Contribution au dépliage des réseaux de Petri et à l'analyse des processus de branchement.

Maurice Comlan

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Maurice COMLAN

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
Docteur de l'Université d'Abomey-Calavi
sous le sceau de l'Université Bretagne Loire

École doctorale : Sciences et Technologies de l'Information, et Mathématiques

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN)

Soutenue le 03 novembre 2016

Contribution au dépliage des réseaux de Petri et à l'analyse des processus de branchement.

JURY

Président :	**M. Christian ATTIOGBÉ**, Professeur des Universités, Université de Nantes
Rapporteurs :	**M. Patrick MARTINEAU**, Professeur des Universités, Université de Tours
	M. Eric NIEL, Professeur des Universités, INSA Lyon
Examinateur :	**M. Florent FRIZON DE LAMOTTE**, Professeur des Universités, Université de Bretagne Sud
Invités :	**M. Sébastien LAHAYE**, Professeur des Universités, Université d'Angers
	M. David DELFIEU, Maître de conférences, Polytech'Nantes - IRCCyN
	* * *
Directeurs de thèse :	**M. Olivier H. ROUX**, Professeur des Universités, Ecole Centrale Nantes - IRCCyN
	M. Antoine VIANOU, Professeur des Universités, UAC/EPAC (Bénin)

A toi l'enfant que je ne connaîtrai jamais . . .

Remerciements

Les travaux présentés dans ce mémoire ont été effectués au sein de l'Institut de Recherche en Communication et Cybernétique de Nantes (IRCCyN, UMR CNRS 6597) en France et au sein du Laboratoire d'ElectroTechnique, de Télécommunications et d'Informatique Appliquée (LETIA) au Bénin. Je remercie les différents responsables et tout le personnel de ces deux laboratoires pour m'avoir accueilli, pour avoir mis à ma disposition les ressources nécessaires et pour leurs disponibilités.

Je ne sais pas si des remerciements, même les plus sincères et les plus profonds, suffiront pour exprimer ma reconnaissance à Monsieur David Delfieu sans qui cette thèse ne serait possible car au delà d'un simple encadrant, tu as été un ami. Tu as accepté de proposer et d'encadrer mon sujet de DEA malgré la distance, puis le sujet de thèse. Je tiens alors à te remercier pour tes compétences, ton soutien, ta disponibilité, tes conseils et pour m'avoir conduit et guidé dans le domaine de la recherche. Aussi je tiens à remercier Monsieur Olivier H. Roux pour avoir accepté diriger mes travaux en France, ta simplicité et ta rigueur m'ont beaucoup aidé. Nos différentes discussions ont toujours été riches et m'ont éclairé. De même, je remercie Monsieur Antoine Vianou mon directeur de thèse à l'Université d'Abomey-Calavi (Bénin) et Monsieur Médésu Sogbohossou pour avoir joué l'intermédiaire avec David et pour tes conseils.

Je tiens à remercier tout particulièrement Monsieur Patrick Martineau et Monsieur Eric Niel qui ont accepté lire ce manuscrit et en être les rapporteurs. Merci aussi à Monsieur Christian Attiogbé pour avoir présidé mon jury de soutenance et à Messieurs Florent de Lamotte et Sébastien Lahaye pour avoir participer au jury.

Je n'oublie pas tous les doctorants de l'IRCCyN et du LETIA que j'ai côtoyés et sur qui j'ai pu compter : Toussaint, Franck, Louis, Patrick, Sibiath, Romanus, Karamatou, Louis-Mari, Armel, Géraud . . . La liste est trop longue pour tous les citer.

Merci à mes parents, à toutes mes sœurs et à Anne pour leur amour, leur compréhension et leur tolérance : ce travail est le vôtre. Un merci particulier

à Gisèle pour ce que tu es.

Un merci à tout le personnel de TBC Sarl : Appolinaire, Gamaliel, Nasirou, ... pour avoir créé pour moi une seconde famille. Un merci spécial à Appolinaire Dossa pour avoir été le grand frère que je n'ai pas eu, à Collins pour avoir été un deuxième papa pour moi, et à tous mes amis.

Merci particulier à Rodolphe, Kazesse, Gildas et à tous ceux que j'ai rencontrés et dont je n'ai pas cité les noms, je vous dis MERCI.

Table des matières

Liste des tableaux

Liste des figures

1

Introduction générale

Sommaire

1.1 Contexte général

Les systèmes temps réel, apparaissent aujourd'hui, de façon omniprésente dans l'industrie et dans notre vie de tous les jours. Ils se déclinent en diverses technologies : des automatismes industriels dans les systèmes de contrôle de procédés, des systèmes à micro-contrôleurs dans nos voitures ou de puissants microprocesseurs dans nos smartphones ou nos tablettes.

