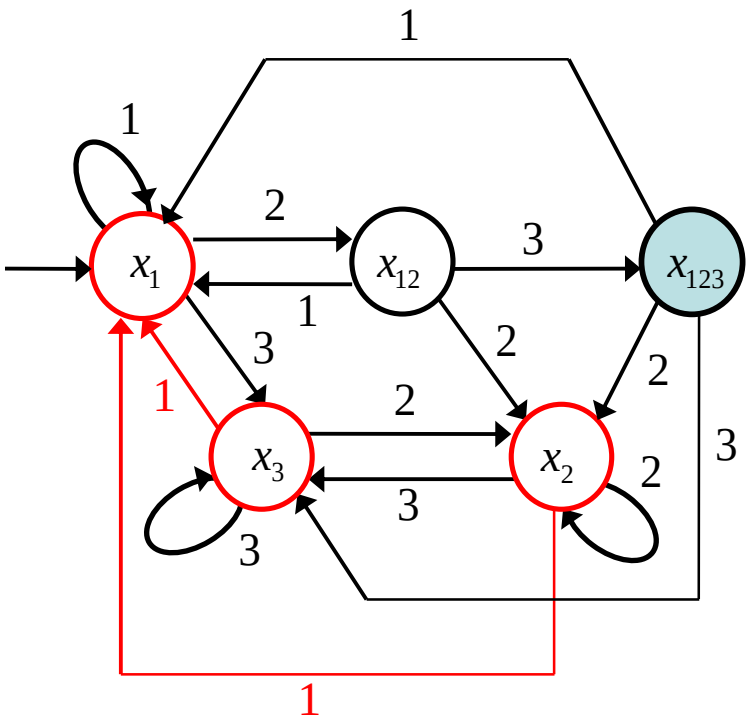


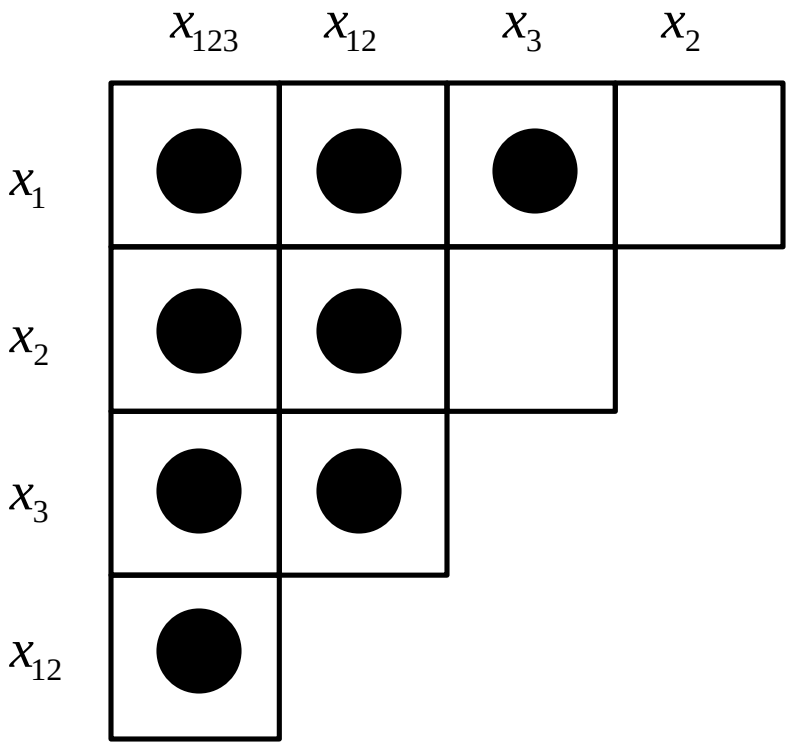
Exemple : application au détecteur de séquence de chiffres



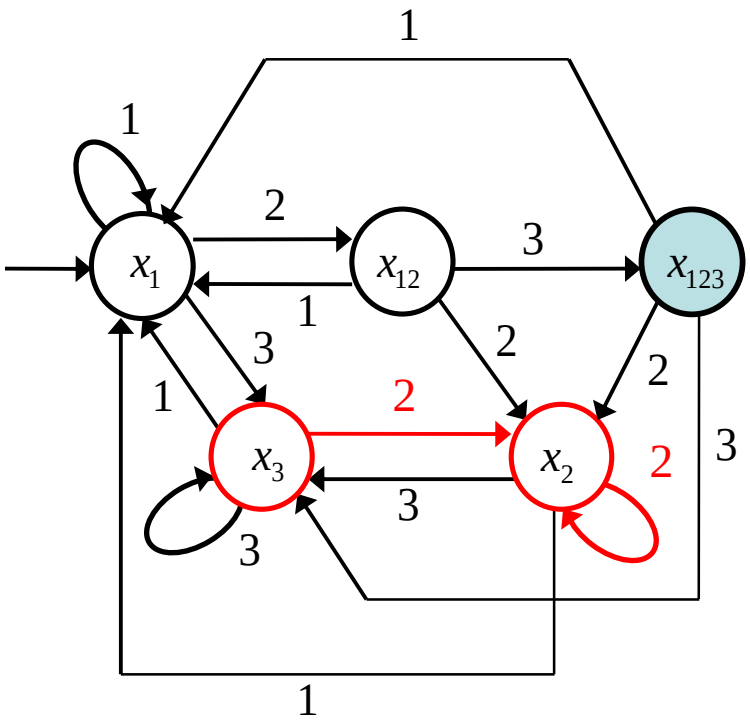


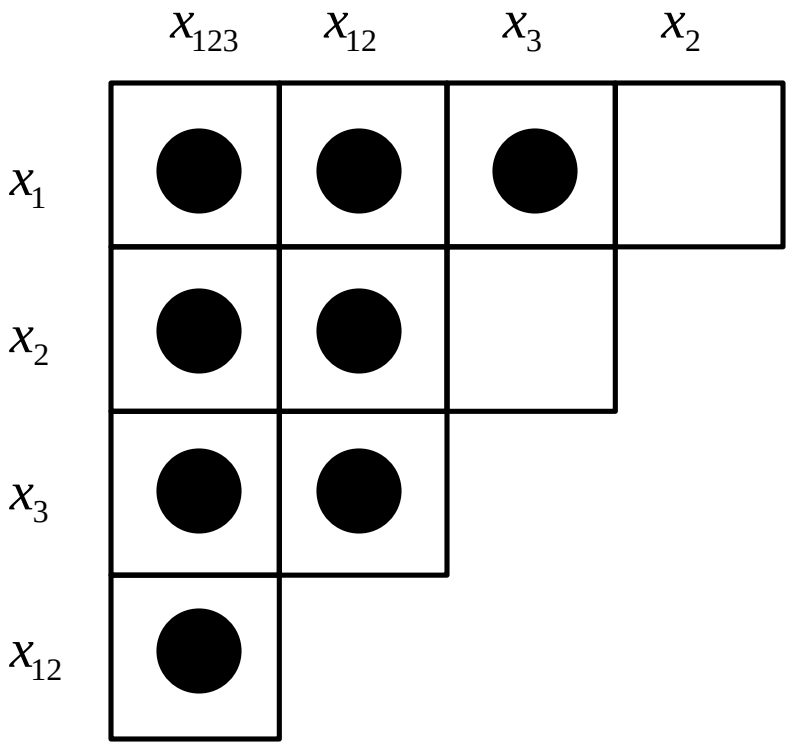
Exemple : application au détecteur de séquence de chiffres



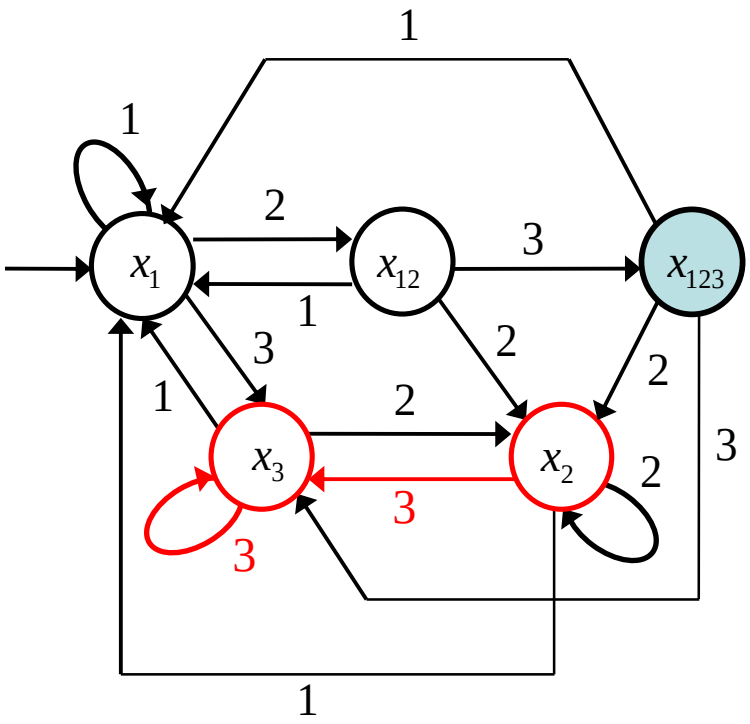


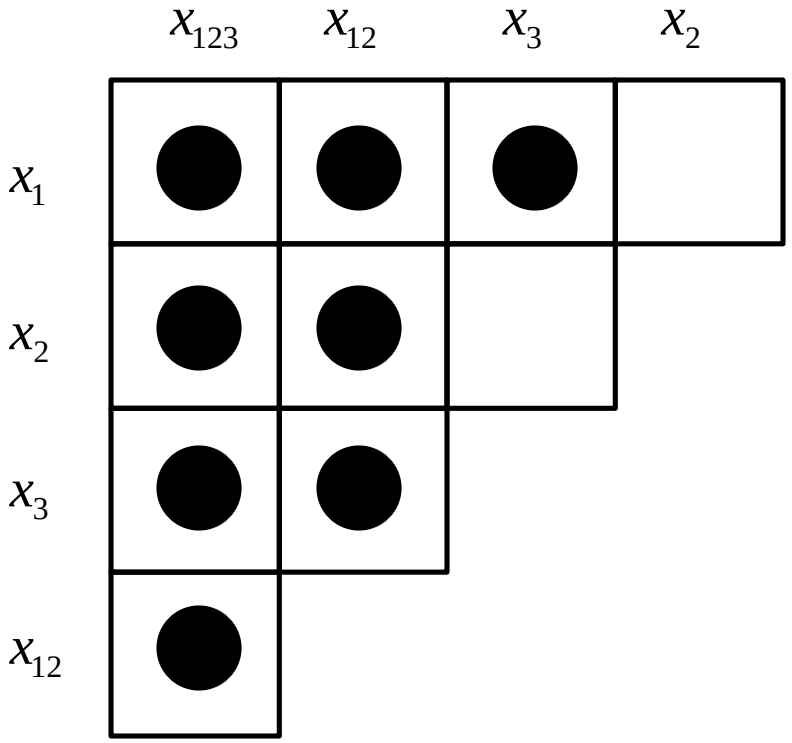
Exemple : application au détecteur de séquence de chiffres



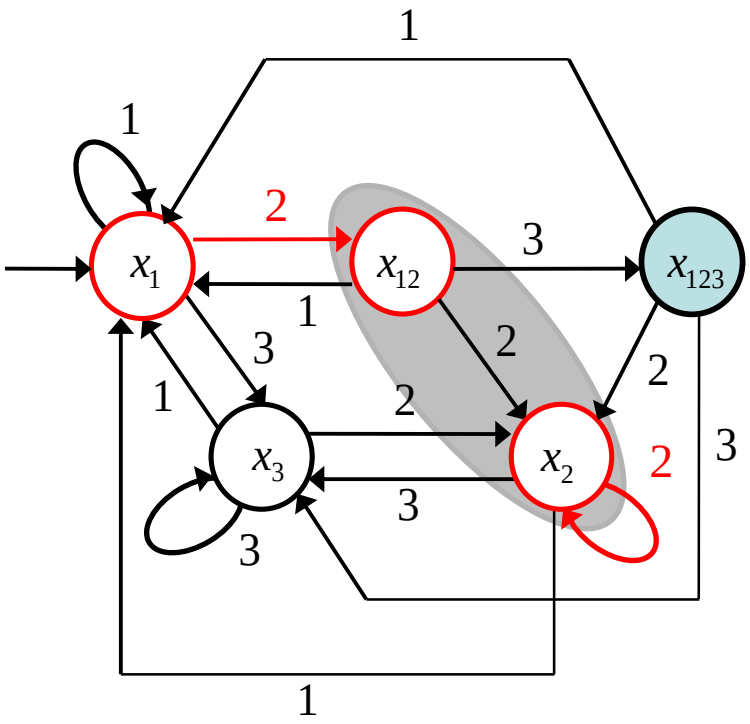


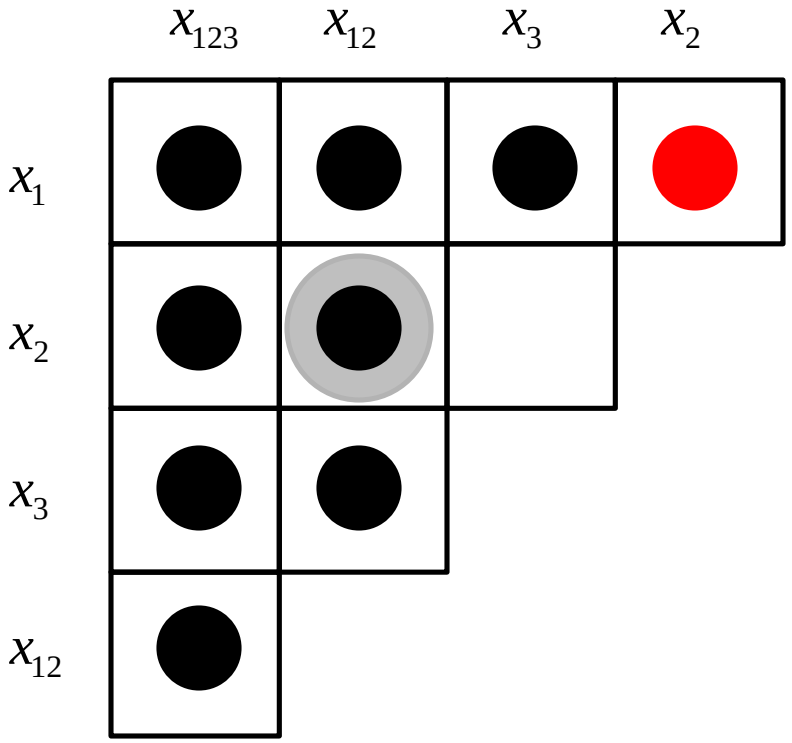
Exemple : application au détecteur de séquence de chiffres



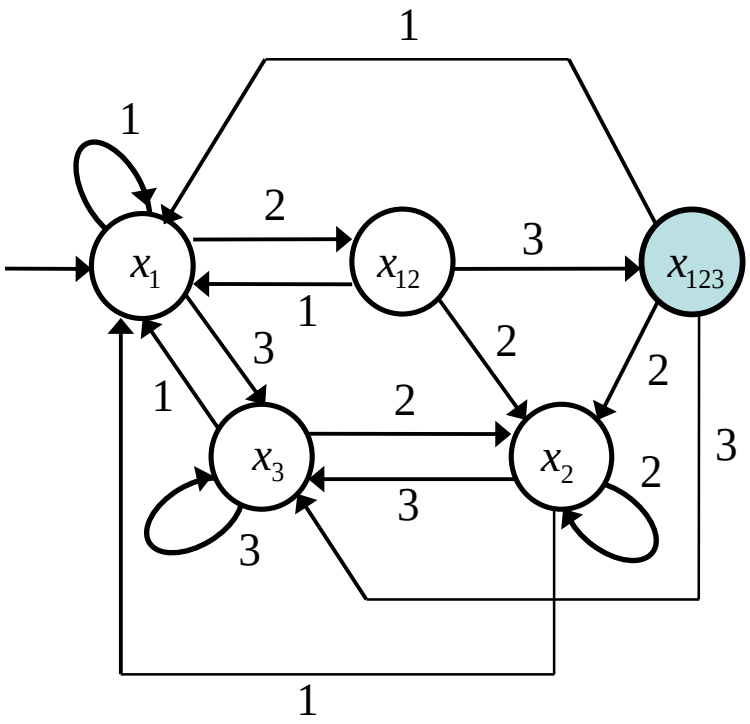


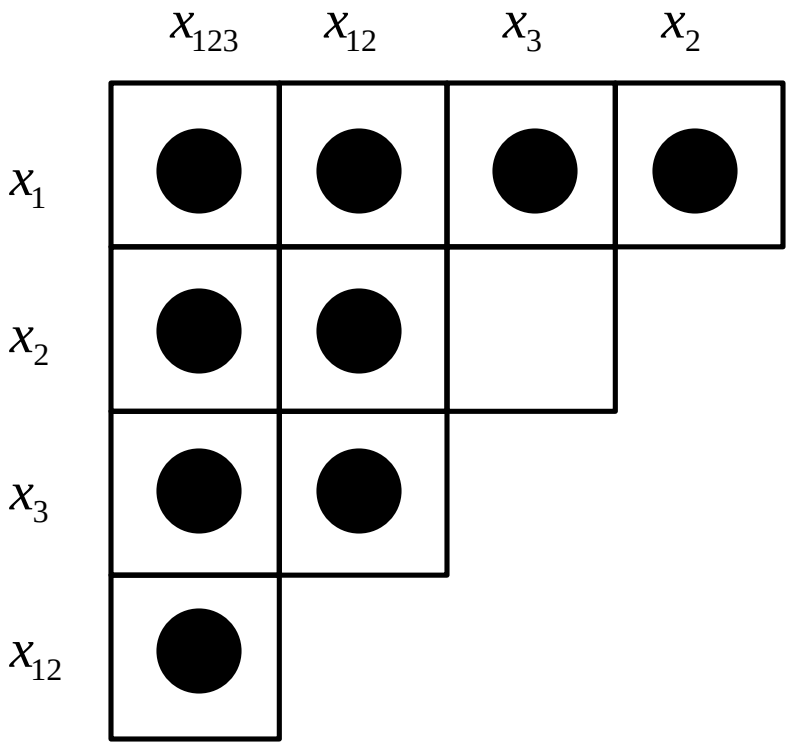
Exemple : application au détecteur de séquence de chiffres





Exemple : application au détecteur de séquence de chiffres





2.2.4 Systèmes à événements discrets comme automates à états

On veut utiliser les automates comme modèles non temporisés de SED.
Notre motivation est qu'un automate fini a déjà l'avantage d'être assez proche d'un système dynamique si l'on

- assimile $\boldsymbol{X}$ à l'espace d'états,
- convient que les entrées sont des chaînes construites à partir d'un alphabet $\boldsymbol{E}$,
- définit la transition d'un état x à un état x' selon $x' = f(x, e)$.

Pour se rapprocher encore d'un système dynamique, on va apporter quelques modifications à la définition d'automate fini de la section 2.2.2.

- On permet à $\boldsymbol{X}$ et $\boldsymbol{E}$ d'être dénombrables.
- On reconnait que certains événements ne peuvent pas se produire dans certains états et on associe un ensemble d'événements possibles à chaque état.
- On ne distingue pas d'ensemble d'états finaux.

Le modèle résultant est un ***automate à états modifié***. L'essentiel demeure et nous continuerons de le qualifier ***automate à états***.

Définition : Un automate à états est un quintuplet $(\boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\Gamma}, f, x_0)$ où
$\boldsymbol{E}$ est un *ensemble* dénombrable *d'événements*,
$\boldsymbol{X}$ est un *espace d'états* dénombrable,
$\boldsymbol{\Gamma}(x)$ est un ensemble d'événements *possibles* ou *permis*, défini pour tout $x \in \boldsymbol{X}$ avec $\boldsymbol{\Gamma}(x) \subseteq \boldsymbol{E}$,
f est une *fonction de transition d'état*, $f : \boldsymbol{X} \times \boldsymbol{E} \to \boldsymbol{X}$, définie seulement pour $e \in \boldsymbol{\Gamma}(x)$ quand l'état est x; $f(x, e)$ n'est pas définie pour $e \notin \boldsymbol{\Gamma}(x)$,
x_0 est un état initial, $x_0 \in \boldsymbol{X}$.

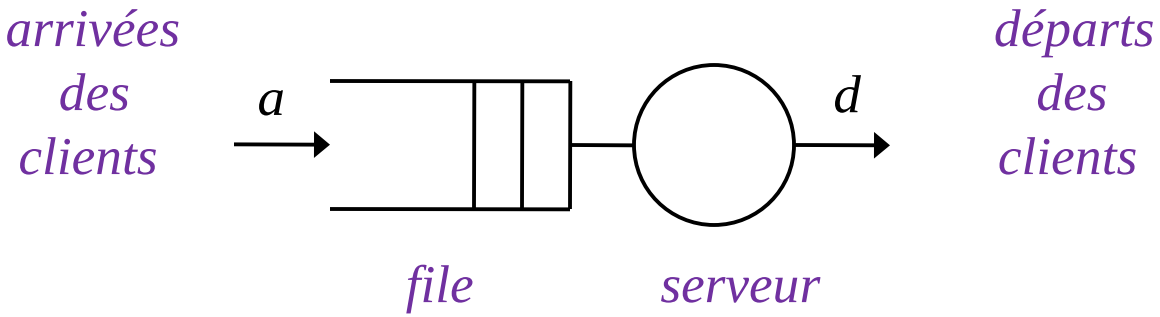
L'état initial est essentiel dans la définition d'un SED. Cependant on est parfois intéressé par la description d'un système où tout état peut être pris comme point de départ. Ainsi on peut aussi définir un SED comme un quadruplet $(\boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\Gamma}, f\ )$, omettant l'état initial.

Dans le diagramme des transitions d'états, $\boldsymbol{\Gamma}(x)$ est représenté implicitement en incluant seulement les arcs émanant de x correspondant aux éléments de $\boldsymbol{\Gamma}(x)$.

2.2.5 Modèles d'automates à états pour les systèmes de files d'attente

Les systèmes de files d'attentes sont une classe importante de SED.

Exemple de la station de service de base :

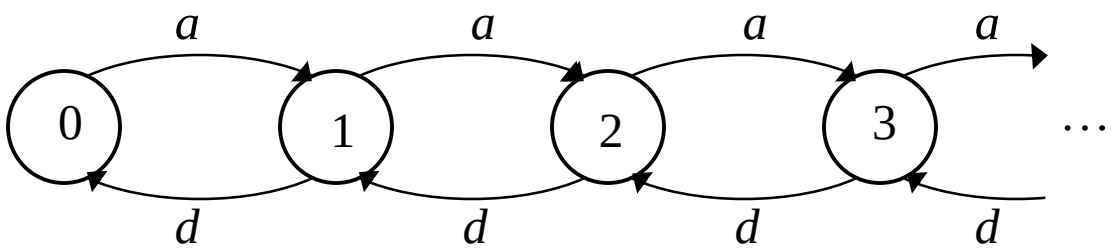


On peut définir un automate à états selon :

$$\boldsymbol{E} = \{a, d\}, \qquad \boldsymbol{X} = \{0, 1, 2, \dots\},$$
$$\boldsymbol{\Gamma}(0) = \{a\}, \qquad \boldsymbol{\Gamma}(x) = \{a, d\} \text{ pour tout } x > 0,$$
$$f(x, a) = x + 1 \text{ pour tout } x \geq 0,$$
$$f(x, d) = x - 1 \text{ pour tout } x > 0.$$

L'état x représente le nombre de clients dans la file, client servi inclus.

Diagramme des transitions d'états associé :



L'espace d'états est infini mais dénombrable.

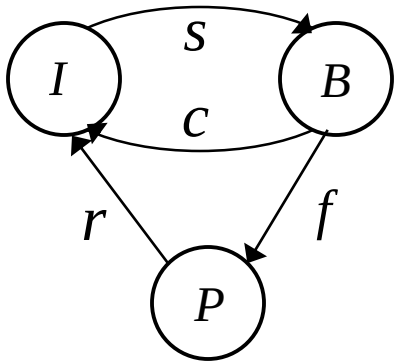
Considérons maintenant un serveur réaliste.

- L'état du serveur peut être inactif (*I*), actif (*B* pour *busy*), ou hors service (*P* pour panne). Quand le serveur tombe en panne, le client qu'il sert est perdu, donc, lorsqu'il est réparé, il se retrouve dans l'état inactif.
- Les événements entrées de ce système sont de quatre types :
  - *s* : le **s**ervice débute (***s****tarts*)
  - *c* : le service se **c**onclut (***c****ompletes*)
  - *f* : le serveur s'e**ff**ondre (***f****ails*)
  - *r* : le serveur est **r**éparé (***r****epaired*)

Un modèle d'automate à états pour le serveur est donné par :

$$\boldsymbol{E} = \{s, c, f, r\}, \quad \boldsymbol{X} = \{I, B, P\},$$
$$\boldsymbol{\Gamma}(I) = \{s\}, \quad f(I, s) = B,$$
$$\boldsymbol{\Gamma}(B) = \{c, f\}, \quad f(B, c) = I, \quad f(B, f) = P,$$
$$\boldsymbol{\Gamma}(P) = \{r\}, \quad f(P, r) = I$$

Le diagramme des transitions d'états correspondant est



Intuitivement, l'événement « le service commence » devrait se produire immédiatement après l'entrée dans l'état *I*. Ce n'est pas possible quand la file est vide, mais le modèle n'a pas connaissance de la longueur de la file. Du coup, les événements *s* sont traités comme des observations exogènes.

2.2.6 Automates à états avec sortie

On est prêts à établir un modèle pour les SED analogue à celui utilisé pour les systèmes dynamiques à variables (d'état) continues.

Pour ce faire, on inclut dans le modèle d'automate un ensemble de sortie $\boldsymbol{Y}$, et une fonction de sortie $g : \boldsymbol{X} \times \boldsymbol{Y} \to \boldsymbol{Y}$.

L'ensemble $\boldsymbol{Y}$, comme l'ensemble des événements $\boldsymbol{E}$, est un alphabet dans lequel des événements sont sélectionnés pour former une séquence, ou chaîne, de sortie.

