IMED JARRAS

VÉRIFICATION ET SYNTHÈSE D'UN RÉSEAU DE PETRI RELATIVEMENT À UNE SPÉCIFICATION LOGIQUE TEMPORELLE

Mémoire
présenté
à la Faculté des études supérieures
de l'Université Laval
pour l'obtention
du grade de Maître ès Sciences (M.Sc.)

Département d'Informatique
FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL

Juillet 1995

Résumé

Nous présentons dans ce mémoire une méthode de vérification et de synthèse d'un réseau de Petri relativement à une spécification logique temporelle.

La vérification consiste à décider si un réseau de Petri R vérifie une spécification f écrite en formules de la logique temporelle. Pour ce faire, il suffit de construire un graphe qui combine le graphe de couverture du réseau R et l'automate de Büchi acceptant les modèles de la formule f. Le réseau R vérifie la spécification f si le graphe construit contient un chemin qui est licite vis à vis du réseau R, et qui est accepté par l'automate de Büchi.

Quant à la synthèse, elle consiste à construire un réseau de Petri satisfaisant une spécification f. L'algorithme de synthèse prend en entrée un réseau de Petri R et une formule f de la logique temporelle, et donne en sortie un nouveau réseau R' satisfaisant f. Le réseau R' est obtenu à partir du réseau R auquel on a ajouté des places, des transitions et des arcs. La "réutilisation" du réseau original R a évidemment l'avantage de réduire considérablement le coût du développement.

Nous présentons enfin un algorithme de synthèse pour un système composé d'un contrôleur et d'un ensemble d'agents. Cette synthèse est aussi basée sur la modification des réseaux de Petri originaux donnés en entrée.

Avant-propos

Louange à notre Seigneur “ALLAH” qui nous a doté de la merveilleuse faculté de raisonnement. Louange à notre Créateur qui nous a incité à acquérir le savoir. C'est à lui que j'adresse toute ma gratitude en premier lieu.

En second lieu, je tiens à remercier mon directeur de recherche, le professeur Jules Desharnais, qui a toujours été disponible malgré ses nombreuses occupations, et dont les encouragements et les conseils judicieux me furent d'une très grande utilité. Je ne saurais trop lui témoigner ma gratitude.

Les professeurs Brahim Chaïb-draa et Thien Vo-Dai ont accepté de lire et de faire partie du jury; je les prie de croire à toute ma reconnaissance.

Ma gratitude va également au *gouvernement Tunisien* et à *Recherches Bell Northern Ltée* pour le support financier qu'ils ont bien voulu m'accorder.

Je voudrais aussi exprimer ma profonde gratitude à mes parents pour tout l'amour qu'ils n'ont pas cessés de me donner, à mon frère Mourad, à mes beaux parents et à mon beau frère.

Je remercie sincèrement tous les “habitués” des salles 3917 et 3918 du pavillon Pouliot pour leur esprit de camaraderie, en particulier Slim Sayadi et Slim Ben Lamine.

Je remercie également Jasmina Mérar et Mérièm Cammoun pour leur participation à la correction de ce mémoire.

Enfin, à mon épouse Maha, j'adresse mes remerciments pour sa confiance, sa patience, son soutien incessant et son encouragement durant la rédaction de ce mémoire et tout au long de mes études en informatique.

Table des matières

Liste des Tables

Liste des figures

Chapitre 1

Introduction

On distingue principalement deux types de programmes. Le premier est vu comme une *fonction* d'un état initial à un état final; dans le cas non déterministe comme une *relation* entre les états initial et final. Cette vue est particulièrement appropriée au programme qui accepte toutes ses données en entrée au début de l'exécution et rend à la fin ses sorties. Nous appelons un tel programme *transformationnel*, se référant à son interprétation comme *transformateur d'états*. Comme exemple typique de programmes transformationnels nous citons les programmes non interactifs. Cette classe de programmes dispose aujourd'hui d'outils de spécification et de description basés sur les fonctions d'états, la logique de Hoare ou les transformateurs de prédicats de Dijkstra.

D'autre part, il y a des systèmes qui ne peuvent être couverts par la vue transformationelle, tels les systèmes d'exploitation, les programmes de contrôle de processus, les systèmes de réservation de places, etc. Habituellement, ces programmes ne se terminent pas. D'ailleurs, le but de leur exécution est non pas d'obtenir un résultat final, mais plutôt de maintenir certaines interactions avec leur environnement. Nous appelons de tels systèmes des *systèmes concurrents*. Dans la littérature, pour caractériser cette classe de systèmes, nous trouvons des termes équivalents : *distribués*, *interagissants*, *parallèles*, *réactifs*, *communicants*, etc.

Il est clair que les systèmes concurrents ne peuvent être décrits en référant seulement à leurs états initiaux et finaux. Une description adéquate doit tenir compte de leur comportement permanent qui est une séquence (peut-être infinie) d'états ou d'événements. Plusieurs méthodes formelles ont déjà été proposées pour spécifier, analyser, vérifier et synthétiser les programmes concurrents : réseaux de Petri [Petri62], CCS [Milner89], CSP [Hoare78] et plusieurs logiques modales, en particulier la logique temporelle [Manna87]. Les réseaux de Petri sont appropriés pour la spécification explicite des structures de programmes concurrents, alors que la logique temporelle est fortement recommandée pour la spécification des propriétés et contraintes des programmes. Les réseaux de Petri sont un outil de modélisation graphique et mathématique. Leur champ d'application est très large. Ils sont un outil prometteur pour la description et l'étude des systèmes de traitement d'informations concurrents, asynchrones, distribués, parallèles, non déterministes et/ou stochastiques. Comme outil graphique, les réseaux de Petri permettent la visualisation du comportement dynamique et des activités concurrentes du système. Comme outil mathématique, ils permettent l'analyse

de propriétés importantes, telles que l'absence de situation de blocage ou l'existence d'un régime permanent. Grâce à leur généralité et à leur souplesse, les réseaux de Petri ont été proposés pour une large variété d'applications.

Le formalisme des réseaux de Petri est considéré comme un cas particulier du formalisme général des systèmes de transitions [Arnold92]. Intuitivement, un système de transitions consiste en une description des états possibles d'un système, et de l'effet des différentes actions sur ces états. En définissant le comportement d'un réseau de Petri par l'ensemble de toutes les séquences de transitions observables qui peuvent avoir lieu, un réseau de Petri peut être vu comme un automate qui génère des mots. L'ensemble de tous les mots est appelé langage du réseau de Petri. Les systèmes concurrents apparaissent très souvent comme des processus dynamiques continus dans le temps. Par conséquent, il est très naturel de représenter le comportement d'un réseau de Petri par l'ensemble des séquences infinies de transitions qui peuvent avoir lieu [Valk82]. Les séquences infinies de transitions ont été utilisées pour l'étude des problèmes de blocage et de privation (famine).

Si les réseaux de Petri sont fortement adaptés à la modélisation formelle et graphique, ils sont insuffisants pour décrire les contraintes déclaratives du programme, ce qui est un des points forts de la logique. La logique temporelle introduite par Pnueli [Pnueli77] convient à la vérification et à la synthèse des programmes concurrents [Manna87] et est bien adaptée à la spécifiaction des contraintes de programmes [Manna84]. Elle est un type spécial de logique modale : ce qu'on appelle *monde possible* dans le cas de la logique modale est appelé *date, instant ou position* dans celui de la logique temporelle et la relation d'accessibilité entre mondes peut être vue comme la relation de consécution entre dates [Audureau90]. Une séquence de transitions d'un système peut correspondre à une formule logique temporelle. Il suffit pour cela d'identifier l'*état du système* et l'*instant*. Une fois l'analogie constatée, il est assez simple de trouver un codage approprié des propriétés de programmes dans le langage de la logique temporelle.

Par conséquent, il est très utile de combiner réseaux de Petri et logique temporelle comme un langage pour l'analyse, la vérification et la synthèse des programmes concurrents. Un programme concurrent spécifié par des formules logiques temporelles peut être représenté par un réseau de Petri. Plusieurs classes dans lesquelles réseaux de Petri et logique temporelle sont combinés ont déjà été proposées. Cependant, ces classes sont inadéquates pour la vérification et la synthèse automatique car certaines sont indécidables à l'égard du problème de *vacuité*, i.e. s'il existe une séquence de tirs licite satisfaisant une formule logique temporelle sur un réseau de Petri donné; d'autres sont décidables mais la complexité est équivalente à celle du problème d'accessibilité. La décidabilité du problème de vacuité est nécessaire pour la vérification et la synthèse automatique.

Dans ce mémoire nous considérons une classe combinant réseaux de Petri et logique temporelle linéaire propositionelle, qui est moins expressive que celles mentionnées ci-dessus, mais dont le problème de vacuité est décidable et de complexité équivalente à celle du problème de couverture. Dans cette classe, une séquence de transitions est un modèle de formules logiques temporelles.

Cette étude est principalement une présentation détaillée de l'article d'Uchihira

[Uchihira90]. C'est un article très dense, contenant quelques erreurs et qui fait appel à plusieurs résultats publiés dans d'autres articles. Le mémoire a pour but de donner plus de détails sur les différents concepts utilisés et de faciliter la compréhension de l'article. L'étude porte dans un premier temps sur des définitions générales concernant les réseaux de Petri, la logique temporelle et les automates de Büchi. Dans un deuxième temps, nous présentons les résultats suivants :

1. Il est décidable si un réseau de Petri satisfait une spécification logique temporelle.

2. Pour un réseau de Petri R et une formule f de la logique temporelle linéaire propositionelle donnés, nous construisons, si cela est possible, un réseau R' satisfaisant f en modifiant R.

Dans le chapitre 2, nous rappelons le modèle de base des réseaux de Petri. Nous présentons les aspects statiques et dynamiques de représentation des systèmes parallèles. Puis nous introduisons la logique temporelle linéaire propositionelle. Nous décrivons la sémantique des opérateurs temporels et nous présentons quelques-unes de leurs propriétés.

Le chapitre 3 introduit l'automate de Büchi, automate fini sur les mots infinis. Nous présentons le théorème de Wolper [Wolper83b] qui associe à chaque formule logique temporelle linéaire propositionelle f un automate de Büchi acceptant les modèles de f.

La vérification des réseaux de Petri est présentée au chapitre 4. Étant donnés un réseau de Petri R et une spécification (propriété, contrainte) donnée par une formule de la logique temporelle, un graphe combinant le graphe de couverture de R et l'automate de Büchi acceptant les modèles de f est construit. Si le graphe contient un cycle licite contenant un noeud désigné et qu'on peut trouver un chemin licite allant du noeud initial au noeud désigné, alors le réseau R satisfait la spécification f.

La partie la plus technique de ce mémoire est le chapitre 5. Dans un premier temps, nous y présentons un algorithme permettant, à partir d'un réseau R et d'une spécification f, de construire un réseau R' vérifiant f. Dans un second temps, nous appliquons la méthode de synthèse d'une façon compositionnelle pour modifier un programme concurrent représenté par un réseau de Petri de manière à satisfaire une spécification f donnée.

La matière présentée dans le chapitre 2 est surtout tirée de manuels sur les réseaux de Petri [Leeuwen90, Reisig85, Vidal92] et la logique temporelle [Audureau90, Manna91]. L'algorithme de construction d'automate de Büchi est dû aux travaux de Wolper, Sistla et Vardi [Wolper83b, Wolper87, Vardi86]. Les chapitres 4 et 5 s'inspirent principalement des travaux d'Uchihira [Uchihira90] et de Valk et Jantzen [Valk85].

Chapitre 2

Préliminaires

Ce chapitre, essentiellement un chapitre de définitions, est composé de deux parties. La première partie rassemble les outils qu'il convient de connaître pour étudier les réseaux de Petri. La seconde partie est consacrée à l'introduction de la Logique Temporelle Linéaire Propositionnelle. Le chapitre ne fait que rappeler les éléments nécessaires à la compréhension des autres chapitres. Par conséquent, nous supposons que le lecteur a certaines notions de réseaux de Petri et de logique temporelle. Nous supposons en plus connues les notions classiques de la théorie des ensembles, de l'algèbre linéaire, de la logique propositionnelle et des prédicats du premier ordre.

2.1 Les réseaux de Petri

Les réseaux de Petri ont été définis en 1962 par Carl Adam Petri [Petri62] dans sa thèse "Kommunikation mit Automaten". Ils permettent, en particulier, de modéliser et d'analyser des systèmes de processus concurrents et parallèles évoluant dans le temps de façon discrète.

2.1.1 Définition. (Réseau de Petri) [Reisig85, Vidal92] Un réseau de Petri (RDP) est la donnée d'un ensemble fini de *places*, d'un ensemble fini de *transitions* et d'une fonction dite *fonction de poids*. Ceci définit la structure statique du système. L'état de celui-ci se modélise à l'aide d'un *marquage* que l'on fait évoluer en franchissant des transitions, ce qui correspond à exécuter les actions qui leur sont associées. Formellement, un réseau de Petri peut être représenté par un quadruplet $R = (P, T, W, m_0)$ où :

$$\begin{aligned} P &= \{p_1, p_2, ..., p_n\} \text{ est l'ensemble des places,} \\ T &= \{t_1, t_2, ..., t_l\} \text{ est l'ensemble des transitions,} \\ W &: (P \times T) \cup (T \times P) \rightarrow \mathbf{N} \text{ est la fonction de poids et} \\ m &: P \rightarrow \mathbf{N} \text{ est la fonction de marquage.} \end{aligned}$$

Le marquage initial est donné par m_0. $m(p) = k$ signifie que la place contient k marques (jetons). On dit aussi que le marquage de p est k. $W(p, t) = k$ signifie que la transition

t utilise k jetons dans la place p; de façon duale, $W(t,p) = k$ signifie que la transition t crée k jetons dans la place p. □

2.1.2 Exemple. Soit le réseau de Petri $R = (P, T, W, m_0)$ où :

$$P = \{p_1, p_2, p_3\} \text{ et}$$
$$T = \{t_1, t_2, t_3, t_4\};$$

définissons W et m_0 de la façon suivante :

$$\begin{array}{rclrclrclrcl} W(p_1,t_2) & = & 1; & W(p_2,t_1) & = & 1; & W(p_2,t_3) & = & 3; & W(p_3,t_4) & = & 1; \\ W(t_1,p_1) & = & 1; & W(t_2,p_2) & = & 1; & W(t_3,p_3) & = & 1; & W(t_4,p_2) & = & 3; \\ m_0(p_1) & = & 0; & m_0(p_2) & = & 3; & m_0(p_3) & = & 0. & & & \end{array}$$

Pour tous les couples (x, y) non spécifiés ci-dessus, $W(x, y) = 0$. Par exemple, $W(p_1, t_3) = W(t_3, p_1) = 0$. □

Cet exemple est utilisé dans toute la section 2.1.

2.1.1 Représentation graphique

Un réseau de Petri peut être vu [Vidal92] comme un graphe bipartie où :

les places sont représentées par des cercles,

les transitions par des rectangles (ou traits).

Un arc relie une place p à une transition t ssi[1] $W(p,t) \neq 0$. Un arc relie une transition t à une place p ssi $W(t,p) \neq 0$. La représentation graphique du réseau de Petri de l'exemple 2.1.2 est donnée à la figure 2.1.

Règles de franchissement d'une transition

Les règles suivantes permettent de représenter l'évolution des systèmes à modéliser.

1. Une transition t est dite *franchissable* (enabled) pour le marquage m si pour toute place p telle que $W(p,t) \neq 0$ est marquée d'au moins $W(p,t)$ jetons, i.e. si

   $$\forall p \in P, m(p) \geq W(p,t).$$

   On note ceci $m(t\rangle$.

2. Une transition t est franchie en retirant $W(p,t)$ jetons de chaque place p telle que $W(p,t) \neq 0$, et en ajoutant $W(t,p')$ jetons à chaque place p' telle que $W(t,p') \neq 0$, i.e. si t est franchissable pour m, alors le franchissement de t fait passer le marquage m au marquage m' de la façon suivante :

   $$\forall p \in P, m'(p) = m(p) \Leftrightarrow W(p,t) + W(t,p).$$

   On note ceci $m(t\rangle m'$.

[1] Abréviation de "si et seulement si"

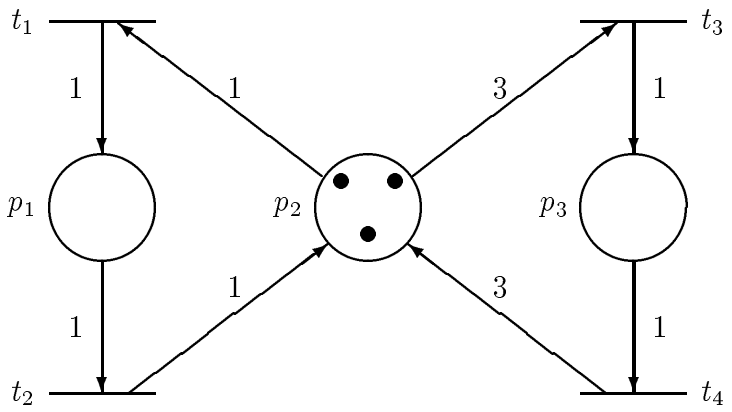


Figure 2.1 : Représentation graphique d'un réseau de Petri.

La notion de franchissement de transitions peut alors être étendue aux séquences de transitions.

2.1.3 Définition. Soit $s = t_1 t_2 \cdots t_n$ une séquence de transitions. La séquence s est franchissable à partir de m et conduit au marquage m' ce qui sera noté $m(s\rangle m'$ ssi il existe des marquages $m_0 = m, m_1, \cdots, m_n = m'$ tels que $\forall i : 0 \leq i \leq n \Leftrightarrow 1 : m_i(t_{i+1}\rangle m_{i+1}$ □

2.1.4 Exemple.

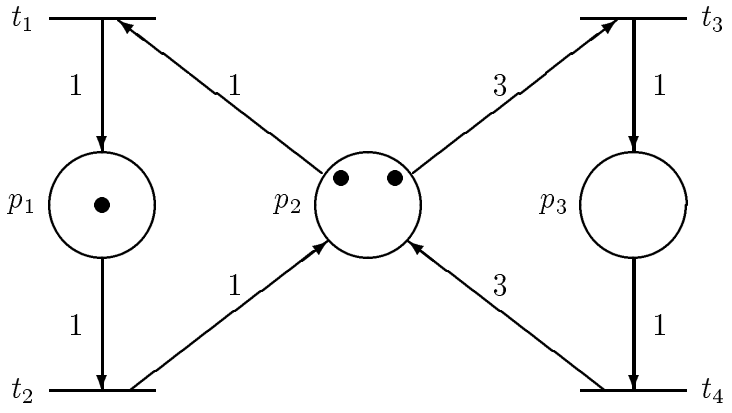


Figure 2.2 : Le réseau de Petri après le franchissement de t_1.

Pour le réseau R de la figure 2.1, seules les transitions t_1 et t_3 sont franchissables. En choisissant de franchir t_1 (figure 2.2), t_3 devient infranchissable, puisque le nombre

de jetons dans p_2 a diminué ($m(p_2) = 2 < W(p_2, t_3) = 3$). En contrepartie, la transition t_2 devient franchissable car $m(p_1) = 1 = W(p_1, t_2)$. □

2.1.2 Représentation matricielle

Il est possible de représenter par des matrices la fonction W [Vidal92]. On appelle matrice *précondition pré* la matrice de dimensions (p, n) à coefficients dans $\mathbf{N}$, où p est le nombre de places, n est le nombre de transitions et $pré(i, j) = W(p_i, t_j)$. On appelle matrice *postcondition post* la matrice de dimensions (p, n) à coefficients dans $\mathbf{N}$ définie par $post(i, j) = W(t_j, p_i)$. La matrice $C = post \Leftrightarrow pré$ est appelée *matrice d'incidence*. $pré(i, j)$ indique le nombre de marques que doit contenir la place p_i pour que la transition t_j soit franchissable. De façon duale, $post(i, j)$ contient le nombre de marques déposées dans la place p_i à la suite du franchissement de la transition t_j. Un marquage m sera alors représenté par un vecteur de dimension p à coefficients dans $\mathbf{N}$. On le note $\overline{m}$.

2.1.5 Exemple. La représentation matricielle du réseau de la figure 2.2 est comme suit :

$$pré = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array},$$

$$post = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array},$$

$$C = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 1 & \Leftrightarrow 1 & 0 & 0 \\ \Leftrightarrow 1 & 1 & \Leftrightarrow 3 & 3 \\ 0 & 0 & 1 & \Leftrightarrow 1 \end{pmatrix} \end{array}.$$

Le marquage initial est

$$m_0 = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \\ \begin{pmatrix} 0 \\ 3 \\ 0 \end{pmatrix} \end{array}.$$

□

2.1.6 Notation. Nous notons $W(., t_i) = pré(., i)$ le $i^{ème}$ vecteur colonne de la matrice $pré$ et $W(p_j, .) = pré(j, .)$ le $j^{ème}$ vecteur ligne de la matrice $pré$. □

2.1.7 Définition. (Vecteur caractéristique d'une transition) [Vidal92] Le *vecteur caractéristique* de la transition t_i, noté $\overline{t}_i$, est défini comme suit :

$$\begin{aligned} \overline{t}_i(j) &= 0 \text{ pour } j \neq i, 1 \leq j \leq l, \text{ où } l \text{ est le nombre de transitions,} \\ \overline{t}_i(i) &= 1. \end{aligned}$$

Si $\overline{m} \geq (pr\acute{e} \cdot \overline{t}) = W(\cdot, t)$, où le "$\cdot$" dans $pr\acute{e} \cdot \overline{t}$ dénote le produit matriciel, alors t est franchissable pour m. Le franchissement de t aboutit au marquage m' défini par :

$$\overline{m}' = \overline{m} + post \cdot \overline{t} \Leftrightarrow pr\acute{e} \cdot \overline{t} = \overline{m} + C \cdot \overline{t}.$$

□

2.1.8 Exemple. Considérons le marquage initial m_0 et la transition t_3 ci-dessous :

$$\overline{m}_0 = \begin{pmatrix} 0 \\ 3 \\ 0 \end{pmatrix}, \quad \overline{t}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}.$$

On a

$$\overline{m}' = \overline{m}_0 + C \cdot \overline{t}_3 = \begin{pmatrix} 0 \\ 3 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & \Leftrightarrow 1 & 0 & 0 \\ \Leftrightarrow 1 & 1 & \Leftrightarrow 3 & 3 \\ 0 & 0 & 1 & \Leftrightarrow 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

□

La notion de vecteur caractéristique d'une transition peut alors être étendu au vecteur caractéristique d'une séquence.

2.1.9 Définition. (Vecteur caractéristique d'une séquence) [Vidal92] Soit $s = t_1 t_2 \ldots t_n$ une séquence finie de transitions franchissables (séquence de tir licite) à partir de m. On appelle *vecteur caractéristique* de s, le vecteur $\overline{s}$ dont la composante $\overline{s}(i)$ détermine le nombre d'occurrences de la transition t_i dans s. Si $m(s\rangle m'$ alors

$$\overline{m}' = \overline{m} + C \cdot \overline{s}$$

□

2.1.10 Exemple. Soit $s = t_1 t_1 t_2$ qui consiste à tirer deux fois t_1 et une fois t_2. Selon la définition 2.1.9, on peut écrire :

$$\overline{s} = \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

on aboutit alors au marquage m' suivant :

$$\overline{m}' = \overline{m}_0 + C \cdot \overline{s} = \begin{pmatrix} 0 \\ 3 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & \Leftarrow 1 & 0 & 0 \\ \Leftarrow 1 & 1 & \Leftarrow 3 & 3 \\ 0 & 0 & 1 & \Leftarrow 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}.$$

Autrement dit, si à partir du marquage initial m_0, on tire deux fois la transition t_1 et une fois la transition t_2, alors on aboutit au marquage $\overline{m}'$. Dans le schéma ci-dessous, on tire dans l'ordre et à partir du marquage m_0 les transitions t_1, t_1 et t_2.

$$\begin{pmatrix} 0 \\ 3 \\ 0 \end{pmatrix} \overset{t_1}{\Leftrightarrow} \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} \overset{t_1}{\Leftrightarrow} \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} \overset{t_2}{\Leftrightarrow} \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}.$$

□

2.1.11 Notation.

1. Soit t une transition, on note

   $$\Delta(t) = C.\overline{t} = (W(t,p_1) \Leftarrow W(p_1,t), \cdots, W(t,p_n) \Leftarrow W(p_n,t)).$$

   De plus, si $t_1, t_2, \cdots, t_k$ sont des transitions alors

   $$\Delta(t_1 t_2 \cdots t_k) = \Delta(t_1) + \Delta(t_2) + \cdots + \Delta(t_k).$$

2. On note $m(s\rangle$ le fait qu'une séquence s (finie ou infinie) soit franchissable à partir de m.

2.2 Propriétés des réseaux de Petri

Une fois qu'un réseau de Petri a été défini, on peut toujours l'analyser, pour déterminer ses propriétés. Celles-ci dépendent de la topologie et du marquage initial du réseau. Nous présentons dans cette section certaines propriétés qui nous seront utiles par la suite. Nous invitons le lecteur à consulter [Reisig85] pour découvrir d'autres propriétés des réseaux de Petri.

2.2.1 Définition. (Réseau k-borné) [Vidal92] Un réseau de Petri est dit *k-borné* ssi le nombre de jetons dans n'importe quelle place ne peut dépasser k. □

2.2.2 Définition. (Réseau sauf) [Vidal92] Un réseau de Petri est dit *sauf* ou (*safe*) ssi il est 1-borné. □

2.2.3 Exemple. Pour le réseau de l'exemple 2.1.2 et pour le même marquage initial m_0, on peut facilement vérifier que les places p_1 et p_2 peuvent avoir au maximum 3 jetons alors que la place p_3 ne peut avoir plus d'un jeton. On peut donc conclure que le réseau de Petri est 3-borné. De plus, si son marquage initial était $m_0 = (0, 1, 0)$, il serait sauf. □

Le concept de réseau k-borné est généralement utilisé pour modéliser les systèmes réels dont les ressources sont nécessairement finies. De plus, lorsqu'il s'agit d'un système logique, il est commode de le modéliser avec un réseau sauf.

La propriété de *monotonie* définie ci-dessous, traduise l'existence de conditions minimales de franchissement d'une transition. Elle est liée au fait que les conditions pour qu'une action soit possible découlent exclusivement de la présence de ressources en nombre suffisant.

2.2.4 Proposition. (Propriété de monotonie) [Vidal92] *Si* $m' \geq m$ *alors pour toute séquence* s, $m(s\rangle m_1 \Rightarrow (\exists m'_1 : m'(s\rangle m'_1$ *et* $m'_1 \geq m_1)$.

Démonstration. Par récurrence sur la longueur de la séquence s. Pour les séquences de longueur 1, i.e. pour toute transition t, et pour tout p dans P, nous avons $m(p) \geq W(p,t)$ d'une part, et $m'(p) \geq m(p)$ d'autre part. Par suite, $m'(p) \geq W(p,t)$, la transition t est franchissable à partir de m'. On a $m'_1(p) = m'(p) + W(t,p) \Leftrightarrow W(p,t) \geq m(p) + W(t,p) \Leftrightarrow W(p,t) = m_1(p)$; d'où le résultat. La récurrence est ensuite immédiate. □

Une méthode naturelle pour examiner un sytème consiste à étudier tous les états possibles de ce système. S'il est fini, nous pouvons par une étude exhaustive déterminer les propriétés du système.

2.2.5 Définition. (Marquage accessible) [Vidal92] Un marquage m est dit *accessible* à partir du marquage m_0 ssi il existe une séquence de tir s telle que

$$m_0(s\rangle m.$$

On note $Acc(R, m_0)$ l'ensemble des marquages du réseau R accessibles à partir de m_0. □

Une des méthodes les plus naturelles pour analyser un réseau de Petri consiste à le faire évoluer systématiquement à partir du marquage initial, pour déterminer quelles propriétés sont satisfaites ou non, par exploration exhaustive de tous les marquages accessibles. Ceci nous amène à construire un graphe de marquages.

2.2.6 Définition. (Graphe des marquages) [Vidal92] Soit R un réseau de Petri et m_0 son marquage initial. Le graphe des marquages $GA(R, m_0)$ est défini par les deux conditions suivantes :

- l'ensemble des sommets est l'ensemble $Acc(R, m_0)$;
- il existe un arc étiqueté par t de m à m' ssi $m(t\rangle m'$. □

$$\begin{pmatrix}3\\0\\0\end{pmatrix} \underset{t_1}{\overset{t_2}{\rightleftarrows}} \begin{pmatrix}2\\1\\0\end{pmatrix} \underset{t_1}{\overset{t_2}{\rightleftarrows}} \begin{pmatrix}1\\2\\0\end{pmatrix} \underset{t_1}{\overset{t_2}{\rightleftarrows}} \begin{pmatrix}0\\3\\0\end{pmatrix} \underset{t_4}{\overset{t_3}{\rightleftarrows}} \begin{pmatrix}0\\0\\1\end{pmatrix}$$

Figure 2.3 : Graphe des marquages accessibles.

2.2.7 Exemple. La figure 2.3 représente le graphe des marquages accessibles du réseau de la figure 2.1 ayant comme marquage initial $m_0 = (0, 3, 0)$. □

Le graphe des marquages n'est pas toujours constructible. En effet, $Acc(R, m_0)$ peut être trop grand, voire même infini, lorsque le réseau n'est pas borné. Cependant, il existe une alternative à ce graphe, le *graphe de couverture* [Reisig85], qui n'est autre qu'une "compression" du graphe des marquages. Il permet à chaque marquage accessible d'être soit représenté explicitement par un noeud du graphe, soit "couvert" par un noeud. En premier lieu et avant de donner la définition de ce graphe, il convient de définir ω qui représente un nombre infini.

2.2.8 Définition. [Vidal92] Soit $\omega \notin \mathbf{N}$ (ensemble des entiers naturels) et $\mathbf{N}_\omega = \mathbf{N} \cup \{\omega\}$. Les opérations $+, \Leftrightarrow$ et la relation $<$ sont étendues à N_ω de la manière suivante : pour tout $n \in \mathbf{N}$,

$$\begin{aligned} n &< \omega, \\ n + \omega &= \omega + \omega = \omega + n = \omega, \\ \omega \Leftrightarrow n &= \omega, \\ n \Leftrightarrow \omega &\text{ n'est pas défini.} \end{aligned}$$

On étend $+$, $\Leftrightarrow$ et $<$ aux vecteurs de $\mathbf{N}_\omega^m$ de manière analogue à celle utilisée pour étendre $+$, $\Leftrightarrow$ et $<$ aux vecteurs de $\mathbf{N}^m$. L'addition de deux vecteurs donne un vecteur dont les composantes sont égales à la somme des composantes correspondantes des deux vecteurs. La soustraction est similaire à l'addition à l'exeption évidemment qu'on soustrait au lieu d'additionner. Pour la comparaison, on compare les composantes correspondantes entre elles : si toutes les composantes du premier vecteur sont inférieures aux composantes correspondantes du deuxième vecteur alors le premier vecteur est inférieur au deuxième et vice-versa. Si certains composantes sont inférieures et certaines ne le sont pas alors les deux vecteurs ne sont pas comparables. □

Sans nous appesantir sur les détails, nous pouvons maintenant définir le graphe de couverture.

2.2.9 Algorithme. (Construction du graphe de couverture) [Uchihira90] Soit le réseau de Petri $R = (P, T, W, m_0)$, comportant n places et ayant comme marquage initial m_0. On note $GC(R, m_0) = (S, X)$ le graphe de couverture de R construit comme suit :

1. S est un ensemble de noeuds étiquetés par des éléments de $\mathbf{N}_\omega^n$; $S := \{m_0\}$.

2. X est un ensemble d'arcs (x, x') étiquetés par des éléments de T; $X := \emptyset$.
3. Répéter l'étape 4 jusqu'à ce que tous les noeuds soient traités.
4. Soit $x = (x_1, x_2, \cdots, x_n)$ un noeud non traité. Pour chaque transition t franchissable à partir de x, faire :
   (a) Créer un nouveau noeud candidat $x' = (x'_1, x'_2, \cdots, x'_n)$.
   (b) Pour tout i, $1 \leq i \leq n$, poser $x'_i := x_i \Leftrightarrow W(p_i, t) + W(t, p_i)$.
   (c) S'il existe un noeud r sur un chemin allant de la racine m_0 à x tel que
   $$(\forall i : 1 \leq i \leq n : r_i \leq x'_i),$$
   alors, pour chaque i tel que $r_i < x'_i$, poser $x'_i := \omega$.
   (d) Si le noeud x' n'appartient pas à S, alors ajouter le noeud x' à S et l'arc (x, x') étiqueté par t à X, sinon ajouter seulement l'arc (x, x') étiqueté par t à X. □

2.2.10 Remarques.

1. On peut remarquer qu'à l'étape 4.b et avec les règles données en 2.2.8, on a $x'_i = \omega$ si $x_i = \omega$.
2. L'algorithme de la construction du graphe de couverture s'arrête toujours, car :
   - On ne peut avoir aucune branche de longueur infinie. En effet, d'après le lemme de Karp et Miller [Karp69], toute suite infinie de vecteurs formés d'entiers positifs ou nuls $v_1, v_2, \cdots, v_k, \cdots$ est telle qu'elle contient au moins deux éléments (en fait une infinité de couples) v_i et v_j avec $i < j$ telles que $v_i \leq v_j$.
   - Le nombre de branches est fini, car pour chaque marquage le nombre de transitions franchissables est fini (inférieur ou égal au nombre de transitions du réseau de Petri). □

2.2.11 Exemple. (Producteur/consommateur) Le réseau de la figure 2.4 représente un processus producteur/consommateur. En partant du marquage

$$m_0 = (1, 0, 0, 1, 0, 0)$$

et en franchissant la séquence de transitions

$$s_1 = \underbrace{t_1 t_2 t_5}_{1} \underbrace{t_1 t_2 t_5}_{2} \cdots \underbrace{t_1 t_2 t_5}_{n},$$

le réseau aboutit au marquage

$$m_1 = (1, 0, 0, 1, 0, n).$$

En effet, la place p_6 joue le rôle d'un tampon, le nombre de jetons qu'elle contient indique le nombre d'objets produits et non consommés. Si on veut consommer un jeton

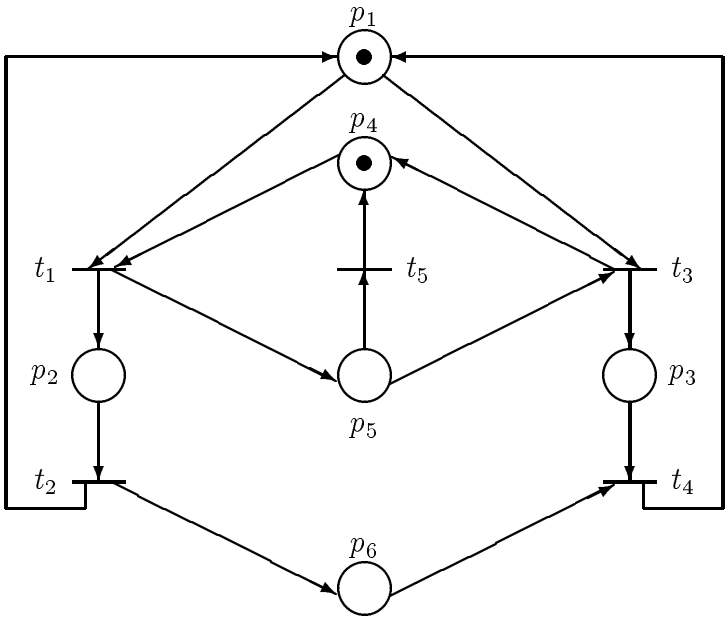


Figure 2.4 : Producteur/Consommateur.

de p_6 alors on doit tirer la séquence $s_2 = t_1t_2t_5 \cdots t_1t_2t_3t_4$. On remarque bien qu'avec ce réseau, on peut toujours produire sans jamais consommer. En revanche, on ne peut consommer plusieurs fois successivement. Par contre, si on éliminait la transition t_5, il y aurait alternance entre la production et la consommation. Le graphe de couverture de ce réseau est donné à la figure 2.5.