Définition :

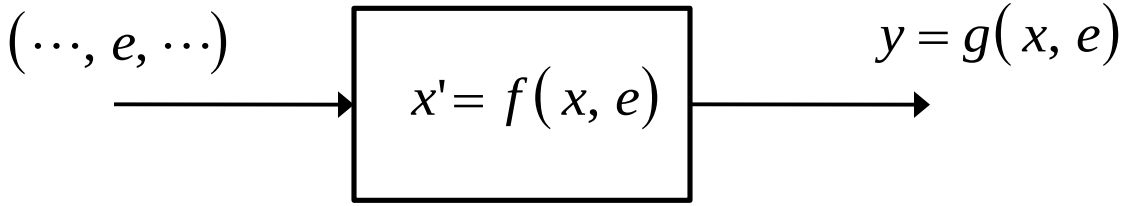
Un automate à états avec sortie est un septuplet $(\boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\Gamma}, f, x_0, \boldsymbol{Y}, g)$ où

$(\boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\Gamma}, f, x_0)$ est un automate à états,

$\boldsymbol{Y}$ est un *espace de sortie* dénombrable,

g est une *fonction de sortie*, $g : \boldsymbol{X} \times \boldsymbol{E} \to \boldsymbol{Y}$, définie seulement pour $e \in \boldsymbol{\Gamma}(x)$ quand l'état est x; $g(x, e)$ n'est pas définie pour $e \notin \boldsymbol{\Gamma}(x)$

Modélisation d'un SED par un automate à états avec sortie :



L'entrée est une chaîne d'événements $(\ldots, e, \ldots)$ sélectionnés dans $\boldsymbol{E}$.
La dynamique du système est capturée par l'équation de transition d'état $x' = f(x, e)$, où x' est le nouvel état résultant de l'occurrence de l'événement e. Si un événement e se produit en entrée alors que l'état est x et e n'est pas dans l'ensemble $\boldsymbol{\Gamma}(x)$, l'événement est ignoré par l'automate et l'état reste x. La sortie est une séquence dont les éléments sont déterminés par $y = g(x, e)$.

Exemple – un protocole simple de communication :
Ce protocole est employé par un transmetteur sur un canal de communication.
Lorsqu'un message arrive, il est traité si le transmetteur est inactif et est ignoré sinon. Le traitement consiste en deux actions :
- rangement d'une copie du message et envoi du message sur le canal, et
- initialisation du décompte d'une durée fixée (délai).

Si le message est effectivement transmis, un acquittement est reçu et la copie n'est pas conservée. Si le délai a expiré, le transmetteur suppose que le message a été perdu et le retransmet.

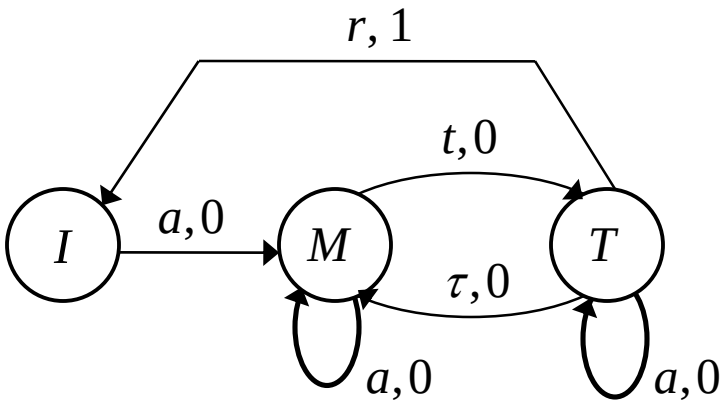
L'ensemble des événements en entrée est $\boldsymbol{E} = \{a, t, \tau, r\}$ où

a : arrivée d'un message
t : transmission (envoi) d'un message
τ : expiration du délai
r : réception d'un acquittement

L'espace d'états est $\boldsymbol{X} = \{I, M, T\}$ où I représente l'état inactif, M la présence d'un message à transmettre, et T l'état de transmission.

L'ensemble de sortie est $\boldsymbol{Y} = \{0, 1\}$ où 1 indique qu'un message a été transmis avec succès; sinon la sortie est 0. Clairement on doit avoir un 1 en sortie chaque fois que se produit un événement r.

Le modèle complet est décrit dans le diagramme des transitions d'états qui suit.



Les événements *a* sont toujours permis, mais il ne causent une transition d'état que lorsque le transmetteur est inactif

Une alternative au diagramme des transitions d'états est la ***table des transitions d'état*** ou ***des nouveaux états.*** La table contient autant de lignes qu'il y a d'éléments dans ***X*** et autant de colonnes que d'éléments dans ***E :***

		événement			
		a	t	τ	r
état courant	I	$M, 0$			
	M	$M, 0$	$T, 0$		
	T	$T, 0$		$M, 0$	$I, 1$

2.3 Réseaux de Petri

Les réseaux de Petri fournissent une alternative aux automates pour construire des modèles non-temporisés de SED. Ces modèles furent initialement développés par C. A. Petri, au début des années 1960.

Les réseaux de Petri ressemblent aux automates dans la mesure où ils représentent de façon explicite la fonction de transition des SED.

Comme un automate, un réseau de Petri manipule des événements selon certaines règles. Une de ses caractéristiques est d'inclure des conditions explicites selon lesquelles un événement peut se produire; cette propriété permet de représenter des SED très généraux, dont le fonctionnement dépend de schémas de contrôle potentiellement complexes.

Il est pratique de décrire cette représentation sous forme graphique, au moins pour des petits systèmes. Les graphes de réseaux de Petri ainsi obtenus sont intuitifs et fournissent d'importantes informations structurelles sur le système.

Nous verrons qu'un automate d'état fini peut toujours être représenté par un réseau de Petri; par contre la réciproque n'est pas vraie. Ainsi, les réseaux de Petri peuvent représenter une classe de langages plus grande que la classe des langages réguliers.
Un autre intérêt de l'utilisation des réseaux de Petri comme modèles de SED est le riche ensemble de techniques d'analyse développées afin de les étudier. Ces techniques comprennent l'analyse de l'ensemble des états accessibles et des méthodes d'algèbre linéaire. Elles s'appliquent non seulement aux réseaux de Petri non-temporisés mais aussi aux réseaux de Petri temporisés.

2.3.1 Notations et Définitions

Dans les réseaux de Petri, les événements sont associés aux transitions.
Pour qu'une transition puisse se produire, plusieurs conditions doivent être satisfaites. L'information associée à ces conditions est contenue dans des places.
Certaines places constituent l' « entrée » d'une transition; elles sont associées aux conditions requises pour que la transition se produise.
D'autres places constituent la « sortie » d'une transition; elles sont associées aux conditions qui sont affectées lorsque cette transition se produit.

Les transitions, les places, et certaines relations entre elles définissent les éléments de base d'un graphe de réseau de Petri.

Un réseau de Petri contient deux types de nœuds, les places et les transitions, et des arcs les reliant. C'est un graphe biparti au sens qu'un arc ne peut pas connecter deux nœuds de même type; les arcs connectent des places à des transitions et des transitions à des places.

Définition : Un ***graphe de réseau de Petri*** est un graphe biparti pondéré $(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w)$, où

$\boldsymbol{P}$ est l'ensemble fini des *places* (l'un des types de sommets du graphe),
$\boldsymbol{T}$ est l'ensemble fini des *transitions* (l'autre type de sommets du graphe),
$\boldsymbol{A} \subseteq (\boldsymbol{P} \times \boldsymbol{T}) \cup (\boldsymbol{T} \times \boldsymbol{P})$ est l'ensemble des arcs des places vers les transitions et des transitions vers les places du graphe,
$w : \boldsymbol{A} \to \{1, 2, 3, \dots\}$ est la *fonction de ponderation* sur les arcs.

On suppose que $(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w)$ n'a pas de places ou de transitions isolées.
En general on représentra l'ensemble des places par $\boldsymbol{P} = \{p_1, p_2, \dots, p_n\}$ et l'ensemble des transitions par $\boldsymbol{T} = \{t_1, t_2, \dots, t_m\}$, ainsi, $|\boldsymbol{P}| = n$ et $|\boldsymbol{T}| = m$.
Un arc typique est de la forme (p_i, t_j) ou (t_j, p_i) et le poids associé à un arc est un entier positif.

Nous pourrions considérer des ensembles ***P*** et ***T*** dénombrables, plutôt que finis, comme on l'a fait pour les automates. Il s'avère, cependant, qu'un nombre fini de transitions et de places est presque toujours adéquat pour modéliser les SED rencontrés en pratique.

Un graphe de réseau de Petri a une structure un peu plus compliquée qu'un diagramme de transition d'état d'un automate :

- les nœuds d'un diagramme de transition d'état correspondent aux états, sélectionnés dans un ensemble unique ***X***. Dans un graphe de réseau de Petri, les nœuds sont soit des places, sélectionnées dans l'ensemble ***P***, soit des transitions, sélectionnées dans l'ensemble ***T***.
- dans un diagramme de transition d'état il y a un seul arc pour chaque événement causant une transition d'état. Dans un graphe de réseau de Petri, des arcs multiples peuvent connecter deux nœuds, ou bien, un poids est affecté à chaque arc pour représenter le nombre d'arcs. C'est une structure de multigraphe.

Lorsqu'on décrit un graphe de réseau de Petri, il est pratique d'utiliser $\boldsymbol{I}(t_j)$ pour représenter l'ensemble des places à l'entrée de la transition t_j.

De même $\boldsymbol{O}(t_j)$ représente l'ensemble des places en sortie de la transition t_j. Ainsi

$$\boldsymbol{I}(t_j) = \{p_i \in \boldsymbol{P}: (p_i, t_j) \in \boldsymbol{A}\},\ \boldsymbol{O}(t_j) = \{p_i \in \boldsymbol{P}: (t_j, p_i) \in \boldsymbol{A}\}$$

Une notation similaire peut être utilisée pour décrire les transitions en entrée et en sortie d'une place donnée p_i : $\boldsymbol{I}(p_i)$ et $\boldsymbol{O}(p_i)$.

On représentera les poids par des arcs multiples dans un graphe lorsqu'ils sont petits, sinon l'écriture d'un poids sur l'arc reliant deux nœuds est une représentation bien plus efficace. Par défaut, si aucun poids n'est inscrit sur un arc d'un graphe de réseau de Petri, il est supposé égal à 1.

Exemple : Considérons le graphe de réseau de Petri simple défini par

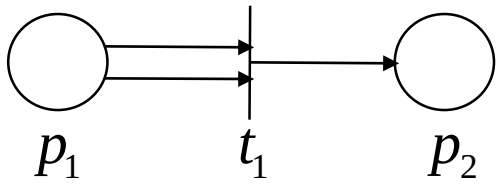
$$\boldsymbol{P} = \{p_1, p_2\} \quad \boldsymbol{T} = \{t_1\} \quad \boldsymbol{A} = \{(p_1, t_1), (t_1, p_2)\}$$

$$w(p_1, t_1) = 2 \quad w(t_1, p_2) = 1$$

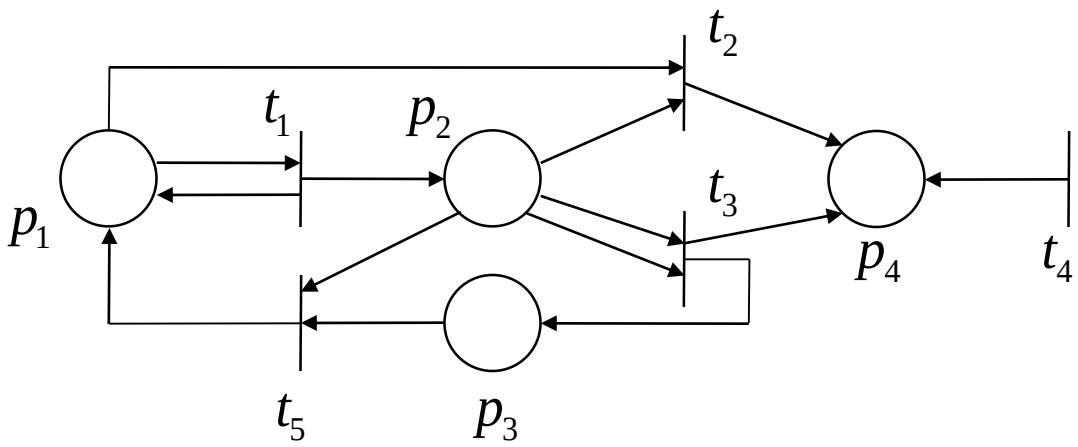
Dans ce cas, $\boldsymbol{I}(t_1) = \{p_1\}$ et $\boldsymbol{O}(t_1) = \{p_2\}$.

Le graphe de réseau de Petri correspondant est montré ci-dessous.

Le fait que $w(p_1, t_1) = 2$ est indiqué par la presence de deux arcs entrants de la place p_1 vers la transition t_1.



Exemple : Considérons le graphe de réseau de Petri suivant



Le réseau de Petri qu'il représente est spécifié par

$$\boldsymbol{P} = \{p_1, p_2, p_3, p_4\} \qquad \boldsymbol{T} = \{t_1, t_2, t_3, t_4, t_5\}$$

$$\boldsymbol{A} = \{(p_1, t_1), (p_1, t_2), (p_2, t_2), (p_2, t_3), (p_2, t_5), (p_3, t_5),$$
$$(t_1, p_1), (t_1, p_2), (t_2, p_4), (t_3, p_3), (t_3, p_4), (t_4, p_4), (t_5, p_1)\}$$

$$w(p_1, t_1) = 1 \quad w(p_1, t_2) = 1 \quad w(p_2, t_2) = 1$$
$$w(p_2, t_3) = 2 \quad w(p_2, t_5) = 1 \quad w(p_3, t_5) = 1$$
$$w(t_1, p_1) = 1 \quad w(t_1, p_2) = 1 \quad w(t_2, p_4) = 1$$
$$w(t_3, p_3) = 1 \quad w(t_3, p_4) = 1 \quad w(t_4, p_4) = 1 \quad w(t_5, p_1) = 1$$

On remarque que la transition t_4 n 'a aucune place en entrée. Si on identifie les transitions à des événements et les places à des conditions associées aux occurrences des événements, alors l'événement correspondant à t_4 a lieu de manière inconditionnelle. Au contraire, l'événement correspondant à la transition t_2, par exemple, dépend de certaines conditions, associées aux places p_1 et p_2.

2.3.2 Marquages et espace d'état d'un réseau de Petri

Reprenons l'idée que les transitions dans un graphe de réseau de Petri représentent les événements pilotant un SED, et que les places décrivent les conditions sous lesquelles ces événements peuvent se produire. Dans ce cadre, nous avons besoin d'un mécanisme indiquant si ces conditions sont en fait remplies ou non.

Ce mécanisme est obtenu en assignant des jetons aux places. Un jeton est quelque chose que l'on « met dans une place » essentiellement pour indiquer que la condition décrite par cette place est satisfaite. La façon dont les jetons sont affectés à un graphe de réseau de Petri définit un marquage.

Formellement, un marquage d'un réseau de Petri $(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w)$ est une fonction

$$x : P \to \mathbb{N} = \{0, 1, 2, \dots\}.$$

Ainsi, le marquage x définit le vecteur ligne

$$\boldsymbol{x} = [x(p_1), x(p_2), \dots, x(p_n)],$$

où n est le nombre de places dans le réseau de Petri.

La $i$ème composante de ce vecteur désigne le nombre (entier non-négatif) de jetons dans la place p_i, $x(p_i) \in \mathbb{N}$.

Dans un graphe de réseau de Petri, un jeton est représenté par un petit disque situé dans la place appropriée.

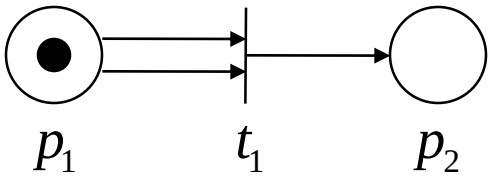
Définition : Un ***réseau de Petri marqué*** est un quintuplet $(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w, x)$ où
$(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w)$ est un graphe de réseau de Petri, et
x est un marquage de l'ensemble des places $\boldsymbol{P}$;
$\boldsymbol{x} = [x(p_1), x(p_2), \dots, x(p_n)] \in \mathbb{N}^n$ est le vecteur ligne associé à x.

Exemple : graphe de réseau de Petri simple avec marquages
Revenant au graphe de réseau de Petri défini par

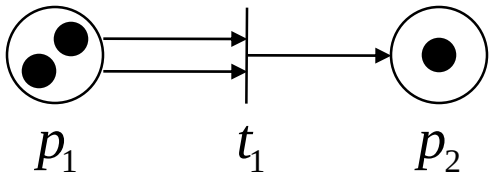
$$\boldsymbol{P} = \{p_1, p_2\} \quad \boldsymbol{T} = \{t_1\} \quad \boldsymbol{A} = \{(p_1, t_1), (t_1, p_2)\}$$
$$w(p_1, t_1) = 2 \quad w(t_1, p_2) = 1,$$

voici deux marquages possibles, définis par les vecteurs lignes

$$x_1 = [1,\ 0] \quad \text{et} \quad x_2 = [2,\ 1]$$



$\boldsymbol{x}_1 = [1,\ 0]$



$\boldsymbol{x}_2 = [2,\ 1]$

Pour simplifier, on appelera dorénavant un réseau de Petri marqué juste un « réseau de Petri ». De plus, puisque nos efforts de modélisation de systèmes fondés sur le concept d'état, dans le cas d'un réseau de Petri on identifiera le marquage des places avec l'***état*** du réseau de Petri.
Autrement dit, on définit l'état d'un réseau de Petri comme étant son ***vecteur*** ligne ***de marquage***

$$\boldsymbol{x} = [x(p_1), x(p_2), \ldots, x(p_n)].$$

Le nombre de jetons attribués à une place est un entier non-négatif arbitraire, pas nécessairement borné. Ainsi le nombre d'états qu'il est possible d'avoir est, en général, infini et l'espace d'état $\boldsymbol{X}$ d'un réseau de Petri avec n places est défini par tous les vecteurs n-dimensionnels dont les composantes sont des entiers non-negatifs, c.-à-d. $\boldsymbol{X} = \mathbb{N}^n$.

Les définitions ci-dessus ne décrivent pas explicitement le mécanisme de transition d'état des réseaux de Petri. C'est clairement un point crucial puisque nous voulons utiliser les réseaux de Petri pour modéliser des SED dynamiques.

Il s'avère que le mécanisme de transition d'état réside dans la structure même du graphe du réseau de Petri.

Afin de définir le mécanisme de transition d'état, nous devons en premier lieu introduire la notion de ***transition autorisée***. Essentiellement, pour qu'une transition $t \in \boldsymbol{T}$ se produise ou soit « permise », il faut que chaque place (c.-à-d. condition) en entrée de la transition contienne au moins un jeton. En fait, puisque l'on peut avoir des poids sur les arcs des places aux transitions, on utilise une définition un peu plus générale.

Définition : Une transition $t_j \in \boldsymbol{T}$ dans un réseau de Petri est dite ***autorisée*** si

$$x(p_i) \geq w(p_i, t_j) \text{ pour toute } p_i \in \boldsymbol{I}(t_j)$$

Ainsi, la transition t_j dans le réseau de Petri est premise quand le nombre de jetons dans p_i est au moins aussi grand que le poids de l'arc connectant p_i à t_j, pour toutes les places p_i en entrée de la transition t_j.

Dans la dernière figure

- avec l'état $\boldsymbol{x}_1$, $x(p_1) = 1 < w(p_1, t_1) = 2$ donc t_1 n'est pas autorisée, mais
- avec l'état $\boldsymbol{x}_2$, on a $x(p_1) = 2 = w(p_1, t_1)$, et t_1 est autorisée.

Comme les places sont associées aux conditions pour qu'une transition se produise, une transition est donc autorisée lorsque toutes les conditions requises pour qu'elle se produise soient satisfaites; les jetons constituent le mécanisme utilisé pour déterminer la satisfaction des conditions.
L'ensemble des transitions permises dans un état donné du réseau de Petri est équivalent à l'ensemble des événements actifs dans un état donné d'un automate.
On est maintenant prêt à décrire l'évolution dynamique des réseaux de Petri.

2.3.3 Dynamique des réseaux de Petri

Dans les automates, le mécanisme de transition d'état est tout simplement inscrit dans les arcs qui connectent les nœuds (états) du diagramme de transition d'état ou, équivalemment par la fonction de transition f.
Le mécanisme de transition d'état des réseaux de Petri est fourni par les jetons se déplaçant dans le réseau et donc changeant l'état du réseau de Petri.

Quand une transition est autorisée, on dit qu'elle est franchissable ou qu'elle peut se produire (le terme « franchir » est standard dans les publications sur les réseaux de Petri). La fonction de transition d'état d'un réseau de Petri est définie par le changement de l'état du réseau de Petri dû au franchissement d'une transition autorisée.

Définition : (***dynamique d'un réseau de Petri***)
La fonction de transition d'état, $f : \mathbb{N}^n \times \boldsymbol{T} \to \mathbb{N}^n$, du réseau de Petri $(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w, x)$ est définie pour la transition $t_j \in \boldsymbol{T}$ si et seulement si

$$x(p_i) \geq w(p_i, t_j) \text{ pour toute } p_i \in \boldsymbol{I}(t_j). \qquad (2.1)$$

Si $f(\boldsymbol{x}, t_j)$ est définie, on a alors $\boldsymbol{x}' = f(\boldsymbol{x}, t_j)$, où

$$x'(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i), \quad i = 1, \ldots, n \qquad (2.2)$$

La condition (2.1) garantit que la fonction de transition d'état est définie seulement pour des transitions autorisées; une « transition autorisée » est donc équivalente à « événement possible » dans un automate. Mais alors que la fonction de transition d'état d'un automate est arbitraire, elle dépend de la structure du graphe d'un réseau de Petri. Ainsi, l'état suivant défini par (2.2) dépend explicitement des places en entrée et en sortie d'une transition et des poids des arcs connectant ces places à la transition.

Selon (2.2), si p_i est une place en entrée de t_j, elle perd autant de jetons que le poids de l'arc de p_i à t_j; si c'est une place en sortie de t_j, elle gagne autant de jetons que le poids de l'arc de t_j à p_i.
Si p_i est une place à la fois en entrée et en sortie de t_j, alors (2.2) enlève $w(p_i, t_j)$ jetons de p_i, et y remet immédiatement $w(t_j, p_i)$ nouveaux jetons.

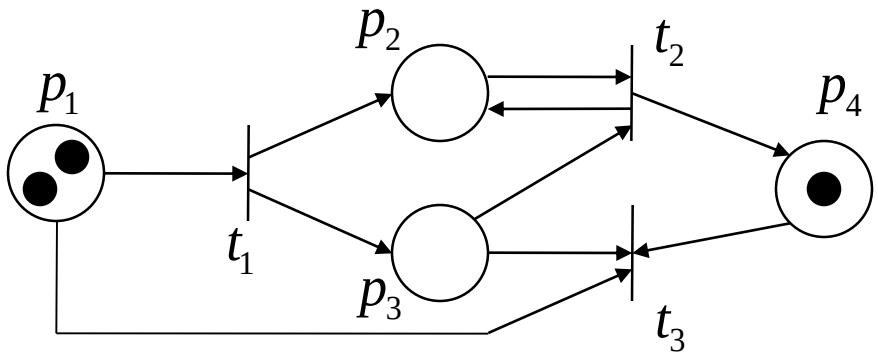
Le nombre de jetons n'est pas nécessairement conservé quand une transition est franchie. C'est clair d'après (2.2), puisqu'il est tout à fait possible que

$$\sum_{p_i \in \boldsymbol{P}} w(t_j, p_i) > \sum_{p_i \in \boldsymbol{P}} w(p_i, t_j) \text{ ou } \sum_{p_i \in \boldsymbol{P}} w(t_j, p_i) < \sum_{p_i \in \boldsymbol{P}} w(p_i, t_j).$$

Il est tout à fait possible que dans un réseau de Petri, après quelques franchissements de transitions, l'état résultant soit $\boldsymbol{x} = [0, \ldots, 0]$, ou que le nombre de jetons dans une ou plusieurs places croisse de manière non bornée après un nombre arbitrairement grand de franchissements.

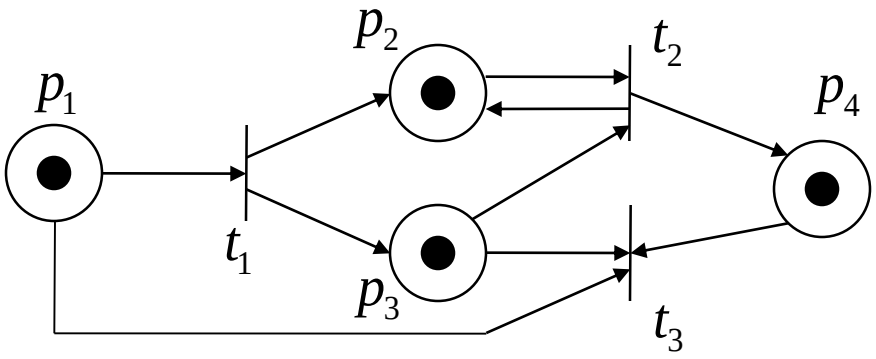
Ce dernier phénomène est une différence majeure avec les automates, où, par définition, les automates d'état fini ont seulement un nombre fini d'états. Par contraste, un graphe de réseau de Petri fini peut donner un réseau de Petri avec un nombre d'états non borné.

Exemple : séquences de franchissements de transitions dans un RdP



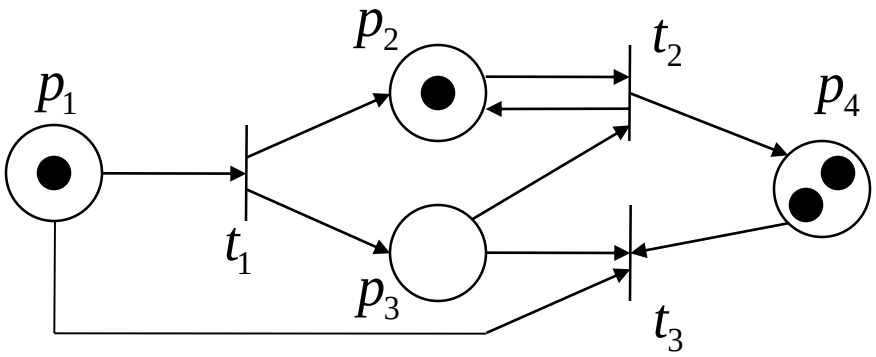
État initial $\boldsymbol{x}_0 = [2, 0, 0, 1]$
seule la transition t_1 est permise

Le franchissement de t_1 amène le RdP dans l'état $\boldsymbol{x}_1$



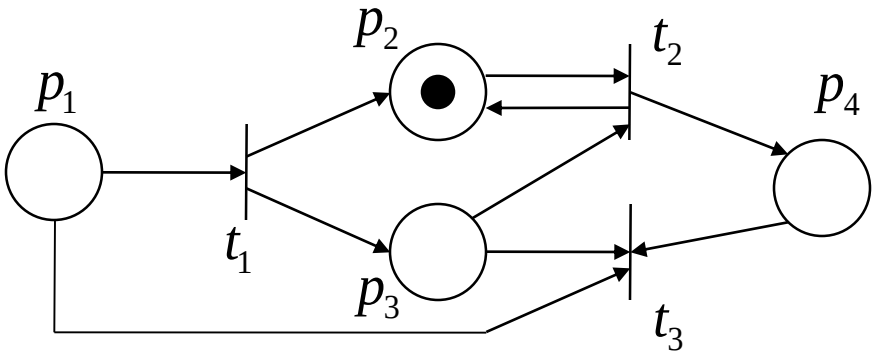
État $\boldsymbol{x}_1 = [1, 1, 1, 1]$
t_1, t_2 et t_3 sont permises

3 scénarios



Scénario du franchissement de t_2
→ état $\boldsymbol{x}_2 = [1, 1, 0, 2]$

seule t_1 est autorisée



Scénario du franchissement de t_3
→ état $\boldsymbol{x}_2 = [0, 1, 0, 0]$

pas de transition autorisée :
état de « blocage »

L'exemple qui précède montre que la séquence selon laquelle les transitions sont franchies n'est pas pré-specifiée dans un réseau de Petri. Dans l'état x_1 n'importe laquelle des trois transitions peut être ensuite franchie. C'est comme avoir trois événements dans l'ensemble des événements actifs associé à l'un des états d'un automate modélisant un SED.

Dans l'étude des réseaux de Petri non-temporisés en tant que modèles de SED, nous devons examiner toute séquence possible de transitions (événements), comme nous l'avons fait pour les automates.

Une observation importante sur le comportement dynamique des réseaux de Petri est qu'on ne peut pas nécessairement accéder à n'importe quel état dans $\mathbb{N}^n$ à partir d'un état initial donné. Ainsi, dans l'exemple de graphe de RdP simple avec l'état initial $\boldsymbol{x}_2 = [2, 1]$ on voit que le seul état accessible à partir de $\boldsymbol{x}_2$ est $[0, 2]$.

Cela conduit à définir l'ensemble des états accessibles, $\boldsymbol{R}[(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w, x)]$, du réseau de Petri $(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w, x)$. Pour ce faire, on doit d'abord étendre la fonction de transition f du domaine $\mathbb{N}^n \times \boldsymbol{T}$ au domaine $\mathbb{N}^n \times \boldsymbol{T}^*$, de la même façon qu'on a étendu la function de transition des automates dans la section 2.2.2 :

$f(\boldsymbol{x}, \varepsilon) := \boldsymbol{x}$ (ici ε représente l'absence de franchissement de transition.)

$f(\boldsymbol{x}, st) := f(f(\boldsymbol{x}, s), t)$ pour $s \in \boldsymbol{T}^*$ et $t \in \boldsymbol{T}$

Définition : L'ensemble des ***états accessibles*** du réseau de Petri $(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w, x)$ est

$$\boldsymbol{R}[(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w, x)] \coloneqq \{\boldsymbol{y} \in \mathbb{N}^n : \exists s \in \boldsymbol{T}^* \, [f(\boldsymbol{x}, s) = \boldsymbol{y}]\}$$

Ces définitions, de la forme étendue de la fonction de transition d'état et de l'ensemble des états accessibles, supposent que les transitions autorisées sont franchies une à la fois.

Dans l'exemple du réseau de Petri simple, après la première transition, où les trois transitions t_1, t_2 et t_3 sont autorisées, les transitions t_1 et t_2 peuvent être franchies simultanément (l'ordre n'importe pas), puisqu'elles « consomment » des jetons d'ensembles de places disjoints : $\{p_1\}$ pour t_1 et $\{p_2 , p_3\}$ pour t_2.

Puisqu'on s'intéresse à tous les états accessibles, et qu'on étiquettera les transitions avec des noms d'événements et considérera les langages représentés par des réseaux de Petri, on excluera désormais les franchissements simultanés de transitions et on supposera que les transitions sont franchies une par une.

Équations d'état

Retournons à l'équation (2.2) décrivant comment la valeur de l'état d'une place individuelle change quand une transition est franchie.
Il n'est pas difficile de voir comment générer une équation vectorielle à partir de (2.2), afin de spécifier $\boldsymbol{x'} = [x'(p_1), x'(p_2), \ldots, x'(p_n)]$, le nouvel état du réseau de Petri étant donné l'état courant $\boldsymbol{x} = [x(p_1), x(p_2), \ldots, x(p_n)]$ et le fait qu'une transition particulière, t_j , a été franchie.

Pour ce faire, on définit le vecteur de franchissement ***u***, un vecteur ligne de dimension *m* de la forme

$$\boldsymbol{u}_j = [0, \ldots, 0, 1, 0, \ldots, 0], \qquad (2.3)$$

où le seul 1 apparaît en *j*ème position, $j \in \{1, \ldots, m\}$, pour indiquer le franchissement de la *j*ème transition est en cours. En outre, on définit la matrice d'incidence d'un réseau de Petri, *B*, une matrice $m \times n$ dont l'élément (j, i) est de la forme

$$b_{ji} = w(t_j, p_i) - w\,(p_i, t_j) \qquad (2.4)$$

correspondant à la différence des poids qui apparaît dans (2.2) en mettant à jour $x(p_i)$.

Utilisant la matrice d'incidence B, on peut maintenant écrire une équation de mise à jour du vecteur d'état

$$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{u}B \qquad (2.5)$$

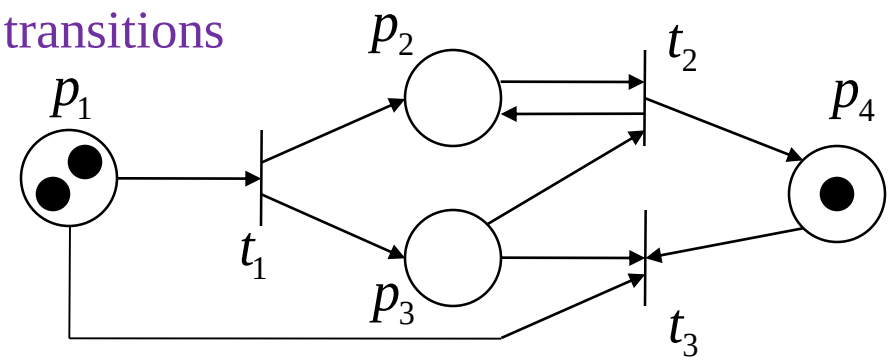
qui décrit le processus de transition d'état suite à une « entrée » $\boldsymbol{u}$, c.-à-d. une transition particulière.

La $i$ème équation dans (2.5) est précisément l'équation (2.2). On voit donc que $f(\boldsymbol{x}, t_j) = \boldsymbol{x} + \boldsymbol{u}_j B$, où $f(\boldsymbol{x}, t_j)$ est la fonction de transition définie plus haut. L'argument t_j dans cette fonction indique que c'est la $j$ème composante de $\boldsymbol{u}$ qui est non-nulle.

L'équation d'état fournit un outil algébrique pratique et une alternative à des méthodes purement basées sur les graphes pour décrire le processus de franchissement des transitions et de changement d'état d'un RdP.

Exemple : franchissements de transitions

état initial $x_0 = [2, 0, 0, 1]$



On peut d'abord, en inspectant le graphe du réseau de Petri, écrire la matrice d'incidence, qui est dans ce cas

$$B = \begin{bmatrix} -1 & 1 & 1 & 0 \\ 0 & \mathbf{0} & -1 & 1 \\ -1 & 0 & -1 & -1 \end{bmatrix}$$

Ainsi, l'élément $(1,2)$ est donné par $w(t_1, p_2) - w(p_2, t_1) = 1 - 0$.

Utilisant (2.5), l'équation quand la transition t_1 est franchie à l'état $\boldsymbol{x}_0$ est

$$\boldsymbol{x}_1 = [2\ \ 0\ \ 0\ \ 1] + [1\ \ 0\ \ \ 0] \begin{bmatrix} -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & 0 & -1 & -1 \end{bmatrix}$$

$$= [2\ \ 0\ \ 0\ \ 1] + [-1\ \ 1\ \ 1\ \ 0] = [1\ \ \ 1\ \ \ 1\ \ \ 1]\,,$$

qui est bien le vecteur d'état obtenu précédemment.

On détermine de même $\boldsymbol{x}_2$, résultant du franchissement de t_2 à partir de $\boldsymbol{x}_1$,

$$\boldsymbol{x}_2 = [1\ \ 1\ \ 1\ \ 1] + [0\ \ 1\ \ \ 0] \begin{bmatrix} -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & 0 & -1 & -1 \end{bmatrix}$$

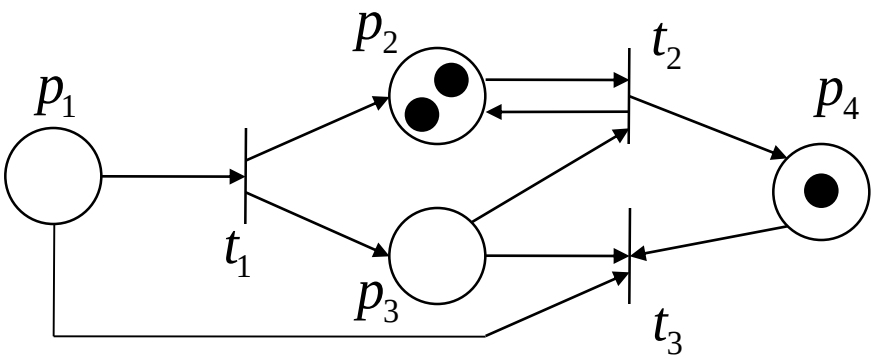
$$= [1\ \ 1\ \ 1\ \ 1] + [0\ \ 0\ \ -1\ \ 1] = [1\ \ \ 1\ \ \ 0\ \ \ 2]$$

Vu comme un système dynamique, un réseau de Petri génère des trajectoires similaires à celles d'un automate. Specifiquement, la trajectoire d'un RdP est une séquence d'états $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots\}$ résultant d'une séquence de franchissements de transitions en entrée $\{e_1, e_2, \ldots\}$ où $e_k = t_j$ est la *k*ème transition franchie.

Étant donné l'état $\boldsymbol{x}_0$, la séquence complète des états peut être générée selon

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{t}_k) = \boldsymbol{x}_k + \boldsymbol{u}_k B$$

où $\boldsymbol{u}_k$ contient l'information sur la *k*ème transition franchie.
Si un état est atteint tel qu'il n'est plus possible de franchir de transition, on dit que l'exécution du réseau de Petri bloque dans cet état (comme dans l'état $[0, 2, 0, 1]$).



De la même façon que l'on a étendu la function de transition f du domaine $\mathbb{N}^n \times \boldsymbol{T}$ au domaine $\mathbb{N}^n \times \boldsymbol{T}^*$,

$f(\boldsymbol{x}, \varepsilon) \coloneqq \boldsymbol{x}$

$f(\boldsymbol{x}, st) \coloneqq f(f(\boldsymbol{x},s),t)$ pour $s \in \boldsymbol{T}^*$ et $t \in \boldsymbol{T}$,

l'état $\boldsymbol{x}_{k+1}$ peut être obtenu directement selon :

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_0, e_1 e_2 \ldots e_k) = \boldsymbol{x}_0 + \boldsymbol{u}B \qquad (2.6)$$

où la $j$ème composante de $\boldsymbol{u}$ est égale au nombre de fois que la transition t_j est franchie.

Cette équation est appelée l'***équation fondamentale*** du réseau de Petri et le vecteur $\boldsymbol{u}$ est appelé le ***vecteur caractéristique*** de la séquence de transitions $e_1 e_2 \ldots e_k$.

Attention : il n'est pas suffisant de trouver un vecteur caractéristique $\boldsymbol{u}$ à composantes non-négatives vérifiant l'équation (2.6) pour être sûr qu'il existe une séquence de transitions effectivement franchissables de l'état $\boldsymbol{x}_0$ à l'état $\boldsymbol{x}_{k+1}$, et plus généralement d'un état $\boldsymbol{x}$ vers un état $\boldsymbol{x}'$.

2.3.4 Langages de réseaux de Petri

Jusqu'à présent, nous nous sommes concentrés sur la dynamique de l'état des réseaux de Petri, où les transitions sont énumérées en tant qu'éléments de l'ensemble ***T***. Nous avons supposé que les transitions correspondent aux événements, mais nous n'avons pas fait d'énoncés précis au sujet de cette correspondence.

Si on veut considérer les réseaux de Petri comme un formalisme de modélisation pour représenter des langages, comme nous l'avons fait pour les automates, alors il faut spécifier précisément à quel événement correspond chaque transition; cela permettra de spécifier les langages représentés (« reconnu » et « marqué ») par un réseau de Petri.

Soit ***E*** l'ensemble des événements du SED considéré et dont l'on veut modéliser le langage à l'aide d'un RdP. On pourrait, bien sûr, imposer que le modèle « réseau de Petri » du système soit tel que chaque transition dans ***T*** corresponde à un événement distinct de l'ensemble ***E*** des événements, et vice-versa. Mais ce serait inutilement restrictif; dans un automate, deux arcs différents (issus de deux états différents) peuvent être exécutés avec le même événement. Cela conduit à la définition d'un réseau de Petri ***étiquetté***.

Définition : Un ***réseau de Petri étiquetté*** est un octuplet $N = (\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w, \boldsymbol{E}, \ell, x_0, \boldsymbol{X}_m)$ où

$(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w)$ est un graphe de réseau de Petri,

$\boldsymbol{E}$ est un ensemble fini (alphabet) d'événements,

$\ell : \boldsymbol{T} \rightarrow \boldsymbol{E}$ est une fonction d'étiquettage qui associe un événement à chaque transition et peut être étendue à une application $\boldsymbol{T}^* \rightarrow \boldsymbol{E}^*$ de la façon usuelle,

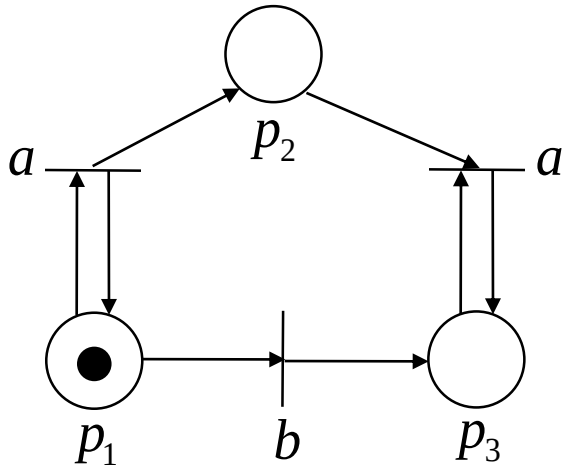
x_0 est un marquage initial de l'ensemble des places $\boldsymbol{P}$,

$\boldsymbol{X}_m \subset \boldsymbol{R}[(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w, x)]$ est un ensemble fini de marquages finaux.

La fonction d'étiquettage ℓ est une function d'étiquettage ε-libre, c.-à-d. qu'aucune transition n'est étiquettée avec le mot vide ε et deux (ou plus) transitions peuvent avoir la même etiquette.

Exemple : Réseau de Petri étiquetté

$\boldsymbol{E} = \{a, b\}$
$x_0 = [1, 0, 0]$
$\boldsymbol{X}_m = \{[0, 0, 1]\}$



Définition : Le ***langage reconnu*** par le réseau de Petri étiquetté $N = (\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w, \boldsymbol{E}, \ell, \boldsymbol{x}_0, \boldsymbol{X}_m)$ est

$$\mathcal{L}(N) := \{\ell(s) \in E^* : s \in \boldsymbol{T}^* \text{ et } f(\boldsymbol{x}_0, s) \text{ est définie}\}$$

Le ***langage marqué*** par N est

$$\mathcal{L}_m(N) := \{\ell(s) \in \mathcal{L}(N) : s \in \boldsymbol{T}^* \text{ et } f(\boldsymbol{x}_0, s) \in \boldsymbol{X}_m\}$$

On voit que ces définitions sont en totale cohérence avec les définitions correspondantes pour les automates.
Elles reflètent l'hypothèse qu'un seul événement peut se produire à un instant donné, ce qui donne des ***langages séquentiels***.
Le langage $\mathcal{L}(N)$ représente toutes les chaînes d'étiquettes de transitions qui sont obtenues avec toutes les séquences (finies) possibles de franchissements de transitions dans N, partant de l'état initial $\boldsymbol{x}_0$ de N; le langage marqué $\mathcal{L}_m(N)$ est le sous-ensemble des chaînes qui laissent le réseau de Petri dans un état qui est un membre de l'ensemble des états marqués donné dans la définition de N.

La classe des langages représentables par des réseaux de Petri étiquettés est

$$\mathcal{PNL} := \{\boldsymbol{K} \subseteq \boldsymbol{E}^* : \exists N = (\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A}, w, \boldsymbol{E}, \ell, \boldsymbol{x}_0, \boldsymbol{X}_m)\ [\mathcal{L}_m(N) = \boldsymbol{K}]\}.$$

Les propriétés de $\mathcal{PNL}$ dependent fortement des hypothèses spécifiques faites au sujet de ℓ (injective ou non) et $\boldsymbol{X}_m$ (fini ou infini).

Exemple : Les langages associés à l'exemple de réseau de Petri étiquetté sont $\mathcal{L}(N) = \{a^m \mid m \geq 0\} \cup \{a^m b a^n \mid m \geq n \geq 0\}$
et $\mathcal{L}_m(N) = \{a^m b a^m \mid m \geq 0\}$.
Ces langages ne sont pas des langages réguliers; aucun ne peut être accepté par un automate d'état fini.

2.3.5 Modèles de réseau de Petri pour les systèmes de files d'attente

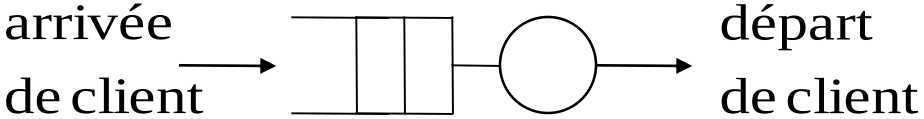
On a vu en 2.2.5 comment les automates peuvent être utilisés pour représenter le comportement dynamique d'une station de service de base. On peut répéter ce processus au moyen d'une structure de réseau de Petri.
On commence par énumérer les événements (transitions) pilotant le système : a : le client **a**rrive
s : le **s**ervice débute (***starts***)
c : le service se **c**lôt et le client part.
On forme l'ensemble des transitions $\boldsymbol{T} = \{a, s, c\}$. (Dans cet exemple, on n'a pas besoin de considerer les RdP étiquettés; on peut supposer que $\boldsymbol{E} = \boldsymbol{T}$ et que ℓ est une bijection entre ces deux ensembles.)

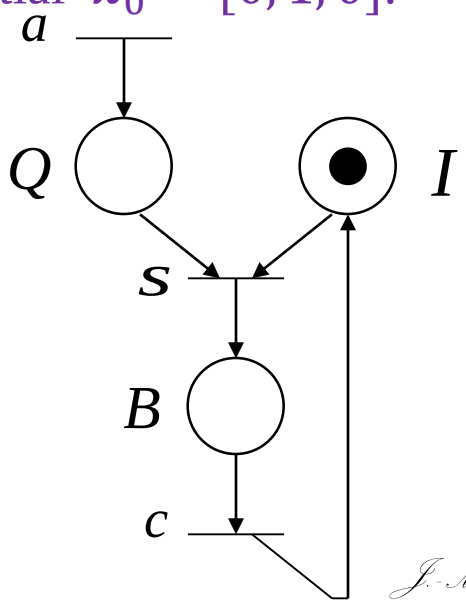
La transition a est spontanée et se produit sans conditions (places en entrée). Par contre, la transition s depend de deux conditions : la présence de clients dans la file d'attente, et le serveur étant inactif.

On représente ces deux conditions par deux places en entrée de cette transition, Q (**q**ueue) and place I (serveur **i**nactif).

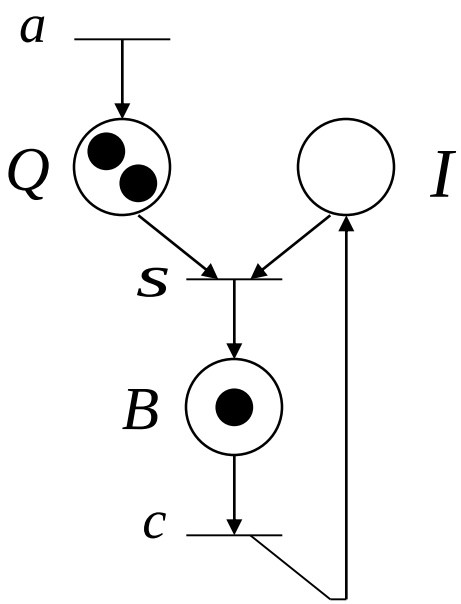
Finalement, la transition c exige que le serveur soit occupé, on introduit donc une place en entrée B (***b**usy server*) pour cela. Ainsi, notre ensemble de places est $P = \{Q, I, B\}$.

Le graphe de réseau de Petri complet est montré ci-dessous, en regard de la station de service de base qu'il modélise. Aucun jeton n'est placé dans Q, indiquant que la file est vide, et un jeton est placé dans I, indiquant que le serveur est inactif. Ceci définit l'état initial $\boldsymbol{x}_0 = [0, 1, 0]$.





Tant qu'il y a un jeton dans Q, une transition c autorise toujours une transition s. Si on suppose que la transition autorisée est franchie immédiatement, ce modèle devient au modèle d'automate utilisant seulement deux événements (arrivées a et départs d), mais spécifiant un ensemble d'événements permis $\boldsymbol{\Gamma}(0) = \{a\}, \boldsymbol{\Gamma}(x) = \{a, d\}$ pour tout $x > 0$.
Puisque la transition a est toujours autorisée, on peut générer diverses trajectoires possibles. À titre d'exemple, la figure ci-dessous montre l'état $[2, 0, 1]$ resultant de la séquence de franchissements de transitions $asaacsa$. Cet état correspond à deux clients en attente dans la queue, tandis qu'un troisième est servi (le premier arrivé dans sequence est déjà parti après la transition c).



Un modèle un peu plus détaillé de la même station de service de base pourrait inclure la transition additionnelle

d : **d**épart du client

qui nécessite la condition F (client **f**ini).

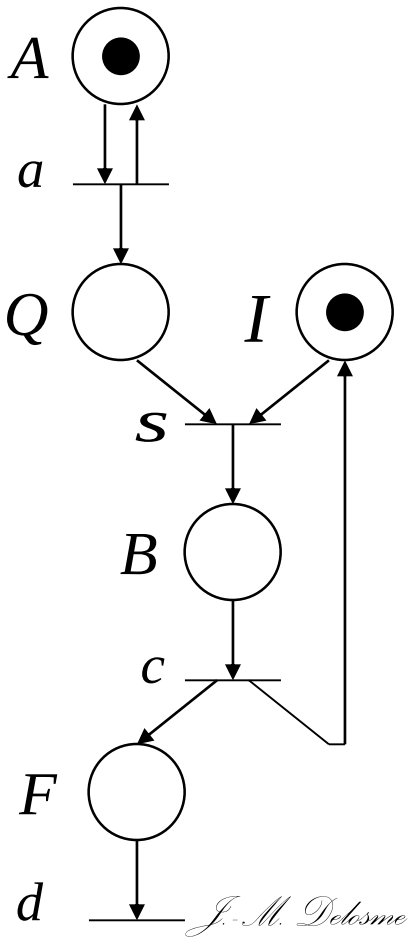
Dans ce cas, la transition c signifie juste que le « service se **c**lôt ».

En sus, le processus d'arrivée externe peut aussi être representé par une place, appelée A, en entrée de la transition a dans la mesure où un jeton est toujours maintenu dans A pour garder la transition a permise.

Ainsi, dans ce modèle alternatif, on a

$$\boldsymbol{T} = \{a, s, c, d\} \text{ et } \boldsymbol{P} = \{A, Q, I, B, F\}$$

Le modèle resultant est montré dans l'état $[1, 0, 1, 0, 0]$.



En comparant ce modèle avec l'automate utilisant seulement les événements $\{a, d\}$, on peut voir que les événements c et d sont combinés en un seul, puisque la transition c autorise toujours la transition d.

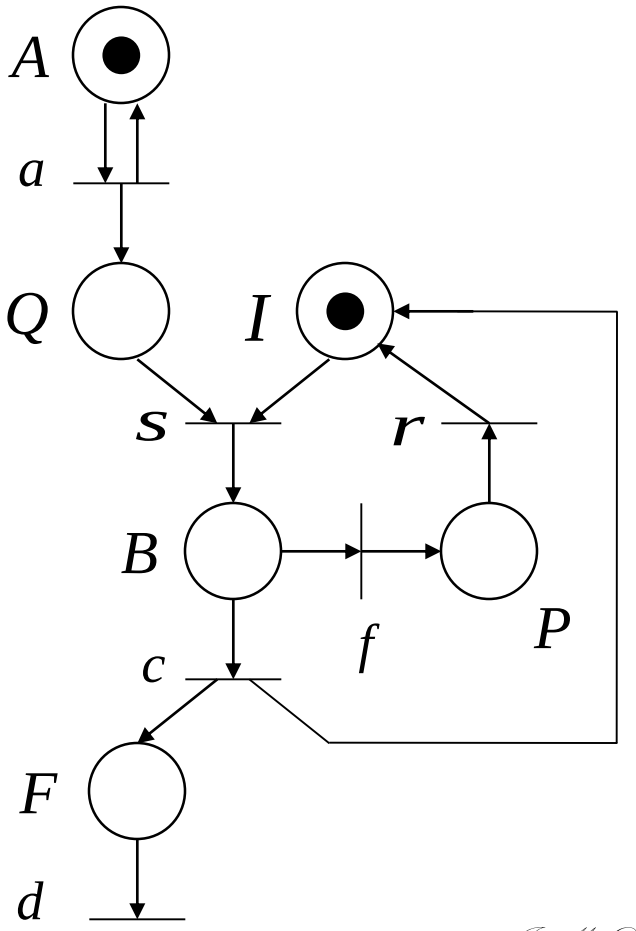
Ceci est encore une indication de la flexibilité de tout processus de modélisation; en général on n'a peut être pas besoin de la transition additionnelle d, mais des applications existent où différencier la « clôture du service » d'un client et son « départ » est utile, sinon nécessaire.

Le modèle peut être modifié davantage si l'on tient compte de la possibilité que le serveur tombe en panne (comme dans le modèle d'automate de la section 2.2.5). Dans ce cas, on introduit deux nouvelles transitions : f : le serveur s'effondre
r : le serveur est réparé
et une place P (serveur en panne) qui est en entrée de la transition r.
Ainsi,

$$\boldsymbol{T} = \{a, s, c, d, f, r\}$$

et $$\boldsymbol{P} = \{A, Q, I, B, F, P\}.$$

Le modèle resultant est montré dans l'état initial $[1, 0, 1, 0, 0, 0]$.