Un cas intéressant serait de prendre $W(p_6, t_4) > 1$ (prenons $W(p_6, t_4) = 2$). Chaque fois que la séquence t_3t_4 est tirée, deux jetons de p_6 sont consommés. Ce qui signifie que préalablement, on doit produire aux moins deux jetons dans p_6. Contrairement au premier cas, des situations de *blocage* peuvent avoir lieu. En particulier, si la séquence suivante est tirée :

$$s_1 = \underbrace{t_1t_2t_5t_1t_2t_3t_4}_{1}\underbrace{t_1t_2t_5t_1t_2t_3t_4}_{2} \cdots \underbrace{t_1t_2t_5t_1t_2t_3t_4}_{n} t_1t_2t_3$$

pour tout $n \geq 0$, on aboutit au marquage $m_b = (0, 0, 1, 1, 0, 1)$ qui est un blocage, i.e. aucune transition n'est franchissable à partir de m_b. Le graphe de couverture de ce réseau est donné à la figure 2.5 de laquelle on élimine l'arc t_4 du sous-graphe :

$$(0, 0, 1, 1, 0, 1) \overset{t_4}{\nleftrightarrow} (1, 0, 0, 1, 0, 0).$$

□

2.2.12 Remarques.

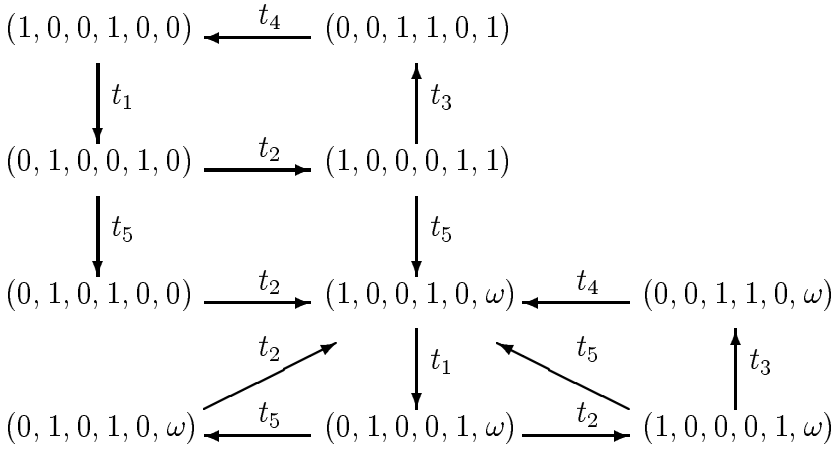


Figure 2.5 : Le graphe de couverture associé au réseau de la figure 2.4.

- Si le réseau est borné, le graphe de couverture coïncide avec le graphe des marquages.
- Étant donné que l'ensemble des sommets du graphe de couverture ne décrit pas l'ensemble des marquages accessibles, un chemin

  $$c \quad = \quad t_1, t_2, \cdots, t_k$$

  du graphe $GC(R, m_0)$ ne constitue pas toujours une séquence de tir licite. Par exemple, pour le RDP de la figure 1.4 avec $W(p_6, t_4) = 2$, la séquence

  $$s_1 \quad = \quad t_1 t_2 t_5 t_1 t_2 t_3 t_4 t_1 t_2 t_3 t_4$$

  est un chemin dans $GC(R, m_0)$. Par contre, elle n'est pas une séquence de tir licite. □

2.3 Langage associé à un réseau de Petri

Avant de présenter la définition du *langage associé à un réseau de Petri*, nous rappelons quelques notions de base sur les mots finis ou infinis; ces notions sont développées davantage dans [Vidal92]. Notons que les termes *séquence* et *mot* sont synonymes.

2.3.1 Définition. (Mot fini) Soit Σ un alphabet (ensemble de lettres). On note par $\Sigma^\star$ le monoïde libre généré par Σ, i.e. l'ensemble des mots finis ayant Σ comme alphabet, incluant le mot vide λ. On dénote par $|u|$ la longueur du mot $u \in \Sigma^\star$, i.e. le nombre de lettres qu'il contient. Pour tout n, $1 \leq n \leq |u|$, on note par $u(n)$ le $n^{ième}$ élément de u. □

Une notion fondamentale pour notre propos est celle de *préfixe (facteur gauche)*. Si $v, u \in \Sigma^\star$, le produit vu est défini par :

$$\begin{aligned} vu(n) &= v(n) \quad \text{si } 1 \leq n \leq |v| \text{ et} \\ vu(n+|v|) &= u(n) \quad \text{si } 1 \leq n \leq |u|. \end{aligned}$$

Le mot $v \in \Sigma^\star$ est alors dit préfixe de $u \in \Sigma^\star$ ssi il existe $w \in \Sigma^\star$ tel que $u = vw$. Nous notons $v \leq u$ la relation "v est un préfixe de u". Pour tout $i \leq |u|$, on note $u[i]$ le préfixe de u de longueur i. On note

$$FG(u) = \{v \in \Sigma^\star / \exists w \in \Sigma^\star : vw = u\}$$

l'ensemble des préfixes de $u \in \Sigma^\star$.

2.3.2 Définition. (Mot infini) On note $\mathbf{N}^+ = \mathbf{N} \setminus \{0\}$ l'ensemble des nombres positifs. Une séquence infinie sur Σ est une fonction $\omega : \mathbf{N}^+ \to \Sigma$. Comme $\omega(i)$ dénote le $i^{ème}$ élément de ω, on peut écrire $\omega = \omega(1)\omega(2)\omega(3)\cdots$. On note par Σ^ω l'ensemble de tous les mots infinis sur l'alphabet Σ. $\Sigma^\infty = \Sigma^\omega \cup \Sigma^\star$ représente l'ensemble de tous les mots finis ou infinis sur Σ. □

On étend la notion de *préfixe (facteur gauche)* aux mots infinis de la façon suivante : si $v \in \Sigma^\star$ et $u \in \Sigma^\omega$, le produit vu est défini par :

$$\begin{aligned} vu(n) &= v(n) \quad \text{si } 1 \leq n \leq |v| \\ vu(n+|v|) &= u(n) \quad \text{si } n \in \mathbf{N}^+. \end{aligned}$$

Le mot $v \in \Sigma^\star$ est alors dit préfixe de $u \in \Sigma^\omega$ ssi il existe $w \in \Sigma^\omega$ tel que $u = vw$. Si nous pouvons multiplier un mot infini à gauche par un mot fini, nous ne pouvons en faire autant à droite. Dans [Nivat77], l'auteur adopte la convention suivante :

$$\forall u \in \Sigma^\omega, v \in \Sigma^\star \cup \Sigma^\omega : uv = u.$$

On note

$$FG(u) = \{u[i] / i \in \mathbf{N}^+\}$$

l'ensemble des préfixes de $u \in \Sigma^\omega$.

2.3.3 Notation. Pour tout $u \in \Sigma^\star$, $u^\star$ dénote une séquence finie de u et u^ω une séquence infinie de u. Par exemple, si $u = aba$ alors

$$\begin{aligned} u^\star &= (aba)(aba)\cdots(aba) \text{ et} \\ u^\omega &= (aba)(aba)\cdots. \end{aligned}$$

□

2.3.4 Définition. (RDP étiqueté) [Uchihira90, Vidal92] Soit Σ un alphabet. On définit un *réseau de Petri étiqueté* par un couple (R, h) où $R = (P, T, W, m_0)$ est un réseau de Petri et $h : T \to \Sigma \cup \{\lambda\}$ est une fonction dite *fonction d'étiquetage*. On étend de façon naturelle la fonction h à $h : T^\infty \to \Sigma^\infty$ par $h(\theta)(i) = h(\theta(i))$ pour tout $\theta \in T^\infty$ et où $1 \leq i \leq |\theta|$ si $\theta \in T^\star$ et $i \in \mathbf{N}^+$ si $\theta \in T^\omega$. □

Les transitions d'un RDP représentent les événements (ou actions) d'un système. Une façon générale de montrer que deux événements t_1 et t_2 sont identiques à l'observateur est de leur attribuer la même étiquette; i.e. $h(t_1) = h(t_2) = x \in \Sigma$. Si on veut ignorer une transition, on peut l'étiqueter par le mot vide λ.

2.3.5 Définition. (ω-langage) [Vidal92] Un *langage* L sur l'alphabet Σ est un sous-ensemble $L \subseteq \Sigma^\star$ de mots finis. $FG(L) = \bigcup\{FG(v)/v \in L\}$ est l'ensemble des préfixes de L. Un *ω-langage* est un sous-ensemble $L_\omega \subseteq \Sigma^\omega$ de mots infinis. $FG(L_\omega) = \bigcup\{FG(v)/v \in L_\omega\}$ est l'ensemble des préfixes de L_ω. □

Soit $V \subseteq \Sigma$, l'ensemble des étiquettes visibles (par l'observateur); $h(t) \in \Sigma \Leftrightarrow V$ est invisible. Nous définissons l'opérateur de restriction "/" comme suit :

$$h/V : T \to V \cup \{\lambda\} \quad \text{tel que} \quad \begin{array}{lcll} h/V(t) & = & h(t) & \text{si} \quad h(t) \in V \\ h/V(t) & = & \lambda & \text{si} \quad h(t) \notin V. \end{array}$$

Nous notons $I : T \to T$ la fonction identité, définie par $I(t) = t$ pour tout $t \in T$. On utilise parfois l'abréviation θ/V pour désigner $I/V(\theta)$ (où $\theta \in T^\infty$).

Nous pouvons maintenant définir le langage associé à un RDP.

2.3.6 Définition. (Langage d'un RDP) [Uchihira90] Soit $R = (P, T, W.m_0)$ un réseau de Petri. On pose $F(R, m_0) = \{v \in T^\star / m_0(v\rangle\}$ ($F(R, m_0)$ est l'ensemble des séquences de tir licites finies) et $F_\omega(R, m_0) = \{v \in T^\omega / m_0(v\rangle\}$ ($F_\omega(R, m_0)$ est l'ensemble des séquences de tir licites infinies du RDP R). En utilisant l'extension de h à $h : T^\infty \to \Sigma^\infty$, on obtient

$$\begin{array}{rcll} L(R,h) & = & \{h(\theta) \in \Sigma^\star / \theta \in F(R)\} & \text{le langage de } R, \\ L_\omega(R,h) & = & \{h(\theta) \in \Sigma^\omega / \theta \in F_\omega(R)\} & \text{le } \omega\text{-langage de } R \text{ et} \\ L_{\lambda\omega}(R,h) & = & \{h(\theta) \in \Sigma^\infty / \theta \in F_\omega(R)\} & \text{le } \lambda\omega\text{-langage de } R. \end{array}$$

□

$L_\omega(R, h)$ ne contient que des mots infinis. Cependant, $L_{\lambda\omega}(R, h)$ peut contenir des mots finis comme $u = u\lambda^\omega \in \Sigma^\star$.

Dans un système complexe où chaque composante élémentaire peut être représentée par un RDP, il est nécessaire que ces composantes puissent être combinées entre elles de façon à assurer la spécification désirée du système global. Un des apports de cette combinaison est le *rendez-vous multiple* [Lloret90], assurant une synchronisation entre n ($n \geq 0$) transitions de composantes différentes. Nous introduisons par la suite l'opérateur de composition "$|_V$" qui joue un rôle important dans la synthèse compositionnelle.

2.3.7 Définition. (Composition des réseaux de Petri étiquetés) [Uchihira90] Soient les RDP $RE_1 = (R_1, h_1)$, $RE_2 = (R_2, h_2)$ et un ensemble d'étiquettes $V \subset \Sigma$

où :

$$\begin{aligned} R_1 &= (P_1, T_1, W_1, m_{01}), \\ R_2 &= (P_2, T_2, W_2, m_{02}), \\ T_1 \cap T_2 &= \emptyset, \\ P_1 \cap P_2 &= \emptyset, \\ h_1 &: T_1 \to \Sigma \cup \{\lambda\}, \\ h_2 &: T_2 \to \Sigma \cup \{\lambda\}. \end{aligned}$$

On introduit le RDP étiqueté $RE = (R, h) = RE_1|_V RE_2$, appelé la composition de RE_1 et RE_2 par rapport à V et défini par $RE = (P, T, W, m_0)$ où :

$$P = P_1 \cup P_2,$$

$$T = \acute{T_1} \cup \acute{T_2} \cup \acute{T} \text{ où } \begin{cases} \acute{T_1} = \{t \in T_1 / h_1(t) \notin V\}, \\ \acute{T_2} = \{t \in T_2 / h_2(t) \notin V\}, \\ \acute{T} = \{t_{ij} / h_1(t_i) = h_2(t_j) \in V, t_i \in T_1, t_j \in T_2\}, \end{cases}$$

$$W(p,t) = \begin{cases} W_1(p,t) & \text{pour } p \in P_1 \text{ et } t \in \acute{T_1}, \\ W_2(p,t) & \text{pour } p \in P_2 \text{ et } t \in \acute{T_2}, \\ W_1(p,t_i) & \text{pour } p \in P_1 \text{ et } t = t_{ij} \in \acute{T}, \\ W_2(p,t_j) & \text{pour } p \in P_2 \text{ et } t = t_{ij} \in \acute{T}, \end{cases}$$

$$W(t,p) = \begin{cases} W_1(t,p) & \text{pour } p \in P_1 \text{ et } t \in \acute{T_1}, \\ W_2(t,p) & \text{pour } p \in P_2 \text{ et } t \in \acute{T_2}, \\ W_1(t_i,p) & \text{pour } p \in P_1 \text{ et } t = t_{ij} \in \acute{T}, \\ W_2(t_j,p) & \text{pour } p \in P_2 \text{ et } t = t_{ij} \in \acute{T}, \end{cases}$$

$$m_0(p) = \begin{cases} m_{01}(p) & \text{pour } p \in P_1, \\ m_{02}(p) & \text{pour } p \in P_2, \end{cases}$$

$$h(t) = \begin{cases} h_1(t) & \text{pour } t \in \acute{T_1}, \\ h_2(t) & \text{pour } t \in \acute{T_2}, \\ h_1(t)(= h_2(t)) & \text{pour } t \in \acute{T}. \end{cases}$$

□

2.3.8 Exemple. [Uchihira90] Nous montrons dans la figure 2.6 un exemple de composition des réseaux de Petri étiquetés (R_1, h_1) et (R_2, h_2). Dans cet exemple, $h_1(t_1) = b$, $h_2(t_4) = c$, $h_1(t_2) = h_2(t_3) = a$ et $V = \{a\}$.

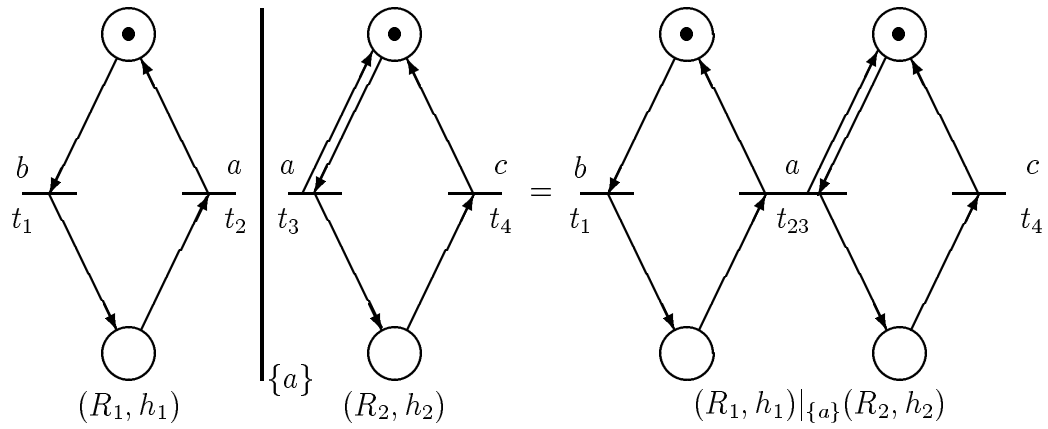


Figure 2.6 : Composition des réseaux de Petri.

Les langages d'un reseau de Petri obtenu par composition peuvent être déduit à partir des langages des composantes.

2.3.9 Lemme. [Uchihira90] *Soient* $RE = (R, h)$, $RE_1 = (R_1, h_1)$ *et* $RE_2 = (R_2, h_2)$ *des réseaux de Petri étiquetés. Si* $RE = RE_1|_\Sigma RE_2$, *alors* $L(RE) = L(RE_1) \cap L(RE_2)$, $L_\omega(RE) = L_\omega(RE_1) \cap L_\omega(RE_2)$ *et* $L_{\lambda\omega}(RE) \supset L_{\lambda\omega}(RE_1) \cap L_{\lambda\omega}(RE_2)$.

2.4 La logique temporelle

La *logique temporelle* est un type spécial de *logique modale*; elle permet de voir la validité des formules qui évoluent dans le temps. Pour ce faire, elle dispose d'opérateurs pour le passé, le présent et le futur. Dans [Audureau90, Manna91, Pnueli77] les auteurs ont démontré que la logique temporelle est un bon formalisme pour la spécification et la vérification des programmes parallèles et des systèmes réactifs. De plus, elle est très bien adaptée pour exprimer des propriétés comme la terminaison, l'équité, l'absence de blocage et bien d'autres. Dans cette partie, nous nous intéressons seulement à la *Logique Temporelle Linéaire Propositionnelle* (LTLP).

2.4.1 La Logique Temporelle Linéaire Propositionnelle

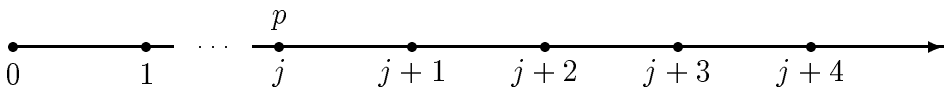
La *LTLP* [Leeuwen90] est une extension de la logique propositionnelle classique, à laquelle, en plus des opérateurs usuels ($\vee$, $\wedge$, $\neg$ etc.), on ajoute des *opérateurs temporels*. Nous introduisons, par la suite, les définitions de ces opérateurs, puis nous décrivons la syntaxe et la sémantique de la LTLP.

Pour chaque opérateur et chaque formule temporelle, nous présentons une définition [Manna91] de leur interprétation dans un modèle donné. Le modèle σ sur lequel on raisonne est une suite infinie de valuations sur des propositions atomiques. Si une

formule p est vraie à une *position (date)* j, $j \geq 0$, dans la séquence σ, on dit que σ satisfait p en j, ce qu'on note

$$(\sigma, j) \models p$$

et qu'on peut schématiser par :

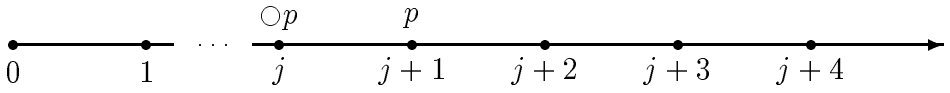


L'opérateur prochain (next) ◯

Si p est une formule temporelle, alors $\bigcirc p$ (lire *prochaine fois* p), en est une. Sa sémantique est :

$$(\sigma, j) \models \bigcirc p \text{ ssi } (\sigma, j+1) \models p.$$

La formule $\bigcirc p$ est vraie à la position j ssi p est vraie à la position $j+1$. Nous pouvons la schématiser par :

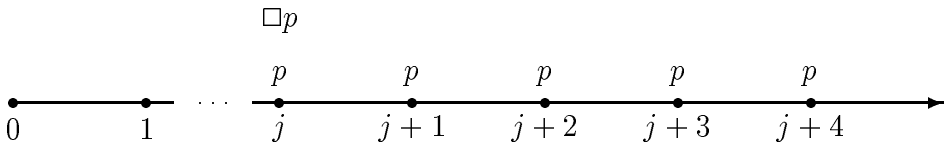


L'opérateur toujours (henceforth, always) □

Si p est une formule temporelle, alors $\Box p$ (lire *toujours* p) en est une. Sa sémantique est:

$$(\sigma, j) \models \Box p \text{ ssi } \forall k \geq j : (\sigma, k) \models p.$$

La formule $\Box p$ est vraie à la position j ssi p est vraie en toute position supérieure ou égale à j. Nous pouvons la schématiser par :



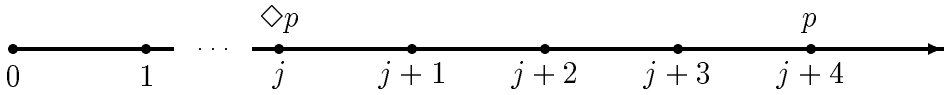
Il est facile de voir que si $\Box p$ est vraie à la position j, alors elle l'est pour toute position $k \geq j$.

L'opérateur nécessairement (eventually, sometime) $\Diamond$

Si p est une formule temporelle, alors $\Diamond p$ (lire *nécessairement* p) en est une. Sa sémantique est :

$(\sigma, j) \models \Diamond p$ ssi $(\sigma, k) \models p$ pour un certain $k \geq j$.

La formule $\Diamond p$ est vraie à la position j ssi p est vraie en une certaine position k supérieure ou égale à j. Nous pouvons la schématiser par :



Il est intéressant aussi de montrer que si $\Diamond p$ est vraie à la position j, alors elle l'est à n'importe quelle position k, $0 \leq k \leq j$. L'opérateur *nécessairement* est le dual de l'opérateur *toujours* :

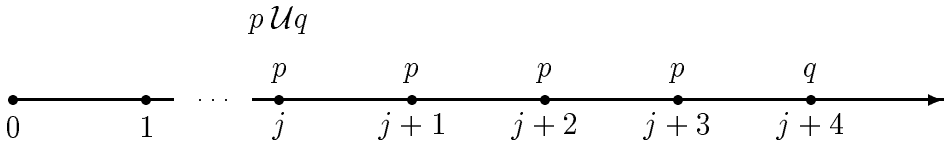
$(\sigma, j) \models \Diamond p$ ssi $(\sigma, j) \models \neg\Box\neg p$.

L'opérateur jusqu'à (Until) $\mathcal{U}$

La formule $p\,\mathcal{U}q$ (lire p *jusqu'à* q), combine les deux formules $\Diamond q$ et $\Box p$ en prédisant une occurrence future de q tout en exigeant que p reste vraie au moins jusqu'à la première occurrence de q. Formellement,

$$(\sigma, j) \models p\,\mathcal{U}q \quad \text{ssi} \quad \begin{array}{l} \text{il existe un } k \geq j, \text{ tel que } (\sigma, k) \models q \text{ et} \\ \text{pour tout } i, j \leq i < k, (\sigma, i) \models p. \end{array}$$

On la schématise par :



Notons que si q est vraie à la position j, alors $p\,\mathcal{U}q$ l'est aussi pour n'importe quelle formule p. De plus, si $p\,\mathcal{U}q$ est vraie à la position j, alors $\Diamond q$ l'est aussi.

Il est aussi utile d'introduire l'opérateur $\mathcal{W}$ (*jusqu'à faible*), défini comme suit :

$(\sigma, j) \models p\mathcal{W}q$ ssi $(\sigma, j) \models p\,\mathcal{U}q$ ou $(\sigma, j) \models \Box p$.

Syntaxe

Les formules de la *LTLP* sont formées à partir :

- d'un ensemble de *propositions atomiques*

$$Prop = \{p, q, r, \cdots\},$$

- des connecteurs booléens : $\wedge$, $\neg$,
- des opérateurs temporels : $\bigcirc$, $\mathcal{U}$.

Les règles de formation des formules sont les suivantes :

- toute variable propositionnelle $p \in Prop$, Vrai et Faux sont des formules,
- si f_1 et f_2 sont des formules alors $f_1 \wedge f_2$, $\neg f_1$, $\bigcirc f_1$ et $f_1 \mathcal{U} f_2$ en sont aussi.

Sémantique

Étant données une suite infinie de valuations sur *Prop* et une position $j, j \geq 0$, on définit la sémantique complète de la LTLP par :

- $(\sigma, j) \models p$ ssi p est vraie à la position j,
- $(\sigma, j) \models p \wedge q$ ssi $(\sigma, j) \models p$ et $(\sigma, j) \models q$,
- $(\sigma, j) \models \neg p$ ssi il est faux que $(\sigma, j) \models p$,
- $(\sigma, j) \models p \mathcal{U} q$ ssi $(\exists k : ((\sigma, k) \models q)$ et $(\forall i : j \leq i < k \Rightarrow (\sigma, i) \models p))$,
- $(\sigma, j) \models \bigcirc p$ ssi $(\sigma, j+1) \models p$.

Nous utilisons $\Diamond f$ pour abrévier Vrai$\mathcal{U} f$ et $\Box f$ comme abréviation de $\neg\Diamond\neg f$. De même $f_1 \vee f_2$, $f_1 \rightarrow f_2$, $f_1 \leftrightarrow f_2$ et $f_1 \mathcal{W} f_2$ représentent $\neg(\neg f_1 \wedge \neg f_2)$, $\neg f_1 \vee f_2$, $(f_1 \rightarrow f_2) \wedge (f_2 \rightarrow f_1)$ et $f_1 \mathcal{U} f_2 \vee \Box f_1$ respectivement.

2.4.2 Propriétés

La *logique temporelle* est riche de propriétés fort intéressantes. La section ci-après présente quelques-unes d'entre elles, tirées du livre de Manna et Pnueli [Manna91].

Satisfaction et validité

Pour une formule p et un modèle σ, on écrit

$$\sigma \models p \text{ ssi } (\sigma, 0) \models p$$

et on dit que σ *satisfait* p. Une formule est dite *valide* si $\sigma \models p$ pour tout modèle σ.

Équivalence et congruence

Deux formules p et q sont dites *équivalentes*, ce qui est noté par

$$p \sim q,$$

si $p \leftrightarrow q$ est une formule valide. Ceci signifie que p et q ont la même valeur de vérité à la première position dans chaque modèle. Les formules p et q sont dites *congruentes*, ce qui est noté

$$p \approx q,$$

si $\Box(p \leftrightarrow q)$ est valide. Ceci signifie que p et q ont la même valeur en toutes positions et pour chaque modèle. Nous abrévions

$$\begin{array}{lll} \Box(p \rightarrow q) & \text{par} & p \Rightarrow q \quad \text{et} \\ \Box(p \leftrightarrow q) & \text{par} & p \Leftrightarrow q. \end{array}$$

Dualité

Chaque opérateur a son dual. En effet, on peut montrer ce qui suit :

$$\begin{array}{rcl} \neg\Box p & \Leftrightarrow & \Diamond\neg p, \\ \neg\Diamond p & \Leftrightarrow & \Box\neg p, \\ \neg\bigcirc p & \Leftrightarrow & \bigcirc\neg p, \\ \neg(p\,\mathcal{U}q) & \Leftrightarrow & (\neg q)\mathcal{W}(\neg p \wedge \neg q), \\ \neg(p\mathcal{W}q) & \Leftrightarrow & (\neg q)\,\mathcal{U}(\neg p \wedge \neg q). \end{array}$$

Implication

Les formules valides suivantes décrivent des implications dans un seul sens :

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} \Box p & \Rightarrow & p, & \Box p & \Rightarrow & \Diamond p, & \Box p & \Rightarrow & \bigcirc p, \\ \Box p & \Rightarrow & \bigcirc\Box p, & \bigcirc p & \Rightarrow & \Diamond p, & p & \Rightarrow & \Diamond p, \\ p\,\mathcal{U}q & \Rightarrow & p \vee q, & p\,\mathcal{U}q & \Rightarrow & \Diamond q, & p\mathcal{W}q & \Rightarrow & p \vee q, \\ q & \Rightarrow & p\,\mathcal{U}q, & q & \Rightarrow & p\mathcal{W}q. & & & \end{array}$$

Idempotence

Un opérateur est dit *idempotent* si une double application donne le même résultat qu'une seule application.

$$\begin{array}{rcl@{\qquad}rcl} \Box\Box p & \Leftrightarrow & \Box p, & \Diamond\Diamond p & \Leftrightarrow & \Diamond p, \\ p\,\mathcal{U}(p\,\mathcal{U}q) & \Leftrightarrow & p\,\mathcal{U}q, & (p\,\mathcal{U}q)\,\mathcal{U}q & \Leftrightarrow & p\,\mathcal{U}q, \\ p\mathcal{W}(p\mathcal{W}q) & \Leftrightarrow & p\mathcal{W}q, & (p\mathcal{W}q)\mathcal{W}q & \Leftrightarrow & p\mathcal{W}q. \end{array}$$

Absorption

Voici deux formules d'absorption :

$$\begin{aligned} \Diamond\Box\Diamond p &\Leftrightarrow \Box\Diamond p, \\ \Box\Diamond\Box p &\Leftrightarrow \Diamond\Box p. \end{aligned}$$

L'idempotence et l'absorption sont deux propriétés très intéressantes, puisqu'elles permettent de réduire la taille des formules de la forme :

$$O_1 O_2 \cdots O_k p,$$

où O_i est soit $\Box$ soit $\Diamond$. Par la règle d'idempotence nous pouvons réduire la formule

$$O_1 O_2 \cdots \Box\Box \cdots O_k p$$

en

$$O_1 O_2 \cdots \Box \cdots O_k p$$

et avec le même principe on traite chaque occurence de $\Diamond\Diamond$. Ainsi, la suite O_i devient une suite alternative de $\Box$ et $\Diamond$. En appliquant les règles d'absorption, on obtient une de ces quatres formules :

$$\Box p, \qquad \Diamond p, \qquad \Diamond\Box p, \qquad \Box\Diamond p.$$

Distributivité

Nous avons les relations de *distributivité* suivantes :

$$\begin{aligned} \Diamond(p \vee q) &\Leftrightarrow \Diamond p \vee \Diamond q, \\ \Box(p \wedge q) &\Leftrightarrow \Box p \wedge \Box q, \\ (p \wedge q)\,\mathcal{U} r &\Leftrightarrow (p\,\mathcal{U} r) \wedge (q\,\mathcal{U} r), \\ p\,\mathcal{U}(q \vee r) &\Leftrightarrow (p\,\mathcal{U} q) \vee (q\,\mathcal{U} r), \\ (p \wedge q)\ \mathcal{W} r &\Leftrightarrow (p\ \mathcal{W} r) \wedge (q\ \mathcal{W} r), \\ p\ \mathcal{W}(q \vee r) &\Leftrightarrow (p\ \mathcal{W} q) \vee (q\ \mathcal{W} r). \end{aligned}$$

L'opérateur *prochain* $\bigcirc$ distribue sur tous les connecteurs booléens :

$$\begin{aligned} \bigcirc(p \vee q) &\Leftrightarrow \bigcirc p \vee \bigcirc q, \\ \bigcirc(p \wedge q) &\Leftrightarrow \bigcirc p \wedge \bigcirc q, \\ \bigcirc(p \rightarrow q) &\Leftrightarrow \bigcirc p \rightarrow \bigcirc q, \\ \bigcirc(p \leftrightarrow q) &\Leftrightarrow \bigcirc p \leftrightarrow \bigcirc q. \end{aligned}$$

Il y a aussi des implications qui ne peuvent être étendues à des équivalences :

$$\begin{aligned} \Box p \vee \Box q &\Rightarrow \Box(p \vee q), \\ \Diamond(p \wedge q) &\Rightarrow \Diamond p \wedge \Diamond q, \\ (p\,\mathcal{U} r) \vee (q\,\mathcal{U} r) &\Rightarrow (p \vee q)\,\mathcal{U} r, \\ p\,\mathcal{U}(q \wedge r) &\Rightarrow (p\,\mathcal{U} q) \wedge (p\,\mathcal{U} r). \end{aligned}$$

Expansion

Voici trois formules d'*expansion* :

$$\begin{aligned} \Diamond p &\Leftrightarrow p \vee \bigcirc \Diamond p, \\ \Box p &\Leftrightarrow p \wedge \bigcirc \Box p, \\ p \,\mathcal{U}\, q &\Leftrightarrow q \vee (p \wedge \bigcirc (p \,\mathcal{U}\, q)). \end{aligned}$$

Ces formules décrivent les opérateurs $\Diamond$, $\Box$ et $\mathcal{U}$ comme des points fixes. Par exemple, $\Diamond p$ est un point fixe de la fonction $f(x) = p \vee \bigcirc x$.