2.3.6 Analyse qualitative

Les réseaux de Petri procurent des outils pour l'analyse qualitative des SED. Nous l'illustrons avec la solution logicielle du problème de la section critique basée sur ***l'algorithme de Peterson :***

```
Tour := 1;
D1 := faux;   D2 := faux;
parbegin
        P1; P2
parend;
```

P_1

```
répéter
        <section restante>
        D1 := vrai;
        Tour := 2;
        tant que (D2 et Tour = 2) faire rien;
        <section critique>
        D1 := faux;
jusqu'à faux;
```

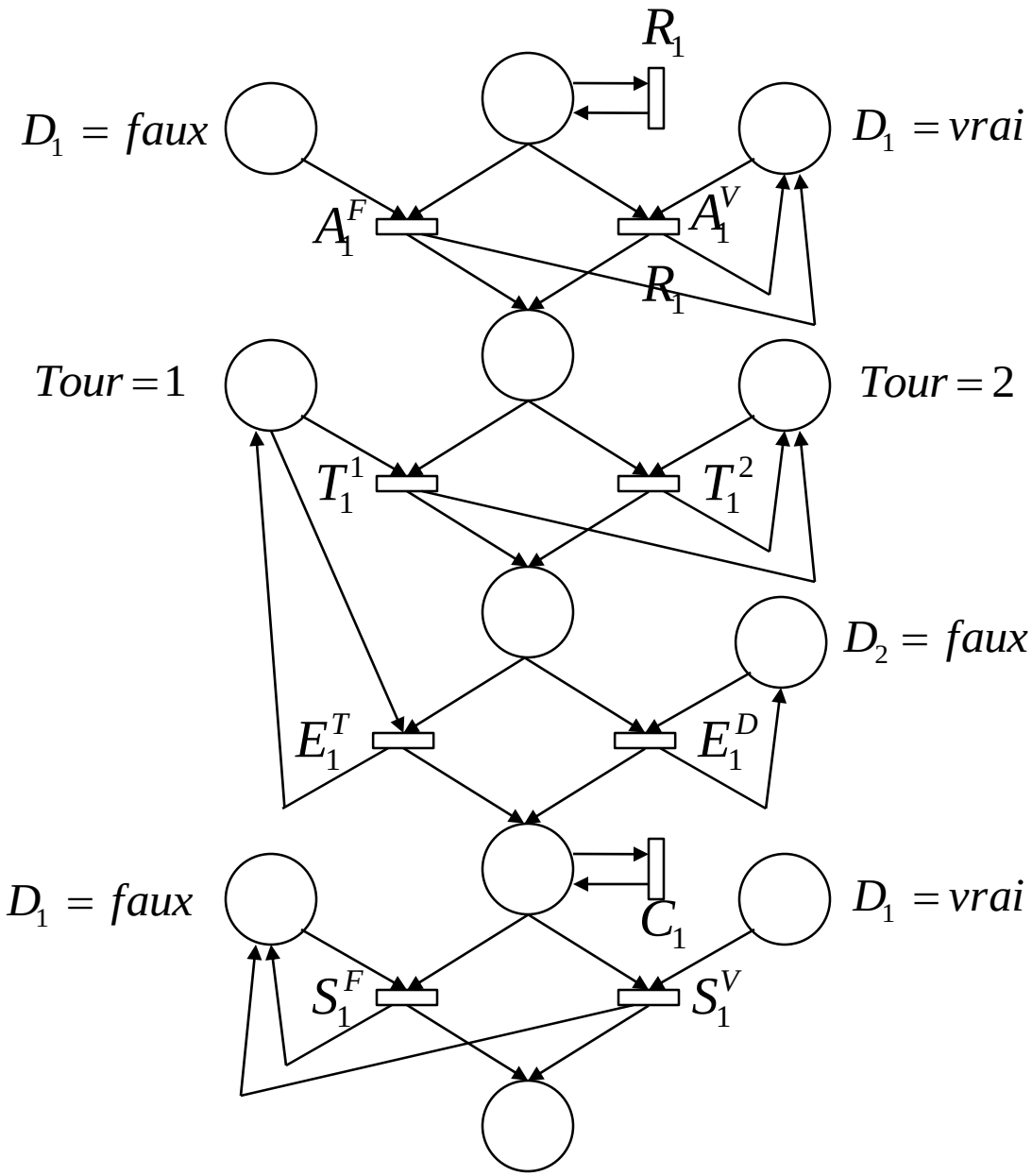
P_2

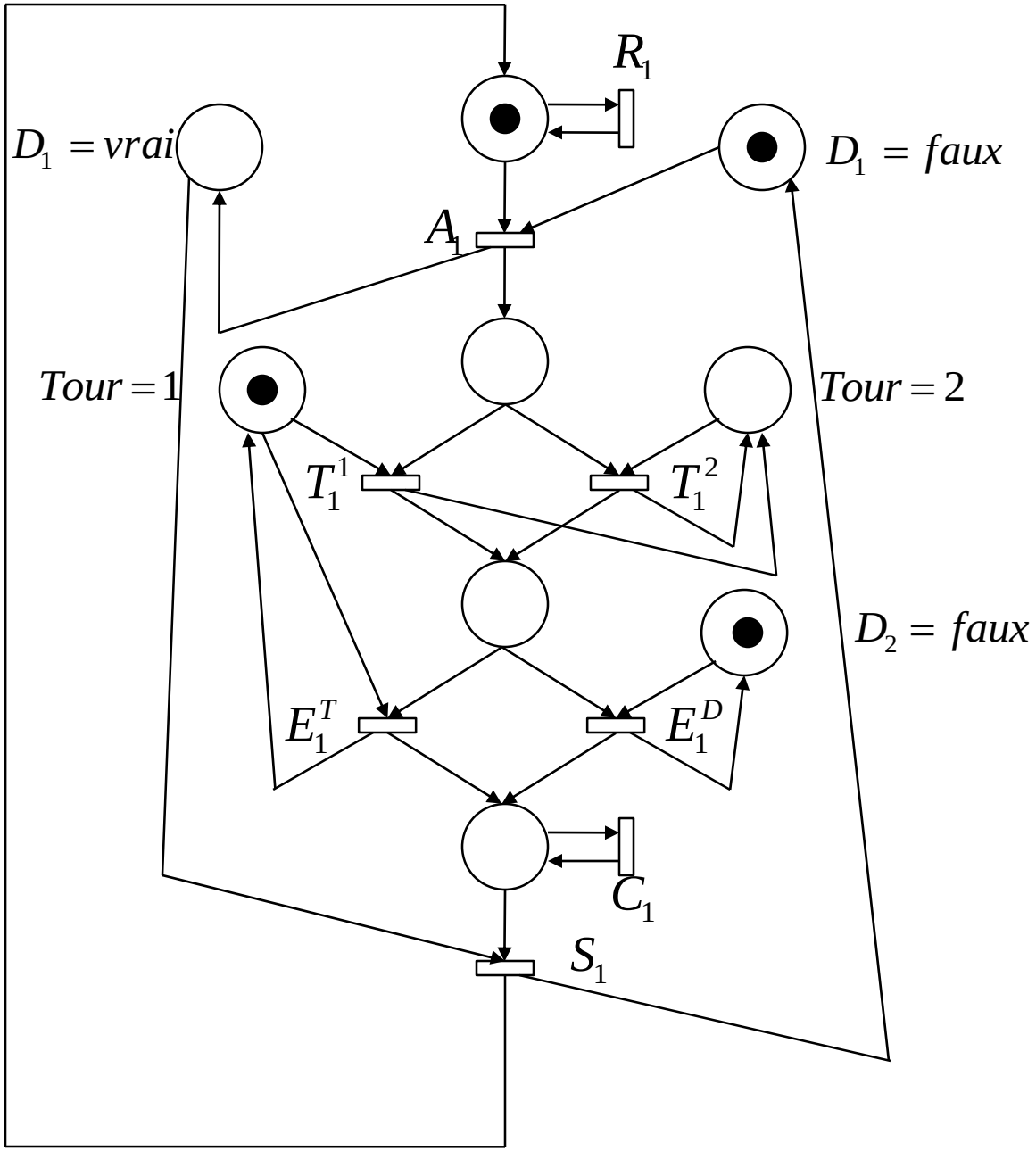
```
répéter
        <section restante>
        D2 := vrai;
        Tour := 1;
        tant que (D1 et Tour = 1) faire rien;
        <section critique>
        D2 := faux;
jusqu'à faux;
```

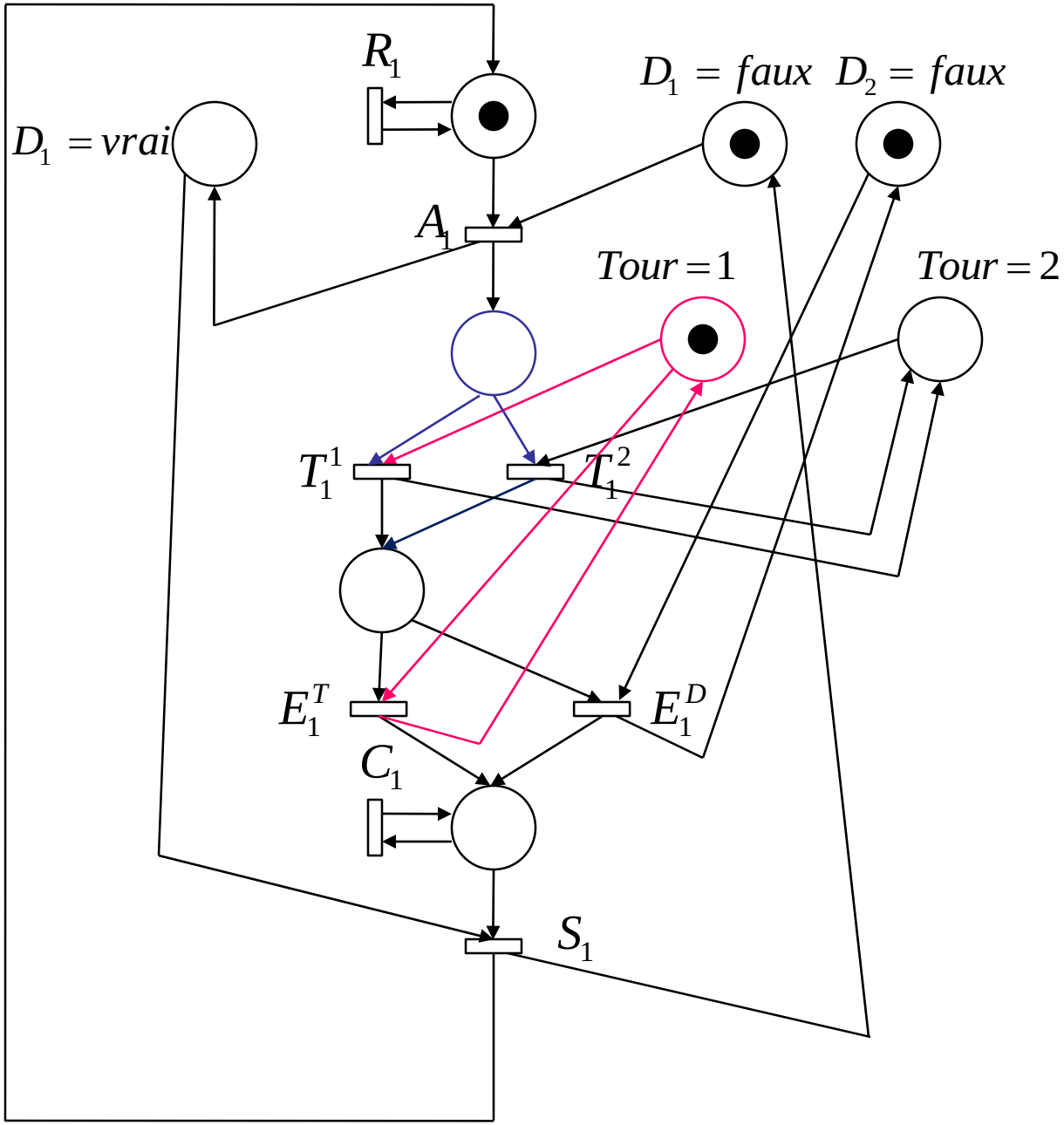
répéter

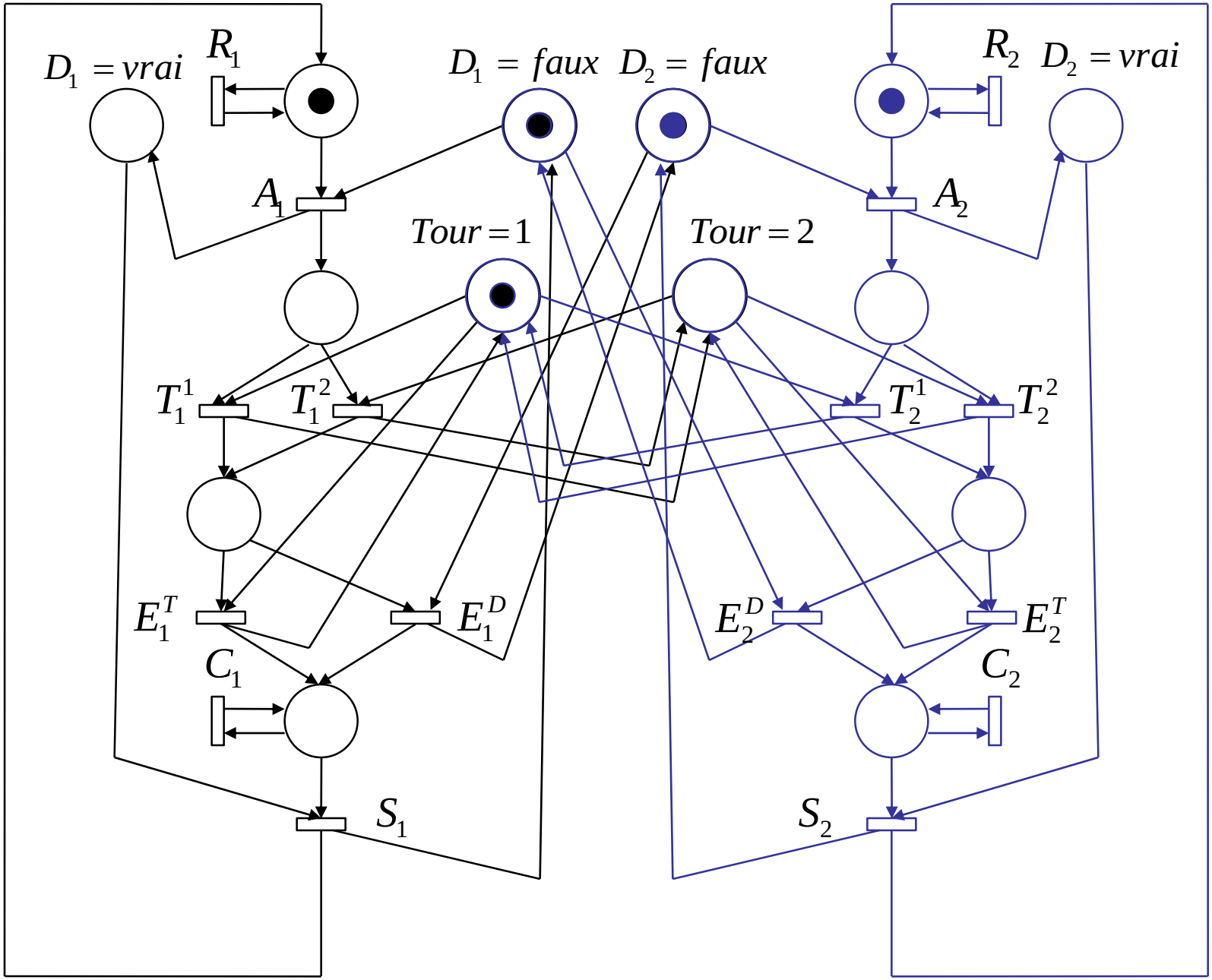
$<\text{section restante}>$	R_1
$D_1 := vrai;$	A_1
$Tour := 2;$	T_1
$\text{tant que } (D_2 \text{ et } Tour = 2) \text{ faire } rien;$	E_1
$<\text{section critique}>$	C_1
$D_1 := faux;$	S_1

jusqu'à $faux;$


R_1
$D_1 = faux$
$D_1 = vrai$
A_1^F
A_1^V
R_1
$Tour = 1$
$Tour = 2$
T_1^1
T_1^2
$D_2 = faux$
E_1^T
E_1^D
$D_1 = faux$
$D_1 = vrai$
C_1
S_1^F
S_1^V


R_1
$D_1 = vrai$
$D_1 = faux$
A_1
$Tour = 1$
$Tour = 2$
T_1^1
T_1^2
$D_2 = faux$
E_1^T
E_1^D
C_1
S_1


R_1
$D_1 = faux$ $D_2 = faux$
$D_1 = vrai$
A_1
$Tour = 1$
$Tour = 2$
T_1^1
T_1^2
E_1^T
E_1^D
C_1
S_1



répéter

< section restante >

$D_1 := vrai$;

$Tour := 2$;

tant que (D_2 et $Tour = 2$) faire *rien* ;

< section critique >

$D_1 :=$ *faux* ;

jusqu'à *faux* ;

répéter

< section restante >

$D_2 := vrai$;

$Tour := 1$;

tant que (D_1 et $Tour = 1$) faire *rien* ;

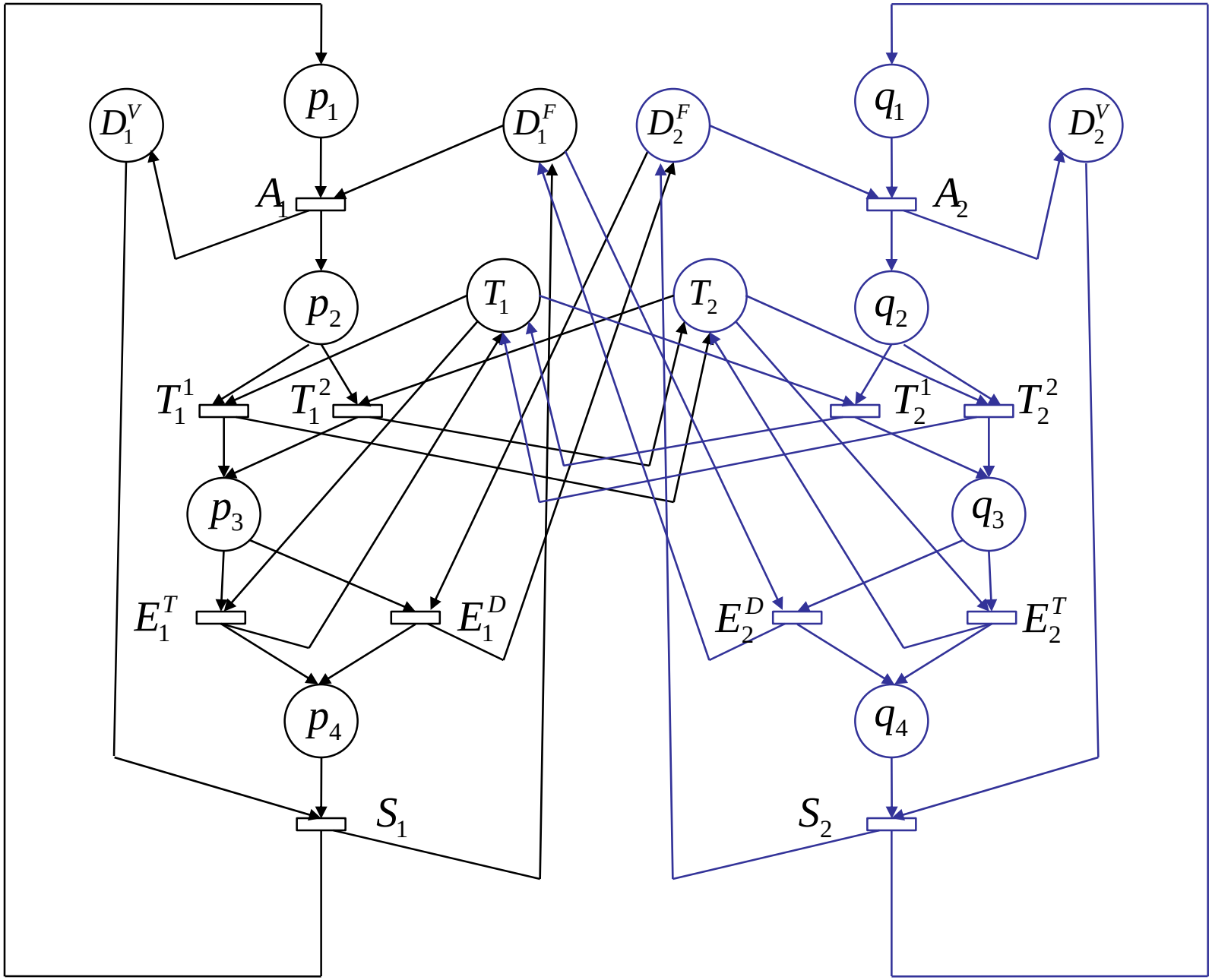
< section critique >

$D_2 :=$ *faux* ;

jusqu'à *faux* ;

Exclusion mutuelle :

à tout instant, au plus l'un des deux processus est en train d'exécuter une instruction de sa section critique



Notation pour l'équation d'état

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

Matrice B dans l'équation d'état

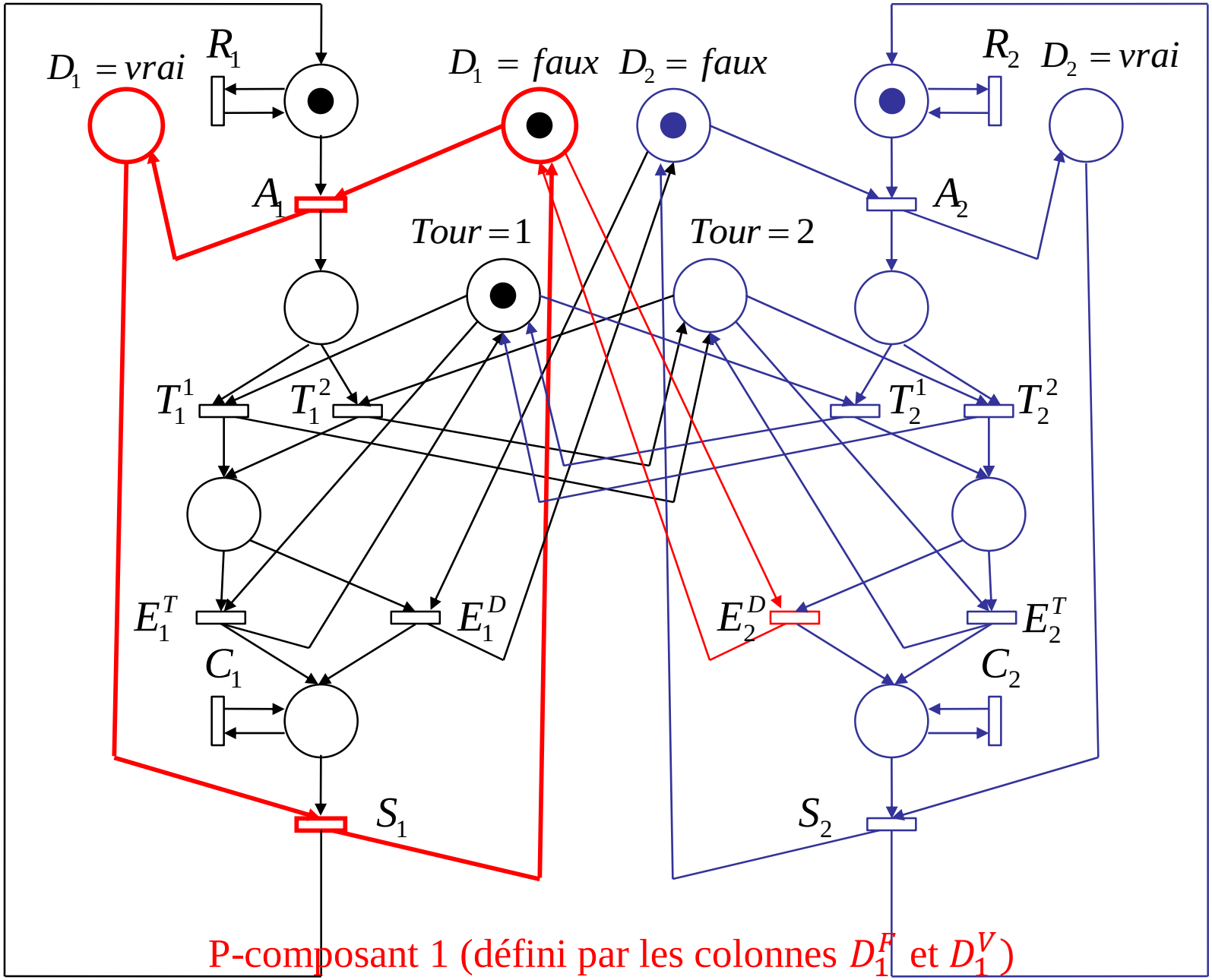
$$\boldsymbol{v}_1 = [\,1,\ 1,\ 0\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]^T$$

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

P-invariant : $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{u}B$ et $B\boldsymbol{v} = 0 \Rightarrow \boldsymbol{x}'\boldsymbol{v} = \boldsymbol{x}\boldsymbol{v} + \boldsymbol{u}B\boldsymbol{v} = \boldsymbol{x}\boldsymbol{v}$

Conservation de jetons (somme pondérée en général)

P-invariant 1 (défini par le vecteur v_1)



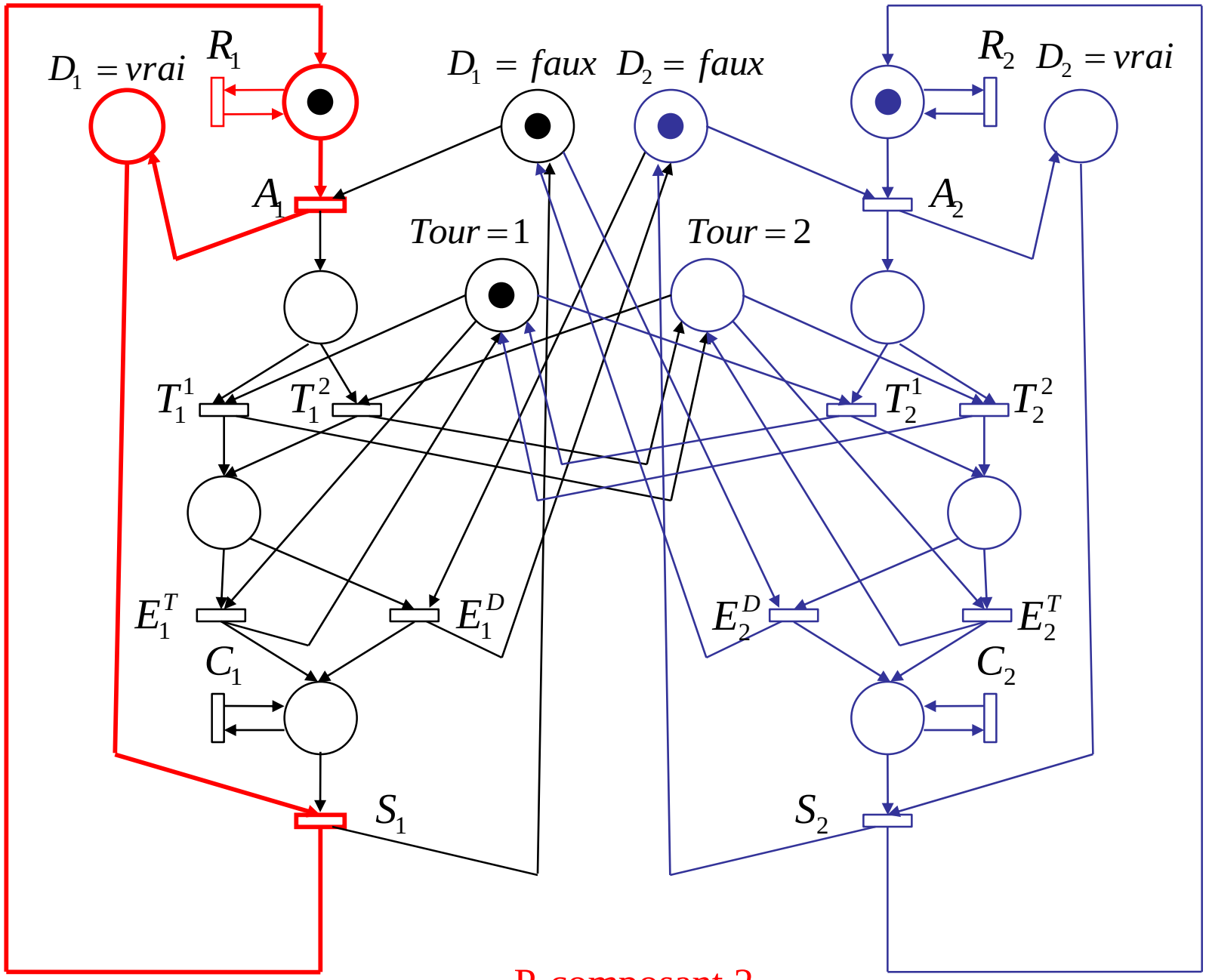
P-composant 1 (défini par les colonnes D_1^F et D_1^V)

(Places D_1^F et D_1^V et transitions incidentes à ces places)

$$\boldsymbol{v}_2 = [\,0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]^T$$

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

P-invariant 2

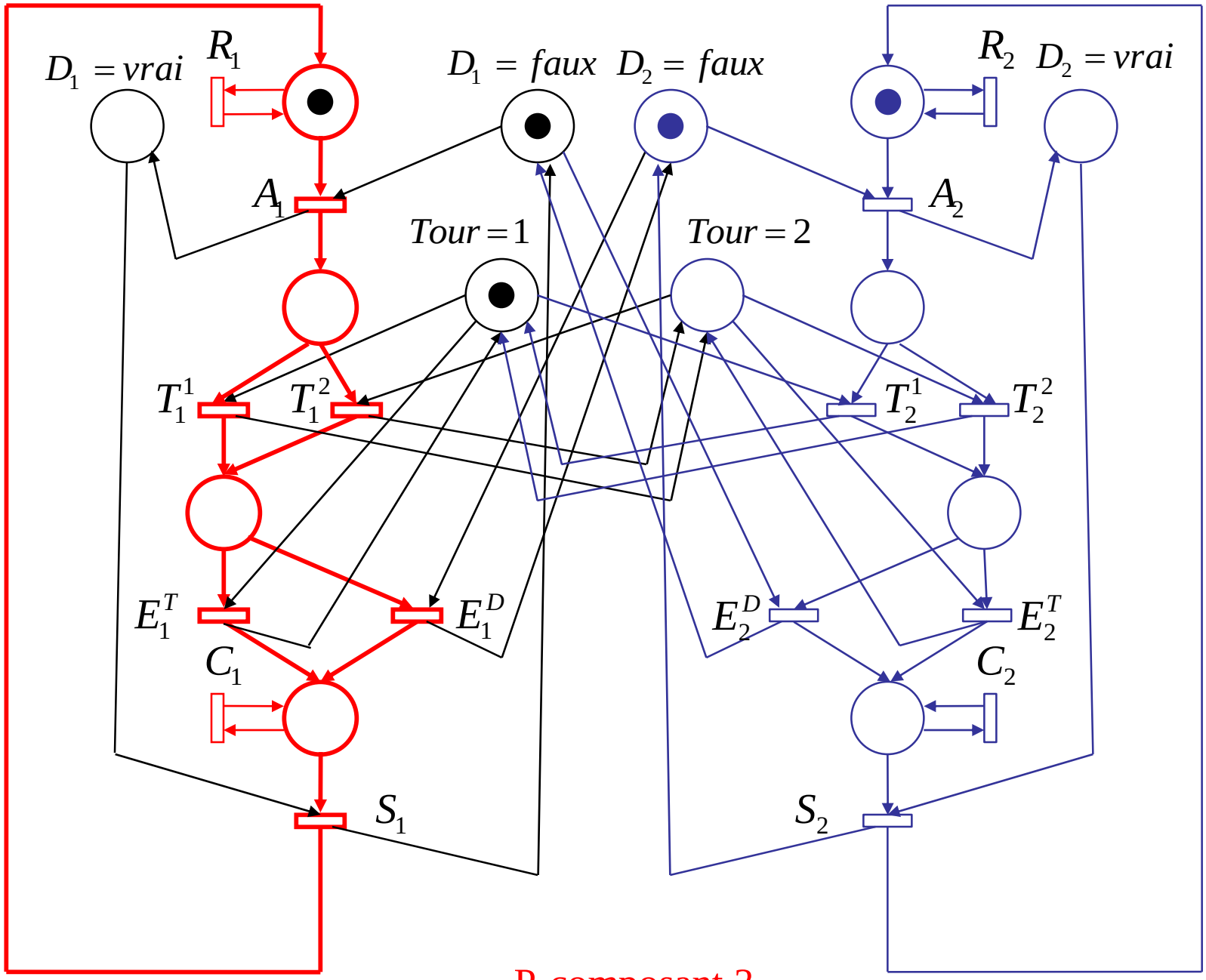


P-composant 2

$$\boldsymbol{v}_3 = [\,0,\ 0,\ 1,\ 1,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]^T$$

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

P-invariant 3



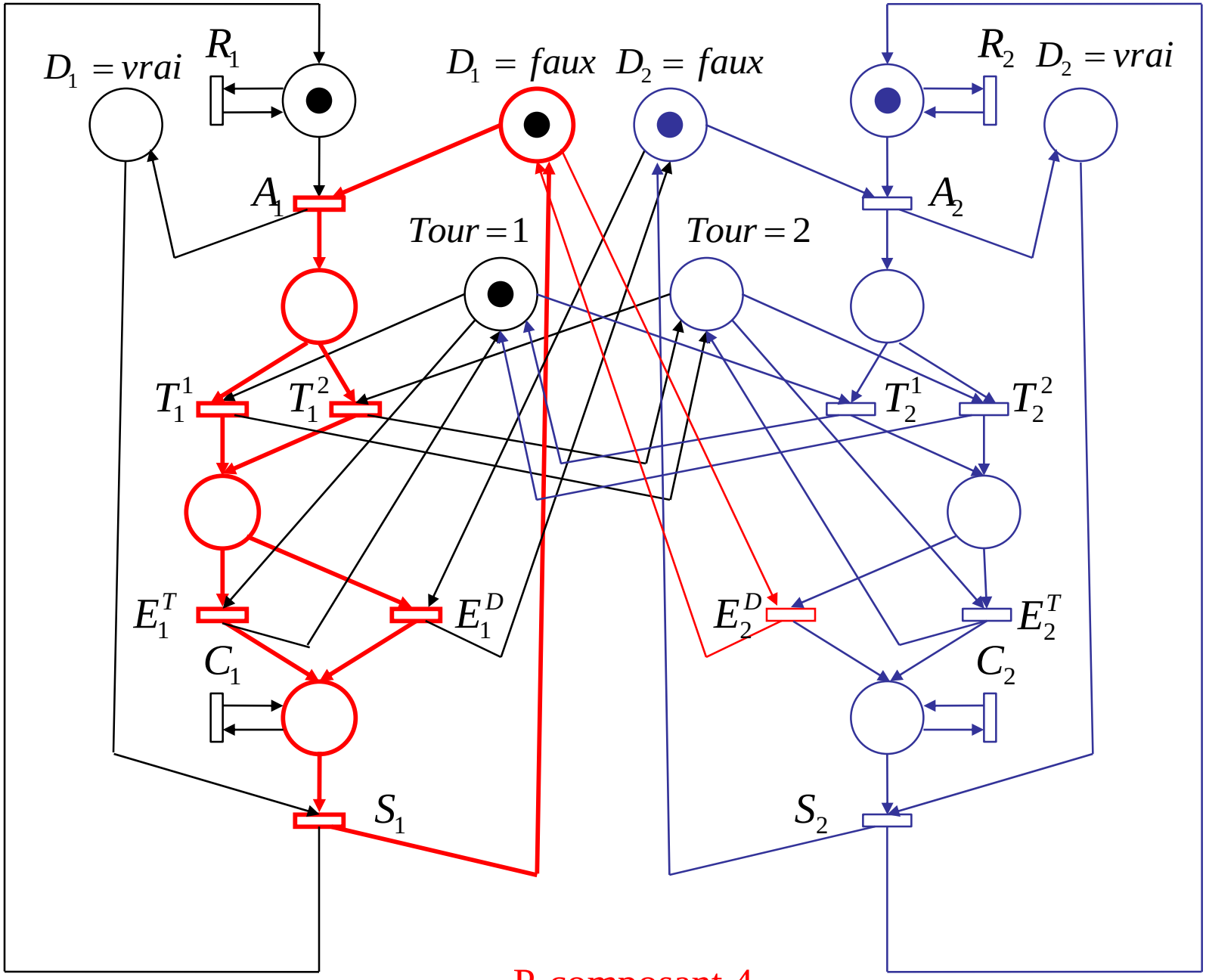
P-composant 3

$$\boldsymbol{v}_4 = [\ 1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]^T$$

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

P-invariant 4

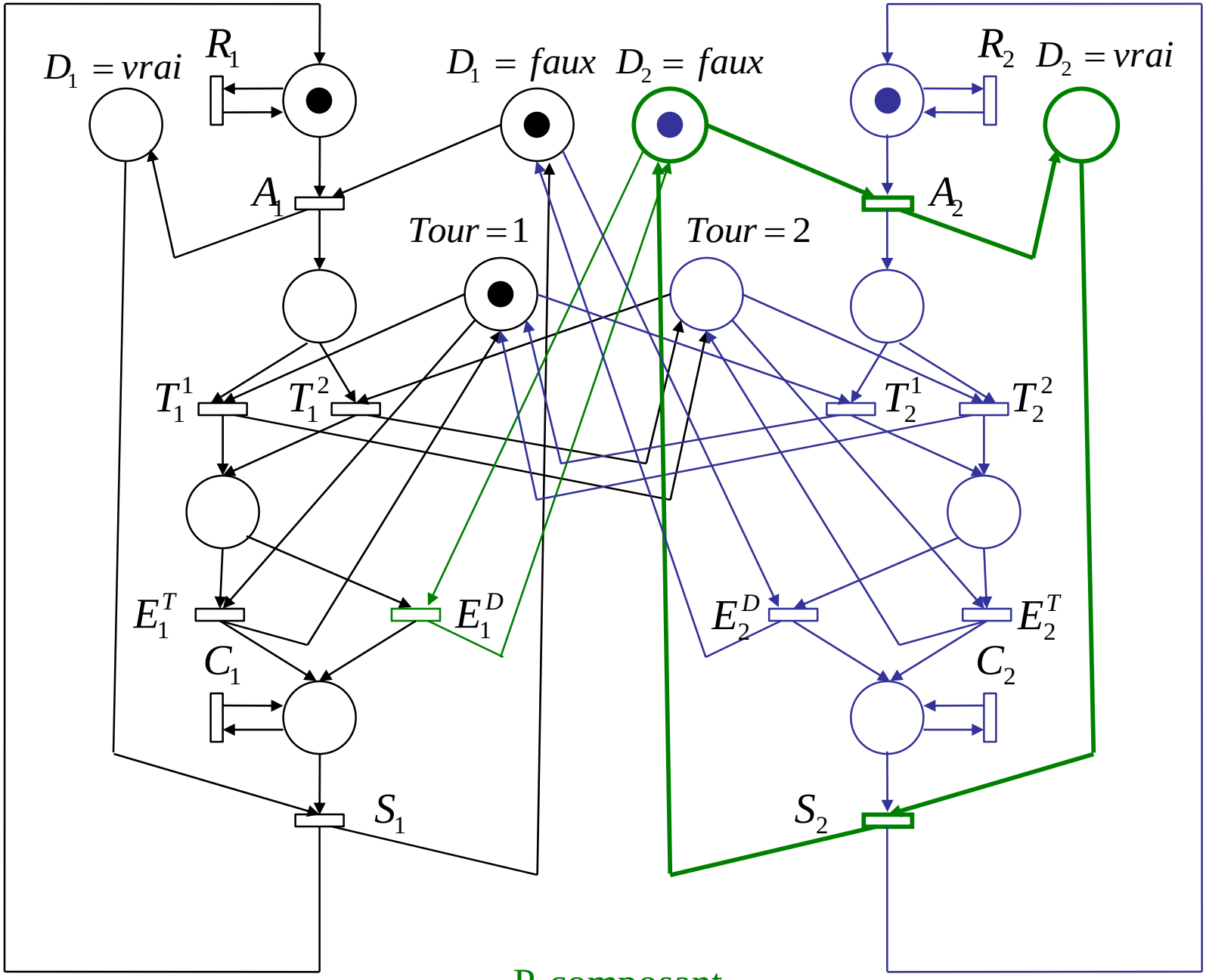
(pas indépendant : $\boldsymbol{v}_1+\boldsymbol{v}_3=\boldsymbol{v}_2+\boldsymbol{v}_4$)



P-composant 4

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

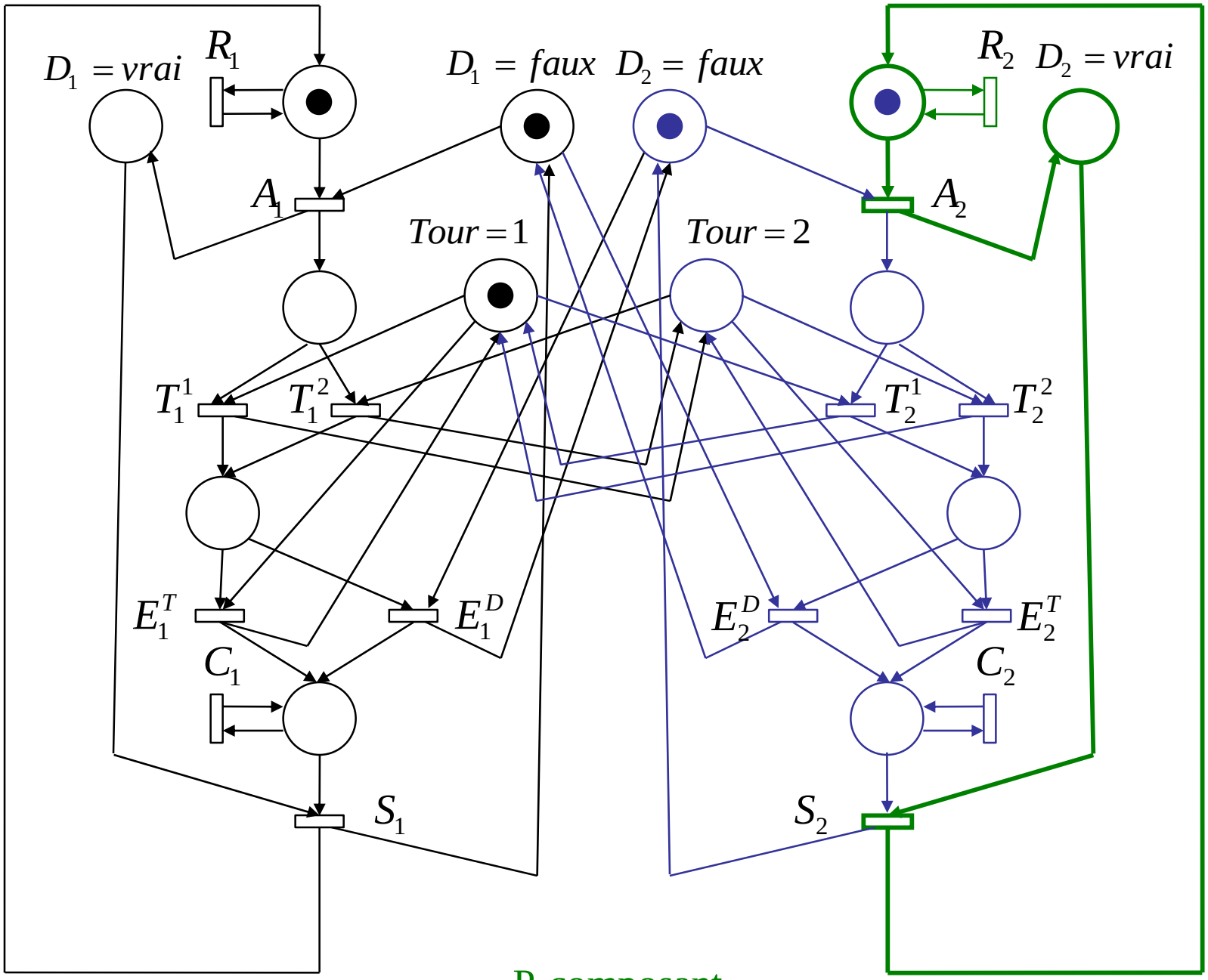
P-invariant



P-composant

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

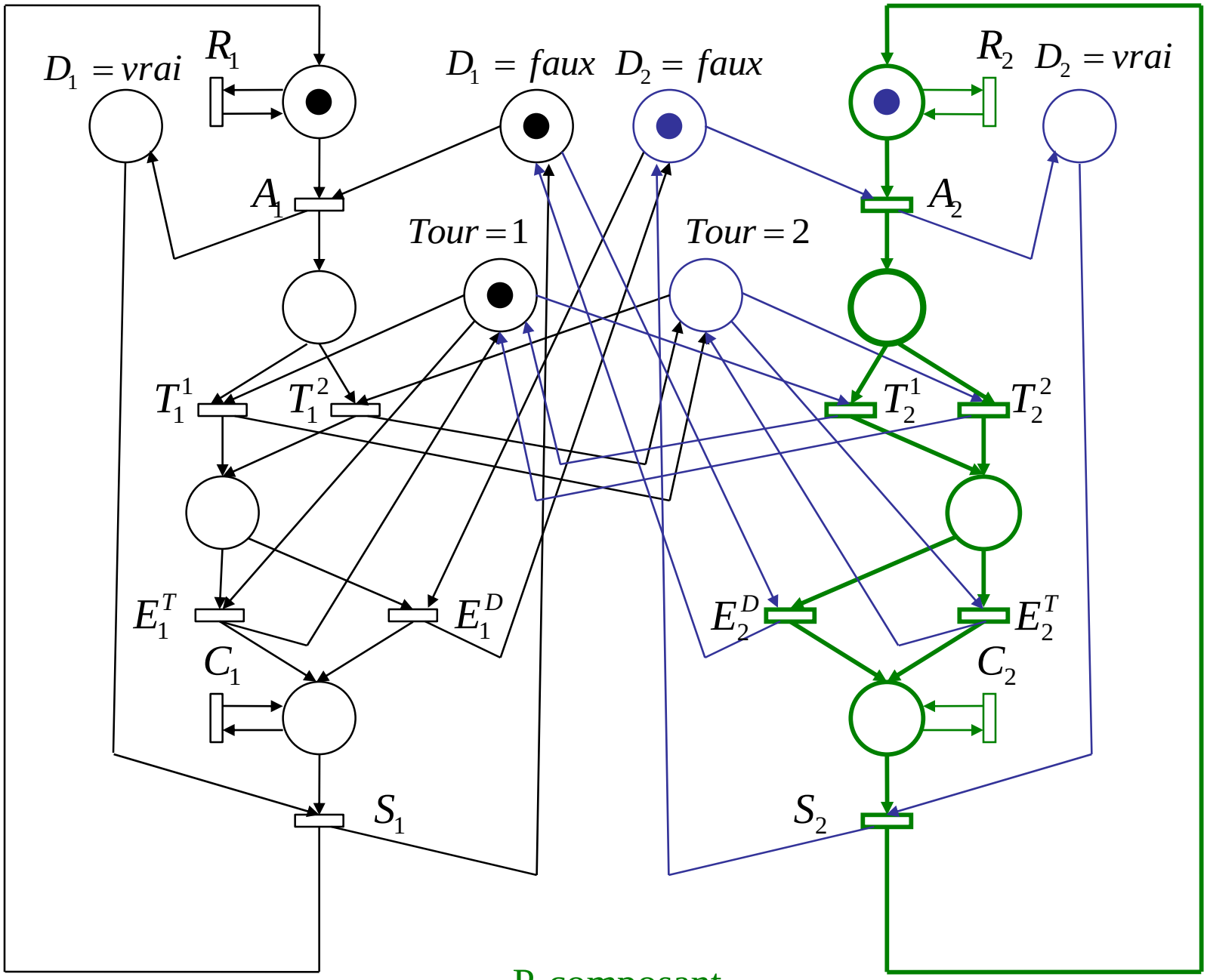
P-invariant



P-composant

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

P-invariant

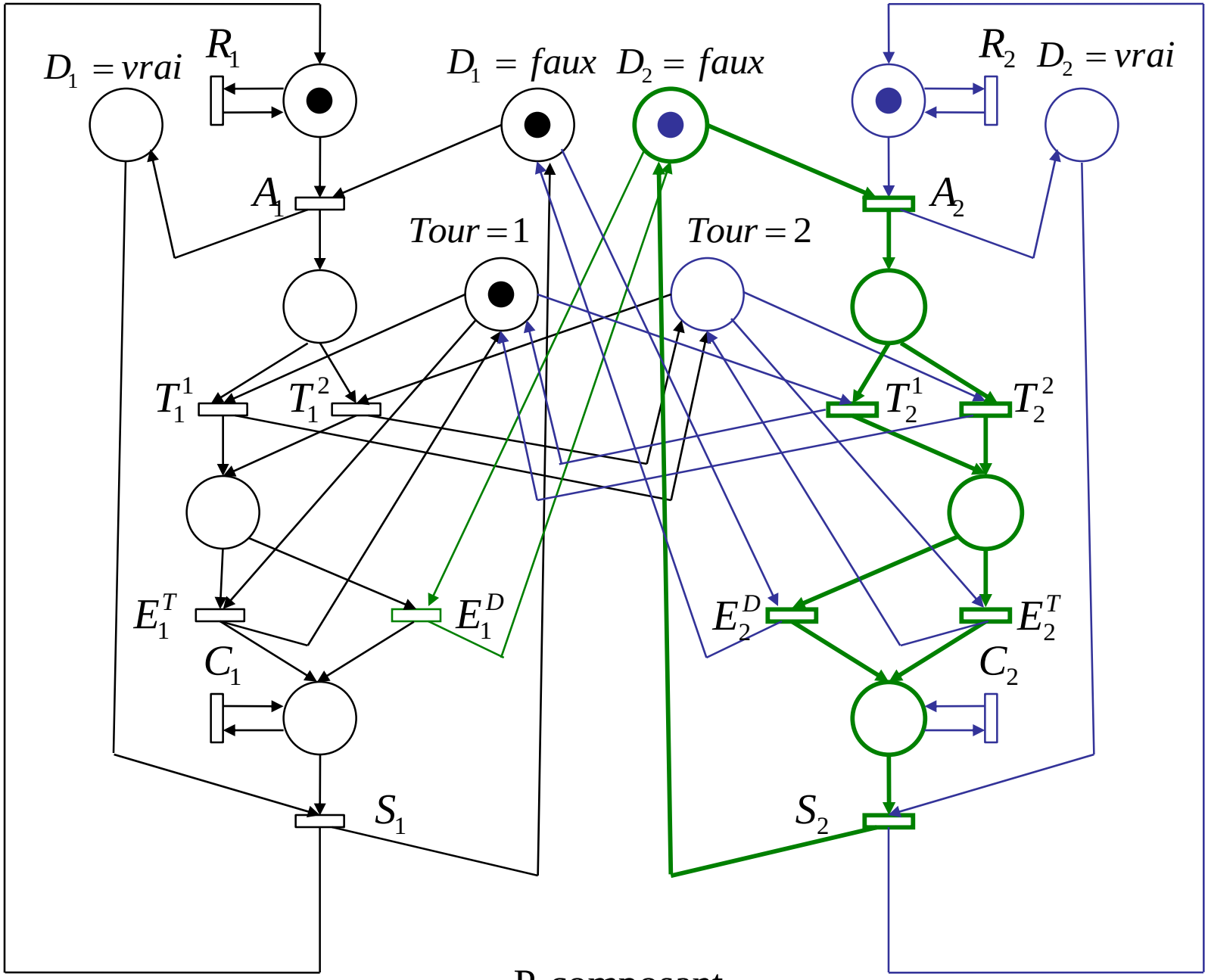


P-composant

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

P-invariant

(4 P-invariants associés à P_2 dont 3 indépendants)