Chapitre 3

Automates de Büchi

Les automates finis sur les mots infinis sont introduits, dans les années soixante, par Büchi [Büchi62], McNaughton [McNaughton66] et Rabin [Rabin69]. Leurs travaux ont ouvert des champs d'intérêt entre la théorie des automates et d'autres domaines de l'informatique comme la logique. Ces types d'automates ont connu une expansion considérable lorsqu'on s'est aperçu de leur adéquation à la description des systèmes à exécution infinie (*nonterminating computation*). Dans le cas propositionnel, une telle exécution est vue comme une séquence infinie de valeurs de vérité d'une formule propositionnelle. Dans [Pnueli77], la logique temporelle a été introduite pour raisonner sur de telles séquences.

Certains travaux [Vardi86, Wolper83a, Wolper83b] ont établi une relation ferme entre la logique temporelle et la théorie des langages ω-réguliers. Les langages ω-réguliers sont analogues aux langages réguliers, mais définis sur les mots infinis plutôt que les mots finis. La notion de ω-régularité est bien fondée et sa théorie est bien développée [Choueka74]. Une des caractéristiques d'un langage ω-régulier est qu'il existe un automate de Büchi qui le reconnaît.

Un automate de Büchi est un automate fini non déterministe muni d'une condition d'acceptation. Un mot u est accepté ssi l'automate peut lire u en passant par des états désignés infiniment souvent.

Dans [Wolper87], il a été démontré que les automates de Büchi et plusieurs types de logiques temporelles ont exactement la même puissance d'expression; en d'autres termes, la classe des ensembles de séquences décrites par ces logiques coïncide avec la classe des langages ω-réguliers. Pour ces logiques, décider de la satisfaction des formules revient à construire un automate de Büchi acceptant exactement les séquences satisfaisant ces formules.

Dans ce chapitre, nous introduisons la notion d'automate fini sur les mots infinis. Nous présentons en particulier les automates de Büchi et nous montrons comment des propriétés exprimées sous forme de formules LTLP peuvent être représentées par de tels automates.

La construction de l'automate de Büchi est tirée des articles [Wolper83b, Wolper83a, Vardi86]. L'algorithme présenté dans chacun de ces articles est légèrement différent des deux autres. Nous avons adopté celui de [Vardi86] dans lequel l'automate de Büchi possède un seul état initial. Ceci répond mieux à nos besoins pour l'étude de la vérification et de la synthèse des réseaux de Petri aux chapitres 4 et 5.

3.1 Automates finis

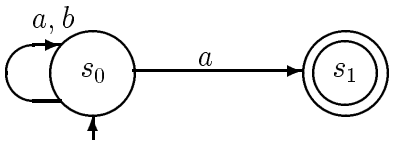
3.1.1 Définition. (Automate fini) [Autebert94] Un automate fini est un 5-uple $\mathcal{A} = (\Sigma, S, \rho, s_0, F)$ où :

- Σ est un alphabet,
- S est un ensemble fini d'états,
- $\rho : S \times \Sigma \to 2^S$ est une fonction décrivant les transitions,
- $s_0 \in S$ est l'état initial et
- $F \subset S$ est l'ensemble des états finaux.

□

À tout automate, on peut associer un graphe $\langle \Sigma, S, \rho \rangle$ où S est l'ensemble des sommets du graphe (représentés par des cercles), Σ l'ensemble des étiquettes d'arcs et ρ l'ensemble des arcs étiquetés. L'état initial est pointé par une petite flèche et les états finaux sont représentés par deux cercles concentriques.

3.1.2 Exemple. Soit l'automate fini $\mathcal{A} = (\Sigma, S, \rho, s_0, F)$ où $\Sigma = \{a, b\}$, $S = \{s_0, s_1\}$, $\rho(s_0, a) = \{s_0, s_1\}$, $\rho(s_0, b) = \{s_0\}$, $\rho(s_1, a) = \rho(s_1, b) = \emptyset$ et $F = \{s_1\}$. Alors on peut représenter $\mathcal{B}$ par le graphe suivant :



□

Appelons H l'ensemble fini des arcs du graphe associé à l'automate :

$$H \quad = \quad \{(s_i, a_j, s_{i+1}) / s_{i+1} \in \rho(s_i, a_j)\}.$$

En considérant l'ensemble H comme un alphabet, on dira qu'un mot w de $H^\star$ est un chemin dans $\mathcal{A}$ qui mène de l'état s_0 à l'état s_n s'il s'écrit :

$$w \quad = \quad (s_0, a_1, s_1)(s_1, a_2, s_2) \cdots (s_{n-1}, a_n, s_n).$$

3.1.3 Définition. (Trace) [Autebert94] On appelle *trace* la fonction $t : H \to \Sigma$ définie par :

$$\forall s \in S : \forall s' \in S : \forall a \in \Sigma : t((s, a, s')) = a.$$

On étend t, d'une façon naturelle, à $t : H^\star \to \Sigma^\star$ par :

$$t((s_0, a_1, s_1)(s_1, a_2, s_2) \cdots (s_{n-1}, a_n, s_n)) \quad = \quad a_1 a_2 \cdots a_n.$$

□

3.1.4 Définition. (Acceptation) [Autebert94] Un mot $u \in \Sigma^\star$ est *accepté* ou (*reconnu*) par un automate fini $\mathcal{A}$ s'il existe un chemin w dans $\mathcal{A}$ qui mène de l'état s_0 à un état s_n de F et qui est tel que $u = t(w)$. On dit aussi que w est une *exécution* de $\mathcal{A}$. L'ensemble de tous les mots acceptés par $\mathcal{A}$, appelé le *langage accepté par* $\mathcal{A}$ et noté $L(\mathcal{A})$, est défini comme suit :

$$L(\mathcal{A}) \quad = \quad \{u \in \Sigma^\star / u \text{ est accepté par } \mathcal{A}\}.$$

□

Le langage accepté par l'automate de l'exemple 3.1.2 est $\{a, b\}^\star a$, i.e. l'ensemble des mots qui commencent par une suite finie (peut-être vide) de a et b, et se termine par un a.

Un automate fini peut être utilisé pour accepter des mots infinis. Il suffit de changer la condition d'acceptation. Nous pouvons citer à titre d'exemple, les trois conditions suivantes [Wolper83b]. Un automate accepte un mot si

1. il accepte certains préfixes par la notion d'acceptation standard des mots finis (définition 3.1.4).
2. il existe une exécution infinie sur le mot.
3. il existe une exécution infinie sur le mot où l'ensemble des états qui sont répétés infiniment souvent satisfait certaines contraintes additionnelles. Par exemple, il contient certains états désignés.

Nous présentons, dans ce qui suit, l'automate de Büchi. Nous montrons le lien entre ce type d'automate et la LTLP. Auparavant, il convient d'étendre la trace t à $t : H^\omega \to \Sigma^\omega$ par :

$$t((s_0, a_1, s_1)(s_1, a_2, s_2)\cdots) \quad = \quad a_1 a_2 \cdots.$$

3.1.5 Définition. (Automate de Büchi) [Büchi62] Un *automate de Büchi* est un 5-uple $\mathcal{B} = (\Sigma, S, \rho, s_0, F)$ avec Σ, S, ρ et s_0 comme dans la définition 3.1.1 et $F \subseteq S$ appelé ensemble d'*états désignés*. Un mot u est accepté si $u \in \Sigma^\omega$ et s'il existe un chemin $w \in H^\omega$ tel que :

- $u = t(w)$ et
- $INF(w) \cap F \neq \emptyset$ où $INF(w)$ représente l'ensemble des états "visités" infiniment souvent en parcourant w; i.e

$$INF(w) \quad = \quad \{s / \forall i \geq 1 : \exists j \geq i : w(j) = (s, a_j, s_j)\}.$$

Le langage accepté par $\mathcal{B}$ est

$$L(\mathcal{B}) \quad = \quad \{u \in \Sigma^\omega / \exists w \in H^\omega : u = t(w) \wedge INF(w) \cap F \neq \emptyset\}.$$

□

3.1.6 Exemple. Considérons l'automate de Büchi de la figure 3.1 où :

- $\Sigma = \{a, b\}$,
- $S = \{s_0, s_1, s_2\}$,
- $\rho(s_0, a) = \{s_1\}$, $\rho(s_0, b) = \emptyset$, $\rho(s_1, a) = \{s_1\}$, $\rho(s_1, b) = \{s_2\}$, $\rho(s_2, a) = \{s_1\}$, $\rho(s_2, b) = \{s_2\}$,
- s_0 est l'état initial,
- $F = \{s_2\}$.

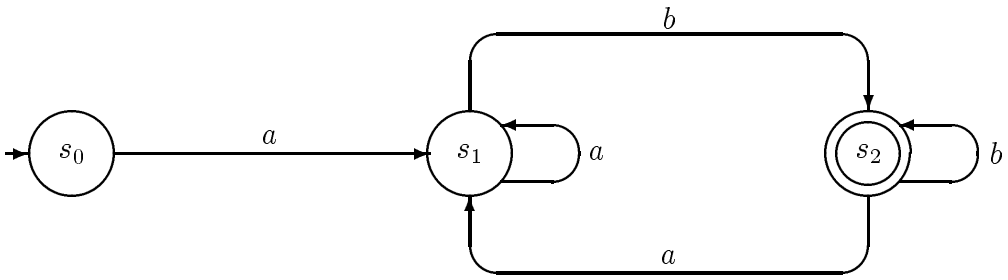


Figure 3.1 : Un automate de Büchi.

Le langage accepté est l'ensemble de tous les mots de la forme $(aa^\star bb^\star)^\omega$. Notons que les mots de la forme $(a^\star bb^\star)a^\omega$ ne sont pas acceptés par cet automate. Intuitivement, un mot est accepté s'il commence par un a et contient infiniment souvent b. □

3.2 Représentation des formules LTLP par des automates de Büchi

Dans cette section, nous établissons une correspondance entre la LTLP et les automates de Büchi. Nous montrons qu'il est possible d'associer à chaque formule f de la logique temporelle, un automate de Büchi acceptant exactement les modèles satisfaisant cette formule.

Dans [Vardi86, Wolper83b], les auteurs ont proposé un algorithme général pour la construction d'un tel automate. Notre contribution s'illustre par l'élaboration d'un exemple assez complexe, chose qui n'est pas faite dans les articles que nous avons consultés. Nous adoptons l'algorithme de construction de [Vardi86] qui offre la particularité d'utiliser un automate de Büchi ayant un seul état initial (et donc correspondant à la définition 3.1.5).

Dans ce qui suit, nous notons $cl(f)$ la clôture de la formule LTLP f. Cet ensemble est constitué de toutes les sous-formules de f et de leur négation. Plus précisément, on définit $cl(f)$ comme suit :

- $f \in cl(f)$,

- $f_1 \wedge f_2 \in cl(f) \rightarrow f_1, f_2 \in cl(f)$,
- $\neg f_1 \in cl(f) \rightarrow f_1 \in cl(f)$,
- $f_1 \in cl(f) \rightarrow \neg f_1 \in cl(f)$,
- $\bigcirc f_1 \in cl(f) \rightarrow f_1 \in cl(f)$,
- $f_1 \ \mathcal{U} f_2 \in cl(f) \rightarrow f_1, f_2 \in cl(f)$.

3.2.1 Théorème. [Vardi86] *Étant donnée une formule LTLP f, on peut construire un automate de Büchi* $\mathcal{B} = (\Sigma, S, \rho, s_0, F)$, *où* $\Sigma = 2^{Prop}$, *tel que* $L(\mathcal{B})$ *est exactement l'ensemble des séquences satisfaisant la formule f.*

Preuve. Pour construire un automate de Büchi acceptant les modèles d'une formule f, nous devons en premier lieu construire un automate sur l'alphabet $2^{cl(f)}$. Cet automate doit reconnaître l'ensemble des mots qui sont des modèles de f. Chaque état de l'automate est étiqueté par des éléments de $cl(f)$ qui sont vrais à cet état.

L'automate de Büchi que nous construisons est une combinaison de deux automates : l'*automate local* et l'*automate de nécessité*. L'automate local garantit qu'il n'y a pas de *propositions incohérentes* dans le modèle et que les opérateurs temporels sont localement satisfaits. Par exemple, en partant de la formule

$$f_1 \mathcal{U} f_2 \quad \leftrightarrow \quad f_2 \vee (f_1 \wedge \bigcirc(f_1 \mathcal{U} f_2)),$$

l'automate local garantit que si $f_1 \mathcal{U} f_2$ est vraie à un état, alors soit f_2 est vraie dans cet état, soit f_1 y est vraie et $f_1 \mathcal{U} f_2$ est vraie au prochain état.

La vérification locale est suffisante sauf pour les formules de nécessité. Ce sont les formules de la forme $f_1 \mathcal{U} f_2$. Le problème avec ces formules est que les conditions locales imposées ne garantissent pas l'accessibilité à un état où f_2 est vraie. Nous confions à l'automate de nécessité le rôle de satisfaire cette nécessité.

Construction de l'automate local.

L'automate local est $\mathcal{B}_L = (2^{cl(f)}, N_L, \rho_L, s_0, N_L)$. L'ensemble des états N_L est formé d'un état initial s_0 et de tous les états s où s est un sous-ensemble de $cl(f)$, maximal et cohérent. En particulier, ils doivent satisfaire les conditions suivantes :

- pour toute formule $f_1 \in cl(f)$, on a $f_1 \in s$ ssi $\neg f_1 \notin s$,
- pour toute formule $f_1 \wedge f_2 \in cl(f)$, on a $f_1 \wedge f_2 \in s$ ssi $f_1 \in s$ et $f_2 \in s$.

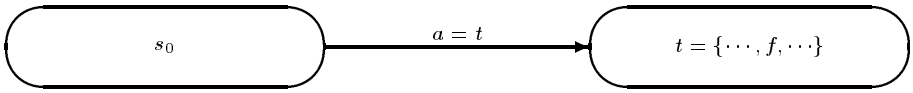
Pour la relation de transition ρ_L, nous avons $t \in \rho_L(s, a)$ ssi $a = t$ (i.e. la transition et l'état d'arrivée ont la même étiquette) et

- soit $s = s_0$ et $f \in t$,
- soit $s \neq s_0$ et
  - pour toute formule $\bigcirc f_1 \in cl(f)$, on a $\bigcirc f_1 \in s \ ssi f_1 \in t$ et

– pour toute formule $f_1 \mathcal{U} f_2 \in cl(f)$, on a $f_1 \mathcal{U} f_2 \in s$ ssi $f_2 \in s$ ou $f_1 \in s$ et $f_1 \mathcal{U} f_2 \in t$.

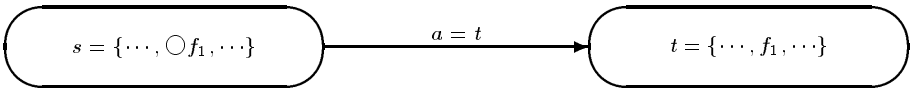
Ainsi, les seules transitions qui peuvent avoir lieu dans l'automate local sont schématisées par ce qui suit :

- si $s = s_0$

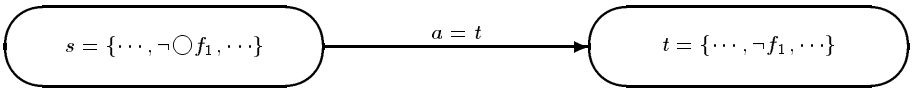


- $s \neq s_0$ et

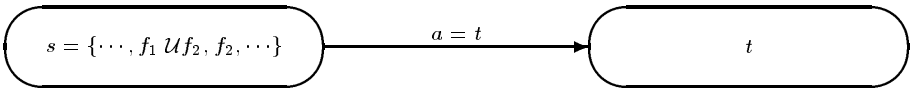
  – pour toute formule $\bigcirc f_1 \in cl(f)$, on a



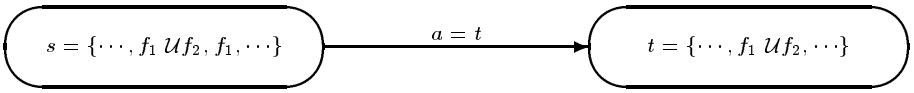
et



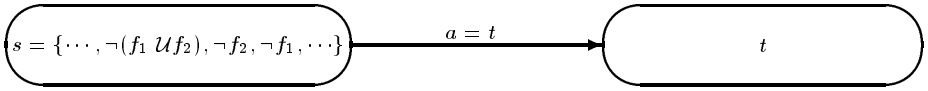
  – pour toute formule $f_1 \mathcal{U} f_2 \in cl(f)$, on a



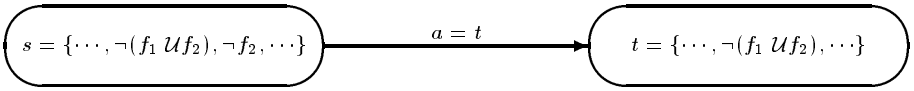
ou



et



ou



L'automate local n'impose aucune condition d'acceptation. Par conséquent l'ensemble des états désignés est N_L.

Construction de l'automate de nécessité

Étant donnée une formule f de la LTPL, nous définissons l'ensemble $e(f)$ comme le sous-ensemble de $cl(f)$ qui contient toutes les formules de la forme $f_1 \mathcal{U} f_2$. L'automate de nécessité est $\mathcal{B}_E = (2^{cl(f)}, 2^{e(f)}, \rho_E, \emptyset, \{\emptyset\})$, où pour la relation de transition ρ_E, on a $t \in \rho_E(s, a)$ ssi $a \in N_L \Leftrightarrow \{s_0\}$ et :

- soit $s = \emptyset$ et pour toute formule $f_1 \mathcal{U} f_2 \in a$, on a $f_1 \mathcal{U} f_2 \in t$ ssi $f_2 \notin a$,
- soit $s \neq \emptyset$ et pour toute formule $f_1 \mathcal{U} f_2 \in s$, on a $f_1 \mathcal{U} f_2 \in t$ ssi $f_2 \notin a$.

Intuitivement, l'automate de nécessité essaie de satisfaire les nécessités dans le modèle, i.e. de garantir à toute formule $f_1 \mathcal{U} f_2$ dans $cl(f)$ d'accéder à un état où la formule f_2 est vraie.

Combinaison des automates

Nous combinons maintenant l'automate local et l'automate de nécessité pour construire l'automate final

$$\mathcal{B}_M \quad = \quad (2^{cl(f)}, N_M, \rho_M, N_{M_0}, F_M).$$

$\mathcal{B}_M$ est obtenu en effectuant le produit cartésien de $\mathcal{B}_L$ et de $\mathcal{B}_E$. L'ensemble des états est

$$N_M \quad = \quad N_L \times 2^{e(f)}.$$

La relation de transitions ρ_M est définie comme suit :

$$(p, q) \in \rho_M((s, t), a) \quad \text{ssi} \quad p \in \rho_L(s, a) \text{ et } q \in \rho_E(t, a).$$

L'état initial est $N_{M_0} = (s_0, \emptyset)$ et l'ensemble des états désignés est $F_M = N_L \times \{\emptyset\}$.

L'automate obtenu accepte des mots sur $2^{cl(f)}$, alors que les modèles de f sont définis par des mots sur 2^{Prop}. La dernière étape de notre construction est d'effectuer la projection de l'automate sur 2^{Prop}. Pour ce faire, il suffit de prendre la projection $b \cap Prop$ de chaque élément $b \in 2^{cl(f)}$. □

3.3 Exemple de construction d'un automate de Büchi

Dans cette section, nous traitons un exemple de construction d'automate de Büchi acceptant les modèles d'une formule LTLP f donnée en entrée. Nous détaillons les différentes étapes de la construction d'une façon claire et nette. L'automate de Büchi que nous allons construire n'est pas réduit, i.e. contient des états et des transitions que nous pouvons éliminer sans changer le langage accepté. Malheureusement, il n'existe pas d'algorithmes de minimisation des automates de Büchi [Howell88]. Nous essayons cependant de donner quelques règles de minimisation que nous appliquerons avec

beaucoup de prudence. Ces règles sont faciles à appliquer sur de petits automates (c'est le cas pour l'exemple à traiter). Mais si tout cela était traité par une machine, nous n'essayerions pas nécessairement de minimiser. Signalons que le langage $L(\mathcal{B}_M)$ accepté par $\mathcal{B}_M$ est donné par

$$L(\mathcal{B}_M) \quad = \quad L(\mathcal{B}_L) \cap L(\mathcal{B}_E)$$

où $L(\mathcal{B}_L)$ et $L(\mathcal{B}_E)$ sont les langages acceptés par $\mathcal{B}_L$ et $\mathcal{B}_E$ respectivement. Ainsi, il nous est possible de réduire les automates $\mathcal{B}_L$ et $\mathcal{B}_E$ sans que cela n'affecte la construction de l'automate $\mathcal{B}_M$. Les types de réductions que nous effectuons dans cet exemple sont les suivants :

- élimination des états qui ne sont pas atteignables à partir de l'état initial;
- élimination des états n'ayant aucune flèche sortante (les états n'ayant pas de successeur). En arrivant à de tels états, l'automate s'arrête, aucune transition ne peut être franchie. La séquence lue est évidement finie et ne peut par conséquent être acceptée par un automate de Büchi;
- élimination des états à partir desquels il n'est plus possible de passer par un état désigné;
- élimination des états à partir desquels les séquences infinies tirées ne passent par aucun état désigné et, que ces mêmes séquences peuvent être tirées à partir d'autres états en passant par au moins un état désigné, i.e. on force l'automate à passer par des états désignés;
- élimination de toutes les transitions arrivant à ou partant d'un état éliminé.

Pour cet exemple, nous allons supposer qu'à chaque instant, une et une seule proposition atomique peut être vraie. Formellement, nous définissons cette exigence de la façon suivante :

3.3.1 Définition. (**Condition d'événement unique**) La condition d'événement unique C_u est donnée par la formule logique temporelle suivante :

$$C_u = \Box((\bigvee_{1\leq i\leq n} p_i) \wedge (\bigwedge_{1\leq i<j\leq n} \neg(p_i \wedge p_j)),$$

où $p_1, \cdots, p_n$ sont toutes des propositions atomiques. □

Pour construire un automate de Büchi $\mathcal{B}_{f'} = (\Sigma, S, \rho, s_0, F)$ capable d'accepter les modèles d'une formule logique temporelle f', nous prenons $\Sigma = 2^{Prop}$ (théorème 3.2.1). Si on exige en plus la condition d'événement unique, on obtient l'automate de Büchi $\mathcal{B}_f = (\Sigma, S, \rho, s_0, F)$, où $f = f' \wedge C_u$, pour lequel nous pouvons changer $\Sigma = 2^{Prop}$ par $\Sigma = Prop$, puisqu'une seule proposition atomique est vraie à un instant donné. Soit

$$f' \quad = \quad \Box(t_1 \rightarrow \bigcirc(t_1 \,\mathcal{U} t_2)) \wedge \Box(t_2 \rightarrow \bigcirc(t_2 \,\mathcal{U} t_1))$$

où t_1 et t_2 sont des propositions atomiques. Posons maintenant

$$f \;=\; f' \wedge C_u$$

et développons f. Nous utilisons les lois de la logique classique et celles présentées à la section 2.4.

$$\begin{aligned}
f \;&=\; f' \wedge C_u \\
&\Leftrightarrow\; \Box\Big(t_1 \rightarrow \bigcirc(t_1 \,\mathcal{U}\, t_2)\Big) \wedge \Box\Big(t_2 \rightarrow \bigcirc(t_2 \,\mathcal{U}\, t_1)\Big) \wedge \Box\Big((t_1 \vee t_2) \wedge \neg(t_1 \wedge t_2)\Big) \\
&\Leftrightarrow\; \Box\Big(\Big(\neg t_1 \vee \bigcirc(t_1 \,\mathcal{U}\, t_2)\Big) \wedge \Big(\neg t_2 \vee \bigcirc(t_2 \,\mathcal{U}\, t_1)\Big) \wedge \Big((t_1 \vee t_2) \wedge (\neg t_1 \vee \neg t_2)\Big)\Big) \\
&\Leftrightarrow\; \Box\Big(\Big(\neg t_1 \vee \bigcirc(t_1 \,\mathcal{U}\, t_2)\Big) \wedge \Big(\neg t_2 \vee \bigcirc(t_2 \,\mathcal{U}\, t_1)\Big) \wedge \Big(t_1 \wedge \neg t_2 \bigvee \neg t_1 \wedge t_2\Big)\Big) \\
&\Leftrightarrow\; \Box\Big(\neg t_1 \wedge \neg t_2 \wedge t_1 \wedge \neg t_2 \\
&\qquad \bigvee \neg t_1 \wedge \neg t_2 \wedge \neg t_1 \wedge t_2 \\
&\qquad \bigvee \neg t_1 \wedge \bigcirc(t_2 \,\mathcal{U}\, t_1) \wedge t_1 \wedge \neg t_2 \\
&\qquad \bigvee \neg t_1 \wedge \bigcirc(t_2 \,\mathcal{U}\, t_1) \wedge \neg t_1 \wedge t_2 \\
&\qquad \bigvee \bigcirc(t_1 \,\mathcal{U}\, t_2) \wedge \neg t_2 \wedge t_1 \wedge \neg t_2 \\
&\qquad \bigvee \bigcirc(t_1 \,\mathcal{U}\, t_2) \wedge \neg t_2 \wedge \neg t_1 \wedge t_2 \\
&\qquad \bigvee \bigcirc(t_1 \,\mathcal{U}\, t_2) \wedge \bigcirc(t_2 \,\mathcal{U}\, t_1) \wedge t_1 \wedge \neg t_2 \\
&\qquad \bigvee \bigcirc(t_1 \,\mathcal{U}\, t_2) \wedge \bigcirc(t_2 \,\mathcal{U}\, t_1) \wedge \neg t_1 \wedge t_2\Big) \\
&\Leftrightarrow\; \Box\Big(\neg t_1 \wedge t_2 \wedge \bigcirc(t_2 \,\mathcal{U}\, t_1) \\
&\qquad \bigvee \neg t_2 \wedge t_1 \wedge \bigcirc(t_1 \,\mathcal{U}\, t_2) \\
&\qquad \bigvee t_1 \wedge \neg t_2 \wedge \bigcirc(t_1 \,\mathcal{U}\, t_2) \wedge \bigcirc(t_2 \,\mathcal{U}\, t_1) \\
&\qquad \bigvee \neg t_1 \wedge t_2 \wedge \bigcirc(t_1 \,\mathcal{U}\, t_2) \wedge \bigcirc(t_2 \,\mathcal{U}\, t_1)\Big) \\
&\Leftrightarrow\; \Box\Big((\neg t_1 \wedge t_2 \wedge \bigcirc(t_2 \,\mathcal{U}\, t_1)) \bigwedge (\mathrm{Vrai} \vee \bigcirc(t_2 \,\mathcal{U}\, t_1)) \\
&\qquad \bigvee (\neg t_2 \wedge t_1 \wedge \bigcirc(t_1 \,\mathcal{U}\, t_2)) \bigwedge (\mathrm{Vrai} \vee \bigcirc(t_2 \,\mathcal{U}\, t_1))\Big) \\
&\Leftrightarrow\; \Box\Big((\neg t_1 \wedge t_2 \wedge \bigcirc(t_2 \,\mathcal{U}\, t_1)) \bigvee (\neg t_2 \wedge t_1 \wedge \bigcirc(t_1 \,\mathcal{U}\, t_2))\Big).
\end{aligned}$$

Sachant que $\Box p \Leftrightarrow \neg(\mathrm{Vrai}\,\mathcal{U}\neg p)$ (voir page 21), nous écrivons, après une application des lois de de Morgan :

$$f \;=\; \neg(\mathrm{Vrai}\,\mathcal{U}(\neg(\neg t_1 \wedge t_2 \wedge \bigcirc(t_2 \,\mathcal{U}\, t_1)) \wedge \neg(\neg t_2 \wedge t_1 \wedge \bigcirc(t_1 \,\mathcal{U}\, t_2)))).$$

Nous posons

$$f \;=\; \neg(\mathrm{Vrai}\,\mathcal{U}(\neg g_1 \wedge \neg g_2))$$

avec

$$\begin{aligned} g_1 &= \neg t_1 \wedge t_2 \wedge \bigcirc(t_2 \mathcal{U} t_1), \\ g_2 &= \neg t_2 \wedge t_1 \wedge \bigcirc(t_1 \mathcal{U} t_2). \end{aligned}$$

Construction de l'automate local

L'ensemble $cl(f)$ (voir page 28) est le suivant :

$$cl(f) = \left\{ \begin{array}{l} f, \text{Vrai}\,\mathcal{U}(\neg g_1 \wedge \neg g_2), g_1 \vee g_2, \neg g_1 \wedge \neg g_2, g_1, \neg g_1, g_2, \neg g_2, \\ t_1, \neg t_1, t_2, \neg t_2, \bigcirc(t_1 \mathcal{U} t_2), \neg\bigcirc(t_1 \mathcal{U} t_2), \bigcirc(t_2 \mathcal{U} t_1), \neg\bigcirc(t_2 \mathcal{U} t_1), \\ t_1 \mathcal{U} t_2, \neg(t_1 \mathcal{U} t_2), t_2 \mathcal{U} t_1, \neg(t_2 \mathcal{U} t_1), \text{Vrai}, \text{Faux} \end{array} \right\}.$$

3.3.2 Remarque. Si $p \mathcal{U} q \in cl(f)$ alors les seuls ensembles qui peuvent être cohérents et contenir seulement p, q et $p \mathcal{U} q$ ou leur négation sont

$$\begin{array}{lll} \{p \mathcal{U} q, q, p\}, & \{\neg(p \mathcal{U} q), \neg q, p\}, & \{p \mathcal{U} q, \neg q, p\}, \\ \{p \mathcal{U} q, q, \neg p\}, & \{\neg(p \mathcal{U} q), \neg q, \neg p\}. & \end{array}$$

□

Pour déterminer l'ensemble des états N_L, i.e. tous les ensembles maximaux et cohérents de $cl(f)$, nous allons construire deux arbres, l'un ayant comme racine la formule f et l'autre, $\neg f$. Les noeuds de chaque arbre sont des formules de $cl(f)$. Rappelons que chaque noeud d'un arbre peut être un successeur (fils) ou le prédécesseur (parent) d'autres noeuds. La racine est le seul noeud qui n'a pas de prédécesseur. Les noeuds qui n'ont pas de successeurs sont appelés des feuilles. On appelle un *chemin* dans l'arbre toute suite de noeuds $n_0, n_1, \cdots, n_k$ tels que :

- n_0 est la racine,
- n_k est une feuille et
- $\forall i : 0 < i < k : n_i$ est un successeur de n_{i-1} et le prédécesseur de n_{i+1}.

Un arbre maximal tel que l'ensemble des formules d'un chemin soit cohérent est construit par la suite. Nous nous contentons de ne présenter que l'arbre ayant f comme racine (voir figure 3.2). Nous montrons que les états contenant la formule $\neg f$ ne sont pas accessibles à partir de l'état initial et que, par conséquent, nous pouvons les éliminer. Nous ne donnerons pas un *algorithme de construction* mais nous détaillerons tout simplement les différents niveaux de construction. La racine a le niveau le plus bas.

niveau 0

On prend comme racine $f = \neg(\text{Vrai}\mathcal{U}(\neg g_1 \wedge \neg g_2))$.