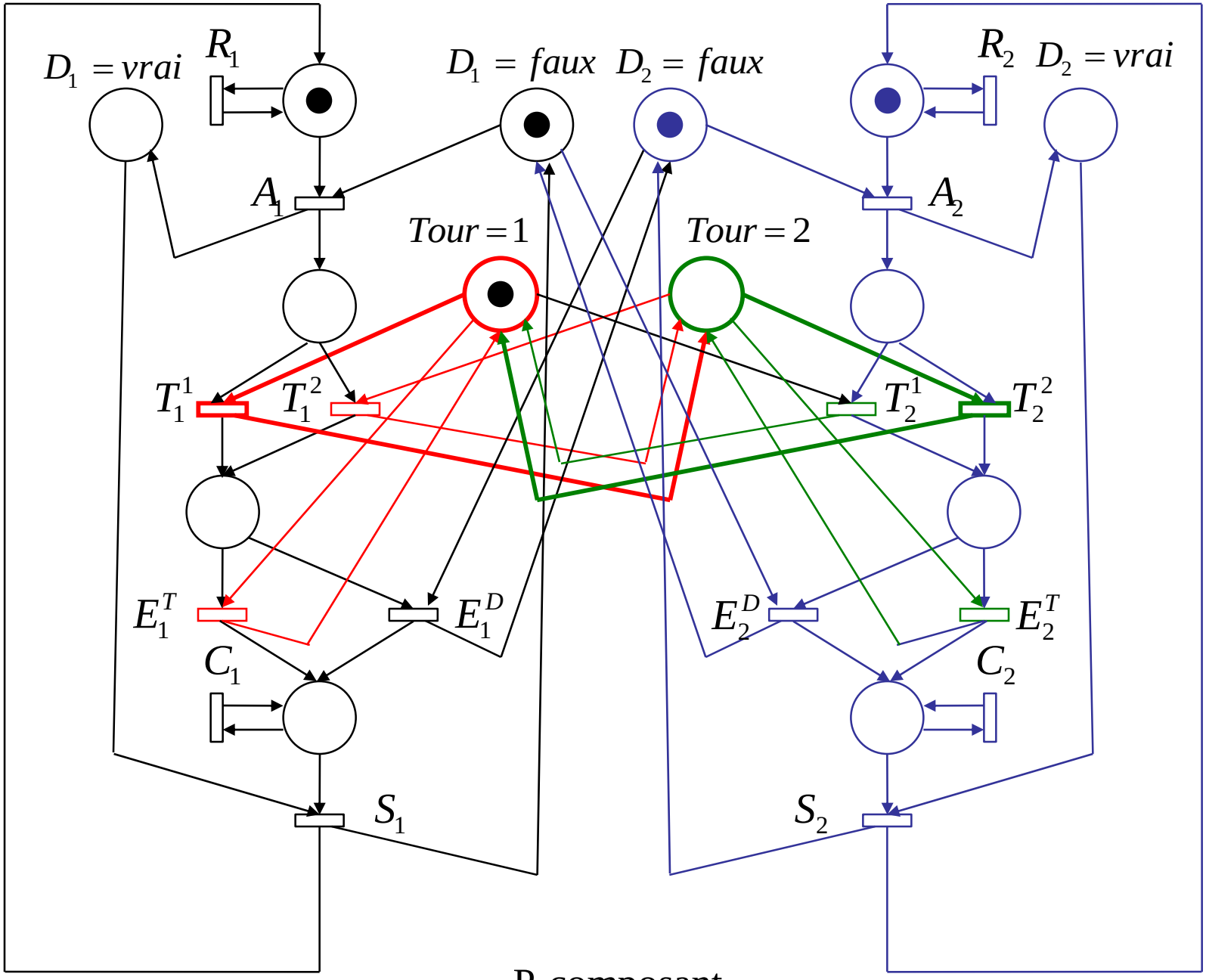


P-composant

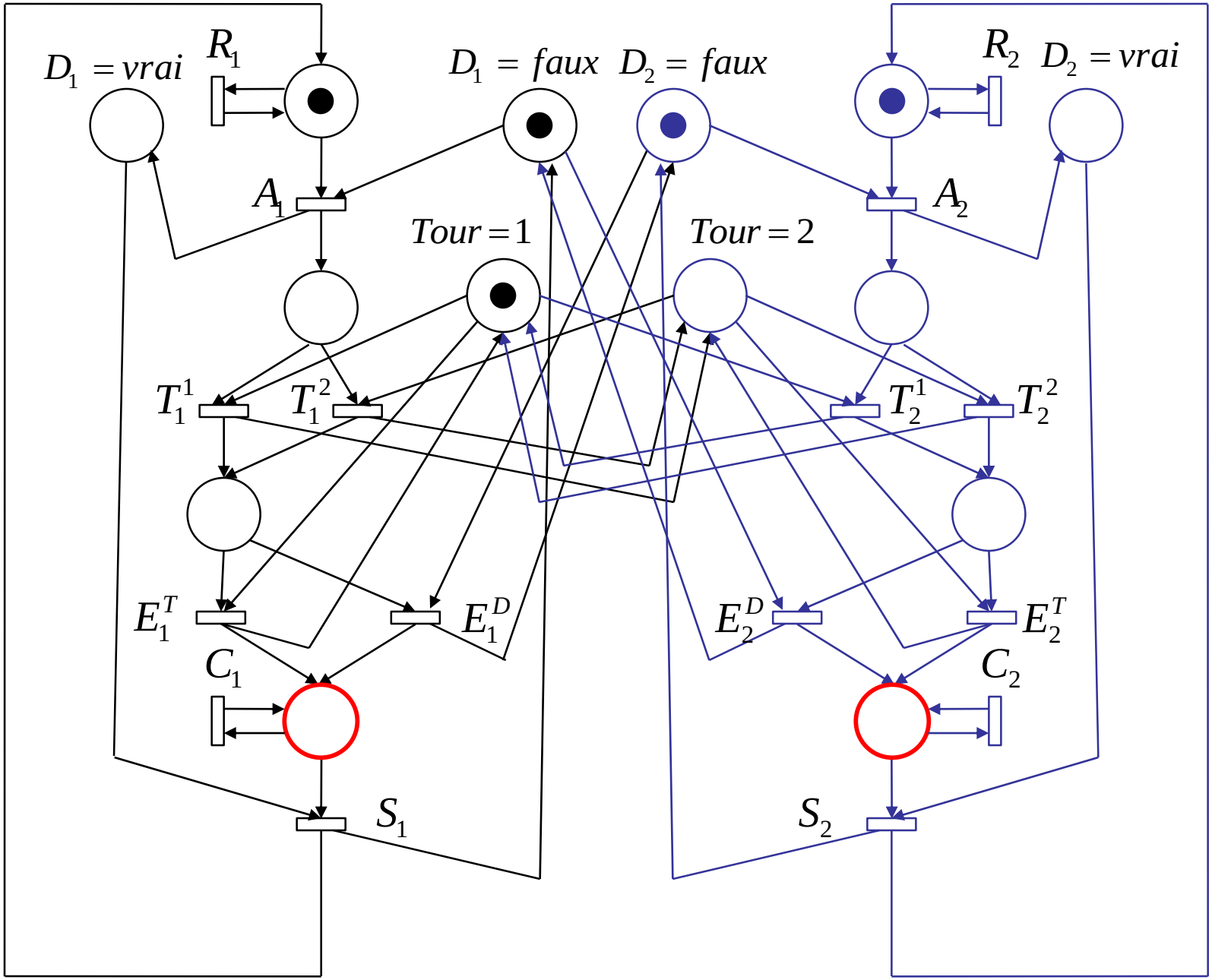
	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

P-invariant

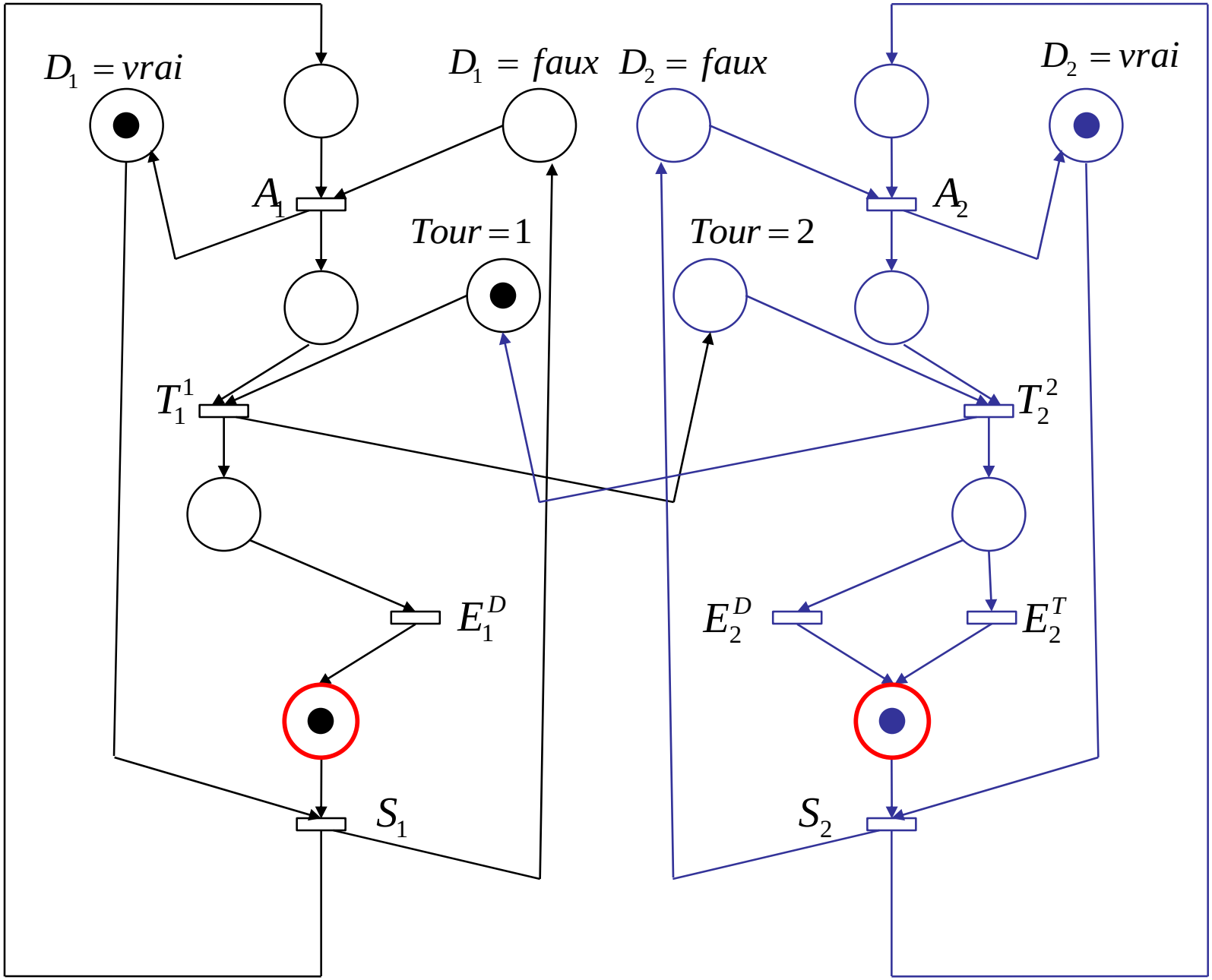
rang = 14 – (3+3+1) = 7



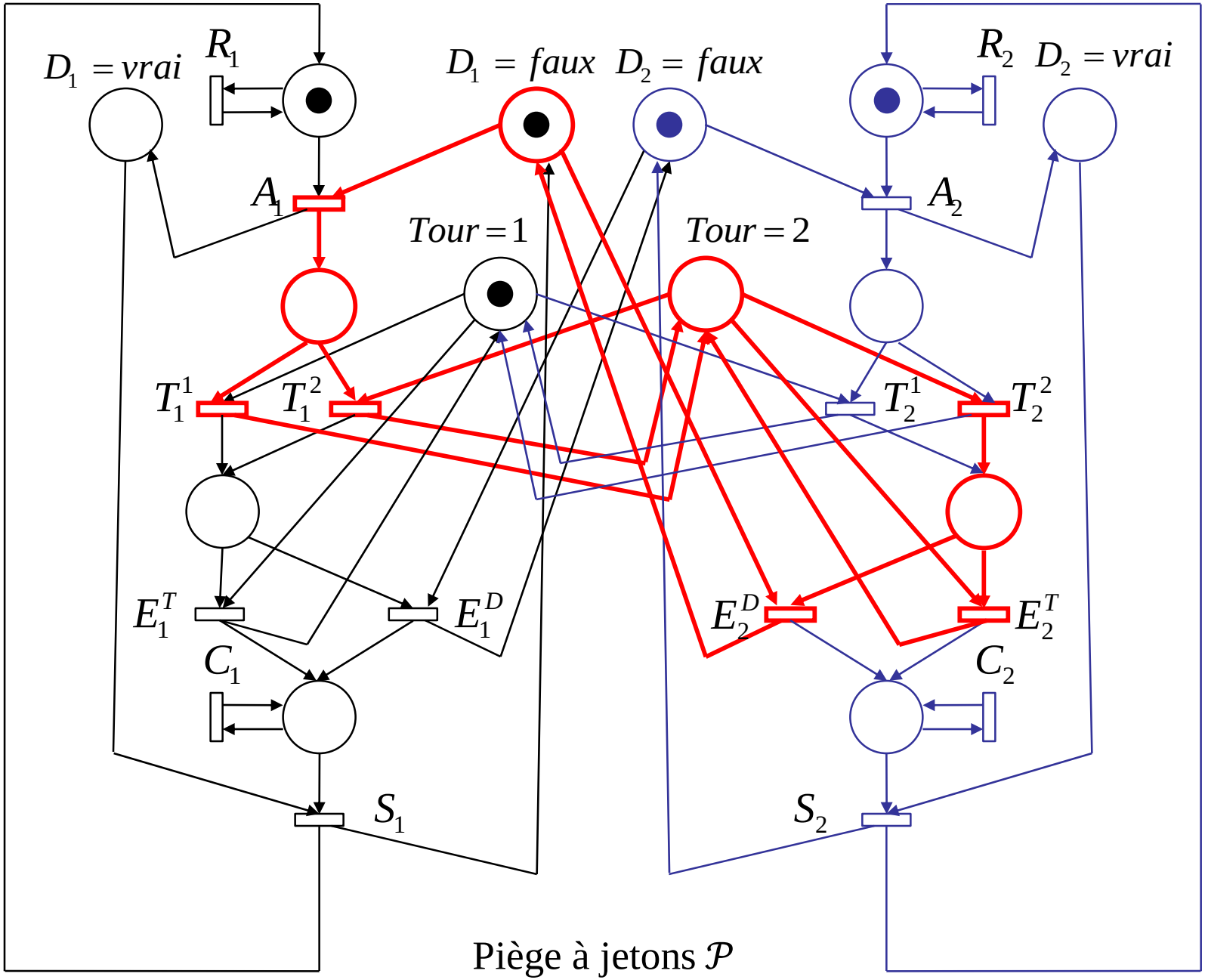
P-composant



L'exclusion mutuelle est-elle garantie ?



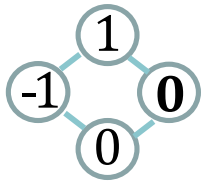
Les P-invariants ne suffisent pas.



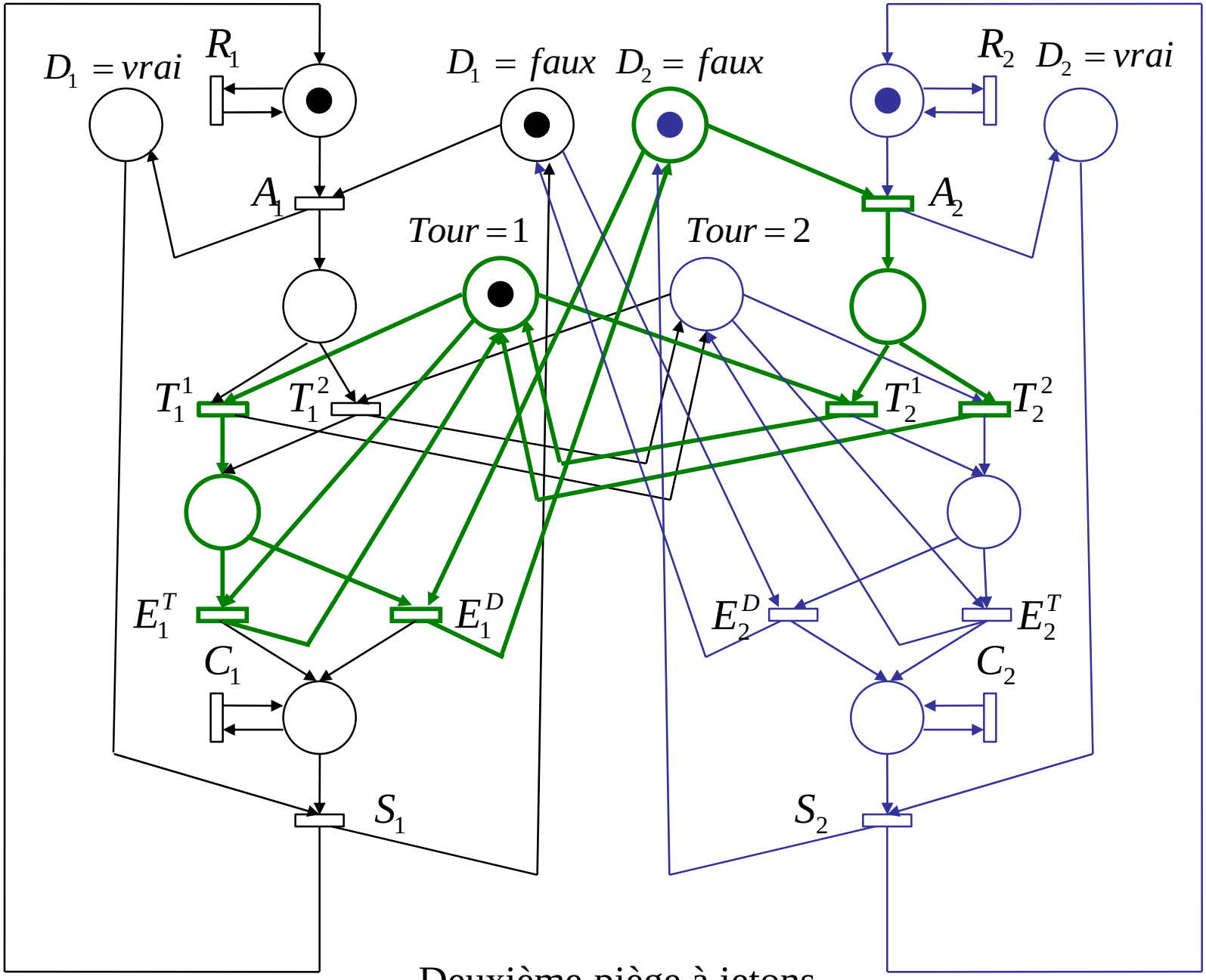
Piège à jetons $\mathcal{P}$

$$\{O(p_i) \mid p_i \in \mathcal{P}\} \subseteq \{I(p_i) \mid p_i \in \mathcal{P}\}$$

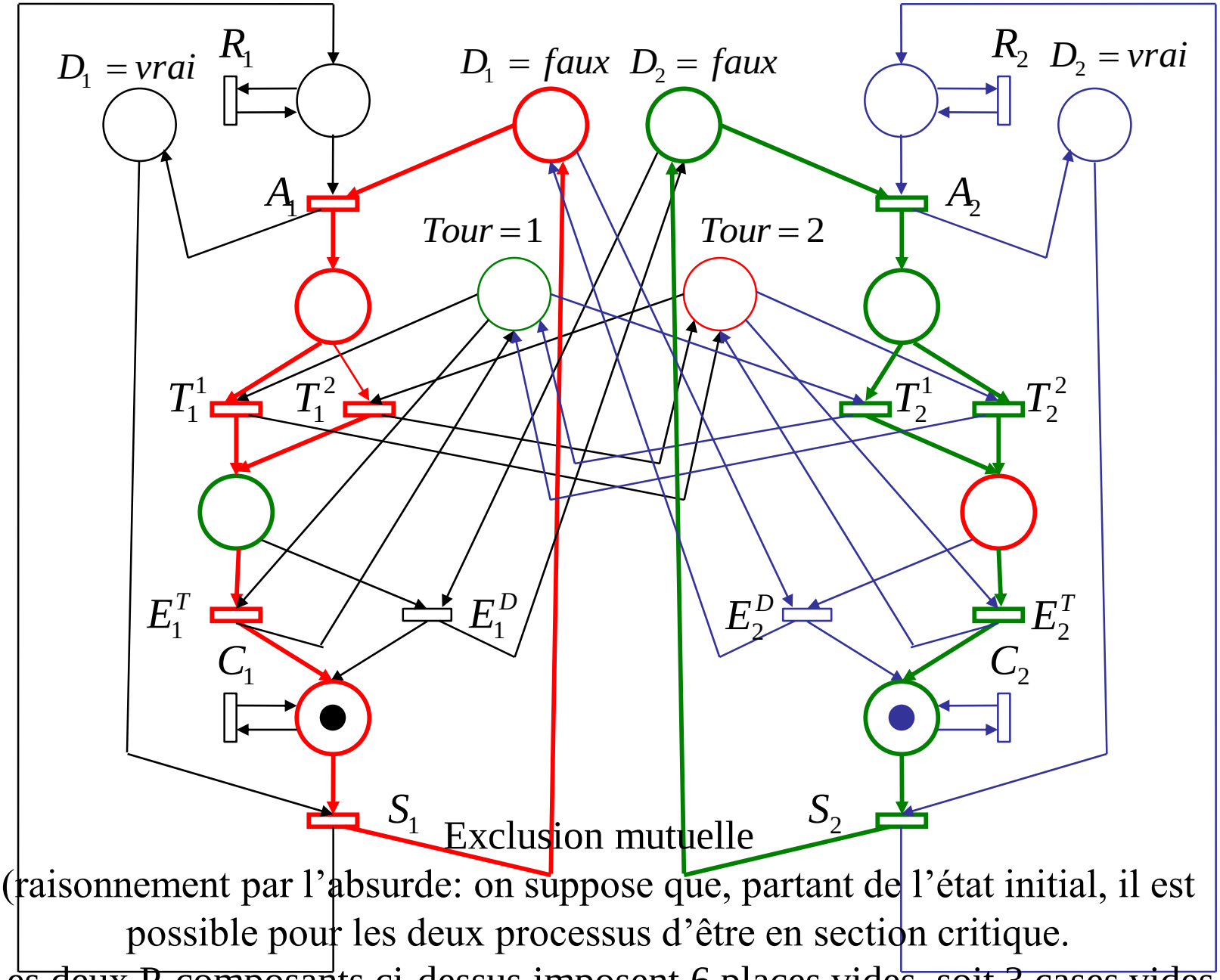
	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	**0**	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	**0**	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	**0**	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	**0**	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	**0**	0	0	0	0	−1	1
E_2^D	**0**	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1



Piège à jetons :
notion un peu plus générale que P-invariant



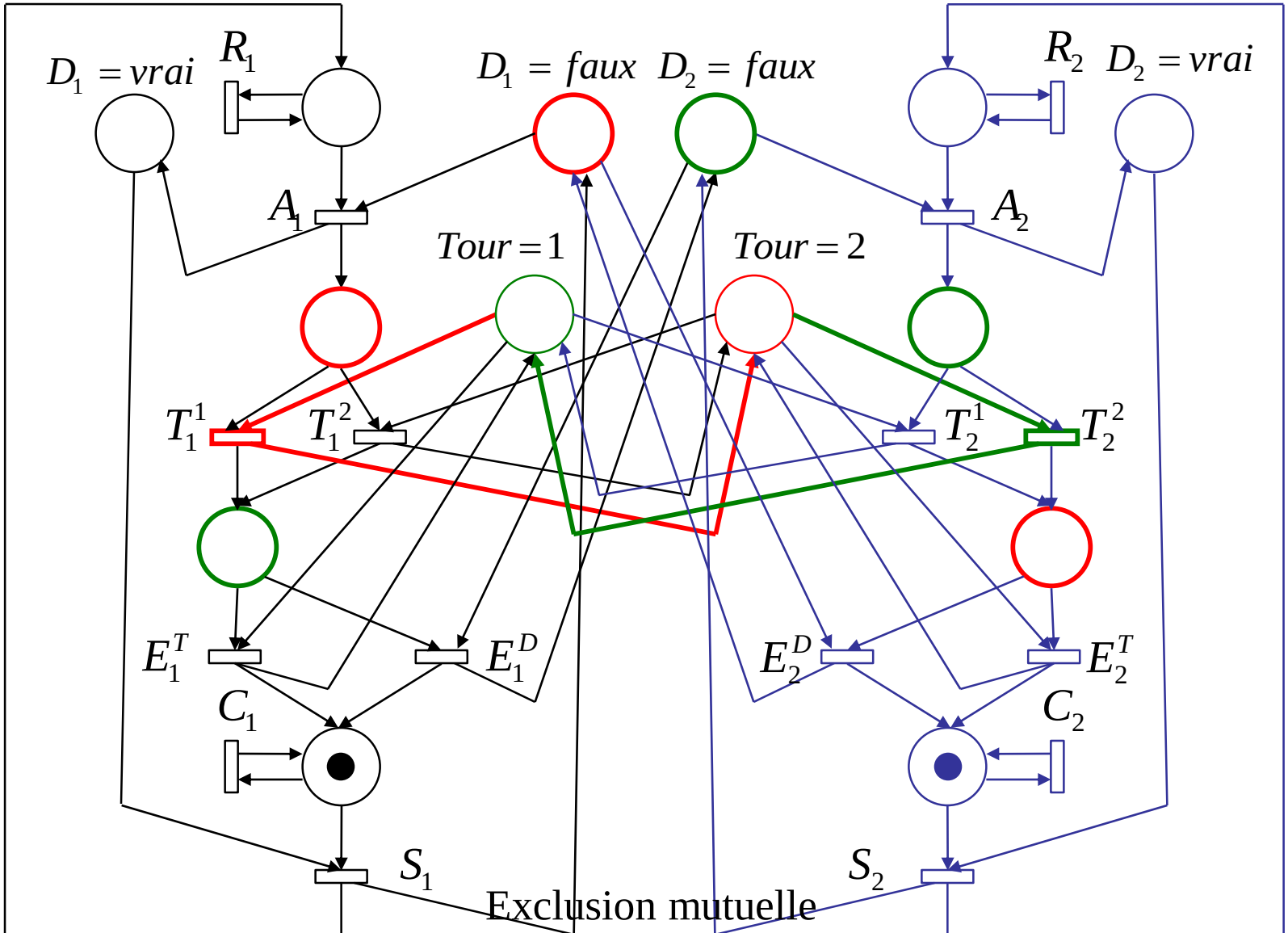
Deuxième piège à jetons



Exclusion mutuelle

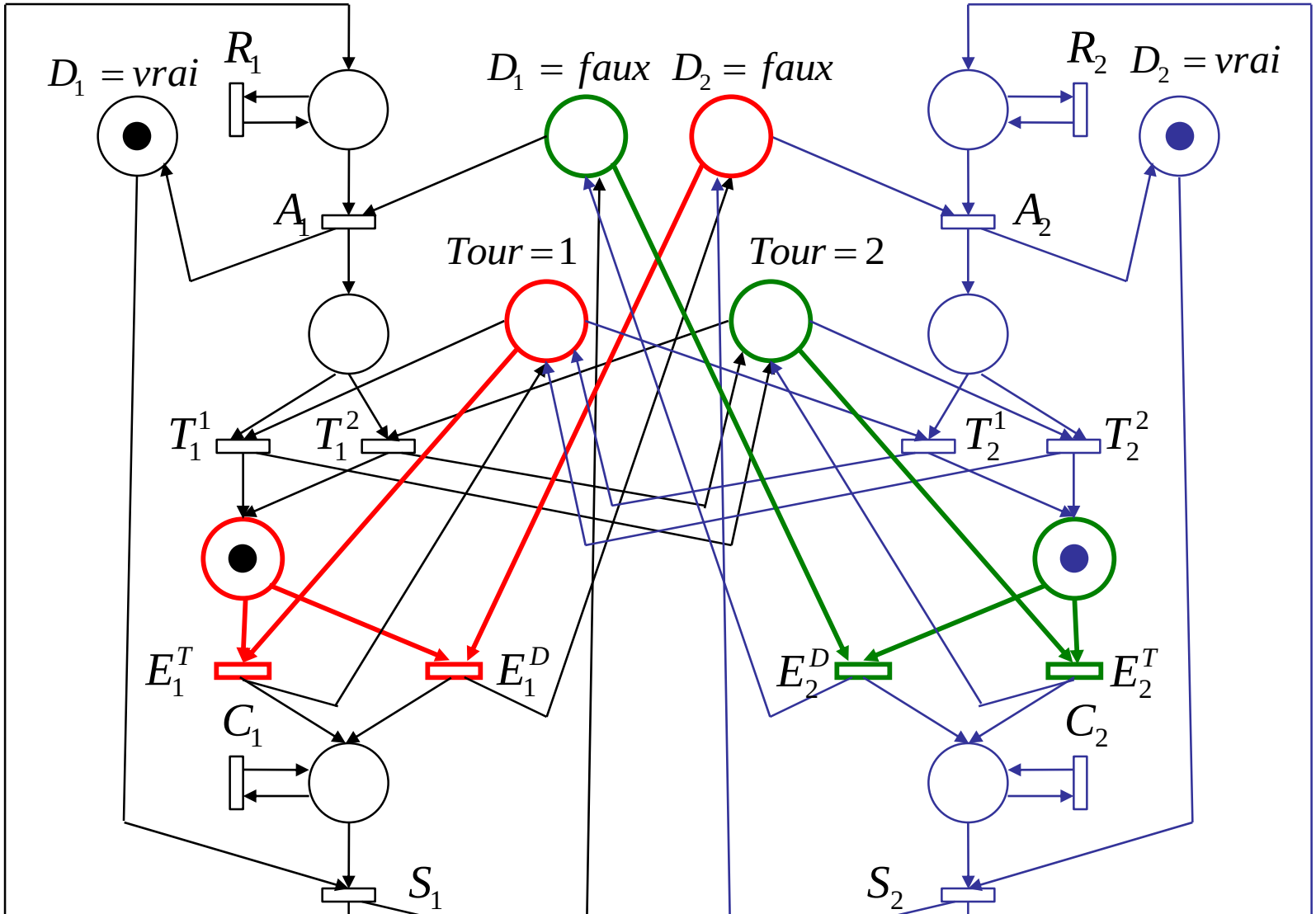
(raisonnement par l'absurde: on suppose que, partant de l'état initial, il est possible pour les deux processus d'être en section critique.

Les deux P-composants ci-dessus imposent 6 places vides, soit 3 cases vides (rouges et vertes) pour chacun des pièges à jetons)



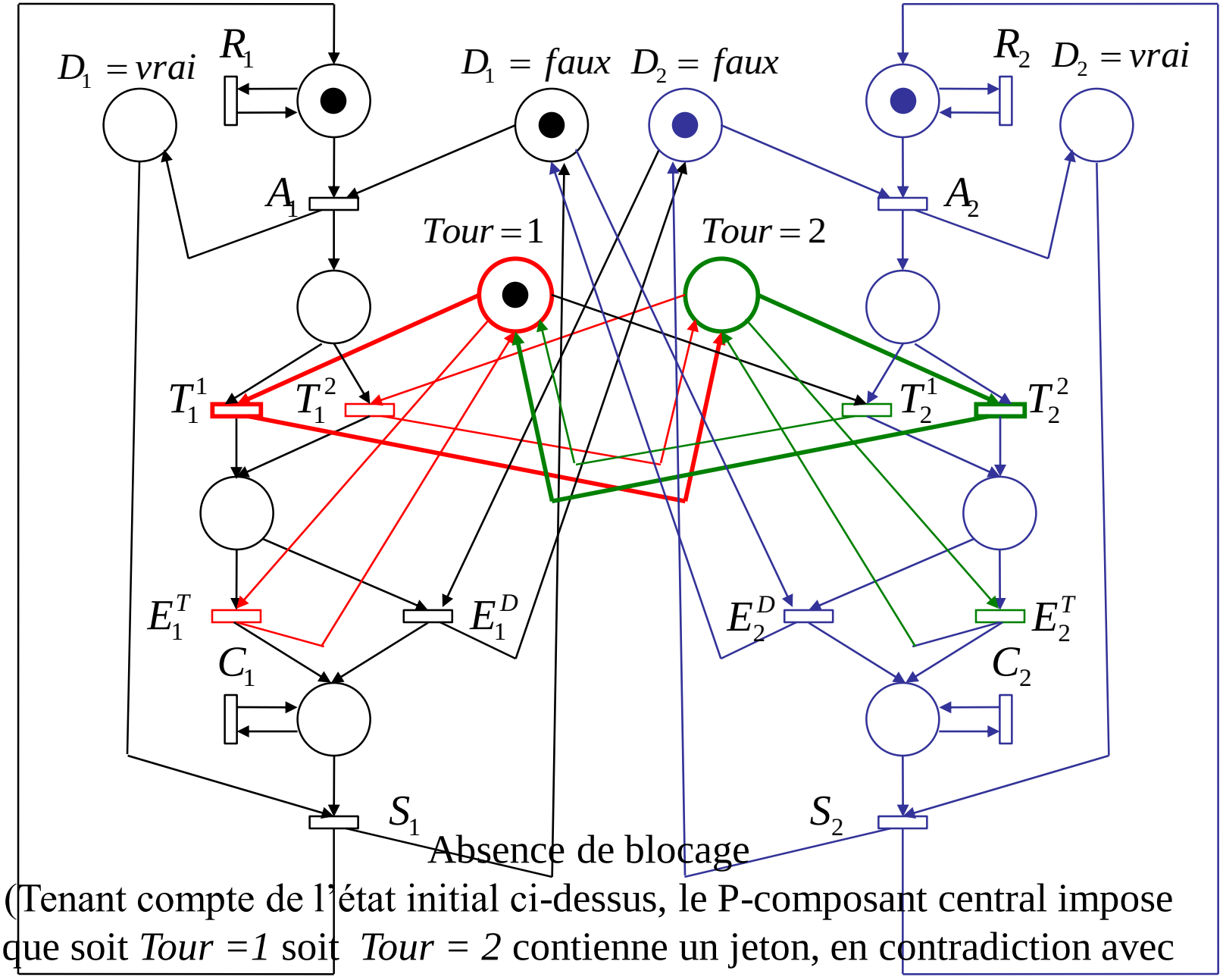
Exclusion mutuelle

(Les pièges à jetons imposent alors que les places *Tour =1* et *Tour = 2* contiennent chacune au moins un jeton, en contradiction avec le fait que le P-composant central ci-dessus contient exactement un jeton)



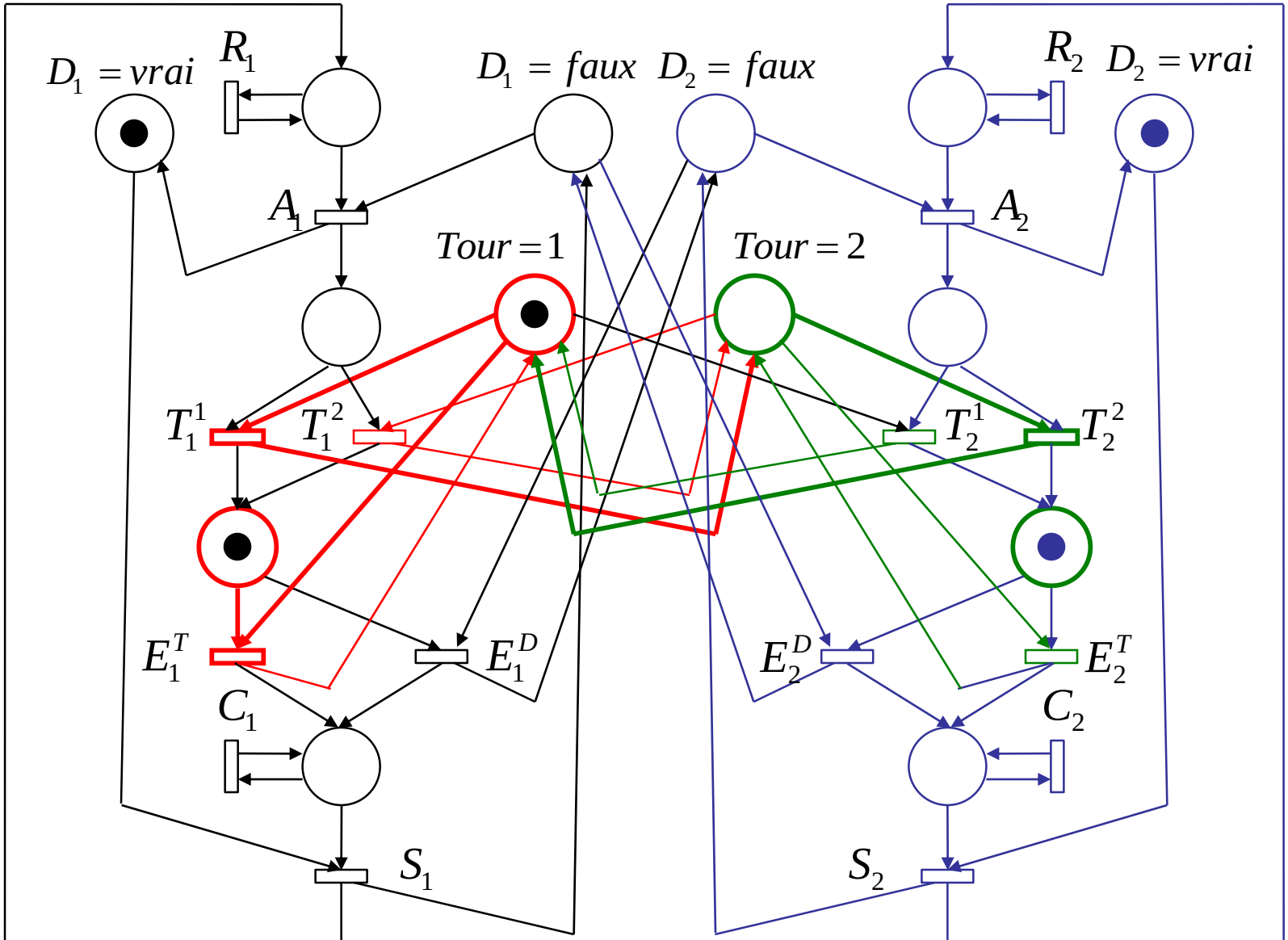
Absence de blocage

(raisonnement par l'absurde : le blocage ne peut se produire qu'au niveau des instructions conditionnelles. Si c'est le cas, cela impose que les quatre places centrales soient vides)



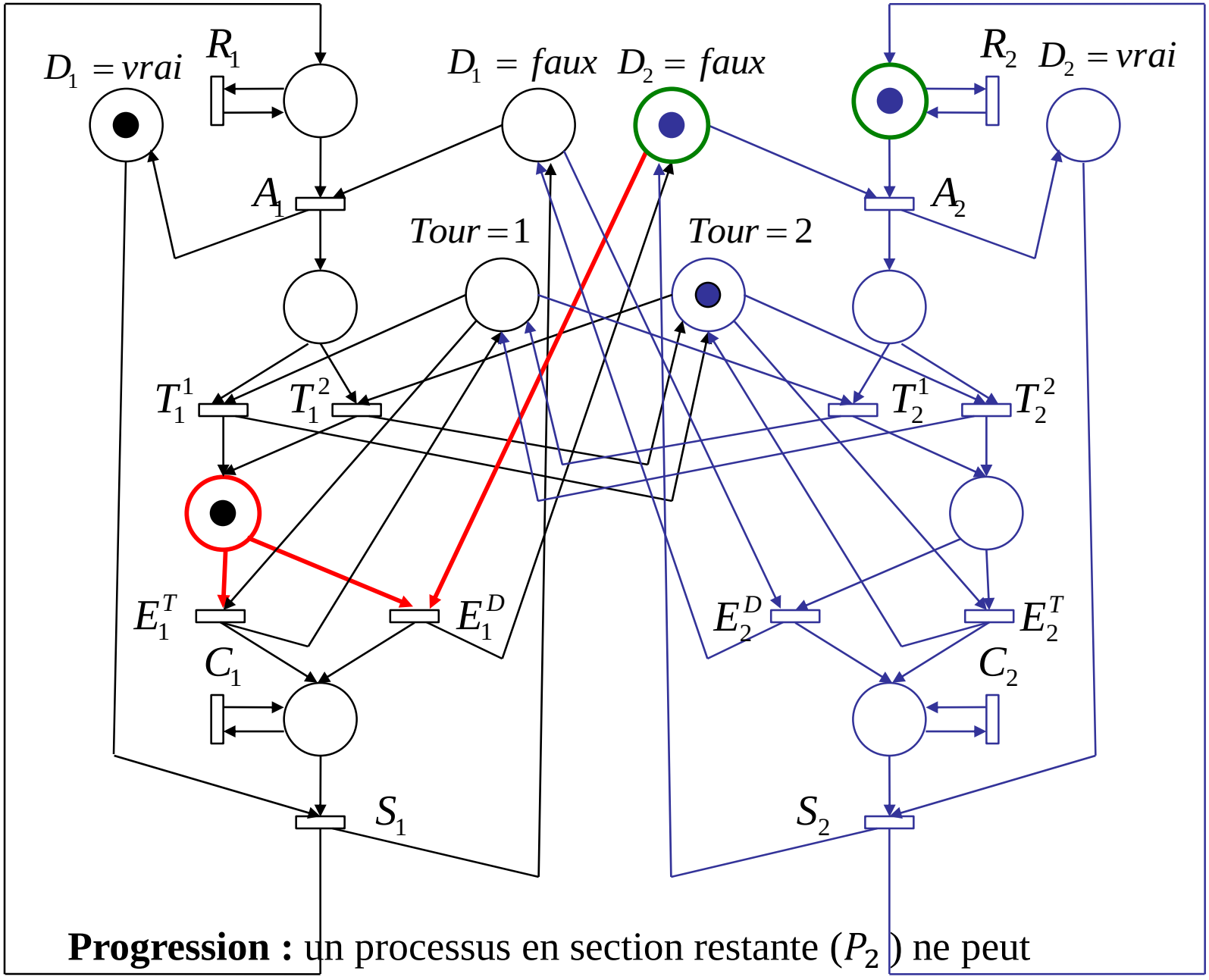
Absence de blocage

(Tenant compte de l'état initial ci-dessus, le P-composant central impose que soit *Tour =1* soit *Tour = 2* contienne un jeton, en contradiction avec la conséquence exposée à la page précédente de l'hypothèse de blocage)



Absence de blocage

(Si *Tour =1* ou *Tour = 2* contient un jeton, il ne peut effectivement pas y avoir de blocage)



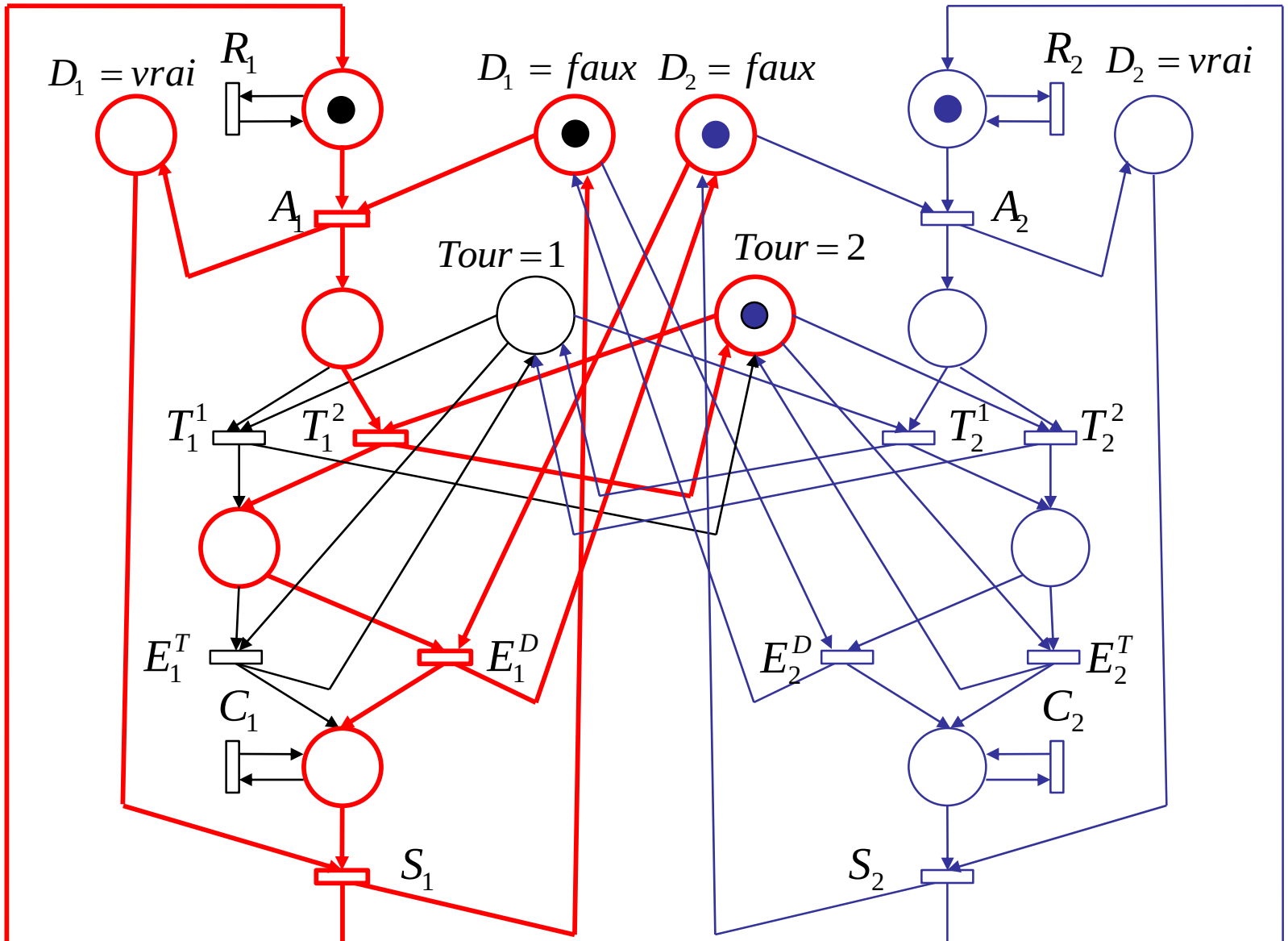
Progression : un processus en section restante (P_2) ne peut empêcher l'autre processus (P_1) de rentrer en section critique.

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

T-invariant associé au processus 1

$$[1,0,1,0,1,1,0,0,0,0,0,0]$$

$$\boldsymbol{uB} = 0 \Rightarrow \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{uB} = \boldsymbol{x}$$



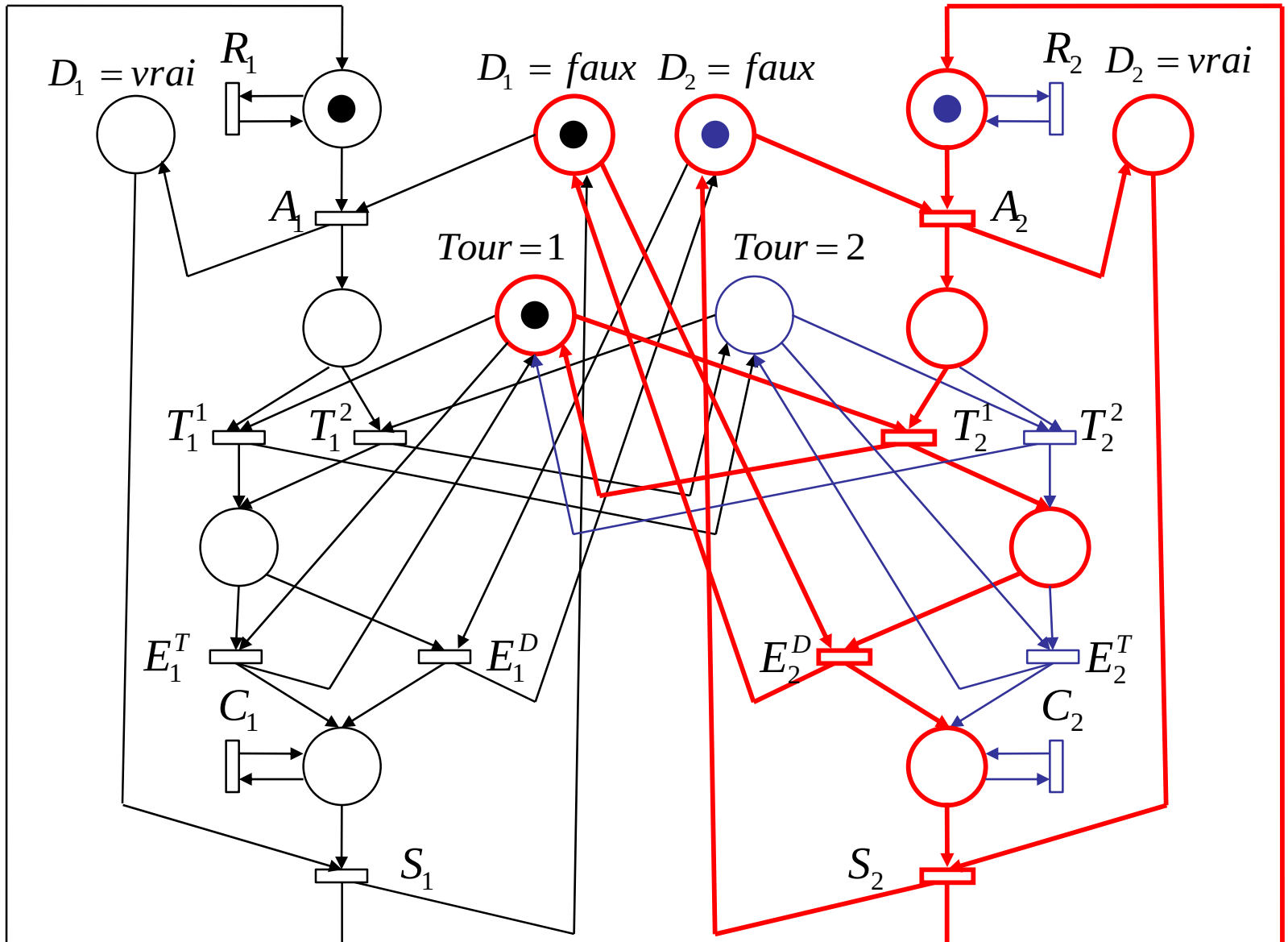
Progression : un processus en section restante (P_2) ne peut empêcher l'autre processus (P_1) de rentrer en section critique.

Le T-composant associé à P_1 montre que la progression est possible**.**

	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

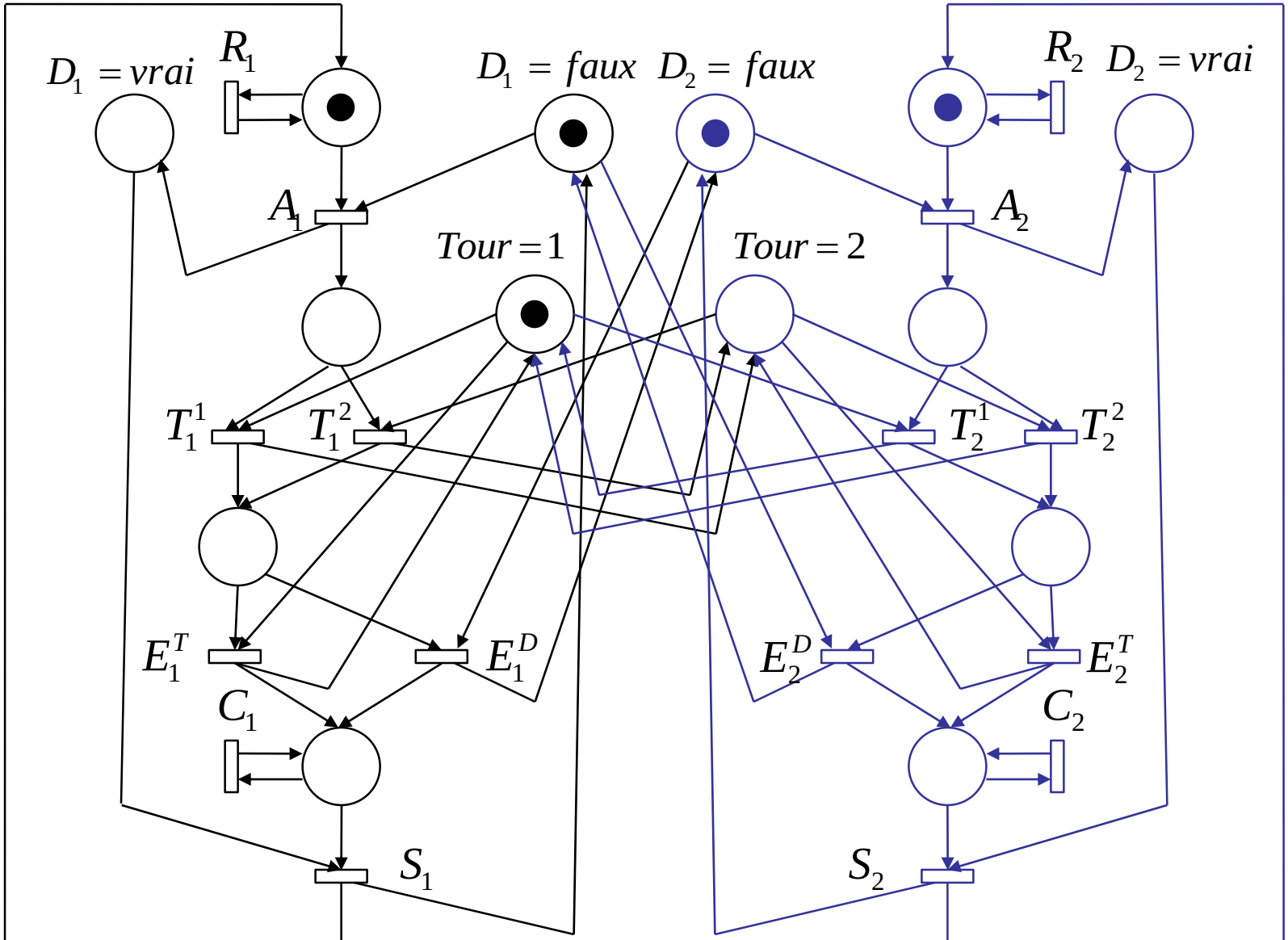
T-invariant associé au processus 2

$[0,0,0,0,0,0,1,1,0,0,1,1]$



Progression : un processus en section restante (P_1) ne peut empêcher l'autre processus (P_2) de rentrer en section critique.