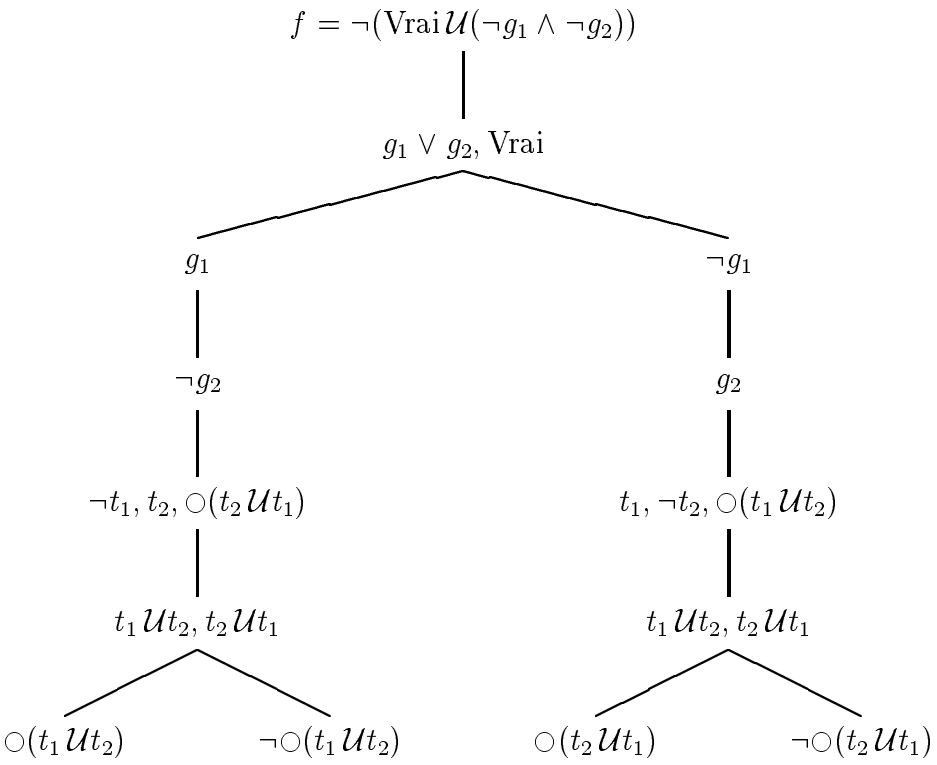


Figure 3.2 : L'arbre maximal et cohérent contenant f.

niveau 1

D'après la remarque 3.3.2, les ensembles

$$\begin{aligned} E_1 &= \{\neg(\text{Vrai}\,\mathcal{U}(\neg g_1 \wedge \neg g_2)), g_1 \vee g_2, \text{Vrai}\} \text{ et} \\ E_2 &= \{\neg(\text{Vrai}\,\mathcal{U}(\neg g_1 \wedge \neg g_2)), g_1 \vee g_2, \text{Faux}\} \end{aligned}$$

peuvent être cohérents. Il est clair que l'ensemble E_2 ne l'est pas, suite à la présence de la valeur Faux dans l'ensemble. Le seul ensemble cohérent est donc E_1. On crée alors un successeur à la racine, ayant comme étiquette $g_1 \vee g_2$,Vrai.

niveau 2

À ce niveau, nous créons deux successeurs, l'un vérifiant g_1, l'autre sa négation.

niveau 3

Pour le noeud de gauche, on doit normalement créer deux noeuds, l'un contenant g_2, le second, $\neg g_2$. Mais $g_1 \wedge g_2 = \text{Faux}$, par définition de g_1 et g_2; l'ensemble $\{f, g_1 \vee g_2, \text{Vrai}, g_1, g_2\}$ est donc incohérent. Ceci explique la création d'un seul successeur. Pour le noeud de droite, il faut créer un seul successeur ayant g_2 comme étiquette; en effet, l'ensemble $\{g_1 \vee g_2, \neg g_1, \neg g_2\}$ est incohérent. Désormais, nous ne nous intéressons qu'à la branche de gauche. La branche de droite est traitée d'une manière symétrique.

niveau 4

En partant de la formule

$$g_1 = \neg t_1 \wedge t_2 \wedge \bigcirc(t_2\,\mathcal{U}\,t_1),$$

nous créons un noeud ayant comme étiquette $\neg t_1, t_2, \bigcirc(t_2\,\mathcal{U}\,t_1)$.

niveau 5

D'après les deux implications suivantes

$$\begin{aligned} t_2 &\Rightarrow t_1\,\mathcal{U}\,t_2 \text{ et} \\ t_2 \wedge \bigcirc(t_2\,\mathcal{U}\,t_1) &\Rightarrow t_2\,\mathcal{U}\,t_1, \end{aligned}$$

nous créons un noeud ayant comme étiquette $t_2\,\mathcal{U}\,t_1, t_1\,\mathcal{U}\,t_2$.

niveau 6

Jusqu'ici tous les noeuds ont été créés suite à des conséquences logiques des noeuds précédents. L'ensemble des noeuds de la branche constitue un ensemble cohérent mais non maximal (voir $cl(f)$: il manque $\bigcirc(t_1\,\mathcal{U}\,t_2)$ ou sa négation). Il est possible dans cet exemple de créer deux branches, l'une ayant comme étiquette $\bigcirc(t_1\,\mathcal{U}\,t_2)$ et l'autre $\neg\bigcirc(t_1\,\mathcal{U}\,t_2)$, sans que les ensembles de chemins soient incohérents. Par exemple, pour

la première branche, $\bigcirc(t_1 \mathcal{U} t_2)$ peut avoir lieu puisque la seule formule qui l'empêche de se produire, c'est-à-dire $\Box t_1$, n'appartient pas à l'ensemble des noeuds.

Finalement, les seuls ensembles maximaux et cohérents de formules de $cl(f)$ contenant f sont les suivants (ce sont les quatre chemins de l'arbre) :

$$\begin{aligned}
l_1 &= \{f, g_1 \vee g_2, \text{Vrai}, g_1, \neg g_2, \neg t_1, t_2, \bigcirc(t_2 \mathcal{U} t_1), t_2 \mathcal{U} t_1, t_1 \mathcal{U} t_2, \bigcirc(t_1 \mathcal{U} t_2)\},\\
l_2 &= \{f, g_1 \vee g_2, \text{Vrai}, g_1, \neg g_2, \neg t_1, t_2, \bigcirc(t_2 \mathcal{U} t_1), t_2 \mathcal{U} t_1, t_1 \mathcal{U} t_2, \neg\bigcirc(t_1 \mathcal{U} t_2)\},\\
l_3 &= \{f, g_2 \vee g_1, \text{Vrai}, g_2, \neg g_1, \neg t_2, t_1, \bigcirc(t_1 \mathcal{U} t_2), t_1 \mathcal{U} t_2, t_2 \mathcal{U} t_1, \bigcirc(t_2 \mathcal{U} t_1)\},\\
l_4 &= \{f, g_2 \vee g_1, \text{Vrai}, g_2, \neg g_1, \neg t_2, t_1, \bigcirc(t_1 \mathcal{U} t_2), t_1 \mathcal{U} t_2, t_2 \mathcal{U} t_1, \neg\bigcirc(t_2 \mathcal{U} t_1)\}.
\end{aligned}$$

Nous obtenons alors $\{s_0, l_1, l_2, l_3, l_4\} \subset N_L$. D'une manière similaire, il serait possible de calculer l'arbre maximal cohérent dont la racine est $\neg f$. Ceci donnerait

$$N_L = \{s_0, l_1, l_2, l_3, l_4, l'_1, l'_2, \cdots\}$$

où $\neg f \in l'_i$ pour tout i. Cependant, nous verrons plus tard qu'en construisant l'automate local, il n'est pas possible d'atteindre aucun des l'_i à partir de s_0. C'est pour cette raison qu'ils ne sont pas calculés. Nous calculons dans ce qui suit les images de s_0, l_1, l_2, l_3 et l_4 par la relation ρ_L et nous montrons que $N_L = \{s_0, l_1, l_2, l_3, l_4\}$.

- Pour $s = s_0$

$$\begin{aligned}
\rho_L(s_0, l_1) &= \{l_1\}, \quad \rho_L(s_0, l_2) = \{l_2\},\\
\rho_L(s_0, l_3) &= \{l_3\}, \quad \rho_L(s_0, l_4) = \{l_4\}.
\end{aligned}$$

- Pour $s = l_1$
  - $\bigcirc(t_1 \mathcal{U} t_2) \in cl(f)$

    On doit avoir $\bigcirc(t_1 \mathcal{U} t_2) \in l_1$ ssi $t_1 \mathcal{U} t_2 \in t$. Puisque $\bigcirc(t_1 \mathcal{U} t_2) \in l_1$, on en déduit $t_1 \mathcal{U} t_2 \in t$.

  - $\bigcirc(t_2 \mathcal{U} t_1) \in cl(f)$

    On doit avoir $\bigcirc(t_2 \mathcal{U} t_1) \in l_1$ ssi $t_2 \mathcal{U} t_1 \in t$. Puisque $\bigcirc(t_2 \mathcal{U} t_1) \in l_1$, on en déduit $t_2 \mathcal{U} t_1 \in t$.

  - $t_1 \mathcal{U} t_2 \in cl(f)$

    On doit avoir $t_1 \mathcal{U} t_2 \in l_1$ ssi $t_2 \in l_1$, ou $t_1 \in l_1$ et $t_1 \mathcal{U} t_2 \in t$. C'est le cas, puisque $t_1 \mathcal{U} t_2 \in l_1$ et $t_2 \in l_1$.

  - $t_2 \mathcal{U} t_1 \in cl(f)$

    On doit avoir $t_2 \mathcal{U} t_1 \in l_1$ ssi $t_1 \in l_1$, ou $t_2 \in l_1$ et $t_2 \mathcal{U} t_1 \in t$. On en déduit $t_2 \mathcal{U} t_1 \in t$, car $t_2 \mathcal{U} t_1 \in l_1, t_1 \notin l_1$ et $t_2 \in l_1$.

  - $\text{Vrai} \mathcal{U} (\neg g_1 \wedge \neg g_2) \in cl(f)$ (i.e. $\neg f$)

On doit avoir Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in l_1$ ssi $\neg g_1 \wedge \neg g_2 \in l_1$, ou Vrai $\in s$ et Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$. On en déduit Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \notin t$, i.e. $\neg f \notin t$, ou encore, de manière équivalente, $f \in t$.

Pour résumer, on doit avoir $t_1 \mathcal{U} t_2 \in t$, $t_2 \mathcal{U} t_1 \in t$ et $f \in t$, i.e.

$$t \in \{l_1, l_2, l_3, l_4\}.$$

Mieux encore,

$$\begin{array}{rclrcl} \rho_L(l_1, l_1) & = & \{l_1\}, & \rho_L(l_1, l_2) & = & \{l_2\}, \\ \rho_L(l_1, l_3) & = & \{l_3\}, & \rho_L(l_1, l_4) & = & \{l_4\}. \end{array}$$

À partir d'ici, nous donnons moins de détails dans l'application des instructions de la page 29.

- Pour $s = l_2$
  - $\bigcirc(t_1 \mathcal{U} t_2) \in cl(f)$

    On doit avoir $t_1 \mathcal{U} t_2 \notin t$.
  - $\bigcirc(t_2 \mathcal{U} t_1) \in cl(f)$

    On doit avoir $t_2 \mathcal{U} t_1 \in t$.
  - $t_1 \mathcal{U} t_2 \in cl(f)$

    Aucune exigence (les conditions de la page 29 sont satisfaites).
  - $t_2 \mathcal{U} t_1 \in cl(f)$

    On doit avoir $t_2 \mathcal{U} t_1 \in t$.
  - Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in cl(f)$ (i.e. $\neg f$)

    On doit avoir Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2)) \notin t$, i.e. $f \in t$.

  Pour résumer, on doit avoir $t_1 \mathcal{U} t_2 \notin t$, $t_2 \mathcal{U} t_1 \in t$ et $f \in t$. Puisque $f \in t$, on doit avoir

  $$t \in \{l_1, l_2, l_3, l_4\}.$$

  Comme aucun $t \in \{l_1, l_2, l_3, l_4\}$ ne peut satisfaire toutes ces conditions, $\rho_L(l_2, a) = \emptyset$ pour tout $a \in \{l_1, l_2, l_3, l_4\}$.

- Pour $s = l_3$
  - De la même manière, on peut facilement vérifier que pour $s = l_3$, on a

    $$\begin{array}{rclrcl} \rho_L(l_3, l_1) & = & \{l_1\}, & \rho_L(l_3, l_2) & = & \{l_2\}, \\ \rho_L(l_3, l_3) & = & \{l_3\}, & \rho_L(l_3, l_4) & = & \{l_4\}. \end{array}$$

- Pour $s = l_4$
  - De la même manière, on peut facilement vérifier que pour $s = l_4$, on a $\rho_L(l_4, a) = \emptyset$ pour tout $a \in \{l_1, l_2, l_3, l_4\}$.

3.3.3 Remarque. À partir de l'état initial s_0, on peut atteindre les états l_1, l_2, l_3 et l_4. De ces derniers, on ne peut jamais accéder à un état contenant $\neg f$. Ainsi, tout état contenant $\neg f$ n'est pas accessible à partir de l'état initial. Par conséquent, l'ensemble des états peut être réduit à :

$$N_L \quad = \quad \{s_0, l_1, l_2, l_3, l_4\}.$$

□

La relation ρ_L est donnée par le tableau suivant :

ρ_L	s_0	l_1	l_2	l_3	l_4
s_0		l_1	l_2	l_3	l_4
l_1		l_1	l_2	l_3	l_4
l_2					
l_3		l_1	l_2	l_3	l_4
l_4					

Les états s_0, l_1, l_2, l_3 et l_4 de la colonne gauche du tableau représentent les états de départ des transitions alors que ceux de la ligne d'en haut représentent les états d'arrivée. S'il y a une transition d'un état s vers un état t alors on inscrit un t à la case (s, t) ayant comme ligne s et t comme colonne (puisque l'étiquette d'un arc est identique à celle de l'état d'arrivée). Par exemple, $(s_0, l_1) = l_1$ signifie que $l_1 \in \rho_L(s_0, l_1)$. Si on n'inscrit rien à la case (s, t), c'est qu'il n'y a pas de transition allant de s vers t. Par exemple, les cases de la ligne l_2 sont vides car $\rho_L(l_2, a) = \emptyset$ pour tout $a \in N_L$.

Remarquons qu'à partir des états l_2 et l_4 on ne peut franchir aucune transition. Par conséquent, toute séquence qui passe par l_2 ou l_4 est finie. Or, les séquences acceptées par un automate de Büchi sont infinies. Suite à ce résultat, nous pouvons éliminer, sans nuire à la construction de l'automate final, les états l_2 et l_4 ainsi que les transitions ayant ces derniers comme états d'arrivée, d'où N_L se réduit à $\{s_0, l_1, l_3\}$. La relation ρ_L après réduction est donnée par le tableau suivant :

ρ_L	s_0	l_1	l_3
s_0		l_1	l_3
l_1		l_1	l_3
l_3		l_1	l_3

L'automate local est donné par la figure 3.3.

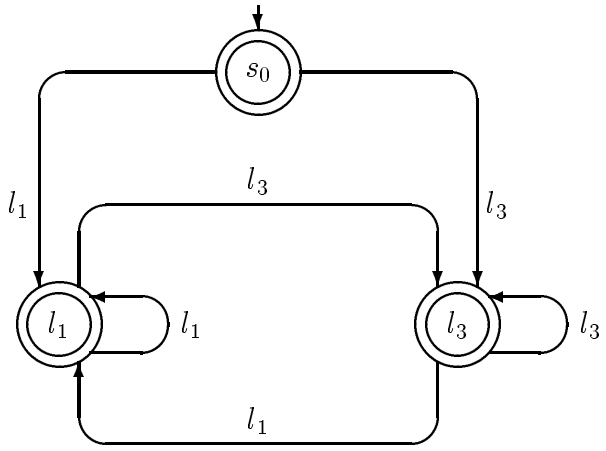


Figure 3.3 : L'automate local

Construction de l'automate de nécessité

Les ensembles $e(f)$ et $2^{e(f)}$ sont définis comme suit :

$$e(f) \;=\; \{t_1\,\mathcal{U}t_2, t_2\,\mathcal{U}t_1, \mathrm{Vrai}\,\mathcal{U}(\neg g_1 \wedge \neg g_2)\};$$

posons

$$\begin{array}{llllllllll} e_0 & = & \emptyset, & e_1 & = & \{t_1\,\mathcal{U}t_2\}, & e_2 & = & \{t_2\,\mathcal{U}t_1\}, & e_3 \;=\; \{\neg f\}, \\ e_4 & = & e_1 \cup e_2, & e_5 & = & e_1 \cup e_3, & e_6 & = & e_2 \cup e_3, & e_7 \;=\; e_4 \cup e_3, \end{array}$$

d'où

$$2^{e(f)} \;=\; \{e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7\}.$$

Construisons maintenant la relation ρ_E. Rappelons tout d'abord que $t \in \rho_E(s,a)$ seulement si $a \in N_L \Leftrightarrow \{s_0\}$ (voir page 31).

- Pour $s = e_0$

  1. pour $a = l_1$
     - $t_1\,\mathcal{U}t_2 \in l_1$

       On doit avoir $t_1\,\mathcal{U}t_2 \in t$ ssi $t_2 \notin l_1$. Puisque $t_2 \in l_1$ on a $t_1\,\mathcal{U}t_2 \notin t$.
     - $t_2\,\mathcal{U}t_1 \in l_1$

       De même, on déduit $t_2\,\mathcal{U}t_1 \in t$.

     Par conséquent,

     $$\rho_E(e_0, l_1) \;=\; \{e_2, e_6\}.$$

2. Pour $a = l_3$
   - $t_1 \mathcal{U} t_2 \in l_3$

     On doit avoir $t_1 \mathcal{U} t_2 \in t$ ssi $t_2 \notin l_3$. Donc $t_1 \mathcal{U} t_2 \in t$.
   - $t_2 \mathcal{U} t_1 \in l_3$

     On doit avoir $t_2 \mathcal{U} t_1 \in t$ ssi $t_1 \notin l_3$. Donc $t_2 \mathcal{U} t_1 \notin t$.

   Par conséquent,

   $$\rho_E(e_0, l_3) = \{e_1, e_5\}.$$

- Pour $s = e_1$

1. Pour $a = l_1$
   - $t_1 \mathcal{U} t_2 \in e_1$

     On doit avoir $t_1 \mathcal{U} t_2 \in t$ ssi $t_2 \notin l_1$. Donc $t_1 \mathcal{U} t_2 \notin t$.

   Par conséquent,

   $$\rho_E(e_1, l_1) = \{e_0, e_2, e_3, e_6\}.$$

2. pour $a = l_3$
   - $t_1 \mathcal{U} t_2 \in e_1$

     On doit avoir $t_1 \mathcal{U} t_2 \in t$ ssi $t_2 \notin l_3$. Donc $t_1 \mathcal{U} t_2 \in t$.

   Par conséquent,

   $$\rho_E(e_1, l_3) = \{e_1, e_4, e_5, e_7\}.$$

- Pour $s = e_2$

1. Pour $a = l_1$
   - $t_2 \mathcal{U} t_1 \in e_2$

     On doit avoir $t_2 \mathcal{U} t_1 \in t$ ssi $t_1 \notin l_1$. Donc $t_2 \mathcal{U} t_1 \in t$.

   Par conséquent,

   $$\rho_E(e_2, l_1) = \{e_2, e_4, e_6, e_7\}.$$

2. pour $a = l_3$
   - $t_2 \mathcal{U} t_1 \in e_2$

     On doit avoir $t_2 \mathcal{U} t_1 \in t$ ssi $t_1 \notin l_3$. Donc $t_2 \mathcal{U} t_1 \notin t$.

   Par conséquent,

   $$\rho_E(e_2, l_3) = \{e_0, e_1, e_3, e_5\}.$$

- Pour $s = e_3$

1. pour $a = l_1$
   - Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in e_3$

On doit avoir Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$ ssi $\neg g_1 \wedge \neg g_2 \notin l_1$. Donc Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$.

Par conséquent,

$$\rho_E(e_3, l_1) = \{e_3, e_5, e_6, e_7\}.$$

2. pour $a = l_3$
   - Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in e_3$

     On doit avoir Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$ ssi $\neg g_1 \wedge \neg g_2 \notin l_3$. Donc Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$.

   Par conséquent,

   $$\rho_E(e_3, l_3) = \{e_3, e_5, e_6, e_7\}.$$

- Pour $s = e_4$

1. pour $a = l_1$
   - $t_1 \mathcal{U} t_2 \in e_4$

     On doit avoir $t_1 \mathcal{U} t_2 \in t$ ssi $t_2 \notin l_1$. Donc $t_1 \mathcal{U} t_2 \notin t$.
   - $t_2 \mathcal{U} t_1 \in e_4$

     On doit avoir $t_2 \mathcal{U} t_1 \in t$ ssi $t_1 \notin l_1$. Donc $t_2 \mathcal{U} t_1 \in t$.

   Par conséquent,

   $$\rho_E(e_4, l_1) = \{e_2, e_6\}.$$

2. pour $a = l_3$
   - $t_1 \mathcal{U} t_2 \in e_4$

     On doit avoir $t_1 \mathcal{U} t_2 \in t$ ssi $t_2 \notin l_3$. Donc $t_1 \mathcal{U} t_2 \in t$.
   - $t_2 \mathcal{U} t_1 \in e_4$

     On doit avoir $t_2 \mathcal{U} t_1 \in t$ ssi $t_1 \in l_3$. Donc $t_2 \mathcal{U} t_1 \notin t$.

   Par conséquent,

   $$\rho_E(e_4, l_3) = \{e_1, e_5\}.$$

- Pour $s = e_5$

1. pour $a = l_1$
   - $t_1 \mathcal{U} t_2 \in e_5$

     On doit avoir $t_1 \mathcal{U} t_2 \in t$ ssi $t_2 \in l_1$. Donc $t_1 \mathcal{U} t_2 \notin t$.
   - Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in e_5$

     On doit avoir Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$ ssi $\neg g_1 \wedge \neg g_2 \notin l_1$. Donc Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$.

   Par conséquent,

   $$\rho_E(e_5, l_1) = \{e_3, e_6\}.$$

2. pour $a = l_3$
   - $t_1 \mathcal{U} t_2 \in e_5$
     On doit avoir $t_1 \mathcal{U} t_2 \in t$ ssi $t_2 \notin l_3$. Donc $t_1 \mathcal{U} t_2 \in t$.
   - Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in e_5$
     On doit avoir Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$ ssi $\neg g_1 \wedge \neg g_2 \notin l_3$. Donc Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$.

   Par conséquent,

   $$\rho_E(e_5, l_3) \quad = \quad \{e_5, e_7\}.$$

- Pour $s = e_6$

1. pour $a = l_1$
   - $t_2 \mathcal{U} t_1 \in e_6$
     On doit avoir $t_2 \mathcal{U} t_1 \in t$ ssi $t_1 \notin l_1$. Donc $t_2 \mathcal{U} t_1 \in t$.
   - Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in e_6$
     On doit avoir Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$ ssi $\neg g_1 \wedge \neg g_2 \notin l_1$. Donc Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$.

   Par conséquent,

   $$\rho_E(e_6, l_1) \quad = \quad \{e_6, e_7\}.$$

2. pour $a = l_3$
   - $t_2 \mathcal{U} t_1 \in e_6$
     On doit avoir $t_2 \mathcal{U} t_1 \in t$ ssi $t_1 \notin l_3$. Donc $t_2 \mathcal{U} t_1 \in t$.
   - Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in e_6$
     On doit avoir Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$ ssi $\neg g_1 \wedge \neg g_2 \notin l_3$. Donc Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$.

   Par conséquent,

   $$\rho_E(e_6, l_3) \quad = \quad \{e_3, e_5\}.$$

- Pour $s = e_7$

1. pour $a = l_1$
   - $t_1 \mathcal{U} t_2 \in e_7$
     On doit avoir $t_1 \mathcal{U} t_2 \in t$ ssi $t_2 \notin l_1$. Donc $t_1 \mathcal{U} t_2 \notin t$.
   - $t_2 \mathcal{U} t_1 \in e_7$
     On doit avoir $t_2 \mathcal{U} t_1 \in t$ ssi $t_1 \notin l_1$. Donc $t_2 \mathcal{U} t_1 \in t$.
   - Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in e_7$
     On doit avoir Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$ ssi $\neg g_1 \wedge \neg g_2 \notin l_1$. Donc Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$

   Par conséquent,

   $$\rho_E(e_7, l_1) \quad = \quad \{e_6\}.$$

2. pour $a = l_3$
   - $t_1 \mathcal{U} t_2 \in e_7$

     On doit avoir $t_1 \mathcal{U} t_2 \in t$ ssi $t_2 \notin l_3$. Donc $t_1 \mathcal{U} t_2 \in t$.
   - $t_2 \mathcal{U} t_1 \in e_7$

     On doit avoir $t_2 \mathcal{U} t_1 \in t$ ssi $t_1 \notin l_3$. Donc $t_2 \mathcal{U} t_1 \notin t$.
   - Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in e_7$

     On doit avoir Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$ ssi $\neg g_1 \wedge \neg g_2 \notin l_3$. Donc Vrai $\mathcal{U}(\neg g_1 \wedge \neg g_2) \in t$.

   Par conséquent,

   $$\rho_E(e_7, l_3) \quad = \quad \{e_5\}.$$

Il convient à présent de représenter la relation ρ_E par le tableau suivant :

ρ_E	e_0	e_1	e_2	e_4	e_3	e_5	e_6	e_7
e_0		l_3	l_1			l_3	l_1	
e_1	l_1	l_3	l_1	l_3	l_1	l_3	l_1	l_3
e_2	l_3	l_3	l_1	l_1	l_3	l_3	l_1	l_1
e_4		l_3	l_1			l_3	l_1	
e_3					l_1, l_3	l_1, l_3	l_1, l_3	l_1, l_3
e_5					l_1	l_3	l_1	l_3
e_6					l_3	l_3	l_1	l_1
e_7						l_3	l_1	

On inscrit a dans la case (e_i, e_j) si $e_j \in \rho_E(e_i, a)$; par exemple, $(e_2, e_1) = l_3$ signifie que $e_1 \in \rho_E(e_2, l_3)$.

Rappelons qu'une suite $u \in \Sigma^\omega$ est acceptée par un automate de Büchi si on peut passer infiniment souvent par certains états désignés. Ce qui signifie que l'automate doit avoir des cycles accessibles à partir de l'état initial et contenant des états désignés. Dans notre exemple, l'automate de Büchi qu'on veut construire est la combinaison de deux automates. Ces derniers doivent aussi avoir des cycles qui passent par les états désignés. Or, on remarque que :

- L'unique état désigné e_0 (voir page 31) est accessible seulement à partir des états e_0, e_1, e_2 et e_4.
- À partir d'un état $e_i \in \{e_3, e_5, e_6, e_7\}$, il n'est plus possible de retourner aux états e_0, e_1, e_2 et e_4. Par conséquent, il n'est plus possible de passer par l'état désigné.

Suite à ces deux remarques, nous pouvons éliminer, afin de réduire l'automate de nécessité, les états e_3, e_5, e_6 et e_7 ainsi que les transitions ayant ces derniers comme état de départ ou d'arrivée. Le nouvel ensemble d'états est composé alors de e_0, e_1, e_2 et e_4. La relation ρ_E est maintenant donnée par le tableau suivant :

ρ_E	e_0	e_1	e_2	e_4
e_0		l_3	l_1	
e_1	l_1	l_3	l_1	l_3
e_2	l_3	l_3	l_1	l_1
e_4		l_3	l_1	

Combinaison des automates

La combinaison de l'automate local avec celui de nécessité donne un troisième automate dont :

- l'état initial est (s_0, e_0),
- les états désignés sont (s_0, e_0), (l_3, e_0) et (l_1, e_0) et
- la relation de transition ρ est représentée par le tableau ci-dessous (seuls l'état initial et les états qui sont la destination d'une transition sont inclus) :

ρ	(s_0, e_0)	(l_1, e_0)	(l_1, e_2)	(l_1, e_4)	(l_3, e_0)	(l_3, e_1)	(l_3, e_4)
(s_0, e_0)			l_1			l_3	
(l_1, e_0)			l_1			l_3	
(l_1, e_2)			l_1	l_1	l_3	l_3	
(l_1, e_4)			l_1			l_3	
(l_3, e_0)			l_1			l_3	
(l_3, e_1)		l_1	l_1			l_3	l_3
(l_3, e_4)			l_1			l_3	

La réduction de cet automate est faite de trois manières. Une par élimination d'états, l'autre par combinaison d'états et la troisième par élimination de transitions. En enlevant un état ou une transition, il faut s'assurer de ne pas réduire le langage de l'automate.

1. Nous pouvons voir facilement que l'état (l_3, e_4) n'est accessible qu'à partir de l'état (l_3, e_1). De plus, les états accessibles à partir de l'état (l_3, e_1) via l'état (l_3, e_4) peuvent aussi être atteints sans passer par l'état (l_3, e_1) et ce, avec les mêmes étiquettes. Par exemple, les transitions $((l_3, e_1), l_3, (l_3, e_4))$, $((l_3, e_4), l_1, (l_1, e_2))$ peuvent être remplacées par $((l_3, e_1), l_3, (l_3, e_1))$, $((l_3, e_1), l_1, (l_1, e_2))$. Nous pouvons donc éliminer l'état (l_3, e_4) ainsi que les transitions qui ont cet état comme état de départ ou d'arrivée. D'une manière similaire, on peut éliminer l'état (l_1, e_4) qui n'est accessible que par l'état (l_1, e_2).

2. On peut combiner les deux états désignés (l_3, e_0) et (l_1, e_0). En effet, ce sont deux états désignés et ils se comportent tous deux de la même façon (les transitions qui quittent ces états ont les mêmes destinations). Une transition qui entre dans l'un de ces états peut donc tout aussi bien entrer dans l'autre.

3. Nous éliminons les deux transitions entre les états (l_3, e_1) et (l_1, e_2). Il suffit de voir que toute séquence infinie qui passe uniquement par ces deux états est de la forme $(l_3^\star l_1^\star)^\omega$. C'est une séquence qui, normalement, devrait être acceptée par l'automate (car elle satisfait la formule). Mais elle ne l'est pas parce qu'elle ne passe jamais par l'état désigné $(l_3, e_0) = (l_1, e_0)$. Cette même séquence, i.e. $(l_3^\star l_1^\star)^\omega$, est acceptée par l'automate si on passe par les séquences reliant (l_3, e_1) et (l_1, e_2) à l'état désigné (l_3, e_0). On remarque qu'en enlevant ces deux transitions l'automate peut quand même toujours faire des transitions étiquetées par l_1 ou par l_3. L'effet du retrait des transitions est de forcer l'automate à passer le plus souvent par l'état désigné.

Finalement, en posant :

$$s_0 = (s_0, e_0), \quad s_1 = (l_3, e_1), \quad s_2 = (l_1, e_2), \quad s_3 = (l_3, e_0) = (l_1, e_0)$$

et en faisant la projection sur $\{t_1, t_2\}$ (voir page 31) on obtient la relation de transition ρ suivante :

ρ	s_0	s_1	s_2	s_3
s_0		t_1	t_2	
s_1		t_1		t_2
s_2			t_2	t_1
s_3		t_1	t_2	

et l'automate final de la figure 3.4.

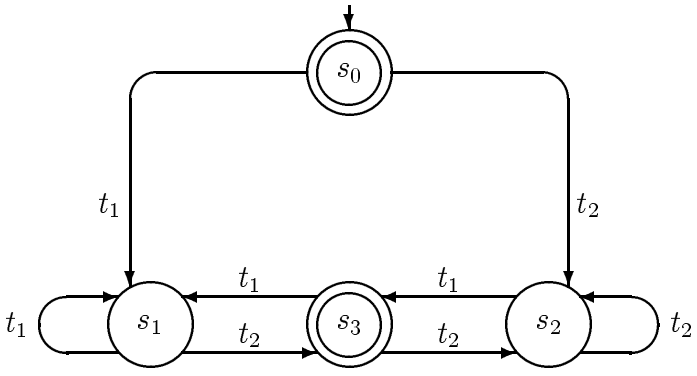


Figure 3.4 : L'automate final

Avant de terminer ce chapitre, nous présentons une définition et un lemme trivial dont nous ne donnons pas la preuve et qui nous sera utile dans les chapitres à venir.

3.3.4 Définition. $L_s(f)$ est un langage infini généré à partir d'une formule LTLP f sous la condition d'événement unique ssi $L_s(f) = L(\mathcal{B}_{f'})$ où $f' = f \wedge C_u$ et *Prop* est l'alphabet de $\mathcal{B}_{f'}$. □

3.3.5 Lemme. *Étant donnée une formule LTLP f, $\overline{L_s(f)} = L_s(\neg f)$, où $\overline{L_s(f)} = Prop^\omega \Leftrightarrow L_s(f)$ est le complément du langage $L_s(f)$.* □

Chapitre 4

Vérification des réseaux de Petri.

Dans ce chapitre, nous montrons comment combiner les réseaux de Petri avec la logique temporelle. Il suffit de savoir à quoi correspond une proposition atomique dans un réseau. Nous présentons quelques-unes de ces correspondances et nous discutons de certains travaux qui ont été effectués sur ces dernières.

Par la suite, nous présentons une méthode qui permet de décider si un réseau de Petri R donné vérifie une propriété f écrite en logique temporelle. Cette méthode est basée sur la combinaison du graphe de couverture du réseau R et de l'automate de Büchi acceptant la formule f.

Si le RDP satisfait la spécification f, il est même possible d'extraire une séquence infinie de transitions licite et satisfaisant f.

Des propriétés comme l'exclusion mutuelle peuvent être prouvées si nous arrivons à trouver la spécification f correspondante. D'autres propriétés, comme la vivacité, sont difficiles à prouver, vu qu'elles sont équivalentes au problème d'accessibilité ($O(exp)$ en espace mémoire). Bien que ce dernier soit décidable, la complexité de l'algorithme est très élevée. Mentionnons que le problème de l'accessibilité consiste à chercher si une configuration du système peut être obtenue et que le problème de la vivacité consiste à vérifier qu'une transition n'est jamais définitivement impossible.

Deux exemples sont traités. Dans le premier, nous cherchons une séquence de transitions infinie et licite dans un RDP satisfaisant une spécification f. Le deuxième traite un problème de décidabilité de l'exclusion mutuelle dans un RDP.

4.1 Combinaison des réseaux de Petri et de la logique temporelle

Il existe plusieurs manières de combiner un réseau de Petri avec la logique temporelle. Le point clé dans la combinaison est de savoir à quoi correspond une proposition atomique de la logique temporelle dans un réseau de Petri. Nous présentons ci-dessous certaines correspondances entre les propositions atomiques de la LTLP et les propriétés des réseaux de Petri :

(a) Une proposition atomique $\mathbf{p}$ est vraie ssi la place p a au moins un jeton.

(b) Une proposition atomique $\mathbf{ge}(\mathbf{p}, \mathbf{c})$ est vraie ssi la place p a au moins c jetons.

(c) Une proposition atomique $\mathbf{en(t)}$ est vraie ssi une transition t est franchissable.

(d) Une proposition atomique $\mathbf{fi(t)}$ est vraie ssi une transition t est tirée.

Noter que (a) est un cas particulier de (b), i.e. $\mathbf{p = ge(p, 1)}$, et que la formule $\mathbf{fi(t)} \Rightarrow \mathbf{en(t)}$ est toujours vraie.

Plusieurs recherches ont montré que le problème de *vacuité*[1] (*emptiness problem*) est décidable lorsqu'il s'agit des réseaux de Petri *bornés* [Uchihira90b], ou *sauf* [Katai82], ou bien lorsque nous faisons appel à une LTLP dite *restreinte* [Howell88]. Pour des réseaux de Petri généraux, le problème devient indécidable [Cherkasova87, Howell88, Suzuki89]. Le problème de *vacuité* consiste à chercher une séquence de transitions licite satisfaisant une formule logique temporelle donnée sur un réseau de Petri donné. La décidabilité de ce problème est nécessaire pour la vérification et la synthèse automatique des programmes. Dans [Uchihira90], les auteurs considèrent les réseaux de Petri généraux avec une LTLP générale et adoptent la correspondance de type (d). Ils considèrent le langage infini $L_\omega(R, h)$ du RDP étiqueté (R, h) et le langage $L_s(f)$ de la formule LTLP f. Ils montrent que si $L_\omega(R, h) \cap L_s(f) \neq \emptyset$ alors le RDP satisfait la spécification f. Une fonction d'étiquetage $h : T \rightarrow \mathit{Prop} \cup \{\lambda\}$ est utilisée pour faire la correspondance entre les transitions t de T et les propositions atomiques (*Prop*) dans f, i.e. $h(t)$ est la formule $\mathbf{fi(t)}$. Certaines transitions peuvent être invisibles à l'utilisateur qui décrit la spécification logique temporelle, i.e. $h(t) = \lambda$ pour les transitions invisibles.

4.2 Vérification des réseaux de Petri.

Dans le processus de vie d'un système, la phase de vérification et de test prend une bonne partie du coût total du développement. C'est une phase nécessaire pour tester le bon fonctionnement du système et pour vérifier si le système développé est le système désiré.

Dans cette section, nous voulons décider si un réseau de Petri R possède des propriétés données sous forme d'une formule f de la logique temporelle. Nous combinons, en premier lieu, l'automate de Büchi acceptant la formule f et le graphe de couverture du réseau R. Nous cherchons, en second lieu, si le graphe obtenu contient un cycle passant par un noeud désigné. Si c'est le cas, alors le chemin partant de l'état initial à un noeud désigné, suivi de la boucle infinie contenant ce même noeud, est une suite infinie de transitions qui vérifie la formule f. Il suffit de vérifier alors si cette suite est licite.

4.2.1 Définition. [Uchihira90] Soient un réseau de Petri $R = (P, T, W, m_0)$, une formule LTLP f et une fonction d'étiquetage $h : T \rightarrow \mathit{Prop} \cup \{\lambda\}$. Nous définissons $L(R, f, h) = L_\omega(R, h) \cap L_s(f)$ où $L_\omega(R, h)$ est le langage infini généré à partir du réseau de Petri étiqueté (R, h) et $L_s(f)$ est le langage infini généré à partir de f sous la condition d'événement unique C_u. $L(R, f, h) \neq \emptyset$ signifie qu'il existe une séquence de transitions de R, licite et satisfaisant f. □

[1] Terme français suggéré par M. Bernard Hodgson.

4.2.2 Théorème. (Vacuité) [Uchihira90] *Le problème de vacuité de $L(R, f, h)$, i.e. $L(R, f, h)$* est-il vide ?, *est décidable pour R, f et h donnés.*

Preuve. Soit $R = (P, T, W, m_0)$. Il suffit de montrer que le problème de vacuité de $L_\omega(R, h) \cap L_s(f)$ est décidable. Au départ, nous construisons un graphe de couverture étendu G à partir de R, h et $L_s(f)$.

1. Construire un automate de Büchi $\mathcal{B}_f = (Prop, S, \rho, s_0, F)$ acceptant le langage $L_s(f)$. Ceci est possible grace au théorème 3.2.1 et aux définitions 3.3.1 et 3.3.4.

2. Construire un graphe de couverture étendu $G = (S_G, X)$ à partir de R, h et $\mathcal{B}_f$. G est un graphe dirigé et étiqueté. Chaque noeud x de G est représenté comme un $(k+2)$-uplet, $x = (x_1, x_2, \cdots, x_k, s, n_f)$ où $|P| = k, x_i \in \mathbf{N}_\omega (1 \leq i \leq k), s \in S$ et

$$n_f \;=\; \begin{cases} 1 & \text{si } s \in F \\ 0 & \text{sinon.} \end{cases}$$

   Chaque arc $e = (x, x')$ est étiqueté avec un élément de T. Une transition $t \in T$ est dite *franchissable* en x si t est franchissable à partir du marquage $(x_1, x_2, \cdots, x_k)$ et $\rho(s, h(t)) \neq \emptyset$, et est dite *localement franchissable* si t est franchissable à partir du marquage $(x_1, x_2, \cdots, x_k)$ et $h(t) = \lambda$. G est construit comme suit :

   (a) S_G est l'ensemble des noeuds de G; $S_G := \{(x_1^0, x_2^0, \cdots, x_k^0, s_0, n_f)\}$ où $(x_1^0, x_2^0, \cdots, x_k^0) = m_0$, s_0 est l'état initial de $\mathcal{B}_f$, et $n_f = 1$ si $s_0 \in F$, sinon $n_f = 0$. Nous posons $q_0 = (x_1^0, x_2^0, \cdots, x_k^0, s_0, n_f)$.

   (b) X est un ensemble d'arcs (x, x') étiquetés par des éléments de T; $X := \emptyset$.

   (c) Répéter tant que tous les noeuds n'ont pas été traités.

      i. Soit $x = (x_1, x_2, \cdots, x_k, s, n_f)$ un noeud non traité. Créer un noeud $x' = (x_1', x_2', \cdots, x_k', s', n_f')$ conformément aux étapes A–C ci-dessous, pour toute transition t franchissable à partir de $(x_1, x_2, \cdots, x_k)$ et pour tout $s' \in \rho(s, h(t))$. De même, créer un noeud $x' = (x_1', x_2', \cdots, x_k', s', n_f')$, où $s' = s, n_f' = 0$ conformément aux étapes A–B pour toute transition t localement franchissable.

         A. Pour tout i, $1 \leq i \leq k$, poser $x_i' := x_i \Leftrightarrow W(p_i, t) + W(t, p_i)$.

         B. S'il existe un noeud $r = (r_1, r_2, \cdots, r_k, s'', n_f'')$ sur un chemin allant de la racine q_0 à x tel que

$$(\forall i : 1 \leq i \leq k : r_i \leq x_i'),$$

            alors pour chaque i tel que $r_i < x_i'$, poser $x_i' := \omega$.

         C. $n_f = 1$ si $s' \in F$, sinon $n_f = 0$.

      ii. Si le noeud x' n'appartient pas à S, alors ajouter le noeud x' à S et l'arc (x, x') étiqueté par t à X, sinon ajouter seulement l'arc (x, x') étiqueté par t à X.

L'algorithme ci-dessus s'arrête toujours, car G est fini, de la même façon que le graphe de couverture normal d'un réseau de Petri (voir remarque 2.2.10).

D'après le théorème 3.11 de Valk et Jantzen [Valk85], nous avons $L_\omega(R,h) \cap L_s(f) \neq \emptyset$ ssi il existe un cycle $c = n_0, n_1, \cdots, n_k, n_0$ dans G tel que n_0 soit un noeud désigné (i.e. $n_f = 1$) et $\Delta(\theta) \geq 0$ où $\theta = t_1, \cdots, t_{k+1}$ est une séquence de transitions sur c, i.e. t_i est l'étiquette de $e = (n_{i-1}, n_i)$. La séquence θ n'est pas vide : elle contient au moins une transition. En effet, pour $k = 0$, on a $\theta = t_1$. Il est important que θ soit non vide pour obtenir une séquence infinie. Notons aussi que θ n'est pas nécessairement licite. La décidabilité de l'existence d'un tel cycle peut être prouvée comme suit : pour tout noeud désigné n_0, l'ensemble de toutes les séquences de transitions θ formant des cycles dans G passant par n_0 peut être représenté par une expression régulière E. Par exemple, l'expression régulière du cycle présenté par la figure 4.1 est donnée par

$$E = t_1(t_3(t_5 + t_6)^\star t_4)^\star t_2.$$

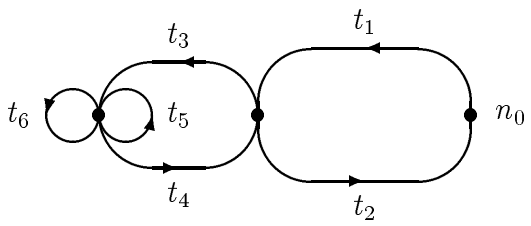


Figure 4.1 : Exemple de cycle c.

Nous voulons décider s'il existe $\theta \in L(E)$ tel que $\Delta(\theta) \geq 0$ où $L(E)$ est l'ensemble des séquences représentées par E. Pour ce faire, nous pouvons considérer que E est commutative. En utilisant les règles suivantes :

$$\begin{aligned} AB &= BA, \\ (A+B)^\star &= A^\star B^\star, \\ (A^\star B)^\star &= \lambda + A^\star B^\star B, \end{aligned}$$

décrites dans le livre de Conway [Conway71], E peut être exprimée comme une somme finie de termes de la forme $\theta_0\theta_1^\star\theta_2^\star \cdots \theta_n^\star$, où chaque θ_i est un cycle et $\theta_i \in T^\star (0 \leq i \leq n)$.

Ces règles ont la propriété de ne pas "décomposer" un cycle. i.e. si, par exemple, t_i et t_j appartiennent au départ à un même cycle alors, après application des règles de Conway, les deux transitions appartiennent toujours au même cycle. Pour mieux illustrer ces propos, appliquons les règles de Conway à l'expression E ci-dessus.

$$\begin{aligned} E &= t_1(t_3(t_5 + t_6)^\star t_4)^\star t_2 \\ &= t_1 t_2 (t_3 t_4 (t_5 + t_6)^\star)^\star \\ &= t_1 t_2 ((t_5 + t_6)^\star t_3 t_4)^\star \\ &= t_1 t_2 (\lambda + (t_5 + t_6)^\star (t_3 t_4)^\star t_3 t_4) \end{aligned}$$

$$= t_1t_2(\lambda + t_5^\star t_6^\star (t_3t_4)^\star t_3t_4)$$
$$= t_1t_2 + t_1t_2t_3t_4t_5^\star t_6^\star (t_3t_4)^\star$$

$E = \theta_{10} + \theta_{20}\theta_{21}^\star\theta_{22}^\star\theta_{23}^\star$ est une somme finie de termes de la forme $\theta_0\theta_1^\star\theta_2^\star\cdots\theta_n^\star$ avec

$\theta_{10} = t_1t_2,$

$\theta_{20} = t_1t_2t_3t_4,$

$\theta_{21} = t_5,$

$\theta_{22} = t_6,$

$\theta_{23} = t_3t_4.$

On remarque bien que chaque θ_{ij} est un cycle y compris θ_{i0} $(1 \leq i \leq 2)$. En utilisant la programmation linéaire, nous pouvons décider pour chaque $\theta_0\theta_1^\star\theta_2^\star\cdots\theta_n^\star$, s'il existe $\alpha_1, \alpha_2, \cdots, \alpha_n \geq 0$ tels que $\alpha_1 + \alpha_2 + \cdots + \alpha_n > 0$ et $\Delta(\theta_0\theta_1^{\alpha_1}\theta_2^{\alpha_2}\cdots\theta_n^{\alpha_n}) = \Delta(\theta_0) + \alpha_1\Delta(\theta_1) + \alpha_2\Delta(\theta_2) + \cdots + \alpha_n\Delta(\theta_n) \geq 0$. □

La contrainte $\alpha_1 + \alpha_2 + \cdots + \alpha_n > 0$ n'a pas été exigée dans l'article d'Uchihira. L'ajout de cette contrainte est l'une des corrections apportées à l'article [Uchihira90]. L'absence de cette contrainte permet à $\alpha_1 = \alpha_2 = \cdots = \alpha_n = 0$ d'être une solution possible, ce qui conduit à $\theta = \lambda$. Or, comme nous l'avons déjà annoncé, λ doit être non vide.

Lorsque $L(R, f, h)$ n'est pas vide, il est possible de trouver au moins une séquence $\theta \in L(R, f, h)$. La méthode pour ce faire est décrite dans le théorème suivant. Ce théorème sera très utile au chapitre suivant.

4.2.3 Théorème. [Uchihira90] *Si $L(R, f, h) \neq \emptyset$, il est possible de construire une séquence de tir licite $\theta_d\theta_c^\omega$ sur R telle que $h(\theta_d\theta_c^\omega) \in L(R, f, h)$.*

Preuve. En premier lieu, redéfinissons la fonction poids Δ (définie pour toute transition $t \in T$) pour tout arc $e = (x, x')$ de G construit selon la preuve du théorème 4.2.2. Nous notons $\Delta(e) = \Delta(t)$ où t est l'étiquette de l'arc e. D'après le théorème 4.2.2, si $L(R, f, h) \neq \emptyset$, il existe un cycle[2] $c = e_0e_1e_2\cdots e_k$ dans G où $e_0 = (x_0, x_1), \cdots, e_k = (x_k, x_0), \Delta(c) \geq 0$ et x_0 est un noeud désigné. Remarquons que le cycle c peut-être illicite. Nous montrons que le cycle c peut être construit et que nous pouvons aussi construire un chemin d du noeud initial q_0 au noeud désigné x_0 de façon que la séquence de transitions soit licite. Le marquage m_{x_0} en x_0 est tel que les places contiennent suffisamment de jetons pour que c soit licite. Par la suite, nous créons, à partir de dc^ω, une séquence de transitions licite $\theta_d\theta_c^\omega$ telle que $h(\theta_d\theta_c^\omega) \in L(R, f, h)$.

Construction du cycle c

Comme dans la preuve du théorème 4.2.2, nous pouvons calculer $\alpha_1, \alpha_2, \cdots, \alpha_n \geq 0$, tels que $\Delta(c) = \Delta(e_0) + \alpha_1\Delta(e_1) + \alpha_2\Delta(e_2) + \cdots + \alpha_n\Delta(e_n) \geq 0$ où e_i peut ne pas être adjacent à e_{i+1}. Chaque α_i correspond au nombre de fois qu'un arc e_i apparaît dans c. Par conséquent, la construction de c peut être réduite au cycle d'Euler, i.e. chaque arc e_i apparaît α_i fois dans c avec $\alpha_i > 0$.

[2] Un cycle peut être décrit soit par des noeuds soit par des transitions reliant des noeuds.

Construction du chemin d

Il est possible de montrer que $d = d_0 c_1^{\alpha_1} d_1 c_2^{\alpha_2} d_2 \cdots c_n^{\alpha_n} d_n$ où c_i est un cycle dans lequel une place de G change de marquage en ω (nous l'appelons $i^{ème}$ ω-place). Nous voulons calculer $\alpha_1, \alpha_2, \cdots, \alpha_n \geq 0$ tels que $\Delta(d_0 c_1^{\alpha_1} d_1 c_2^{\alpha_2} d_2 \cdots c_n^{\alpha_n} d_n) \geq m_{x_0}$. Pour que d soit licite, nous devons résoudre le système récurrent suivant :

(0) $M_0 = m_0 + \Delta(d_0)$, tel que le chemin d_0, issu du marquage m_0, soit licite,

(i) $M_i = M_{i-1} + \alpha_i \Delta(c_i) + \Delta(d_i)$, tel que le chemin $c_i^{\alpha_i} d_i$, issu du marquage M_{i-1}, soit licite,

(n) $M_n \geq m_{x_0}$.

Nous pouvons résoudre ce système de la dernière équation à la première, c'est-à-dire que nous pouvons calculer les α_i selon l'ordre décroissant des indices i indépendamment de $\alpha_1, \alpha_2, \cdots, \alpha_{i-1}$, puisque nous pouvons ignorer la $j^{ème}$ ω-place pour tout $j < i$. □

Pour mieux comprendre cette partie, traitons l'exemple illustratif suivant :

4.2.4 Exemple.

Soit le réseau de Petri de la figure 4.2, tiré de [Vidal92], sur lequel nous voulons vérifier une formule logique temporelle f.

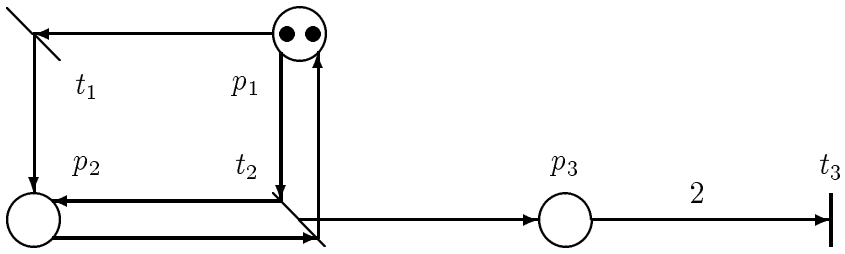


Figure 4.2 : Réseau de Petri R.

Les séquences licites du réseau sont des préfixes des séquences de la forme : $t_1 t_2^{a_1} t_3^{b_1} \cdots t_2^{a_i} t_3^{b_i} t_1 t_3^{b_{i+1}}$ avec :

$$a_1 + \cdots + a_k \quad \geq \quad 2 \cdot (b_1 + \cdots + b_k) \ (1 \leq k \leq i)$$

et

$$2 \cdot (b_1 + \cdots + b_{i+1}) \quad \leq \quad a_1 + \cdots + a_i \quad \leq \quad 2 \cdot (b_1 + \cdots + b_{i+1}) + 1.$$

Le premier franchissement de t_1 est un coup d'envoi pour un traitement répétitif : la transition t_2 est toujours permise alors que la transition t_3 est permise chaque fois que la place p_3 contient au moins deux jetons. Le deuxième franchissement de t_1 est une demande d'interruption du traitement : la transition t_2 n'est plus franchissable et seule la transition t_3 est tirée jusqu'à ce que la place p_3 contienne moins de deux

jetons. Par la suite, aucune transition n'est franchissable. Pour permettre au réseau un régime permanent, il faut interdire le deuxième franchissement de t_1. Cette condition est donnée par la formule logique temporelle

$$f \quad = \quad t_1 \wedge \bigcirc\Box(\neg t_1).$$

L'automate de Büchi acceptant cette formule est donné à la figure 4.3 (théorème 3.2.1).

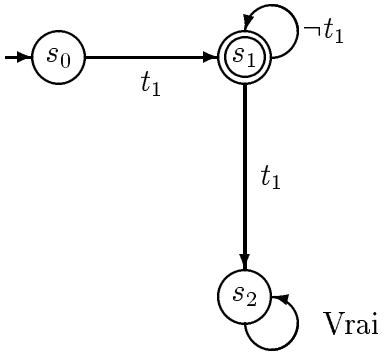


Figure 4.3 : Automate de Büchi acceptant f.

Construisons le graphe de couverture étendu, représenté à la figure 4.4, à partir du réseau R, de la formule f et de la fonction d'étiquetage, que nous supposons égale à la fonction identité I (théorème 4.2.2).

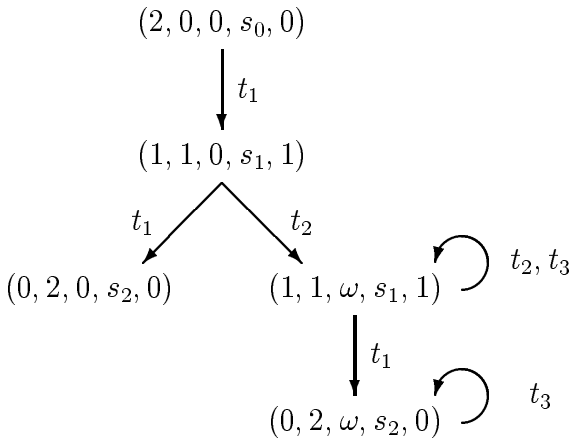


Figure 4.4 : Graphe de couverture étendu G.

Le seul cycle c du graphe G impliquant un état désigné passe par $x_0 = (1, 1, \omega, s_1, 1)$ où s_1 est l'état désigné de l'automate de Büchi. D'où $L(R, f, I) \neq \emptyset$ (théorème 4.2.2). L'expression régulière E associée à c est :

$$\begin{aligned} E \quad &= \quad (t_2 + t_3)^\star \\ &= \quad t_2^\star t_3^\star \qquad \text{d'après les règles de décomposition de Conway.} \end{aligned}$$

E contient un seul terme de la forme $\theta_0\theta_1^\star\theta_2^\star$ avec $\theta_0 = \lambda$, $\theta_1 = t_2$ et $\theta_2 = t_3$. La fonction poids Δ appliquée à l'expression E donne :

$$\Delta(t_2^{\alpha_1} t_3^{\alpha_2}) \quad = \quad \alpha_1\Delta(t_2) + \alpha_2\Delta(t_3).$$

Cherchons α_1 et α_2 tels que $\alpha_1\Delta(t_2)+\alpha_2\Delta(t_3) \geq 0$. Comme $\Delta(t_2) = (0,0,1)$ et $\Delta(t_3) = (0,0,\Leftrightarrow 2)$, on obtient $\alpha_1 \geq 2\alpha_2$. Pour obtenir un cycle d'Euler, il suffit de prendre $\alpha_1 = 2$ et $\alpha_2 = 1$. Le chemin d du noeud $q_0 = (2,0,0,s_0,0)$ au noeud $x_0 = (1,1,\omega,s_1,1)$ est la séquence t_1t_2. Ce chemin ne contient pas de cycles c_i. Pour que c soit licite, nous pouvons choisir $m_{x_0} = (1,1,1)$. Calculons maintenant le marquage initial. On a

$$\begin{aligned} m_{x_0} &= m_0 + \Delta(t_1t_2) \\ m_0 &= m_{x_0} \Leftrightarrow \Delta(t_1t_2) \\ &= (1,1,1) \Leftrightarrow (\Leftrightarrow 1,1,1) \\ &= (2,0,0). \end{aligned}$$

Ainsi, la séquence de transitions $t_1t_2(t_2^2t_3)^\omega$ est une suite infinie licite telle que $t_1t_2(t_2^2t_3)^\omega \in L(R,f,h)$. □

Nous avons jusqu'ici montré que le problème de *vacuité* de $L(R,f,h)$ est décidable et que si $L(R,f,h)$ n'est pas vide, alors il nous est possible d'extraire une séquence de transitions θ contenue dans $L(R,f,h)$. Il nous reste à définir le problème de *vérification* et à montrer qu'il est décidable.

Le programme à vérifer peut être un système concurrent représenté par un réseau de Petri. Vérifier, c'est décider si un réseau de Petri donné satisfait une spécification donnée. La spécification est une formule logique temporelle dans laquelle les propositions atomiques correspondent aux transitions du réseau de Petri.

Ici, nous nous intéressons seulement au langage infini $L_\omega(R,h)$, qui ne tient pas compte des séquences finies, y compris les séquences conduisant à un blocage. C'est pourquoi nous étendons un réseau de Petri R au réseau R_ω qui est sans blocage, et ce, en ajoutant une transition *visible* et *bidon* : *nop* (non-opération, voir figure 4.5). Nous supposons que toute transition est visible et que toute proposition atomique de f correspond à une transition de T. Dans ce cas, la fonction d'étiquetage est la fonction identité I.

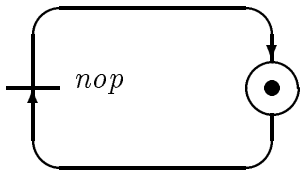


Figure 4.5 : nop (non-opération).

Pour vérifier que le programme satisfait la spécification, il suffit de vérifier que $L_\omega(R_\omega, I) \subset L_s(f)$, ce qui signifie que toute séquence de transitions est un modèle de la formule LTLP f.

4.2.5 Définition. (Vérification) [Uchihira90] Un réseau de Petri R_ω sans blocage satisfait la spécification LTLP f sous la condition d'événement unique C_u ssi $L_\omega(R_\omega, I) \subset L_s(f)$. Décider si R_ω satisfait f est appelé *problème de vérification.* □

4.2.6 Théorème. (Décidabilité) [Uchihira90] *Le problème de vérification est décidable.*

Preuve. D'après le lemme 3.3.5, le problème de vérification ($L_\omega(R_\omega, I) \subset L_s(f)$) peut être réduit au problème de vacuité $L_s(\neg f) \cap L_\omega(R_\omega, I) = L(R_\omega, \neg f, I) = \emptyset$, ce qui est décidable d'après le théorème 4.2.2. □

Le programme de vérification (voir figure 4.6) prend en entrée :

1. Un programme concurrent représenté par un réseau de Petri.
2. Une spécification f représentée par une formule logique temporelle.

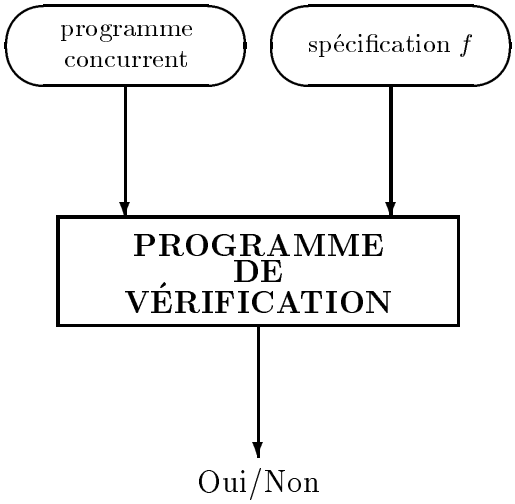


Figure 4.6 : Vérification d'un programme concurrent.

La sortie du programme est :

1. "Oui", si le programme en entrée satisfait la spécification.
2. "Non", sinon.

Indiquons maintenant quelles sont les propriétés que la méthode permet de vérifier.

4.2.1 Ce qui est vérifiable

La méthode présentée dans ce chapitre permet de vérifier des propriétés que nous pouvons exprimer en utilisant des transitions, alors qu'il est difficile de vérifier des propriétés faisant intervenir le nombre de jetons dans les places. Ceci est dû au fait que la correspondance entre les propositions atomiques et les propriétés des RDP que nous avons choisie est de type (d) (voir page 49).

L'exclusion mutuelle

Le problème classique de l'exclusion mutuelle se produit chaque fois que deux processus veulent accéder à une ressource non partageable (exemple : l'accès de deux utilisateurs à une imprimante). Pour l'exemple de la figure 4.7, le processus producteur (t_1, p_3, t_2) et le processus consommateur (t_3, p_6, t_4) communiquent par les tampons p_4 et p_5. Les places p_1 et p_2 assurent l'exclusion mutuelle : soit le producteur, soit le consommateur peut fonctionner. Les deux processus ne peuvent être en même temps dans leur section critique respective, i.e. on ne peut avoir, en même temps, un jeton dans la place p_3 et un autre dans la place p_6. En d'autres termes, les intervalles $[t_1, t_2]$ et $[t_3, t_4]$ ne doivent pas se chevaucher, ce que nous pouvons formuler en logique temporelle par :

$$\Box(t_1 \rightarrow \bigcirc(\neg t_3 \,\mathcal{U}\, t_2)) \wedge \Box(t_3 \rightarrow \bigcirc(\neg t_1 \,\mathcal{U}\, t_4)).$$

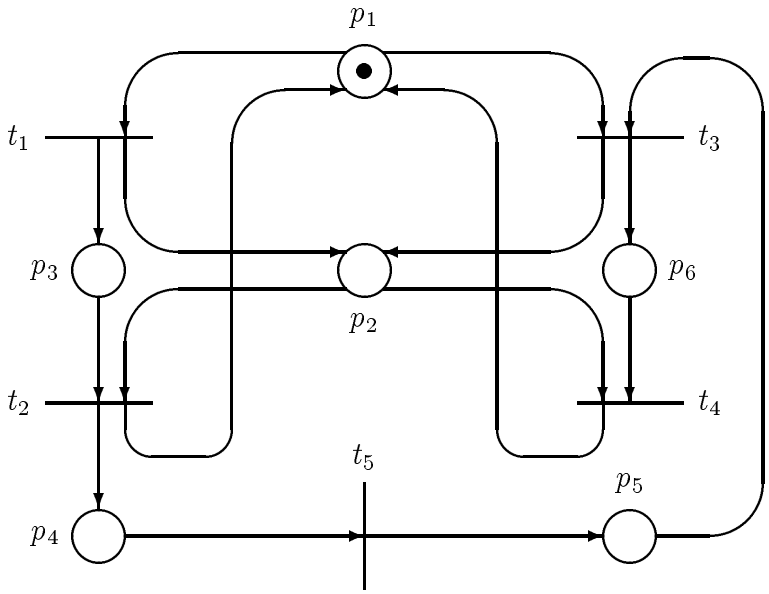


Figure 4.7 : Réseau de Petri R_ω.

Ordre partiel

Il est possible d'exprimer un ordre parmi les tirs de transitions. Par exemple, si nous voulons que deux transitions t_1, t_2 soient tirées à tour de rôle, nous écrivons :

$$\Box(t_1 \rightarrow (\neg t_1 \,\mathcal{U}\, t_2)) \wedge \Box(t_2 \rightarrow \bigcirc(\neg t_2 \,\mathcal{U}\, t_1)).$$

Interdiction de tir

$$\Box(t_1 \rightarrow \bigcirc(\Box \neg t_2))$$

signifie que si à un instant $i \geq 0$, t_1 est tirée alors t_2 n'est jamais tirée à un instant ultérieur j $(j > i)$.

Privation obligatoire

Il est possible de vérifier si une transition t est privée de tir à un instant futur. Cette propriété s'exprime par :

$$\Diamond\Box(\neg t).$$

Borne sur le nombre de jetons contenus dans une place

Ceci peut être vérifié par l'ajout d'une transition bidon b. Par exemple, $\Diamond\Box\neg b$ (figure 4.8) permet de vérifier si la place p est bornée.

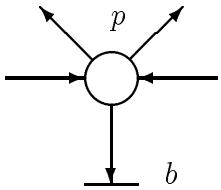


Figure 4.8 : Place bornée.

4.2.2 Ce qui n'est pas vérifiable

Nombre de jetons

Généralement, il est difficile de vérifier le nombre de jetons dans les places. Ce problème est équivalent à la recherche d'un marquage accessible. Notons que le problème d'accessibilité est décidable. Cependant, la complexité de l'algorithme de décision est très élevée, car elle nécessite la construction explicite du graphe d'accessibilité. Le problème d'accessibilité, posé en 1962, a été complètement résolu en 1982 [Kosaraju82, Mayr84].

Possibilité de blocage (Vivacité)

Il n'est pas possible de vérifier si un réseau est bloqué. Ceci est dû à l'introduction de la transition *nop* qui est une transition visible. Elle peut être itérée indéfiniment, en cachant un blocage à l'intérieur du réseau initial.

Quand la formule LTLP est petite, comparée au RDP, la complexité de la méthode de vérification est presque équivalente à celle du problème de couverture ($O(exp)$ en espace mémoire). Cependant, la méthode ne peut vérifier le problème d'accessibilité et la propriété de vivacité dont la complexité est beaucoup plus grande que celle du problème de couverture.

4.2.7 Exemple. [Uchihira90] Comme exemple de programme concurrent à vérifier, considérons un problème d'exclusion mutuelle contenant des tampons non bornés. Le

RDP en entrée, R_ω, est représenté à la figure 4.7, où les places p_5 et p_6 sont des tampons non bornés. Ce RDP est sans blocage, ainsi nous ignorons la transition *nop*. La spécification f impose que les intervalles $[t_1, t_2]$ et $[t_3, t_4]$ satisfassent la condition d'exclusion mutuelle :

$$f \quad = \quad \Box(t_1 \rightarrow \bigcirc(\neg t_3 \,\mathcal{U}\, t_2)) \wedge \Box(t_3 \rightarrow \bigcirc(\neg t_1 \,\mathcal{U}\, t_4)).$$

Nous devons vérifier que $L_\omega(R_\omega, I) \subset L_s(f)$. Ceci peut être réduit au problème de vacuité $L_s(\neg f) \cap L_\omega(R_\omega, I) = L(R_\omega, \neg f, I) = \emptyset$ (voir preuve du théorème 4.2.6). L'automate de Büchi $\mathcal{B}_{\neg f} = (\{t_1, t_2, t_3, t_4, t_5\}, \{s_0, s_1, s_2, s_3\}, \rho, s_0, \{s_3\})$ acceptant $\neg f$ est représenté par la figure 4.9. Cet automate est légérement différent de celui donné par Uchihira qui ne contient pas l'étiquette t_5. Cette modification est nécessaire pour la construction du graphe de couverture étendu. Intuitivement, l'automate $\mathcal{B}_{\neg f}$ accepte toute séquence infinie contenant la suite $t_1(t_1 + t_4 + t_5)^\star t_3$ ou la suite $t_3(t_3 + t_2 + t_5)^\star t_1$. Plus précisement, l'automate $\mathcal{B}_{\neg f}$ accepte le chevauchement des intervalles $[t_1, t_2]$ et $[t_3, t_4]$. Le graphe de couverture étendu G (figure 4.10) est généré à partir du RDP R_ω et $\mathcal{B}_{\neg f}$ (théorème 4.2.2). G ne contient aucun noeud désigné, ce qui signifie que $L(R_\omega, \neg f, I) = \emptyset$. Par conséquent, R_ω satisfait f.

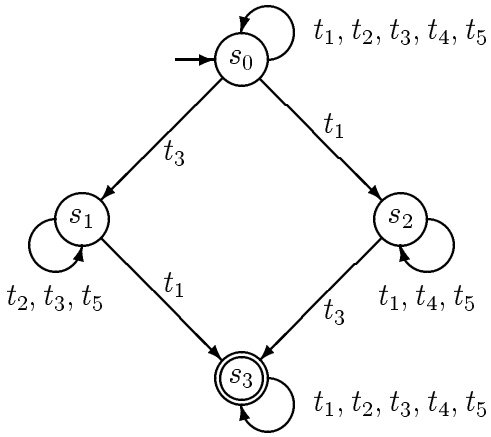


Figure 4.9 : Automate de Büchi $\mathcal{B}_{\neg f}$ acceptant $\neg f$.