Le T-composant associé à P_2 montre que la progression est possible

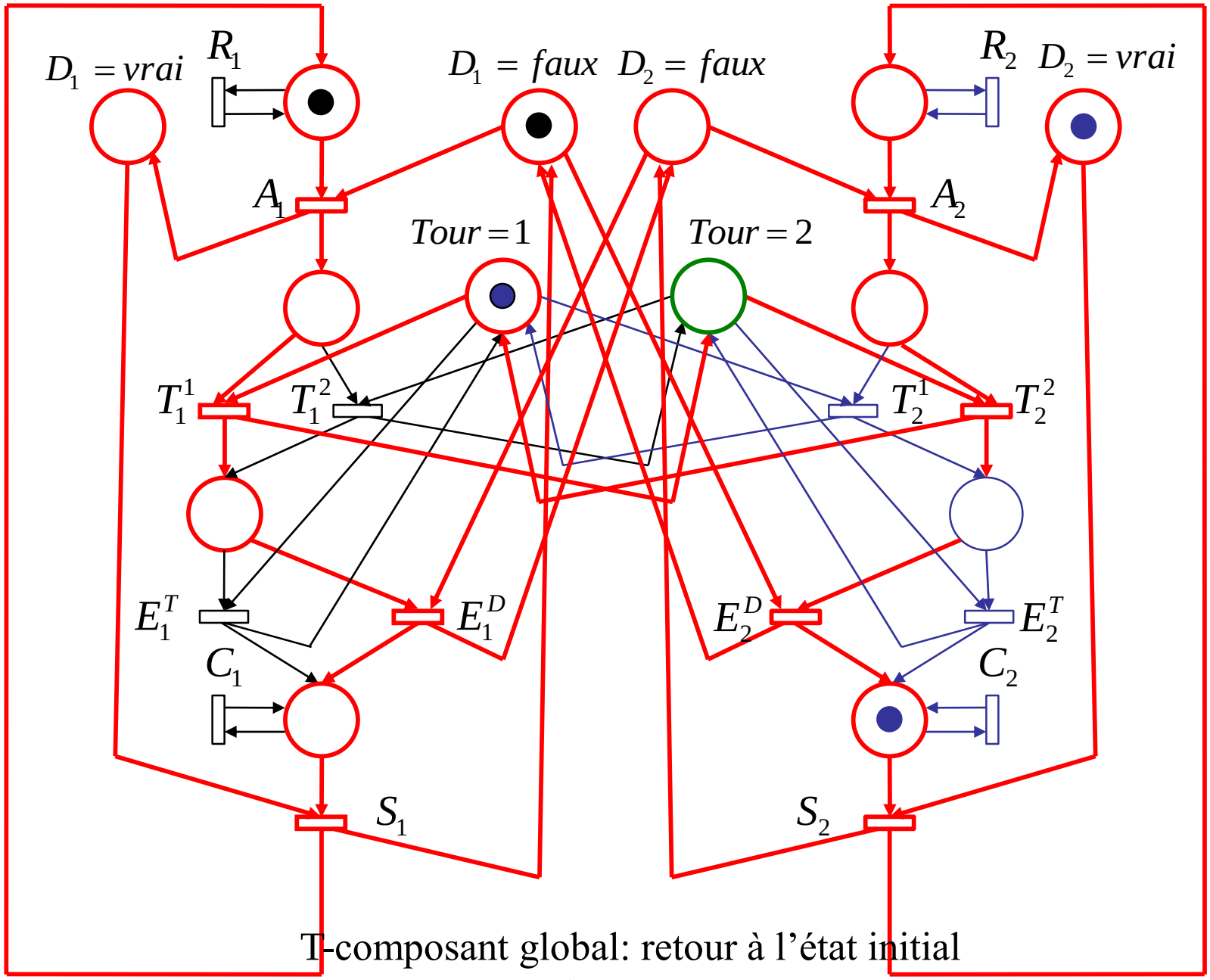


Attente bornée : lorsqu'un processus est en attente de sa section critique, il existe une borne supérieure au nombre de fois où l'autre processus exécute sa section critique.

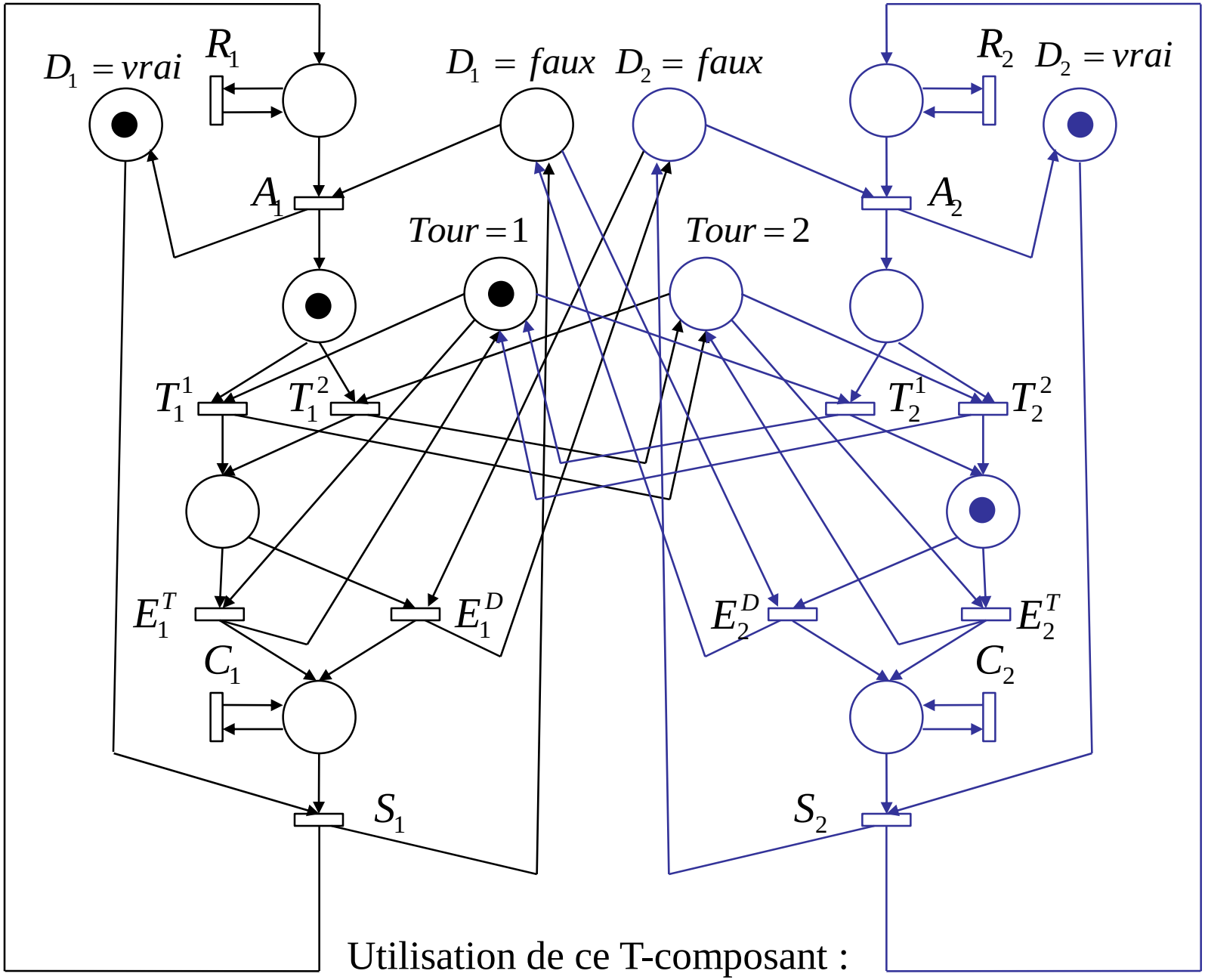
	D_1^F	D_1^V	p_1	p_2	p_3	p_4	T_1	T_2	D_2^F	D_2^V	q_1	q_2	q_3	q_4
A_1	−1	1	−1	1	0	0	0	0	0	0	0	0	0	0
T_1^1	0	0	0	−1	1	0	−1	1	0	0	0	0	0	0
T_1^2	0	0	0	−1	1	0	0	0	0	0	0	0	0	0
E_1^T	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
E_1^D	0	0	0	0	−1	1	0	0	0	0	0	0	0	0
S_1	1	−1	1	0	0	−1	0	0	0	0	0	0	0	0
A_2	0	0	0	0	0	0	0	0	−1	1	−1	1	0	0
T_2^1	0	0	0	0	0	0	0	0	0	0	0	−1	1	0
T_2^2	0	0	0	0	0	0	1	−1	0	0	0	−1	1	0
E_2^T	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
E_2^D	0	0	0	0	0	0	0	0	0	0	0	0	−1	1
S_2	0	0	0	0	0	0	0	0	1	−1	1	0	0	−1

T-invariant global

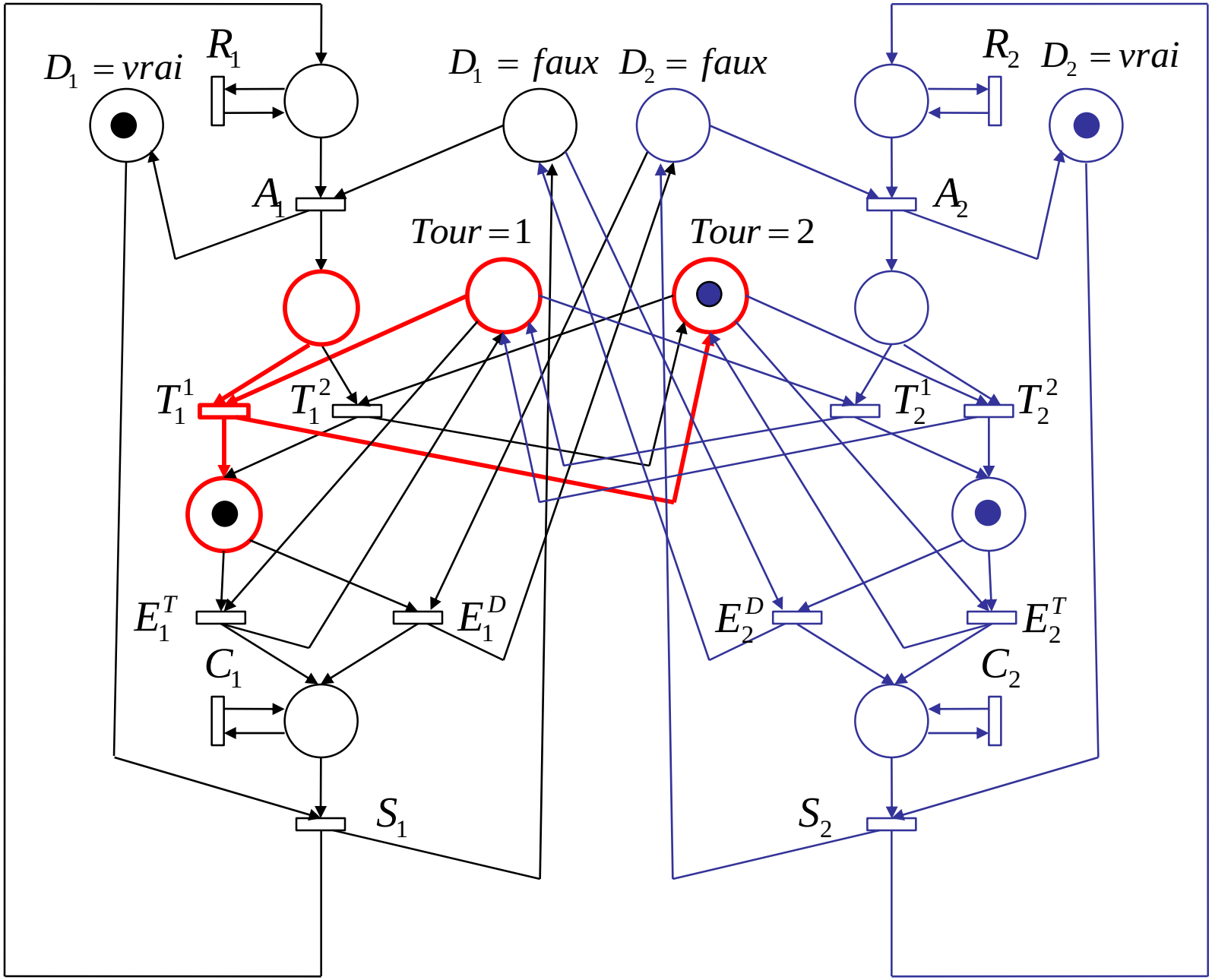
$$\boldsymbol{u} = [1,1,0,0,1,1,1,0,1,0,1,1]$$



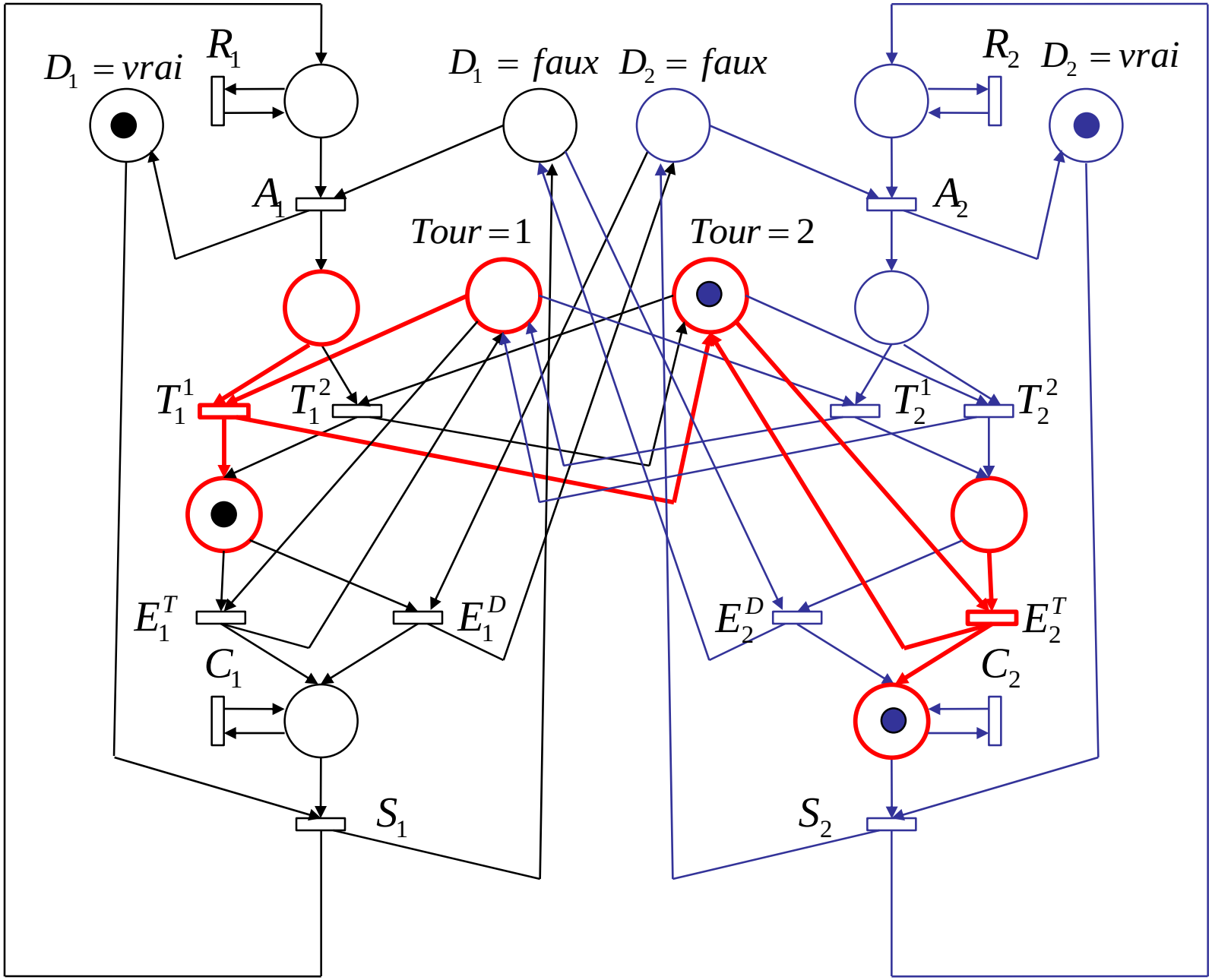
T-composant global: retour à l'état initial
(les transitions franchies sont en rouge)



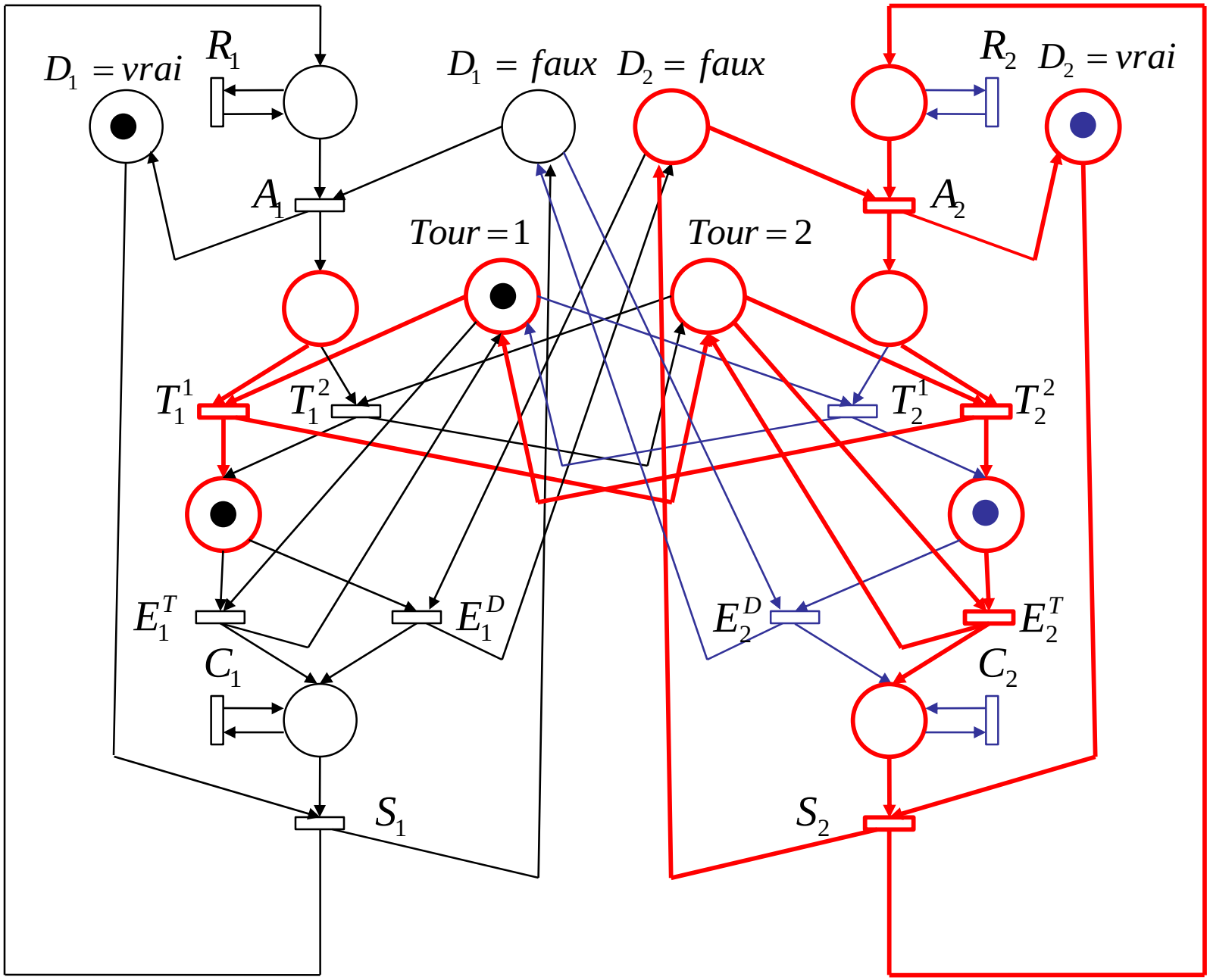
Utilisation de ce T-composant :
Processus 2 en attente de sa section critique



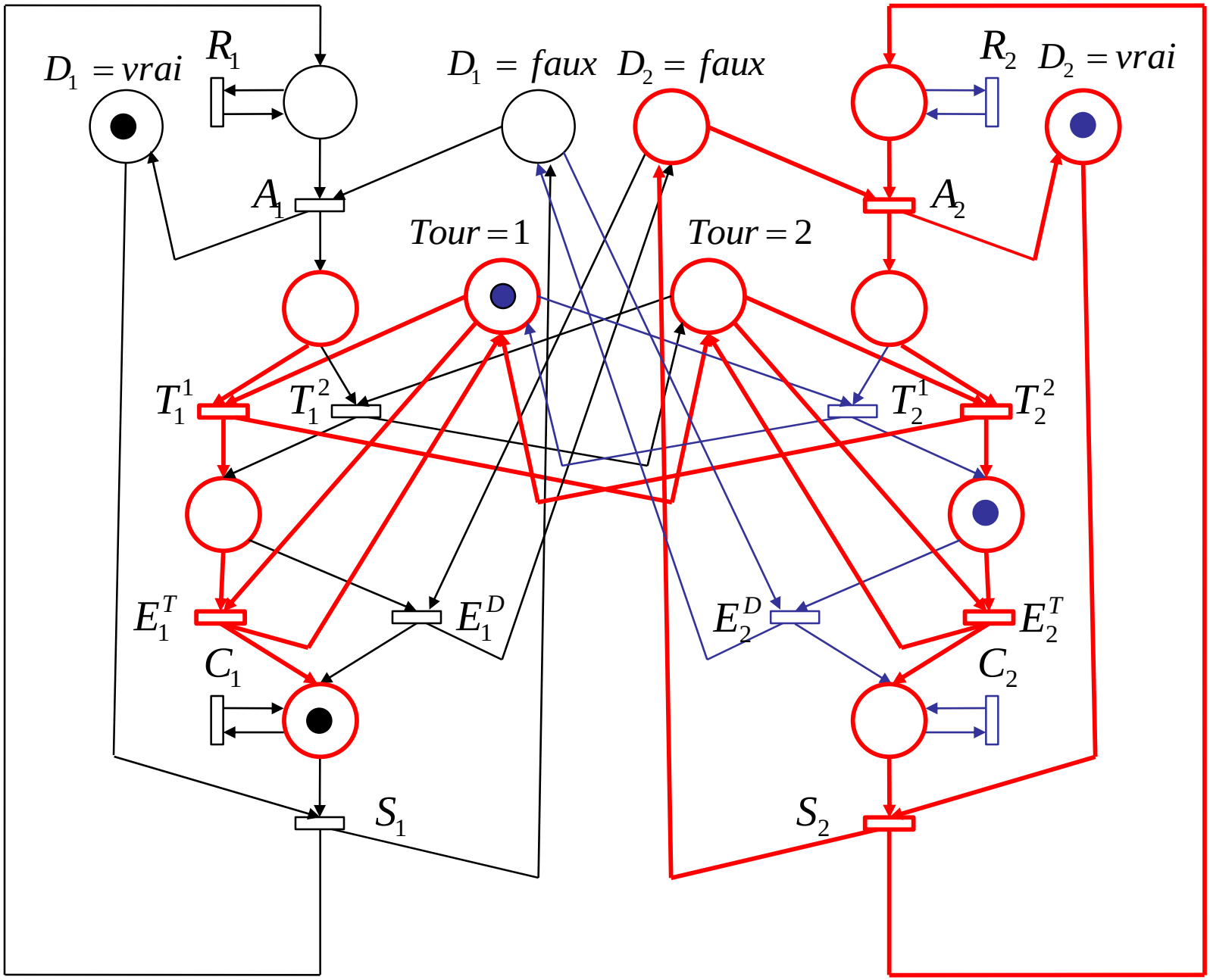
Processus 1 devient aussi en attente de sa section critique



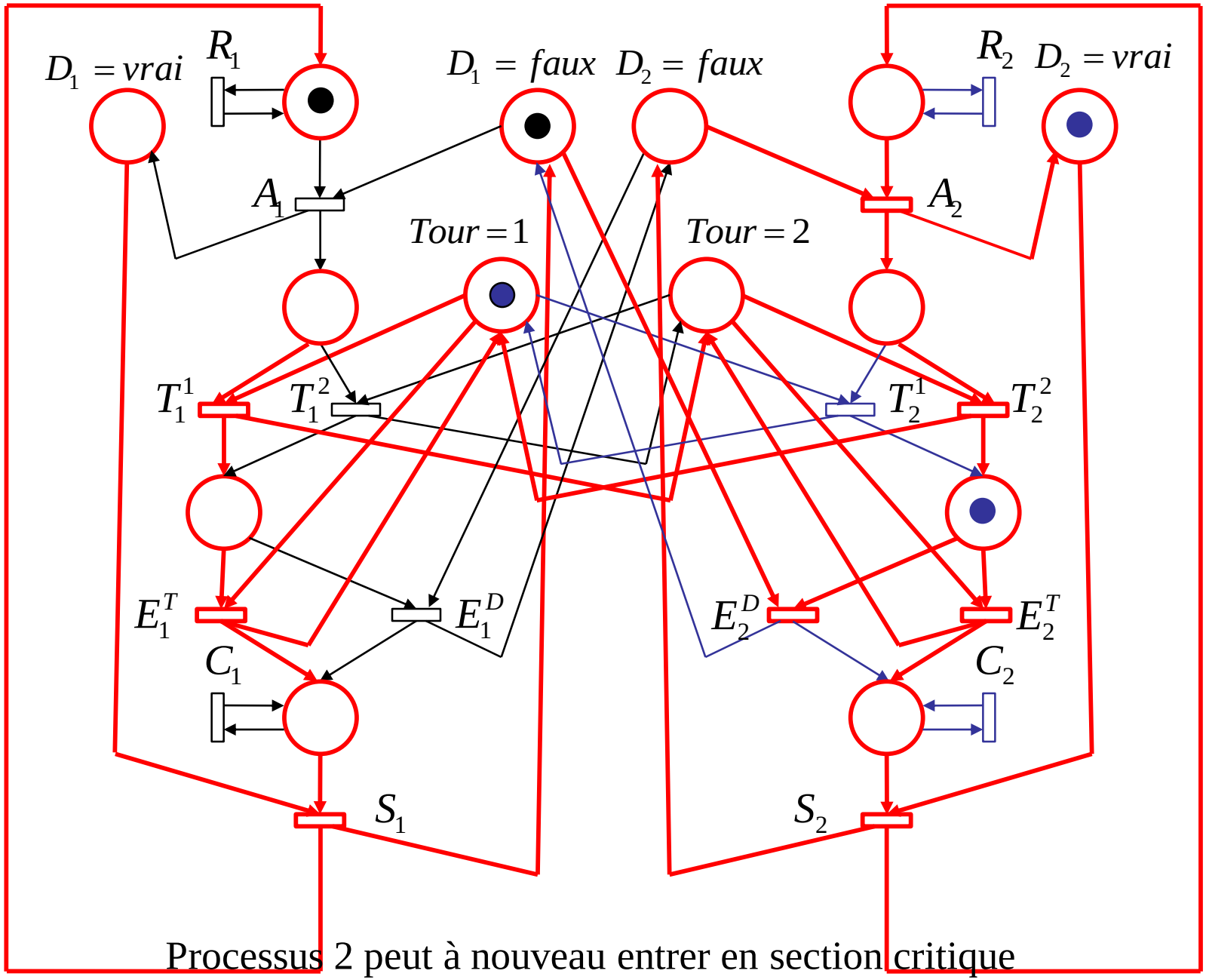
Processus 1 demeure en attente de sa section critique



Processus 2 à nouveau en attente de sa section critique



Processus 2 demeure en attente de sa section critique



Processus 2 peut à nouveau entrer en section critique
(borne égale à 1 pour chacun des processus)