□

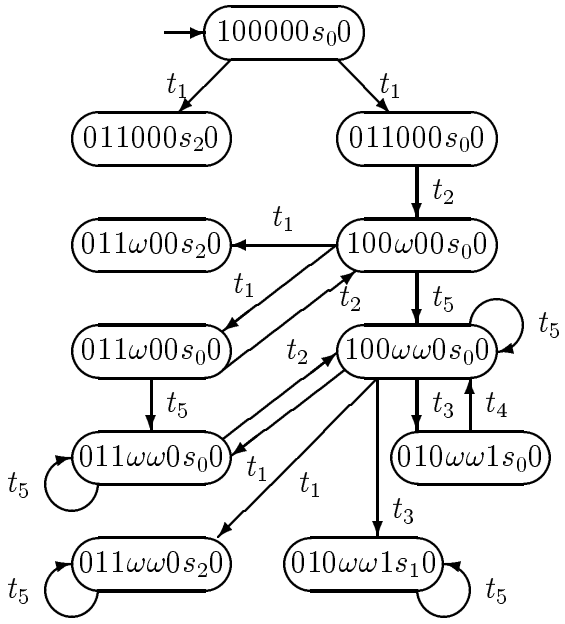


Figure 4.10 : Graphe de couverture étendu G.

Chapitre 5

Synthèse des réseaux de Petri.

Garantir qu'un programme assure de manière correcte l'ensemble des fonctionnalités pour lesquelles il a été réalisé est en soi un problème crucial en informatique séquentielle. Que dire alors lorsqu'il s'agit des systèmes distribués ou concurrents, auxquels s'ajoutent des problèmes de nature temporelle, à savoir la synchronisation et le non-déterminisme. Certaines recherches [Valk85] basées sur le modèle mathématique simple (vecteur d'entiers) sous-jacent aux réseaux de Petri, ont montré qu'il est possible de construire un réseau de Petri dont le langage est la restriction de l'ensemble des comportements du réseau initial aux séquences conduisant à des marquages vérifiant la propriété souhaitée.

Nous appliquons dans ce chapitre ces techniques à la synthèse des réseaux de Petri en utilisant le procédé de la *réutilisation*. Sachant que le coût des tests compte beaucoup plus que le coût du codage, ce procédé permet de minimiser les coûts de production. En effet, nous n'avons pas à refaire des tests et des vérifications qui ont été effectués sur le réseau initial. Nous montrons comment construire un réseau de Petri sans blocage et vérifiant une propriété d'équité, à partir d'un réseau de Petri déjà existant. Nous traitons un exemple d'illustration pour ce type d'application.

Enfin, nous étendons ces techniques à la synthèse compositionnelle de certains types de programmes concurrents. Ces derniers sont constitués d'un contrôleur et d'un ensemble d'agents. Le comportement désiré est donné par une formule LTLP. Et nous terminons par un exemple.

5.1 Représentation finie des ensembles de vecteurs entiers

Le but de cette première section est de donner quelques définitions concernant les *vecteurs d'entiers*, les notions d'*idéaux* de *résidu* et quelques résultats des travaux de Valk et Jantzen. Cette section est très utile pour la compréhension du reste du chapitre. Les définitions sont tirées principalement de [Valk85, Vidal92].

5.1.1 Définition. Soient $\mathbf{Z}$ l'ensemble des entiers et $\mathbf{N}$ l'ensemble des entiers positifs. Si $A, B \subseteq \mathbf{Z}^k$ alors on note $A + B = \{x + y | x \in A, y \in B\}$ et $A^\star = \bigcup_{i \in \mathbf{N}} A_i$ où

$$A_0 = \{(0, 0, \cdots, 0)\}, \quad A_{i+1} = A_i + A$$

et $(0, 0, \cdots, 0)$ est le vecteur nul de dimension k. □

Soient $A = \{(2, 1)\}$ et $B = \{(1, 0), (0, 1)\}$ deux sous-ensembles de $\mathbf{Z}^2$. Alors,

$$A + B = \{(3, 1), (2, 2)\},$$

$$A_0 = \{(0, 0)\}, \quad A_1 = \{(2, 1)\}, \quad A_2 = \{(4, 2)\},$$

On peut déduire facilement par récurrence sur n que

$$A_n = \{(2n, n)\} \text{ pour tout } n \in \mathbf{N},$$

d'où

$$A^\star = \bigcup_{i \in \mathbf{N}} A_i = \{(2n, n) | n \in \mathbf{N}\}.$$

5.1.2 Définition. Soient m et m' deux éléments de $\mathbf{N}_\omega^k$, on définit alors

$$min(m, m') = (min(m(1), m'(1)), min(m(2), m'(2)), \cdots, min(m(k), m'(k)).$$

Si $M, M' \subseteq \mathbf{N}_\omega^k$ alors, on définit

$$min(M, M') = \{min(m, m') | m \in M \text{ et } m' \in M'\}.$$

□

Pour les ensembles A et B définis ci-haut, on a

$$\begin{aligned} min(A, B) &= \{min((2, 1), (1, 0)), min((2, 1), (0, 1))\} \\ &= \{(1, 0), (0, 1)\}. \end{aligned}$$

5.1.3 Définition. (Ensemble régulier) Les sous-ensembles réguliers de $\mathbf{Z}^k$ sont définis comme suit :

(a) Tout sous-ensemble fini de $\mathbf{Z}^k$ est régulier.

(b) Si A et B sont des sous-ensembles réguliers de $\mathbf{Z}^k$ alors $A \cup B, A + B$ et $A^\star$ le sont aussi.

(c) Un ensemble $A \subseteq \mathbf{Z}^k$ est régulier ssi il est obtenu par une combinaison finie des règles (a) et (b). □

Les sous-ensembles A et B présentés ci-haut sont alors réguliers, puisqu'ils sont finis. L'ensemble

$$C = \{(x,y)|(x = 2y+1 \text{ ou } x = 2(y \Leftrightarrow 1)) \text{ et } x,y \in \mathbf{N}\}$$

est-il régulier ?

$$\begin{aligned} C &= \{(x,y)|x = 2y+1 \text{ et } x,y \in \mathbf{N}\} \cup \{(x,y)|x = 2(y \Leftrightarrow 1) \text{ et } x,y \in \mathbf{N}\} \\ &= \{(2n,n)|n \in \mathbf{N}\} + \{(1,0)\} \cup \{(2n,n)|n \in \mathbf{N}\} + \{(0,1)\} \\ &= A^\star + \{(1,0)\} \cup A^\star + \{(0,1)\} \\ &= A^\star + \{(1,0),(0,1)\} \\ &= A^\star + B \end{aligned}$$

Comme C est la somme de deux ensembles réguliers $A^\star$ et B, alors il est aussi régulier.

5.1.4 Définition. (Ensemble linéaire) Un ensemble régulier $R \subseteq \mathbf{Z}^k$ est dit *linéaire*, s'il a la forme $R = \{x\} + B^\star$ pour $x \in \mathbf{Z}^k$ et un sous-ensemble fini $B \subseteq \mathbf{Z}^k$. Un ensemble régulier $R \subseteq \mathbf{Z}^k$ est dit *semi-linéaire*, s'il est une union finie d'ensembles linéaires. □

Si $B = \{(0,0,\cdots,0)\}$ alors le singleton $R = \{x\} + B^\star = \{x\}$ est un ensemble linéaire. L'ensemble $D = \{(2n+1, n+1)|n \in \mathbf{N}\}$ est un ensemble linéaire. En effet, $D = \{(1,1)\} + A^\star$.

5.1.5 Théorème. [Conway71] *Les sous-ensembles réguliers de* $\mathbf{Z}^k$ *(ou* $\mathbf{N}^k$*) sont exactement les sous-ensembles semi-linéaires de* $\mathbf{Z}^k$ *(ou* $\mathbf{N}^k$*).*

5.1.6 Définition. (Région spécifiée par m) Pour tout $m \in \mathbf{N}_\omega^k$, on appelle $reg(m) = \{m' \in \mathbf{N}^k | m' \leq m\}$ la région spécifiée par m. □

Remarquons que $reg(m)$ est fini pour tout $m \in \mathbf{N}^k$.

5.1.7 Lemme. *Pour tout* $m \in \mathbf{N}_\omega^k$*, l'ensemble* $reg(m)$ *est semi-linéaire.*

Prenons par exemple $m = (2,\omega) \in \mathbf{N}_\omega^2$. Alors,

$$\begin{aligned} reg(m) &= \{(2,0),(1,0),(0,0)\} + \{(0,1)\}^\star \\ &= \{(2,0)\} + \{(0,1)\}^\star \cup \{(1,0)\} + \{(0,1)\}^\star \cup \{(0,0)\} + \{(0,1)\}^\star. \end{aligned}$$

$reg(m)$ est une somme de trois ensembles linéaires donc il est semi-linéaire. Nous définissons par la suite, les notions d'*idéal* et de *résidu*.

5.1.8 Définition. (Idéal) Un ensemble $K \subseteq \mathbf{N}^k$ est un *idéal* de $\mathbf{N}^k$ ssi $K = K + \mathbf{N}^k$. □

Intuitivement, un ensemble $K \subseteq \mathbf{N}^k$ est un idéal s'il est stable par passage aux valeurs supérieures. Par exemple, l'ensemble $K = \{(n+5, n+4) | n \in \mathbf{N}\}$ est un idéal puisqu'il peut s'écrire

$$K \;=\; \{(n+5, n+4) | n \in \mathbf{N}\} \;= K + \mathbf{N}^k,$$

alors que $K' = K \cup \{(1,1)\}$ ne l'est pas. En effet, $(2,2) \in K' + \mathbf{N}^k$ mais $(2,2) \notin K'$ d'où $K' \neq K' + \mathbf{N}^k$.

5.1.9 Définition. (Résidu) Soit K un sous-ensemble de $\mathbf{N}^k$. Le *résidu* de K, dénoté $res(K)$, est le plus petit sous-ensemble de K satisfaisant $res(K) + \mathbf{N}^k = K + \mathbf{N}^k$. □

Le résidu d'un ensemble $K \subset \mathbf{N}^k$ est un ensemble fini qui permet d'obtenir tout l'ensemble par passage aux valeurs supérieures. Pour $K = \{(n+5, n+4) | n \in \mathbf{N}\}$ on a $res(K) = \{(5,4)\}$.

5.1.10 Lemme. *Pour tout idéal K de $\mathbf{N}^k$, $res(K)$ est fini et $K = res(K) + \mathbf{N}^k$ est une représentation de K comme un ensemble semi-linéaire.*

Dans le chapitre 2 (proposition 2.2.4), nous avons dit que si une propriété monotone est vraie pour un marquage donné, alors elle reste vraie pour tout marquage plus grand. L'ensemble des marquages K pour lesquels la propriété est vraie est donc un idéal. Décider si un marquage m de K vérifie la propriété revient à chercher dans $res(K)$ un minorant de ce marquage. Cela simplifie évidemment considérablement la procédure de décision de la propriété. Des propriétés comme la *quasi-vivacité*, la *terminaison infinie* et le fait que le réseau soit *non borné* peuvent se décider suivant ce procédé. Pour mettre en oeuvre ce processus de décision, nous disposons d'un critère de calculabilité du résidu d'un idéal de $\mathbf{N}^k$. Définissons, tout d'abord, une propriété *RES* utile pour l'algorithme de construction du résidu.

5.1.11 Définition. (*RES*) Pour tout ensemble $K \subseteq \mathbf{N}^k$, on définit le prédicat $p_K : \mathbf{N}_\omega^k \to \{\text{Vrai, Faux}\}$ par $p_K(m) = (reg(m) \cap K \neq \emptyset)$. Un ensemble K possède la propriété *RES* ssi le prédicat $p_K(m)$ est décidable pour tout $m \in \mathbf{N}_\omega^k$. □

5.1.12 Théorème. *Soit K un idéal de $\mathbf{N}^k$. Alors, $res(K)$ est effectivement calculable ssi K possède la propriété RES.*

Preuve : Supposons en premier que $res(K)$ peut être calculé. Alors, $K = res(K) + \mathbf{N}^k$ donne une représentation semi-linéaire de K. Puisque $reg(m)$ est un ensemble semi-linéaire, la question "$reg(m) \cap K = \emptyset$?" est décidable.

Réciproquement, supposons que la question "$reg(m) \cap K = \emptyset$?" est décidable pour tout $m \in \mathbf{N}_\omega^k$. La méthode suivante peut être utilisée pour calculer effectivement $res(K)$:

Soit K un idéal de $\mathbf{N}^k$ possédant la propriété *RES*, i.e. $p_K(m) = (reg(m) \cap K \neq \emptyset)$ est décidable pour tout $m \in \mathbf{N}^k$.

Algorithme pour calculer $res(K)$

Début (*initialisation*)
$i := 0; M := \{(\omega, \cdots, \omega)\}; R := \emptyset;$
fin := Faux;
Tant que non(fin) faire

Répéter

Choisir un $m \in M$;

Si $p_K(m) =$ Faux alors $M := M \Leftrightarrow \{m\}$

Jusqu'à $p_K(m) =$ Vrai ou $M = \emptyset$;

Si $M = \emptyset$ alors fin := Vrai

Sinon (* Ici $reg(m) \cap K \neq \emptyset$; en conséquence $reg(m)$ contient au moins un élément de $res(K)$ *)

Répéter

Pour toutes les coordonnées i de m, remplacer $m(i)$ dans m par le plus petit $n \in \mathbf{N}$ tel que $p_K(m)$ reste encore Vrai;

Poser $m = (x_1, \cdots, x_k)$ le vecteur trouvé à la dernière étape;

(* le nouveau vecteur $m \in \mathbf{N}^k$ est un élément de $res(K)$ *)

$R := R \cup \{m\}$;

$$M' := \left\{ (y_1, \cdots, y_k) \in \mathbf{N}_\omega^k \;\middle|\; \begin{array}{l} \exists 1 \leq j \leq k : y_j = x_j \Leftrightarrow 1 \text{ et} \\ y_n = \omega \text{ pour tout } n \neq j \end{array} \right\};$$

(* M'_i décrit toutes les régions ne contenant pas l'élément m, i.e. pour $reg(M') := \bigcup_{m' \in M'} reg(m')$ on a $\mathbf{N}^k - reg(M') = \{m\} + \mathbf{N}^k$ *)

$M := min(M, M')$

Finsi

Finfaire
Fin □

Deux notions importantes découlent de la propriété de monotonie (propriété 2.2.4). Il s'agit de *séquence répétitive* et de *régime permanent.*

5.1.13 Définition. (Séquence répétitive) [Vidal92] Une séquence r est *répétitive* ssi pour tout marquage m tel que $m(r\rangle m'$, on a $m' \geq m$. Si $m' > m$, alors r est *répétitive croissante.* □

Une séquence répétitive est une séquence qui ne diminue pas le nombre de marques dans chaque place, c'est en fait une séquence sur laquelle on peut "boucler", i.e. la répéter autant de fois que l'on veut. La propriété suivante décrit le fait que si une telle séquence existe, le réseau admet un régime permanent.

5.1.14 Proposition. (Régime permanent) [Vidal92] *Si w est une séquence répétitive, alors pour tout marquage m tel que $m(w\rangle$, et tout $n \in \mathbf{N}$, on a :*

- $m(w^n\rangle$,
- w^ω *est une séquence infinie admissible pour* m.

Rappelons que $F(R, m_0)$ est l'ensemble des séquences de tir licites finies du réseau de Petri R ayant comme marquage initial m_0 (voir définition 2.3.6). Pour le reste de ce chapitre, nous posons $G_m = GC(R, m)$ le graphe de couverture de R issu de m, avec m dans $\mathbf{N}_\omega^k$ et, pour tout m' étiquette d'un noeud du graphe G_m, $L(G_m, m')$ dénote l'ensemble des chemins du graphe G_m issus de m'. Si X est l'ensemble des noeuds du graphe G_m, alors $L(G_m) = \bigcup_{m' \in X} L(G_m, m')$ est l'ensemble des chemins du graphe.

5.1.15 Lemme. [Vidal92] *Soient R un réseau de Petri et, $m \in \mathbf{N}_\omega^k$ et $m' \in \mathbf{N}_\omega^k$ deux éléments de X.*

(a) Si $m' \in reg(m)$, on a $F(R, m') \subseteq L(G_m, m)$.

(b) Si $v \in L(G_m)$ et si $\Delta(v) \geq 0$, alors $\exists u \in T^\star, \exists m' \in reg(m)$ tels que $uv \in F(R, m')$.

Preuve.

(a) Une séquence franchissable à partir de m' l'est à partir de m du fait de la monotonie. D'où,

(5.1) $F(R, m') \subseteq F(R, m)$.

De plus, si une séquence est franchissable à partir de m alors elle étiquette un chemin du graphe G_m issu de m, i.e. elle est élément de $L(G_m, m)$. Ainsi,

(5.2) $F(R, m) \subseteq L(G_m, m)$.

D'après (5.1) et (5.2), on obtient

(5.3) $F(R, m') \subseteq L(G_m, m)$.

(b) Si $v \in L(G_m)$ et $\Delta(v) \geq 0$ alors v étiquette un cycle du graphe issu d'un noeud m'' avec m'' un noeud du graphe (voir paragraphe juste après la proposition 5.1.14). Nous décomposons P en trois sous-ensembles disjoints :

$$\begin{aligned} P_1 &= \{p \in P, m(p) = m''(p) = \omega\}, \\ P_2 &= \{p \in P, m(p) \in \mathbf{N} \text{ et } m''(p) = \omega\} \text{ et} \\ P_3 &= \{p \in P, m''(p) \in \mathbf{N} \text{ (et donc } m(p) \in \mathbf{N})\}. \end{aligned}$$

Soit $u' \in T^\star$ le chemin (dans G_m) allant de m à m''. Il existe dans u' des séquences répétitives croissantes qui font augmenter le marquage des places de P_2 et qui donc "amènent" des ω dans ces places. Elles peuvent alors être itérées. Nous itérons chacune d'elles de façon à amener dans les places de P_2 suffisamment de

marques pour que v soit franchissable. Le nombre de marques contenues dans les places de P_3 demeure inchangé. En effet, les séquences sont répétitives. Donc, soit le marquage d'une place augmente et alors il devient infini soit il reste fini et constant. Nous obtenons ainsi une séquence u, allant de m à m''' (dans R) avec $m''' \leq m''$. De plus, comme v étiquette un cycle allant de m'' à m'' (dans G_m), le franchissement de v laisse invariant le nombre de marques dans les places de P_3. Donc, la séquence v est franchissable à partir de m''' puisque les places de P_1 n'induisent pas de contraintes; celles de P_2 ont été remplies par u de sorte qu'elles contiennent assez de marques pour permettre le franchissement de v. Enfin, du fait du cycle, celles de P_3 n'ont pas changé. Elles permettent donc toujours le franchissement de v. Puis pour chaque place p de P_1, nous remplaçons la coordonnée infinie $m'(p)$ par :

$$m'(p) \quad = \quad \Sigma_{i=1}^{a} W(p, u_i) + \Sigma_{j=1}^{b} W(p, v_j)$$

avec $u = u_1 \cdots u_a$ et $v = v_1 \cdots v_b$. On obtient ainsi un marquage m' avec $m' \leq m$ et $m'(uv\rangle$. □

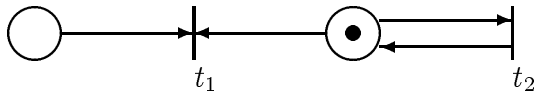
5.1.16 Définition. Soit w une séquence infinie de tir. L'ensemble des transitions de w tirées infiniment souvent est dénoté $In(w)$.

$$In(w) \quad = \quad \{t \in T | \forall i > 0 : \exists j \geq i : t = w(j)\}.$$

□

5.1.17 Définition. (Marquage $\hat{T}$-continuel) [Valk85] Soit $R = (P, T, W, m_0)$ un réseau de Petri. Un marquage $m \in \mathbf{N}^k$ de R est dit $\hat{T}$-continuel pour un sous-ensemble $\hat{T} \subseteq T$ de transitions ssi il existe des séquences infinies $w \in T^\omega$ telles que $m(w\rangle$ et $\hat{T} \subseteq In(w)$. On note CONTINUEL$(\hat{T}) = \{m \in \mathbf{N}^k | m$ est $\hat{T}$-continuel$\}$. □

Soient $R = (P, T, W, m_0)$ le réseau de Petri de la figure ci-dessous et $\hat{T} = \{t_2\}$.



La seule séquence infinie est $W = t_2^\omega$. Le marquage $m = (0, 1)$ est $\hat{T}$-continuel puisque $m(w\rangle$ et $\hat{T} = \{t_2\} \subseteq \ In(w) = \{t_2\}$. CONTINUEL$(\hat{T}) = \{(a, b) | a, b \in \mathbf{N}$ et $b > a\}$. En effet, pour tout marquage (a, b) avec $b > a$, le tir de la transition t_1 dimunie le nombre de jetons dans P_1 et P_2 de 1. Par contre, le tir de la transition t_2 laisse inchangé le marquage. Après a tirs de la transition t_1, on aura comme marquage $m = (0, b \Leftrightarrow a)$ et seule la transition $t_2 \in \hat{T}$ peut être tirée.

D'après la propriété de monotonie des réseaux de Petri (voir proposition 2.2.4), CONTINUEL$(\hat{T})$ est un idéal. Nous devons montrer qu'il satisfait la propriété *RES* (définition 5.1.11).

5.1.18 Théorème. [Valk85] *Soient $R = (P, T, W, m_0)$ un réseau de Petri et $\hat{T} \subseteq T$, alors CONTINUEL($\hat{T}$) satisfait la propriété RES.*

Preuve. Rappelons tout d'abord que $\overline{s}$ est le vecteur caractéristique de la séquence de transitions $s \in T^\star$ (voir définition 2.1.9) et que $\overline{s}(i)$ détermine le nombre d'occurences de la transition t_i dans s. Posons maintenant $e_{\hat{T}} \in \mathbf{N}^l$, où $l = |T|$, défini comme suit :

$$e_{\hat{T}}(i) \quad = \quad \begin{cases} 1 & \text{si } t_i \in \hat{T}, \\ 0 & \text{sinon,} \end{cases}$$

et montrons que

(5.4) *reg*$(m) \cap$ CONTINUEL$(\hat{T}) \neq \emptyset$ ssi $\exists v \in L(G_m) : \Delta(v) \geq 0$ et $\overline{v} \geq e_{\hat{T}}$.

Pour démontrer cette propriété, supposons en premier lieu qu'il existe $v \in L(G_m)$ tel que $\Delta(v) \geq 0$ et $\overline{v} \geq e_{\hat{T}}$. D'après le lemme 5.1.15, il existe $m' \in$ *reg*(m) et une séquence $u \in T^\star$ tels que $m'(uv\rangle$. Comme $\Delta(v) \geq 0$, alors $m'(uv^n\rangle$ pour tout $n \in \mathbf{N}$ (voir proposition 5.1.14) et m' est $\hat{T}$-continuel puisque $\hat{T} \subseteq In(v^\omega)$.

Réciproquement, si $m' \in$ *reg*(m) est $\hat{T}$-continuel, alors il existe une séquence infinie $w \in T^\omega$ telle que $m'(w\rangle$ et $\hat{T} \subseteq In(w)$. Évidemment, w peut être décomposée en $w = w_1 w_2 \cdots$, où $w_i \in T^\star$ et $\overline{w}_i \geq e_{\hat{T}}$. La suite $m'(w_1\rangle m_1, m'(w_1 w_2\rangle m_2, m'(w_1 w_2 w_3\rangle m_3, \cdots$ définit une séquence infinie de marquages $m', m_1, m_2, \cdots$. D'où, il existe nécessairement des indices $i < j$ tels que $m_i \leq m_j$. Définissons $v = w_{i+1} w_{i+2} \cdots w_j$, nous aurons alors $m_i(v\rangle m_j$ avec $\Delta(v) \geq 0$ et $\overline{v} \geq e_{\hat{T}}$. Puisque $m_i \in Acc(R, m')$, il existe alors $u \in T^\star$ tel que $m'(u\rangle m_i$. Ainsi, $uv \in L(G_m, m)$ et $v \in L(G_m)$. Ce qui démontre (5.4).

Maintenant, dans le but de décider s'il existe des $v \in L(G_m)$ avec $\Delta(v) \geq 0$ et $\overline{v} \geq e_{\hat{T}}$ nous procédons comme suit :

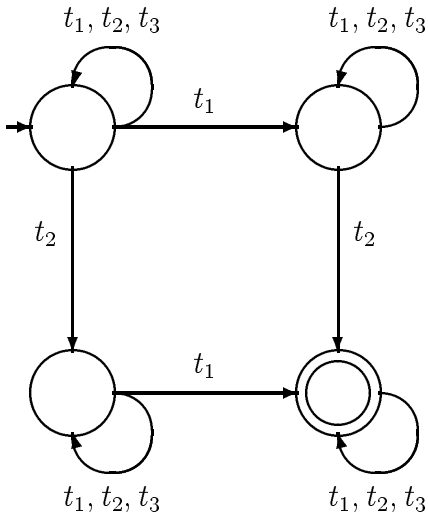
Au début, montrons que $E_R = L(G) \cap \{w \in T^\star | \overline{w} \geq e_{\hat{T}}\}$ est l'intersection de deux sous-ensembles réguliers et par conséquent il est régulier.

- $L(G_m)$ est un langage régulier. En effet, il est possible de construire un automate fini $\mathcal{M}$ dont le langage est exactement $L(G_m)$. Les états de $\mathcal{M}$ sont les noeuds du graphe G_m, l'état initial est le marquage initial du graphe, tous les états sont des états finaux et les étiquettes sur les arcs sont les mêmes que celles du graphe. Comme $L(G_m) = L(\mathcal{M})$ est un langage reconnu par un automate fini, il est alors régulier.

- Le langage $\{w \in T^\star | \overline{w} \geq e_{\hat{T}}\}$ peut être aussi reconnu par un automate fini. Il suffit de construire un automate qui accepte toutes les séquences finies contenant au moins une occurrence de chaque transition t_i telle que la $i^{ème}$ composante de $e_{\hat{T}}$ est égale à 1. Nous illustrons ce propos par un simple exemple dont le nombre de composante de $e_{\hat{T}}$ égale à 1 est 2. Soient $T = \{t_1, t_2, t_3\}$ et $\hat{T} = \{t_1, t_2\}$. $e_{\hat{T}}$ est donné par le vecteur

  $$e_{\hat{T}} \quad = \quad \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}.$$

  $L(\mathcal{M}) = \{w \in T^\star | \overline{w} \geq e_{\hat{T}}\}$ est le langage de l'automate ci-dessous.



Pour cet exemple, le nombre de composante de $e_{\hat{T}}$ égale à 1 est 2. L'automate a la forme d'un carré. Si ce nombre est égale à 3, l'automate aura la forme d'un cube et ainsi de suite.

Une représentation finie de E_R peut être construite à partir du graphe de couverture G_m. Comme E_R est régulier, $\Delta(E_R) = \{\Delta(v) | v \in E_R\}$ est un sous-ensemble régulier, donc semi-linéaire, de $\mathbf{Z}^k$. Alors $S = \Delta(E_R) \cap \mathbf{N}^k$ est un ensemble semi-linéaire de $\mathbf{N}^k$, une représentation duquel nous pouvons effectivement calculer. La question "$S \neq \emptyset$?" est ainsi décidable et est équivalente à :

$$\exists v \in L(G_m) : \Delta(v) \geq 0 \wedge \overline{v} \geq e_{\hat{T}}?$$

Par conséquent, CONTINUEL($\hat{T}$) satisfait la propriété *RES*. □

Le résultat suivant est une conséquence directe de la preuve du théorème 5.1.18.

5.1.19 Théorème. (Décidabilité) [Valk85] *Étant donnés un réseau de Petri $R = (P, T, W, m_0)$ et un ensemble de transitions $\hat{T} \subseteq T$, il est décidable si un marquage m est $\hat{T}$-continuel.*

Preuve. La preuve de l'équation (5.4) montre qu'un marquage m est $\hat{T}$-continuel ssi le graphe de couverture G_m contient un chemin de m' à m'' étiqueté par $u \in T^\star$ et un cycle de m'' à m'' étiqueté par $v \in T^\star$, tel que $\Delta(v) \geq 0$ et que toute transition $t \in \hat{T}$ soit tirée au moins une fois dans v. □

5.1.20 Théorème. *L'ensemble fini res($\hat{T}$-continuel) peut être effectivement calculé.*

Preuve. Elle se déduit du théorème 5.1.12 et du théorème 5.1.18. □

Ayant la possibilité de calculer le résidu $res(K)$ d'un idéal K, nous pouvons maintenant contrôler un réseau de Petri de façon que tout marquage accessible m reste dans K (par exemple K = CONTINUEL($\hat{T}$)).

5.1.21 Définition. Soient R un réseau de Petri, m_0 son marquage initial et $K \subseteq \mathbf{N}^k$. On note

$$\begin{aligned} F_K(R, m_0) &= \{u \in F(R, m_0) | \forall v < u, \forall m' \in \mathbf{N}^k, \text{ si } m_0(v\rangle m', \text{ alors } m' \in K\}, \\ Acc_K(R, m_0) &= \{m \in \mathbf{N}^k | \exists u \in F_K(R, m_0), m_0(u\rangle m\}. \end{aligned}$$

□

$F_K(R, m_0)$ est l'ensemble des séquences dont le franchissement mène toujours à des marquages appartenant à K et $Acc_K(R, m_0)$ est l'ensemble des marquages que l'on peut atteindre sans quitter K.

5.1.1 Application au contrôle d'un réseau de Petri

Parmi tous les comportements possibles d'un réseau, certains peuvent se révéler plus intéressants que d'autres. Par exemple, dans l'optique de l'obtention d'un régime permanent, les comportements conduisant à un blocage sont sans intérêt, et doivent être évités. Le but de l'utilisateur, lors de la spécification d'un système, est donc de contraindre le système de sorte à ne garder que les comportements "favorables".

Supposons que ces comportements soient ceux conduisant à des marquages possédant une propriété monotone (par exemple, tel que le réseau soit à terminaison infinie). L'ensemble des marquages possédant cette propriété est alors un idéal. Nous allons montrer que l'on peut construire un réseau de Petri dont le langage est la restriction de l'ensemble des comportements du réseau initial aux séquences conduisant à des marquages vérifiant la propriété souhaitée, i.e. un réseau à terminaison infinie.

Cette sous-section est extrêmement technique et offre des résultats importants en terme d'analyse et de construction des réseaux par une méthode mathématique fine. Nous construisons donc un nouveau réseau de Petri R_K associé au réseau R et à l'idéal $K = \text{CONTINUEL}(\hat{T})$. Nous modifions les préconditions du réseau de sorte que l'ensemble des marquages accessibles de R_K soit un sous-ensemble de l'idéal K.

5.1.22 Définition. (Réseau contrôlé) [Valk85] Soient $R = (P, T, W, m_0)$ un réseau de Petri et K un idéal. On appelle *réseau contrôlé par* K le réseau $R_K = (P, T', W', m_0)$ où :

$$\begin{aligned} T' &= T_1 \cup T_2 \\ T_1 &= \left\{ t \in T \;\middle|\; \begin{array}{l} \forall m' \in res(K) : \exists m \in res(K) : \forall p \in P \\ \max(m'(p), W(p,t)) + W(t,p) \Leftrightarrow W(p,t) \geq m(p) \end{array} \right\} \\ T_2 &= \{t_m | t \in T \setminus T_1, m \in res(K)\}. \end{aligned}$$

1. Pour $t \in T_1, W'(p,t) = W(p,t)$ et $W'(t,p) = W(t,p)$.

2. Pour $t_m \in T_2$,

$$\begin{aligned} W'(p,t_m) &= max(W(p,t), m(p) \Leftrightarrow W(t,p) + W(p,t)), \\ W'(t_m,p) &= max(W(t,p), m(p)), \end{aligned}$$

i.e.,

$$(W'(p,t_m), W'(t_m,p)) = \begin{cases} (W(p,t), W(t,p)) & \text{si } W(t,p) \geq m(p), \\ (m(p) - W(t,p) + W(p,t), m(p)) & \text{sinon.} \end{cases}$$

□

Nous notons $\rangle_K$ la relation d'accessibilité de ce réseau. Nous définissons de plus la fonction d'étiquetage $h : T' \to T$,

$$h(t') = \begin{cases} t' & \text{si } t' \in T_1 \text{ et} \\ t & \text{si } t' = t_m \in T_2. \end{cases}$$

Ce réseau vérifie les propriétés suivantes :

5.1.23 Proposition. [Valk85] *Soient $R = (P, T, W, m_0)$ un réseau de Petri, K un idéal et $R_K = (P, T', W', m_0)$ le réseau de Petri contrôlé par K. Alors on a*

1. *$m \in K$ et $m(t\rangle_K m' \Rightarrow m' \in K$.*
2. *$\forall t' \in T' : C' \cdot \overline{t'} = C \cdot \overline{h(t')}$ où C' et C sont les matrices d'incidence des réseaux R_K et R.*

Preuve.

1. Nous pouvons écrire $m = n_0 + n$ avec $n_0 \in res(K)$, puisque $m \in K$ et K un idéal.

   (a) Si $t \in T_1$ alors

   $\exists m'' \in res(K)$ tel que $max(n_0(p), W(p,t)) + W(t,p) \Leftrightarrow W(p,t) \geq m''(p)$ pour tout $p \in P$ (par définition de T_1).
   Or, $m(t\rangle_K m'$ donc
   (5.5) $n_0(p) + n(p) = m'(p) \geq W'(p,t) = W(p,t)$.
   On a bien évidemment
   (5.6) $n_0(p) + n(p) \geq n_0(p)$.
   Ceci donne
   (5.7) $n_0(p) + n(p) \geq max(n_0(p), W(p,t))$ (d'après 5.5 et 5.6).
   D'où,

$$\begin{aligned} m'(p) &= n_0(p) + n(p) \Leftrightarrow W(p,t) + W(t,p) \\ &\quad \text{(d'après 5.7)} \\ &\geq max(n_0(p), W(p,t)) \Leftrightarrow W(p,t) + W(t,p) \\ &\geq m''(p). \end{aligned}$$

   Comme K est un idéal et $m'' \in K$, on a alors $m' \in K$.

(b) Si $t = t_{m''} \in T_2$:

- Si $W(t,p) \geq m''(p)$ alors $W'(p, t_{m''}) = W(p,t)$.
  (5.8) $m(t\rangle_K \Rightarrow n_0(p) + n(p) \geq W'(p, t_{m''}) = W(p,t)$.

$$\begin{aligned} m'(p) &= n_0(p) + n(p) + W'(t_{m''}, p) \Leftrightarrow W'(p, t_{m''}) \\ &= n_0(p) + n(p) + W(t,p) \Leftrightarrow W(p,t) \text{ (par hypothèse)} \\ &\geq n_0(p) + n(p) + m''(p) \Leftrightarrow W(p,t) \\ &\quad \text{(d'après 5.8)} \\ &\geq m''(p). \end{aligned}$$

- Si $W(t,p) < m''(p)$ alors $m(t\rangle_K \Rightarrow m(p) \geq W'(p,t)$.
  (5.9) $n_0(p) + n(p) \geq W'(p,t) = m''(p) \Leftrightarrow W(t,p) + W(p,t)$.
  (d'après la définition 5.1.22).

  Donc

$$\begin{aligned} m'(p) &= n_0(p) + n(p) + W(t,p) \Leftrightarrow W(p,t) \\ &\quad \text{(d'après 5.9)} \\ &\geq m''(p). \end{aligned}$$

Par conséquent, on a $m' \geq m'', m'' \in K$, d'où l'on tire $m' \in K$.

2. Rappelons que la matrice d'incidence est donnée par :

$$\begin{aligned} C(i,j) &= post(i,j) \Leftrightarrow pré(i,j) \\ &= W(t_j, p_i) \Leftrightarrow W(p_i, t_j). \end{aligned}$$

En utilisant les équations 1 et 2 de la définition 5.1.22, on a pour tout i :
si $t' \in T_1$,

$$\begin{aligned} C'(i,.) \cdot \overline{t'} &= (W'(., p_i) \Leftrightarrow W'(p_i, .)) \cdot \overline{t'} \\ &= (W(., p_i) \Leftrightarrow W(p_i, .)) \cdot \overline{h(t')} \\ &= C(i,.) \cdot \overline{h(t')} \end{aligned}$$

sinon $t' = t_m$ avec $m \in res(K)$

si $W(t,.) \geq m$ alors

$$\begin{aligned} C'(i,.) \cdot \overline{t'} &= (W'(., p_i) \Leftrightarrow W'(p_i, .)) \cdot \overline{t'} \\ &= (W(., p_i) \Leftrightarrow W(p_i, .)) \cdot \overline{t} \\ &= C(i,.) \cdot \overline{h(t')} \end{aligned}$$

sinon

$$\begin{aligned} C'(i,.) \cdot \overline{t'} &= (W'(., p_i) \Leftrightarrow W'(p_i, .)) \cdot \overline{t'} \\ &= (m \Leftrightarrow (m + W'(p_i, .) \Leftrightarrow W'(., p_i))) \cdot \overline{t} \\ &= (W(., p_i) \Leftrightarrow W(p_i, .)) \cdot \overline{t} \\ &= C(i,.) \cdot \overline{h(t')}. \end{aligned}$$

□

Modulo un étiquetage, le comportement de ce réseau est l'ensemble des comportements du réseau initial permettant de ne pas sortir de l'idéal, ce que décrit le théorème suivant :

5.1.24 Théorème. [Vidal92] *Soient $R = (P, T, W, m_0)$ un réseau de Petri, K un idéal et $R_K = (P, T', W', m_0)$ le réseau contrôlé par K. $\forall m_1 \in K, \forall t \in T$:*

1. *$m_1(t\rangle m_2$ et $m_2 \in K \Leftrightarrow \exists t' \in T' : h(t') = t$ et $m_1(t'\rangle_K m_2$.*
2. *$Acc(R_K, m_1) = Acc_K(R, m_1)$.*
3. *$h(F(R_K, m_1)) = F_K(R, m_1)$.*

Preuve.

1. $\Rightarrow$ Si $t \in T_1$, alors on prend $t' = t$, et le résultat est immédiat. Sinon, soit $m \in res(K)$, tel que $m \leq m_2$ (un tel m existe car $m_2 \in K$). Soit $t' = t_m \in T_2$. On a $m_1(p) = m_2(p) \Leftrightarrow W(t,p) + W(p,t) \geq m(p) \Leftrightarrow W(t,p) + W(p,t)$ pour tout $p \in P$. On a également $m_1(p) \geq W(p,t)$ car $m_1(t\rangle$. Par conséquent, $m_1(p) \geq max(W(p,t), m(p) \Leftrightarrow W(t,p) + W(p,t)) = W'(p,t')$, donc $m_1(t'\rangle_K$. De plus, $m_1(p) + W'(t',p) \Leftrightarrow W'(p,t') = m_1(p) + W(h(t'),p) \Leftrightarrow W(p,h(t')) = m_2(p)$, d'après la proposition 5.1.23. En résumé, $m_1(t'\rangle_K m_2$ et $h(t') = t$. D'où,

   $$m_1(t\rangle m_2 \text{ et } m_2 \in K \Rightarrow \exists t' \in T' : h(t') = t \text{ et } m_1(t'\rangle_K m_2.$$

   $\Leftarrow$ Si $t' \in T_1, t' = h(t')$ et le résultat est immédiat, en utilisant la proposition 5.1.23.

   Sinon $t' = t_m$, on a $m_1(t'\rangle_K$, donc $m_1 \geq W'(p,t') \geq W(p,t)$, de sorte que : $m_1(t\rangle$ et

   $$\begin{aligned} m_2(p) &= m_1(p) + W'(t',p) \Leftrightarrow W'(p,t') \\ &= m_1(p) + W(t,p) \Leftrightarrow W(p,t) \in K \end{aligned}$$

   d'après la proposition 5.1.23, i.e. $m_1(t\rangle m_2$ et $m_2 \in K$. D'où,

   $$m_1(t\rangle m_2 \text{ et } m_2 \in K \Leftarrow \exists t' \in T' : h(t') = t \text{ et } M_1(t'\rangle_K m_2.$$

2. Soit m_2 un marquage. Nous faisons une récurrence sur la longueur de la séquence de transitions w allant de m_1 à m_2 dans $\mathbf{N}^k$.

   (a) Si $|w| = 1$ (prenons $w = t$), alors d'après le point 1 du théorème, on a $m_2 \in Acc_K(R, m_1) \Leftrightarrow m_2 \in Acc(R_K, m_1)$.

   (b) Supposons le résultat acquis pour les séquences de taille n. Soit $w = vt$ avec $|v| = n$.

   $$\begin{aligned} m_2 \in Acc(R_K, m_1) &\Leftrightarrow m_1(vt\rangle_K m_2 \\ &\Leftrightarrow \exists m, m_1(v\rangle_K m \text{ et } m(t\rangle_K m_2 \\ &\Leftrightarrow \exists m, m \in Acc_K(R, m_1) \text{ et } m_2 \in Acc_K(R, m) \\ &\Leftrightarrow m_2 \in Acc_K(R, m_1). \end{aligned}$$

3. Nous procédons de nouveau par récurrence sur la longueur des séquences de transitions.

   (a) Si $|w| = 1$, on applique le point 1 du théorème.

   (b) Nous supposons le résultat acquis pour les séquences de taille n. Si $w = vt$, avec $|v| = n$, alors

$$\begin{aligned} w \in F_K(R, m_1) &\Leftrightarrow \exists m, v \in F_K(R, m_1) \text{ et } t \in F_K(R, m) \text{ et } m_1(v\rangle m \\ &\Leftrightarrow \exists m, v \in h(F(R_K, m_1)) \text{ et } t \in h(F(R_K, m)) \text{ et} \\ &\quad m_1(v\rangle m \\ &\Leftrightarrow vt \in h(F(R_K, m_1)). \end{aligned}$$

□

Suite à ces résultats, nous pouvons ainsi synthétiser ou plus précisément modifier un réseau de Petri déjà existant, de façon à ce qu'il :

- soit sans blocage,
- ne contienne pas de boucles infinies contenant seulement des transitions invisibles.

Les transitions invisibles (voir page 16) sont des transitions que l'usager ne peut observer et qui paraissent comme des transitions internes. Une transition est visible si son étiquette est visible. L'ensemble des transitions visibles est dénoté V. Pour $V \subseteq \Sigma$, une transition étiquetée par λ est invisible. Pour le reste de chapitre, nous supposons que $\Sigma = V$, c'est-à-dire que pour toutes les transitions visibles t on a $h(t) \in \Sigma$ et pour toutes les transitions invisibles t, on a $h(t) = \lambda$. Rappelons que $F_\omega(R, m_0)$ dénote l'ensemble des séquences de tir licites infinies du réseau R ayant comme marquage initial m_0 (voir définition 2.3.6). Nous avons aussi défini les notions de langage $L(R, h)$, de ω-langage $L_\omega(R, h)$ et de $\lambda\omega$-langage $L_{\lambda\omega}(R, h)$ du réseau étiqueté (R, h).

5.2 Synthèse des réseaux de Petri

Dans cette section, nous présentons une méthode de synthèse des réseaux de Petri [Uchihira90]. Elle construit un réseau de Petri satisfaisant à des spécifications logiques temporelles en modifiant légèrement un réseau initial. Avant la description de la méthode, définissons les notions de langage *V-équitable*, de *réseau sans blocage* et de *réseau V-Juste*.

5.2.1 Définition. (Langage *V-équitable***)** [Uchihira90] Soit (R, h) un réseau étiqueté où $R = (P, T, W, m_0)$ un réseau de Petri et h la fonction d'étiquetage. Le langage $L_{\lambda\omega}^{V\text{-eq}}(R, h) \subset L_{\lambda\omega}(R, h)$ est défini comme suit : $L_{\lambda\omega}^{V\text{-eq}}(R, h) = \{h(\theta) \in \Sigma^\infty | \theta \in F_\omega(R, m_0)$ sous la condition de *V-équité*$\}$ où la condition de *V-équité* signifie que si certaines transitions visibles sont infiniment souvent franchissables alors l'une d'elles est nécessairement tirée. □

Rappelons que $L_{\lambda\omega}^{V\text{-eq}}(R,h)$ peut contenir des mots finis tels que $\sigma = \sigma\lambda^\omega$. Nous donnons ci-dessous un exemple et nous discutons de quelques cas particuliers.

5.2.2 Exemple. Soit (R,h) un réseau de Petri étiqueté (figure 5.1), où $R = (P,T,W,m_0)$ avec

$$\begin{aligned} P &= \{p_1,p_2\},\\ T &= \{t_1,t_2,t_3,t_4\},\\ m_0 &= (1,0),\\ h(t_1) &= a,\ h(t_2) = b \text{ et } h(t_3) = h(t_4) = \lambda. \end{aligned}$$

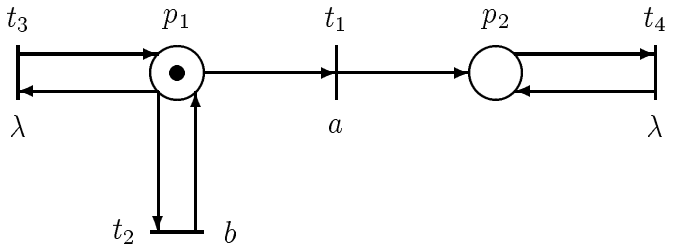


Figure 5.1 : Réseau de Petri étiqueté et transitions visibles.

La seule séquence infinie visible de ce réseau est b^ω. Donc $L_\omega(R,h) = \{b^\omega\}$. Les séquences infinies incluant λ sont $\lambda = \lambda^\omega$ et $b^n a\lambda^\omega = b^n a$ pour tout $n \in \mathbf{N}$. On déduit alors que $L_{\lambda\omega}(R,h) = \{b^\omega\} \cup \{\lambda\} \cup \{b^n a : n \in \mathbf{N}\}$. Parmi les transitions de $L_{\lambda\omega}(R,h)$ seuls b^ω et $b^n a$ pour tout $n \in \mathbf{N}$ sont des transitions *V-équitable*. D'où, $L_{\lambda\omega}^{V\text{-eq}}(R,h) = \{b^\omega\} \cup \{b^n a : n \in \mathbf{N}\}$. Pour cet exemple, on a

$$L_\omega(R,h) \subset L_{\lambda\omega}^{V\text{-eq}}(R,h) \subset L_{\lambda\omega}(R,h).$$

□

5.2.3 Définition. (Réseau sans blocage, réseau *V-Juste*) [Uchihira90] Un réseau de Petri $R = (P,T,W,m_0)$ est *sans blocage* ssi il existe au moins une transition franchissable pour tout marquage accessible. Un réseau de Petri étiqueté (R,h) est dit *V-Juste* ssi $L_{\lambda\omega}^{V\text{-eq}}(R,h) = L_\omega(R,h)$. Cela signifie qu'il n'existe pas de boucle infinie contenant seulement des transitions invisibles (qui paraît comme un blocage de l'extérieur) sous la condition de *V-équité*. □

Soit (R,h) un réseau de Petri étiqueté (figure 5.2), où $R = (P,T,W,m_0)$ avec

$$\begin{aligned} P &= \{p_1,p_2\},\\ T &= \{t_1,t_2,t_3\},\\ m_0 &= (1,0),\\ h(t_1) &= a,\ h(t_2) = b \text{ et } h(t_3) = \lambda. \end{aligned}$$

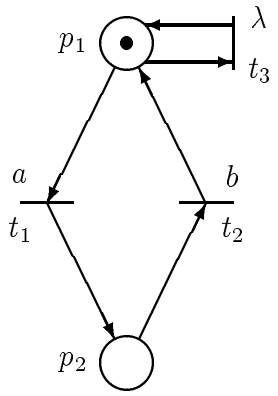


Figure 5.2 : Réseau de Petri *V-Juste*.

Le réseau R est sans blocage et *V-Juste* pour $V = \{t_1, t_2\}$. En effet, t_3, qui est une transition invisible, ne peut être tirée indéfiniment sans que les transitions visibles t_1 et t_2 soient tirées.

$$L_{\lambda\omega}^{V\text{-eq}}(R,h) \;=\; L_\omega(R,h) = (t_1 t_2)^\omega.$$

5.2.4 Théorème. (Synthèse d'un RDP) [Uchihira90] *Si* $\sigma_0\sigma_c^\omega \in L_\omega(R,h)$, *un réseau de Petri* (R_K, h') *peut être construit en ajoutant certaines places, transitions et arcs au réseau initial* R *tel que* R_K *soit sans blocage,* (R_K, h') *soit V-Juste et* $L_{\lambda\omega}^{V\text{-eq}}(R_K, h') = \{\sigma_0\sigma_c^\omega\}$.

Preuve. Il est facile de construire un réseau de Petri étiqueté (R_σ, I) tel que $L_\omega(R_\sigma, I) = \{\sigma_0\sigma_c^\omega\}$. Ensuite, construisons par composition le réseau de Petri $(R_c, h_c) = (R,h)|_\Sigma(R_\sigma, I)$. D'après le lemme 2.3.9, $L_\omega(R_c, h_c) = L_\omega(R,h) \cap L_\omega(R_\sigma, I) = \{\sigma_0\sigma_c^\omega\}$. Finalement, nous pouvons construire (R_K, h') en modifiant (R_c, h_c) tel que R_K soit sans blocage et $L_{\lambda\omega}^{V\text{-eq}}(R_K, h') = L_\omega(R_K, h') = \{\sigma_0\sigma_c^\omega\}$. Le RDP (R_σ, I) a l'identité comme fonction d'étiquetage. Il a donc des transitions étiquetées par des éléments de Σ et les transitions sont aussi des éléments de Σ. Nous pouvons donc choisir $\hat{T}$ égal à l'ensemble des transitions de σ_c (voir théorème 5.1.19) afin d'avoir un régime permanent. Calculons $K = \text{CONTINUEL}(\hat{T})$ et $res(K)$ (théorème 5.1.12). D'après le théorème 5.1.24, nous pouvons construire un nouveau réseau de Petri R_K dont tous les marquages accessibles sont contenus dans K avec le même nombre de places que R_c, mais en ajoutant de nouvelles transitions et de nouveaux arcs. □

5.2.5 Remarque. Cette preuve est légèrement différente de celle de [Uchihira90]. Cette dernière construit un réseau de Petri étiqueté (R_σ, I) tel que $L(R_\sigma, I) = \{\sigma_0\sigma_c^\star\}$. Or, ceci est faux lorsque le cycle σ_c contient plus d'une transition. Nous montrerons dans l'exemple ci-dessous la nécessité de choisir un réseau de Petri étiqueté (R_σ, I) tel que $L_\omega(R_\sigma, I) = \{\sigma_0\sigma_c^\omega\}$. □

L'exemple suivant est une application directe du théorème 5.2.4. Nous abrévions **fi(t)** par **t**.

5.2.6 Exemple. [Uchihira90] Soit $R = (P, T, W, m_0)$ un réseau de Petri (figure 5.3) défini comme suit :

$$\begin{aligned} P &= \{p_1, p_2\}; \\ T &= \{t_0, t_1, t_2, t_3\}; \\ m_0 &= (2, 0); \end{aligned}$$

$$pré = \begin{array}{c} \\ p_1 \\ p_2 \end{array} \begin{array}{c} \begin{array}{cccc} t_0 & t_1 & t_2 & t_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array}, \quad post = \begin{array}{c} \\ p_1 \\ p_2 \end{array} \begin{array}{c} \begin{array}{cccc} t_0 & t_1 & t_2 & t_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

$V = \{t_0, t_1, t_2\}$ est l'ensemble des transitions visibles et $h = I/V$ est la fonction d'étiquetage du réseau R. Étant donné que la séquence $\sigma_0\sigma_c^\omega \in L_\omega(R, h)$ avec $\sigma_0 = t_0$ et $\sigma_c = t_1t_2$, nous pouvons construire un réseau (R_K, h') qui est sans blocage, *V-Juste* et tel que $L_\omega^{\text{V-eq}}(R_K, h') = \{\sigma_0\sigma_c^\omega\}$. Construisons tout d'abord le réseau (R_σ, I) (figure 5.4) tel que $L_\omega(R_\sigma, I) = \{\sigma_0\sigma_c^\omega\}$. Suite à la remarque 5.2.5, nous pouvons montrer que $L(R_\sigma, I) \neq \{\sigma_0\sigma_c^\star\}$, contrairement à ce qu'on trouve dans [Uchihira90]. En effet, $L(R_\sigma, I) = \{\sigma_0(\sigma_c^\star + \sigma_c^\star t_1)\}$. Combinons maintenant les réseaux (R, h) et (R_σ, I) (définition 2.3.7) pour obtenir le réseau (R_c, h) (figure 5.5), i.e. $(R_c, h) = (R, h)|_V(R_\sigma, I)$. Nous donnons ci-dessous les matrices *pré* et *post* du réseau R_c.

$$pré = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{cccc} t_0 & t_1 & t_2 & t_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}, \quad post = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{cccc} t_0 & t_1 & t_2 & t_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array}.$$

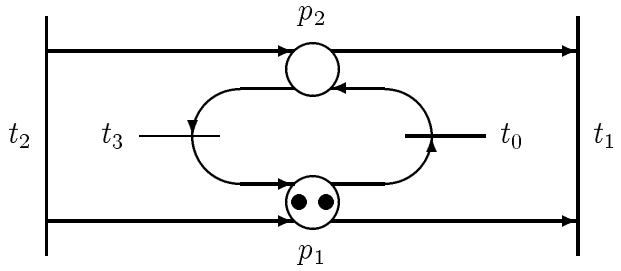


Figure 5.3 : Réseau de Petri initial R.

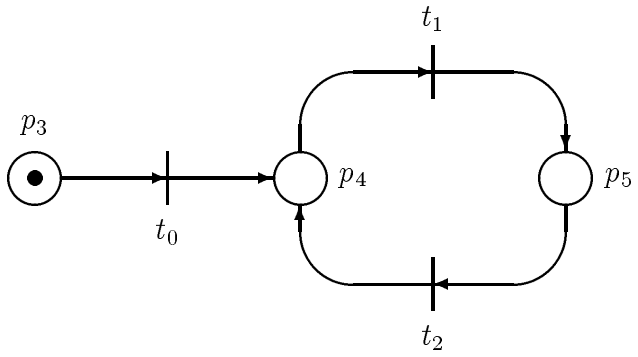


Figure 5.4 : Réseau de Petri R_σ.

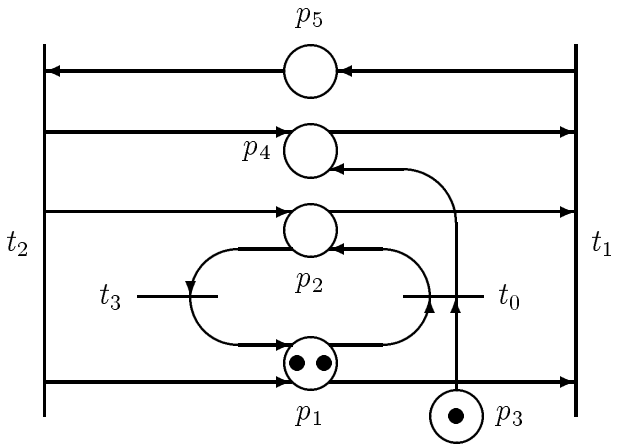


Figure 5.5 : Réseau de Petri R_c.

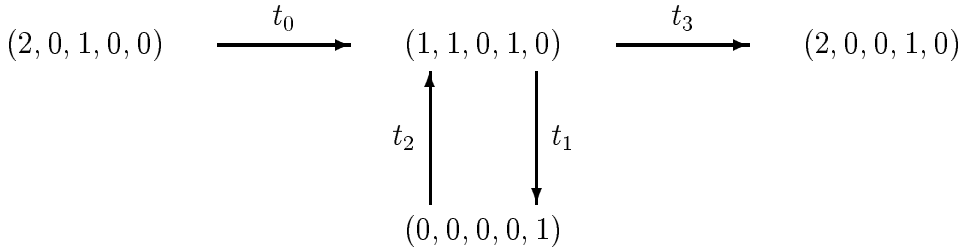


Figure 5.6 : Graphe de couverture du réseau R_c.

Construisons par la suite le réseau R_K tel que décrit dans la définition 5.1.22. Donnons d'abord le graphe de couverture du réseau R_c (figure 5.6). Dans ce cas-ci, le graphe de couverture coïncide avec le graphe d'accessibilité. En effet, le réseau ne contient pas de places non bornées (les marquages ne contiennent pas de ω). La transition t_3 amène à une situation de blocage. Par contre, l'exécution de la séquence infinie $t_0(t_1t_2)^\omega$ permet d'obtenir un régime permanent. L'ensemble $\hat{T}$ est, par conséquent, choisi comme suit :

$$\hat{T} \ = \ \{t_1, t_2\}.$$

L'ensemble des marquages m qui sont $\hat{T}$-continuel (définition 5.1.17) est

$$K \ = \ \{m_0 = (2,0,1,0,0); m_1 = (1,1,0,1,0); m_2 = (0,0,0,0,1)\} + \mathbf{N}^k.$$

Étant donné que K est un idéal, on peut appliquer l'algorithme donné dans la preuve du théorème 5.1.12 pour construire le résidu de K. On obtient :

$$res(K) \ = \ \{m_0, m_1, m_2\}.$$

Dans ce qui suit nous calculons le réseau contrôlé par K, $R_K = (P, T', W', m_0,)$ (voir définition 5.1.22). Rappelons que

$$\begin{aligned}
T' &= T_1 \cup T_2 \\
T_1 &= \left\{ t \in T \;\middle|\; \begin{array}{l} \forall m' \in res(K) : \exists m \in res(k) : \forall p \in P \\ \max(m'(p), W(p,t)) + W(t,p) \Leftrightarrow W(p,t) \geq m(p) \end{array} \right\} \\
T_2 &= \{t_m \mid t \in T \setminus T_1, m \in res(K)\}.
\end{aligned}$$

Calculons en premier lieu l'ensemble T' :

1. $t_0 \notin T_1$.
   En effet, on a
   $$\begin{aligned}
   max(m_1, W(., t_0)) + W(t_0, .) \Leftrightarrow W(., t_0) &= (1,1,1,1,0) + (\Leftrightarrow 1, 1, \Leftrightarrow 1, 1, 0) \\
   &= (0,2,0,2,0) \\
   &\not\geq m, \forall m \in res(K).
   \end{aligned}$$

2. $t_3 \notin T_1$.
   En effet, on a
   $$\begin{aligned}
   max(m_1, W(., t_3)) + W(t_3, .) \Leftrightarrow W(., t_3) &= (1,1,0,1,0) + (1, \Leftrightarrow 1, 0, 0, 0) \\
   &= (2,0,0,1,0) \\
   &\not\geq m, \forall m \in res(K).
   \end{aligned}$$

3. $t_1 \in T_1$ et $t_2 \in T_1$.
   Les deux tableaux suivants (tableau 5.1 et tableau 5.2) justifient l'appartenance de t_1 et t_2 à T_1. Pour des raisons de visibilité des tableaux, nous représentons un vecteur $(a_1, a_2, \cdots, a_n)$ par $a_1a_2\cdots a_n$.

m'	$W(.,t_1)$	$W(t_1,.)$	$\max(m', W(.,t_1)) + W(t_1,.) \Leftrightarrow W(.,t_1) \geq m$
$m_0 = 20100$			$21110 + 00001 \Leftrightarrow 11010 = 10101 \geq m_2$
$m_1 = 11010$	11010	00001	$11010 + 00001 \Leftrightarrow 11010 = 00001 \geq m_2$
$m_2 = 00001$			$11011 + 00001 \Leftrightarrow 11010 = 00002 \geq m_2$

Tableau 5.1 : Vérification de $t_1 \in T_1$.

m'	$W(.,t_2)$	$W(t_2,.)$	$\max(m', W(.,t_2)) + W(t_2,.) \Leftrightarrow W(.,t_2) \geq m$
$m_0 = 20100$			$20101 + 11010 \Leftrightarrow 00001 = 31110 \geq m_1$
$m_1 = 11010$	00001	11010	$11011 + 11010 \Leftrightarrow 00001 = 22020 \geq m_1$
$m_2 = 00001$			$00001 + 11010 \Leftrightarrow 00001 = 11010 \geq m_1$

Tableau 5.2 : Vérification de $t_2 \in T_1$.

Ainsi,

$$T_1 = \{t_1, t_2\}$$
$$T_2 = \{t_{0m_0}, t_{0m_1}, t_{0m_2}, t_{3m_0}, t_{3m_1}, t_{3m_2}\}.$$

Calculons maintenant les matrices $W'(.,t)$ et $W'(t,.)$.

1. Pour tout $t \in T_1$, on a $W'(.,t) = W(.,t)$ et $W'(t,.) = W(t,.)$.
2. Dans le tableau 5.3, nous calculons les vecteurs $W'(.,t)$ et $W'(t,.)$ pour tout $t \in T_2$.

t_m	m	$W(.,t)$	$W(t,.)$	$m \Leftrightarrow W(t,.) + W(.,t)$	$W'(.,t_m)$	$W'(t_m,.)$
t_{0m_0}	m_0			$3 \Leftrightarrow 12 \Leftrightarrow 10$	30200	21110
t_{0m_1}	m_1	10100	01010	20100	20100	11010
t_{0m_2}	m_2			$1 \Leftrightarrow 11 \Leftrightarrow 11$	10101	01011
t_{3m_0}	m_0			11100	11100	20100
t_{3m_1}	m_1	01000	10000	02010	02010	11010
t_{3m_2}	m_2			$\Leftrightarrow 11001$	01001	10001

Tableau 5.3 : Calcul des matrices $W'(t,.)$ et $W'(.,t)$.

Pour ne pas encombrer le schéma de la figure 5.7, nous avons utilisé des arcs à double sens lorsque le cas se présente. Nous espérons que cette représentation aide le lecteur à mieux comprendre le fonctionnement du réseau de Petri R_K.

À partir du marquage initial $m_1 = (2, 0, 1, 0, 0)$ la seule transition franchissable est t_{0m_1}. Par la suite, la boucle infinie $(t_1 t_2)^\omega$ sera itérée, ce qui nous donne un régime permanent.

Avant de terminer cet exemple, résumant ce que nous avons fait. À partir du réseau initial R, on peut tirer la séquence $\sigma_0 \sigma_c^\omega = t_0 (t_1 t_2)^\omega$. Ce réseau est sans blocage mais

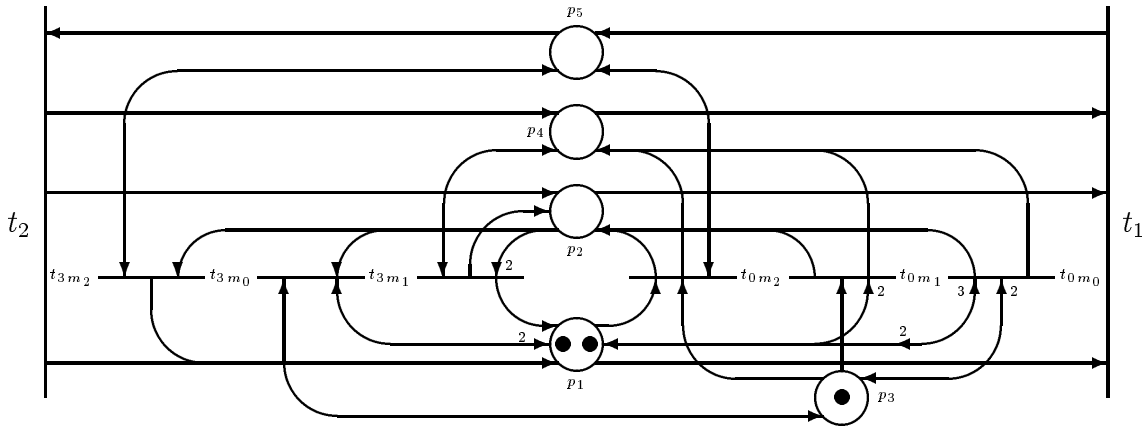


Figure 5.7 : Réseau de Petri R_K.

il n'est pas *V-Juste* puisque le tir de la séquence $(t_0t_3)^\omega \in L_\omega(R,h)$ rend la transition t_1 infiniment franchissable sans jamais être tirée. La composition du réseau R avec le réseau R_σ acceptant seulement la séquence $\sigma_0\sigma_c^\omega$ donne le réseau R_c. Ce réseau est *V-Juste* mais peut se trouver dans une situation de blocage. Il suffit de franchir la transition t_3. En effet, pour la séquence $s = t_0(t_1t_2)^\star t_3$, on obtient $m_0(s\rangle m$ avec $m = (2,0,0,0,0)$. Aucune transition n'est par la suite franchissable. En appliquant la définition 5.1.22 au réseau R_c, nous avons obtenu le réseau R_K qui est sans situation de blocage et *V-Juste*. La seule séquence infinie acceptée par le réseau R_K est $t_{0m_1}(t_1t_2)^\omega$. Ce réseau est différent de celui de [Uchihira90]. Dans cet article, les transitions t_{0m_0}, t_{0m_2}, t_{3m_0} et t_{3m_2} sont éliminées, puisqu'ils ne sont jamais franchis. Des arcs sont aussi éliminés. Cependant, les deux réseaux ont le même langage. □

5.2.7 Corollaire. *Soient (R,h) un réseau de Petri étiqueté et f une formule LTLP. Si $L(R,f,h) \neq \emptyset$, un réseau de Petri étiqueté (R_K,h') peut être construit en ajoutant certaines places, transitions et arcs au réseau R tels que R_K soit sans bloquage, (R_K,h') soit V-Juste et $L_{\lambda\omega}^{V\text{-eq}}(R_K,h') \subset L(R,f,h) \subset L_s(f)$.*

Preuve. Immédiate d'après les théorèmes 4.2.3 et 5.2.4. □

5.3 Synthèse compositionelle de programmes

La représentation graphique ainsi que le formalisme mathématique font des réseaux de Petri un bon modèle pour la description et la spécification des systèmes concurrents. Cependant, les réseaux de Petri de grande taille sont difficiles à manipuler, i.e. il est difficile de déterminer leur comportement ou de prouver quoique ce soit, et leur représentation graphique peut perdre beaucoup de sa clarté. D'où l'intérêt d'une construction modulaire des réseaux de Petri dans laquelle on décompose le système en un ensemble de composantes. Mieux encore, ces dernières, i.e. les composantes, peuvent être des objets déjà utilisés et validés que nous modifions légèrement afin de satisfaire

la spécification donnée [Uchihira90]. Le but principal d'une telle *réutilisation* est de réduire considérablement le coût du développement.

5.3.1 Structure du programme concurrent

Le programme que nous désirons construire consiste en un *contrôleur* et un ensemble d'*agents*. Le contrôleur contrôle séquentiellement chaque agent, tandis que les agents sont indépendants les uns des autres et peuvent ainsi s'exécuter en parallèle. Le contrôleur c et chaque agent a_i communiquent via un ensemble de canaux de communication synchrone C_i, comme en CCS [Milner89]. Nous supposons ici, que le contrôleur et les agents sont déjà construits à partir des composantes réutilisables mais qu'ils devront être modifiés pour satisfaire la spécification. Le contrôleur est représenté par un réseau de Petri $R_c = (P_c, T_c, W_c, m_{c_0})$ et l'agent a_i est représenté par $R_{a_i} = (P_{a_i}, T_{a_i}, W_{a_i}, m_{a_{i0}})$. Les canaux de communication C_i entre R_c et R_{a_i} sont définis comme suit :

$$C_i \quad = \quad \{h_i(t) \in T_c | t \in T_{a_i}\},$$

où $h_i : T_{a_i} \rightarrow T_c \cup \{\lambda\}$ est une fonction d'étiquetage, appelée *fonction de canal*, pour connecter les transitions de R_{a_i} avec celles de R_c. Si $t \in T_{a_i}$ et $h_i(t) = \lambda$ alors t est une transition interne et invisible. Elle ne peut être reliée à une transition de T_c.

5.3.2 Spécification logique temporelle

L'utilisateur spécifie un ensemble de contraintes par une formule LTLP f construite sur l'ensemble des propositions atomiques $Prop \subset T_c$. Le nouveau contrôleur synthétisé ainsi que les nouveaux agents doivent satisfaire f.

Ici, la synthèse du programme concurrent signifie la modification des composantes pour satisfaire la spécification (contraintes) f. Le programme de synthèse prend en entrée :

1. une spécification LTLP f,

2. des programmes réutilisables, i.e. un contrôleur, des agents et des canaux de communication (représentés par des réseaux de Petri $R_c, R_{a_1}, R_{a_2}, \cdots, R_{a_k}$ et des *fonctions de canal* $h_1, h_2, \cdots, h_k$).

Il donne en sortie des programmes synthétisés satisfaisant la spécification f, i.e. un contrôleur, des agents et des canaux de communication (représentés par des réseaux de Petri $R'_c, R'_{a_1}, R'_{a_2}, \cdots R'_{a_k}$ et des *fonctions de canal* $h'_1, h'_2, \cdots, h'_k$).

Nous donnons, dans ce qui suit, la procédure de synthèse du contrôleur puis celle des agents. Cette façon compositionelle est très pratique pour la synthèse des programmes de grande taille.

5.3.3 Synthèse du contrôleur

Nous présentons ci-dessous l'algorithme de synthèse du contrôleur. La première étape de l'algorithme consiste à réduire le plus possible chaque réseau R_{a_i} en un réseau $R_{a_i^r}$ comme c'est fait dans [Lee85]. Intuitivement, la réduction d'un réseau de Petri consiste en la transformation d'un réseau original en un réseau réduit tout en préservant certaines propriétés désirées. L'espace des états accessibles est, par conséquent, réduit et l'analyse du nouveau réseau peut donner assez d'informations pour déduire le comportement du réseau original. Plusieurs méthodes ont été déjà étudiées. On distingue principalement les notions de *macro-place* et de *macro-transition*, qui consistent à remplacer un sous-réseau du réseau original par une place (*macro-place*) ou une transition (*macro-transition*). Le problème complexe de réduction des réseaux de Petri n'a pas été présenté dans cet ouvrage, par crainte d'appesantir le mémoire par des détails qui peuvent nous éloigner du sujet principal. Nous pensons quand même que la méthode est compréhensible malgré ce manque. Cependant, l'implantation de l'algorithme nécessite évidemment l'exploration du problème.

Algorithme de synthèse du contrôleur

1. Réduire chaque réseau de Petri R_{a_i} d'un agent a_i en $R_{a_i^r}$ [Lee85] tel que $L_\omega(R_{a_i}, h_i) = L_\omega(R_{a_i^r}, h_i)$.

2. Construire le réseau de Petri suivant :

$$(R, h) \;=\; (\cdots(((R_c, I)|_{C_1}(R_{a_1^r}, h_1))|_{C_2}(R_{a_2^r}, h_2))|_{C_3}\cdots|_{C_k}(R_{a_k^r}, h_k))$$

   où $|_{C_i}$ est l'opérateur de composition des réseaux de Petri par rapport à C_i (voir définition 2.3.7). Nous abrégeons cette composition par $Sync(R_c, R_{a_1^r}, R_{a_2^r}, \cdots, R_{a_k^r}, h_1, h_2, \cdots, h_k)$, puisqu'elle représente la synchronisation des processus avec les canaux.

3. Si $L(R, f, h/Prop) = \emptyset$ alors

   Afficher "la synthèse est impossible"

   sinon

   construire une séquence infinie de tirs $\theta = \theta_0\theta_c^\omega$ sur R telle que $h/Prop(\theta_0\theta_c^\omega) \in L(R, f, h/Prop)$ (théorème 4.2.3).

4. Construire un réseau de Petri R'_c tel que $L_\omega(R'_c, I) = \{h(\theta_0\theta_c^\omega)\}$. □

Comme à la remarque 5.2.5, nous suggerons que $L_\omega(R'_c, I) = \{h(\theta_0\theta_c^\omega)\}$ à l'opposé de ce qu'on trouve dans [Uchihira90], i.e. $L(R'_c, I) = \{h(\theta_0\theta_c^\star)\}$. Noter que $h(\theta_0\theta_c^\omega) \in T_c^\omega$ et que R'_c, le réseau de Petri du contrôleur synthétisé, est un programme séquentiel déterministe. Il est séquentiel parce qu'il ne contient pas de transitions qui s'exécutent en parallèle et, est déterministe car à partir de tout marquage accessible, une et une seule transition est franchissable.

5.3.4 Synthèse d'agent

Pour chaque agent, nous pouvons construire un réseau de Petri modifié $R'_{a_i} = (P'_{a_i}, T'_{a_i}, W'_{a_i}, m'_{a_{i0}})$ et une fonction d'étiquetage $h'_i : T'_{a_i} \to C_i$, à partir de $R_{a_i^r}$ et $\theta_0\theta_c^\omega$, tels que R'_{a_i} soit sans blocage, (R'_{a_i}, h'_i) soit V_i-*Juste* et $L_{\lambda^\omega}^{V_i\text{-eq}}(R'_{a_i}, h'_i) = \{h(\theta_0\theta_c^\omega)/C_i\}$ (théorème 5.2.4).

5.3.1 Exemple. [Uchihira90] Soit un programme concurrent (figure 5.8) composé :

- d'un contrôleur (figure 5.9) représenté par le réseau de Petri $R_c = (P_c, T_c, W_c, m_{c_0})$ avec $T_c = \{\mathit{début}_1, \mathit{fin}_1, \mathit{début}_2, \mathit{fin}_2\}$,
- de deux agents (figure 5.9) représentés par les réseaux de Petri R_{a_1} et R_{a_2} avec $R_{a_i} = (P_{a_i}, T_{a_i}, W_{a_i}, m_{a_{i_0}})$, $T_{a_i} = \{\mathit{début}, \mathit{fin}, \mathit{Trop\text{-}plein}\}$ et $V_i = \{\mathit{début}, \mathit{fin}\}$ pour $i = 1, 2$.

Dans la figure 5.9, nous avons entouré chaque composante, i.e. le contrôleur et l'agent, par une boucle extérieure. Les lignes fines reliant le contour de l'agent à des transitions indiquent que seules ces transitions sont reliées à des transitions du contrôleur par la *fonction canal* h_i. Les canaux de communication sont définis par :

$$h_i(\mathit{début}) = \mathit{début}_i, \quad h_i(\mathit{fin}) = \mathit{fin}_i, \quad h_i(\mathit{Trop\text{-}plein}) = \lambda,$$

d'où

$$C_i = \{\mathit{début}_i, \mathit{fin}_i, \lambda\}.$$

L'ensemble des propositions $Prop$ est égal à l'ensemble des transitions T_c

$$Prop = T_c = \{\mathit{début}_1, \mathit{fin}_1, \mathit{début}_2, \mathit{fin}_2\}.$$

La formule LTLP f est donnée par :

$$\begin{aligned} f = {} & \Box(\mathit{début}_1 \to \bigcirc(\neg \mathit{début}_2 \,\mathcal{U}\, \mathit{fin}_1) \wedge \\ & \Box(\mathit{début}_2 \to \bigcirc(\neg \mathit{début}_1 \,\mathcal{U}\, \mathit{fin}_2). \end{aligned}$$

Elle signifie qu'à chaque fois que l'agent a_1 franchit $\mathit{début}_1$, l'agent a_2 ne peut franchir $\mathit{début}_2$ avant que l'agent a_1 ne franchisse fin_1 et vice versa.

5.3.2 Remarque. Nous avons choisi de placer toutes les figures de cet exemple à la fin, pour faciliter leur consultation. □

Synthèse du contrôleur

1. Dans notre exemple, les réseaux de Petri R_{a_i} ne contiennent pas de sous-réseaux complexes et par conséquent ne peuvent subir de réductions, donc $R_{a_i} = R_{a_i^r}$.
2. Le réseau $(R, h) = ((R_c, I)|_{C_1}(R_{a_1}, h_1))|_{C_2}(R_{a_2}, h_2))$ est représenté par la figure 5.10.

3. Construisons l'automate de Büchi acceptant la formule f (figure 5.11) et le graphe de couverture étendu G (figure 5.12) comme indiqué aux théorèmes 3.2.1 et 4.2.2.

4. Cherchons un cycle c du graphe G contenant un noeud désigné et tel que $\Delta(c) \geq 0$. Posons c_{q_i} le cycle passant par q_i. La séquence $c^{\omega}_{q_3}$ est une suite infinie de transitions *début*$_1$, *fin*$_1$ et *Trop-plein*. Il est clair que cette suite de transitions n'est pas une séquence équitable vis-à-vis de V_2, puisqu'elle prive les séquences visibles *début*$_2$ et *fin*$_2$ d'être tirées. D'une façon symétrique, $c^{\omega}_{q_4}$ n'est pas V_1-*Juste*. Par conséquent, nous ne considérons que le cycle $c_{q_6} = (\textit{début}_1, \textit{fin}_1, \textit{début}_2, \textit{fin}_2)$. Notons que $\Delta(c_{q_6}) \geq 0$ et que $c^{\omega}_{q_6}$ est une séquence licite, V_1-*Juste* et V_2-*Juste*. Trouvons maintenant un chemin d partant du noeud initial q_0 jusqu'au noeud q_6. Nous pouvons choisir

   $$d \;=\; \textit{début}_1, \textit{fin}_1, \textit{début}_2, \textit{fin}_2.$$

   Ainsi, nous pouvons construire d'une manière déterministe la séquence de transitions licite $\theta_0\theta_c^{\omega}$ à partir du chemin $dc^{\omega}_{q_6}$ et telle que $h/Prop(\theta_0\theta_c^{\omega}) = \theta_0\theta_c^{\omega} \in L(R, f, h)$.

   $$\begin{aligned}\theta_0\theta_c^{\omega} &= (\textit{début}_1, \textit{fin}_1, \textit{début}_2, \textit{fin}_2)(\textit{début}_1, \textit{fin}_1, \textit{début}_2, \textit{fin}_2)^{\omega} \\ &= (\textit{début}_1, \textit{fin}_1, \textit{début}_2, \textit{fin}_2)^{\omega}.\end{aligned}$$

5. Nous pouvons enfin construire le réseau de Petri R'_c du contrôleur synthétisé tel que $L_{\omega}(R'_c, I) = \{h(\theta_0\theta_c^{\omega})\}$. La figure 5.13 représentant le réseau R'_c est tirée de [Uchihira90].

Synthèse d'agent

Appliquons le théorème 5.2.4 pour la construction du réseau de Petri étiqueté (R'_{a_i}, h'_i). La figure 5.14 montre la composition $(R_{a_i^r}, h_i)|_{c_i}(R_{\theta}, I)$ telle que $L_{\omega}(R_{\theta}, I) = \{(\textit{début}_i\, \textit{fin}_i)^{\omega}\}$ et R_{θ} est le réseau acceptant la séquence $\theta = \theta_0\theta_c^{\omega}$ avec $\theta_0 = \emptyset$ et $\theta_c = \textit{début fin}$. Le graphe de couverture du réseau R'_{a_i} est représenté par la figure 5.15. L'ensemble $\hat{T}$ est donné par :

$$\hat{T} \;=\; \{\textit{début}_i, \textit{fin}_i\}.$$

Le résidu de l'ensemble des marquages $K = \{m_1, m_2, m_3, m_4\} + \mathbf{N}^k$ est calculé et donne

$$res(K) \;=\; \{m_1, m_2\}.$$

Par la suite, nous obtenons $\textit{début}_i \in T_1$, $\textit{fin}_i \in T_1$ et *Trop-plein* $\notin T_1$. La transition *Trop-plein* se décompose alors en *Trop-plein*$_{m_1}$ et *Trop-plein*$_{m_2}$. Les vecteurs d'incidence de ces deux transitions sont donnés par :

$$\begin{aligned}&W'(., \textit{Trop-plein}_{m_1}) = (1, 0, 3), \quad W'(., \textit{Trop-plein}_{m_2}) = (0, 1, 5) \\ &W'(\textit{Trop-plein}_{m_1}, .) = (1, 0, 0), \quad W'(\textit{Trop-plein}_{m_2}, .) = (0, 1, 2).\end{aligned}$$

La représentation de l'agent synthétisé est donnée à la figure 5.16. □

Le réseau R'_{a_i} obtenu est aussi légèrement différent de celui de [Uchihira90]. Nous avons calculé

$$W'(., \textit{Trop-plein}_{m_2}) \;=\; (0,1,5) \;\text{ et }\; W'(\textit{Trop-plein}_{m_2}, .) \;=\; (0,1,2)$$

alors que dans [Uchihira90],

$$W'(., \textit{Trop-plein}_{m_2}) \;=\; (0,1,4) \;\text{ et }\; W'(\textit{Trop-plein}_{m_2}, .) \;=\; (0,1,1).$$

Nous pensons que dans [Uchihira90], les auteurs ont procédé en plus à des réductions sans les mentionner. Dans les deux réseaux, i.e. notre réseau et celui de [Uchihira90], le tir de la transition $\textit{Trop-plein}_{m_2}$ consomme 3 jetons. Cependant, pour notre réseau, on doit avoir $m(p_3) \geq 5$ alors que celui de [Uchihira90], on doit avoir $m(p_3) \geq 4$.

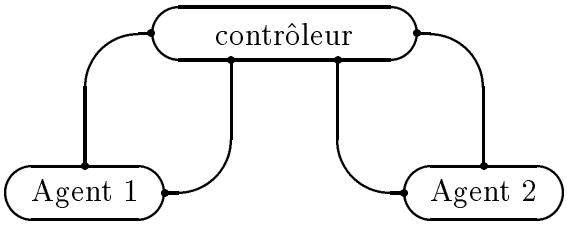


Figure 5.8 : Structure du programme concurrent.

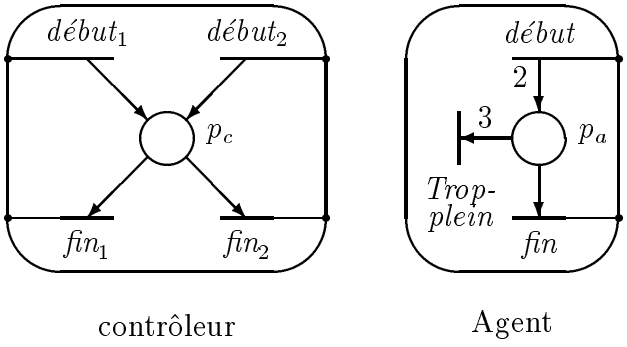


Figure 5.9 : Réseaux de Petri du contrôleur et d'un agent.

5.3.3 Théorème. [Uchihira90] *Si les réseaux de Petri étiquetés* (R'_c, I), (R'_{a_1}, h'_1), (R'_{a_2}, h'_2), $\cdots$, (R'_{a_k}, h'_k), *peuvent être synthétisés suivant la méthode de synthèse présentée ci-dessus à partir des réseaux* $R_c, R_{a_1}, R_{a_2}, \cdots, R_{a_k}$, *des fonctions de canal* $h_1, h_2, \cdots, h_k$, *et d'une fonction logique temporelle* f, *alors le réseau de Petri étiqueté* $(R', h') = \textit{Sync}(R'_c, R'_{a_1}, R'_{a_2}, \cdots, R'_{a_k}, h'_1, h'_2, \cdots, h'_k)$ *est sans blocage, V-Juste et* $L^{\text{V-eq}}_{\lambda\omega}(R', h'/Prop) \subseteq L(R, f, h/Prop) \subseteq L_s(f)$ *sous la condition de V-équité, où* $(R, h) = \textit{Sync}(R_c, R_{a_1}, R_{a_2}, \cdots, R_{a_k}, h_1, h_2, \cdots, h_k)$.

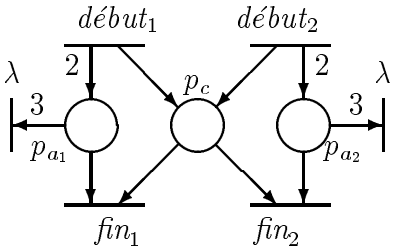


Figure 5.10 : Réseau de Petri composé R.

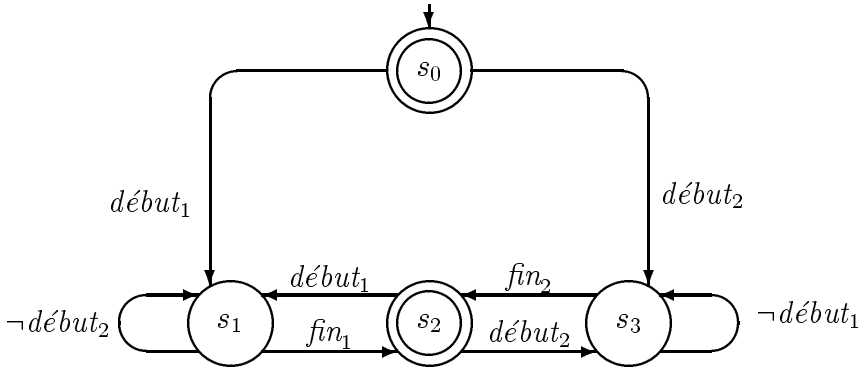


Figure 5.11 : Automate de Büchi.

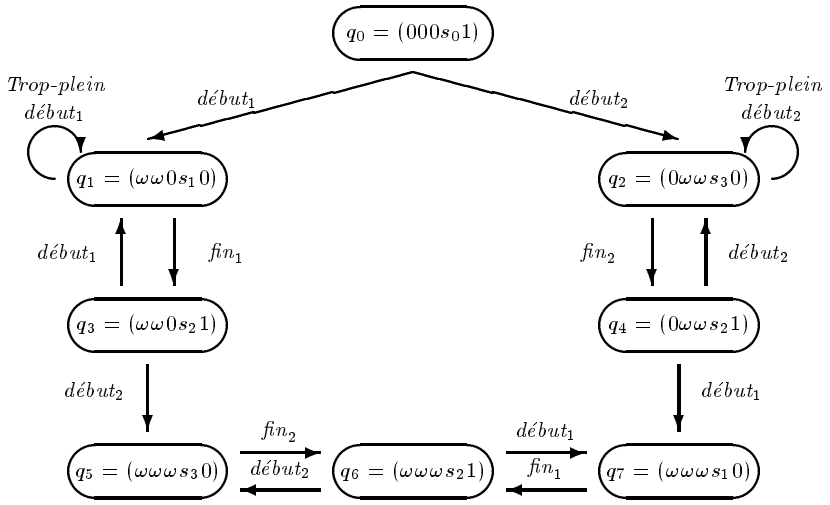


Figure 5.12 : Graphe de couverture étendu G.

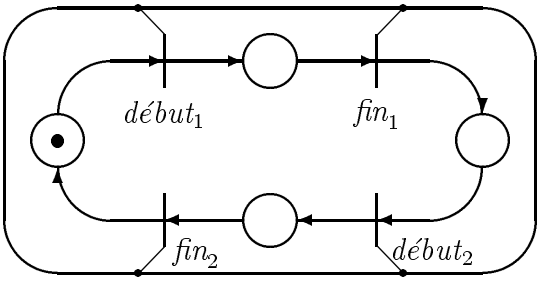


Figure 5.13 : Réseau de Petri du contrôleur synthétisé.

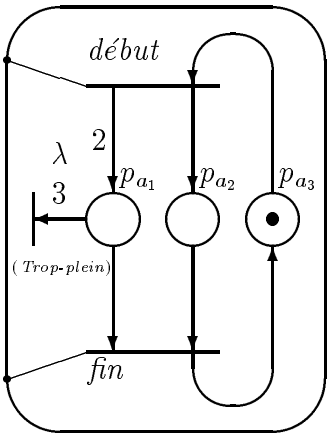


Figure 5.14 : Réseau de Petri d'un agent synthétisé.

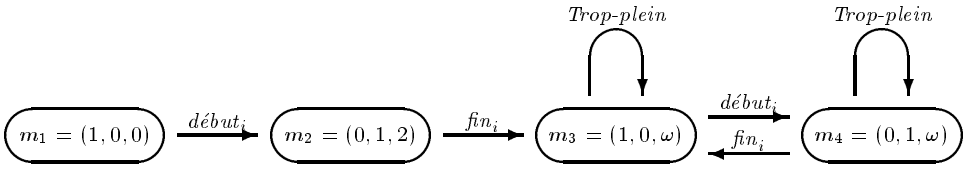


Figure 5.15 : Graphe de couverture du réseau composé $(R_{a_i^r}, h_i)|_{c_i}(R_\theta, I)$.

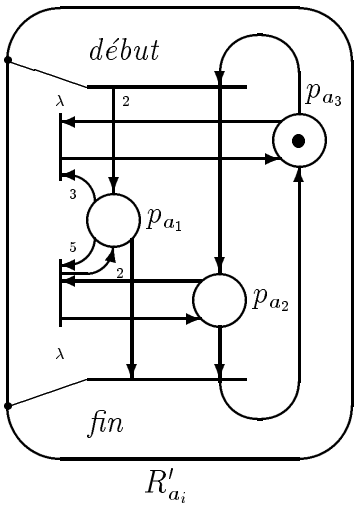


Figure 5.16 : Réseau de Petri d'un agent synthétisé R'_{a_i}.

Preuve. Elle s'ensuit directement des méthodes utilisées dans le théorème 4.2.2 et théorème 5.2.4. □

Nous avons présenté dans ce chapitre une méthode qui, à partir de programmes représentés par des réseaux de Petri, construit des programmes (réseaux de Petri) satisfaisant des formules logiques temporelles. Ces réseaux sont le résultat d'une modification des réseaux initiaux. La méthode peut être utilisée soit pour un simple réseau soit pour un système composé d'un ensemble de réseaux reliés par des *fonctions de canal*.

Chapitre 6

Conclusion

Ce mémoire avait pour objectif essentiel d'exposer et d'éclaircir l'article d'Uchihira [Uchihira90]. L'intérêt porté à un tel article est principalement dû aux points suivants :

1. Le nombre minime de travaux traitant de la combinaison des réseaux de Petri et de la logique temporelle.
2. La complexité et la puissance des techniques utilisées.
3. La décidabilité de la procédure de vérification de programmes concurrents.
4. L'approche compositionnelle de la synthèse de programmes concurrents.
5. La réutilisation des programmes.

La construction d'un automate de Büchi $\mathcal{B}_f$, dont le langage accepté $L(\mathcal{B}_f)$ est exactement l'ensemble des séquences satisfaisant la formule f, constitue notre première difficulté rencontrée. En effet, l'automate obtenu n'est pas minimal. Des recherches bibliographiques nous ont permis de constater que, malheureusement, une technique de minimisation des automates de Büchi n'existe que dans le cas déterministe. Elle est essentiellement la même que celle des automates finis sur les mots finis. Toutefois, dans le cas non déterministe, cette méthode ne peut fonctionner. Le problème vient du fait que la classe des langages reconnaissables par les automates de Büchi déterministes ne coïncide pas avec la classe des langages ω-réguliers [Leeuwen90]. Un peu d'ingéniosité nous a permis d'établir quelques règles de réduction de l'automate. Cependant, la manipulation de ces règles nécessite beaucoup d'attention. Rappelons que le nombre d'états de l'automate est de l'ordre de $2^{O(|f|)}$. L'implantation d'une telle procédure est possible tant que la taille de la formule f n'est pas très grande. Cependant, l'absence d'une méthode de minimisation peut rendre difficile l'étude et l'utilisation de l'automate.

Au chapitre 4, nous avons présenté des correspondances entre les propositions atomiques de la LTLP et les réseaux de Petri. Ces correspondances sont nécessaires pour la vérification des réseaux de Petri. Les propriétés (ou contraintes) à vérifier sont exprimées par des formules logiques temporelles. Nous avons discuté de certaines classes trouvées dans la littérature. Ces classes sont soit indécidables à l'égard du problème de vacuité soit décidables mais leur complexité est équivalente à celle du problème d'accessibilité, ce qui est presque impraticable. Néanmoins, la classe étudiée dans cet ouvrage

est décidable et sa complexité est équivalente à celle du problème de couverture qui est de loin moins complexe que le problème d'accessibilité.

La procédure de vérification présentée dans [Uchihira90] est très élégante. Elle permet non seulement de décider si le réseau de Petri vérifie la propriété donnée, mais, si c'est le cas, elle calcule en plus une séquence de transitions infinies licite et vérifiant la propriété. Certaines propriétés ne peuvent être vérifiées par cette méthode, telle que la possibilité d'atteindre une situation de blocage. En contrepartie, des propriétés comme l'ordre partiel et l'exclusion mutuelle, qui ne sont généralement pas couvertes par l'analyse traditionnelle, peuvent être étudiées par la méthode. Cette dernière remarque montre que la méthode n'est pas complète. Toutefois, elle peut être utilisée pour compléter l'analyse traditionnelle. Par exemple, si nous nous restreignons aux réseaux de Petri bornés, la vérification devient simple [Uchihira90b, Katai82]. Mais un tampon non borné est souvent nécessaire pour spécifier un programme concurrent. Lors de l'analyse des programmes réels, on suppose que le tampon est borné mais avec une capacité aussi large que nécessaire. Remarquons cependant qu'avec la méthode présentée dans cet ouvrage, un réseau de Petri borné à grande capacité aura un graphe de couverture plus large que celui du réseau de Petri non borné.

Dans la dernière partie, une étude de l'article de Valk et Jantzen [Valk85] était nécessaire pour la compréhension de la méthode de synthèse d'Uchihira. Nous avons beaucoup apprécié cet article très technique, basé sur une approche purement mathématique, à savoir les vecteurs entiers et le calcul des résidus. L'idée principale est de modifier un réseau de Petri initial pour satisfaire certaines propriétés et ce, en ajoutant des transitions et des arcs au réseau. L'ensemble P des places reste le même. L'utilisation de la logique temporelle comme langage de spécification différencie la méthode de Uchihira de celle de Valk et Jantzen. La méthode de Uchihira a en plus l'avantage de permettre une approche compositionnelle.

L'algorithme de synthèse a déjà été programmé par l'équipe de Uchihira dans le cadre du projet MENDELS ZONE [Uchihira90b], pour des réseaux de Petri borné. MENDELS ZONE est un système de synthèse de programmes concurrents utilisant la logique temporelle et les réseaux de Petri.

Finalement, mentionons que le contenu du mémoire simplifie beaucoup la tâche pour un usager qui voudrait implanter les algorithmes de vérification et de synthèse (beaucoup plus de détails que l'article d'Uchihira). Une telle implantation fournit un outil pour juger l'efficacité et les limites de la méthode.

Bibliographie

[Alpern89] B. Alpern et F. B. Schneider, Verifying temporal properties without temporal logic, *ACM Trans. on Programming Languages and Systems*, vol. 11, no. 1, janvier 1989, p. 147–167.

[Arnold92] A. Arnold, J. B. Beauquier, B. Bérard et B. Rozoy, *Programmes parallèles, modèles et validation,* Armand Colin, 1992.

[Audureau90] E. Audureau, P. Enjalbert et L. F. Del Cerro, *Logique temporelle : sémantique et validation de programmes parallèles,* Masson, 1990.

[Autebert94] J. M. Autebert, *Théorie des langages et des automates,* Masson, 1994.

[Brams83] G. W. Brams, *Réseaux de Petri : théorie et pratique,* Masson, 1983.

[Büchi62] J. R. Büchi, On a decision method in restricted second-order arithmetic, *Proc. Internat. Congr. Logic. Method and Philos. Sci. 1960.* Stanford University Press, 1962, p. 1–12.

[Cherkasova87] L. A. Cherkasova et V. E. Kotov, The undecidability of propositional temporal logic problems of Petri nets, *Computers and aAtificial Intelligence*, vol. 6, no. 2, 1987, p. 123–130.

[Choueka74] Y. Choueka, Theories of automata on ω-tapes : a simplified Approch, *J. Comput. System Sci.*, vol. 8, 1974, p. 117–141.

[Clarke82] E. M. Clarke and E. A. Emerson, Design and synthesis of synchronisation skeletons using branching time temporal logic, *Lecture Notes in Computer Science 131*, Springer-Verlag, 1982, p. 52–71.

[Conway71] J. H. Conway, *Regular algebra and finite machines,* Chapman and Hall (1971).

[Gjalt91] G. de Jong Gjalt, An automata theoretic approch to temporal logic, *CAV 91, Lecture Notes in Computer Science 575*, Springer-Verlag, juillet 1991, p. 477–487.

[Hoare78] C. A. R. Hoare, Communicating Sequentiel Processes, *Communications ACM*, vol. 21 no. 8, 1987, p. 666–677.

[Howell88] R. R. Howell, L. E. Rosier et H. C. Yen, A taxonomy of fairness and temporal logic problems for Petri nets, *Lecture Notes in Computer Science 324,* Springer-Verlag 1988, p. 351–359.

[Karp69] R. M. Karp et R. E. Miller, Parallel program schemata, *Journal of Computing System Science.*, vol. 4, 1969, p. 147–195.

[Katai82] O. Katai and S. Iwai, Construction of scheduling rules for asynchronous, concurrent systems based on tense logic, *Trans. of SICE*, vol. 18, no. 12, 1982.

[Kosaraju82] S. R. Kosaraju, Decidability of reachability in vector addition systems, *Proceedings of the 14th ann. ACM STOC*, 1982.

[Leeuwen90] J. V. Leeuwen, *Handbook of theoretical computer science, vol B : Formal models and semantics,* The MIT Press/Elsevier, 1990.

[Lee85] K. H. Lee et J. Favrel, Hierarchical reduction method for analysis and decomposition of Petri nets, *IEEE Trans. Syst., Man. and Cybern., SMC-15*, no. 2, 1982, p. 272–280.

[Lloret90] J. C. Lloret, P. Azema et F. Vernadat, Compositional design and verification of communication protocols using labelled Petri nets, *CAV'90, Lecture Notes in Computer Science 531*, Springer-Verlag, 1990, p. 96–105.

[Manna84] Z. Manna et A. Pnueli. Synthesis of communicating processes from temporal logic specification, *ACM Trans. Program. Lang. and Syst.*, vol 6, no. 1, 1984, p. 68–93.

[Manna87] Z. Manna et A. Pnueli, Specification and verification of concurrent programs by $\forall$-automata, *Lecture Notes in Computer Science,* avril 1987, p. 124–164.

[Manna91] Z. Manna et A. Pnueli, *The temporal logic of reactive and concurrent systems : specification,* Springer-Verlag, 1991.

[Mayr84] E. Mayr, An algorithm for the general Petri net reachability problem, *SIAM Journal of computation*, vol. 13, no. 3, 1984.

[McNaughton66] R. McNaughton, Testing and generating infinite sequences by a finite automaton, *Inform. and Control*, vol. 9, 1966, p. 521–530.

[Milner89] R. Milner. *Communication and concurrency,* Prentice-Hall, 1989.

[Nivat77] M. Nivat, Mots infinis engendrés par une grammaire algébrique, *RAIRO informatique théorique*, vol. 11, no. 4, 1977, p. 46–57.

[Petri62] C. A. Petri. Kommunikation Mit Automaten, Institute für instrumentelle mathematik, schriften des IMM, 1962.

[Pnueli77] A. Pnueli, The temporal logic of programs, *Proc* 18^{th} *Ann. Aymp. Foundations of Computer Science,* Providence, RI, New York, IEEE, 1977, p. 46–57.

[Rabin59] M. O. Rabin et D. Scott, Finite automata and their decision problems, *IBM J. Res. Develop.*, vol. 3, 1959, p. 114–125.

[Rabin69] M. O. Rabin, Decidability of second-order theories and automata on infinite trees, *Trans. Amer. Math. Soc.*, vol. 141, 1969, p. 1–35.

[Reisig85] W. Reisig, *Petri nets : an introduction,* EATCS monographs on theoretical computer science, Springer-Verlag, 1985.

[Sistla85] A. P. Sistla, M. Y. Vardi et P. Wolper, The complementation problems for Büchi automata with applications to temporal logic, *Lecture Notes in Computer Science 194,* Springer-Verlag, Juillet 1985, p. 465–474.

[Suzuki89] I. Suzuki et H. Lu, Temporal Petri nets and their application to modelling and analysis of a handshake daisy chain arbiter, *IEEE Trans. Comput.*, vol. 38, no. 5, 1989, p. 696–704.

[Uchihira90] N. Uchihira et S. Honiden, Verification and synthesis of concurrent programs using Petri nets and temporal logic, *The Transactions of the IEICE*, vol. E73, no. 12, decembre 1990, p.2001–2010.

[Uchihira90b] N. Uchihira et al., Concurrent program synthesis : automated reasoning complements software reuse, *IEEE Proc. of 23rd Hawaii International Conference on Systems Science,* 1990, p. 64–73.

[Valk82] R. Valk, Infinite behaviour of Petri nets, *Theoret. Comput. Sci.*, vol. 25, 1983, p. 311–341.

[Valk85] R. Valk et M. Jantzen, The residue of vector sets with applications to decidability problems in Petri nets, *Acta Informatica*, vol. 21, 1985, p. 643–674.

[Vardi86] M. Y. Vardi. et P. Wolper, An automata theoretic approch to automatic program verification, *IEEE Symp. on Logic in Computer Science*, 16-18 juin 1986, p. 332–344.

[Vidal92] G. Vidal-Naquet et A. Choquet Geniet, *Réseaux de Petri et systèmes parallèles,* Armand Colin, 1992.

[Wolper83a] P. Wolper, Temporal logic can be more expressive, *Information and Control*, vol. 56, 1983, p. 72–99.

[Wolper83b] P. Wolper, M. Y. Vardi et A. P. Sistla, Reasoning about infinite computation paths, *Proc.* 24^{th} *Ann. Aymp. on Foundations of Computer Science, Tucson, AZ*, 7-9, novembre 1983, p. 185–193.

[Wolper87] P. Wolper, On the relation of programs and computations to models of temporal logic, *Lecture Notes in Computer Science 398,* Springer-Verlag, 1987, p. 75–123.