De plus, cherchant à créer de la valeur et des services, les objets ou entités de notre quotidien (ordinateurs, téléphones, frigos, télévisions, voitures, maisons ...) deviennent communicants. Ils *interagissent* avec leur environnement. Chaque entité peut parfois fonctionner de manière autonome des autres entités tout en leur fournissant à des instants précis un ou plusieurs services. Une entité ne possède pas un temps illimité pour fournir le service : elle doit le faire suivant un délai ou dans un intervalle de temps qui lui est imparti.

Ces interactions imposent non seulement des contraintes fonctionnelles (justesse des résultats calculés) mais aussi temporelles (enchaînement correct des évènements) et également temps réel (dates d'arrivée des évènements). Si ces interactions sont gérées par le système en harmonie avec le procédé avec lequel il interagit on dit que le système est "temps réel" :

> "*En informatique temps réel, le comportement correct d'un système dépend, non seulement des résultats logiques des traitements, mais aussi du temps auquel les résultats sont produits : un résultat juste mais hors délai est un résultat faux.*" [75]

De l'utilisation des applications temps réel découlent des problèmes spécifiques que nous pouvons classer en deux catégories : l'utilisation de ressources communes et les problèmes temporels [2].

Partage de ressources communes.

L'utilisation d'une même ressource par plusieurs entités peut entrainer des conflits d'accès, auxquels il faut apporter des solutions. Plus particulièrement, il s'agit de garantir : l'exclusion mutuelle à la ressource partagée, l'absence de blocage ou éviter le problème de famine, ...

Contraintes temporelles.

Nous pouvons classer les contraintes temporelles en deux familles : les contraintes temporelles *qualitatives* et *quantitatives*.

- Les contraintes temporelles qualitatives découlent de la communication entre les tâches et s'expriment par des relations de précédence, de synchronisation ou de parallélisme. On veut observer des évènements suivant une planification ou un ordonnancement précis. Les mécanismes de base tels que le rendez-vous, la synchronisation ou la concurrence doivent être garantis.
- Les contraintes temporelles quantitatives, font intervenir des durées et des délais d'exécution. Toute vérification du système prend en compte les spécifications fonctionnelles et temporelles.

Dans le cadre de cette thèse, nous nous intéressons la plupart du temps aux aspects qualitatifs.

En fonction de la criticité des tâches réalisées, un système temps réel est qualifié temps réel souple ou à temps réel dur. En temps *souple*, une défaillance ou un retard à respecter des contraintes de temps de réponse entraîne une dégradation de l'exécution mais n'entraîne pas de conséquences catastrophiques sur l'environnement contrôlé. C'est le cas des systèmes multimédia : si une image est affichée en retard ou une trame de données transmises est perdue, cela ne remet pas en cause le fonctionnement correct de l'ensemble du système. Par contre en temps *dur*, le non respect d'une contrainte temporelle cause une erreur de fonctionnement, et peut entraîner des conséquences catastrophiques sur l'environnement contrôlé (perte de vies humaines, destruction de matériel, impact financier important, ...). C'est le cas des procédés industriels sensibles tels que les transports ferroviaires, les centrales nucléaires, l'ingénierie financière, la médecine assistée par ordinateur, les systèmes embarqués automobiles, ...

La majorité des systèmes de contrôle "temps réel" reposent sur une "électronique numérique". Ils sont composés d'unité de commande, de capteurs et actionneurs. Si les entrées/sorties peuvent être de type discret ou analogique, au sein de l'unité de commande, les applications sont souvent modélisées comme des Systèmes à Evènements Discrets (SED) : machine de Moore, automate, Grafcet, réseau de Petri, ... Les SED [15] s'opposent aux systèmes à variables continues et satisfont les deux propriétés suivantes : l'espace d'états est discret et la transition d'état est déclenchée par occurrence ou franchissement d'évènement. On aura donc besoin de techniques de description formelles qui manipulent des états et de transitions discrètes.

Les systèmes sont de plus en plus complexes car ils peuvent s'exécuter sur des architectures mono ou multiprocesseurs, parallèles ou réparties et peuvent aussi exécuter des tâches qui sont soumises à des contraintes fonctionnelles et temporelles. De par leur complexité, leur caractère critique et

même leur coût de réalisation, de tels systèmes sont difficiles à appréhender et l'étape de spécification de ces systèmes est cruciale. Toute erreur ou mauvaise interprétation du cahier des charges, non détectées à cette étape, se paierait au prix le plus fort. Il est donc nécessaire de disposer d'outils et de modèles fiables pour, d'une part, les modéliser, et d'autre part, pour les vérifier en s'assurant qu'un certain nombre de contraintes sont bien respectées [62]. Nous plaçons notre étude dans le cadre de la modélisation de spécification de la partie contrôle-commande de systèmes temps réel critiques [1]. Dans de tels systèmes les exigences de validation et de vérification formelles correspondent à des normes de spécification (DO-331, DO-333, ...).

Modéliser une spécification, c'est construire un modèle formel d'un système. Un modèle formel permet de représenter fidèlement (le plus fidèlement que possible) le comportement d'un système. Plus précisément, il s'agit d'une modélisation mathématique abstraite et approchée du fonctionnement du système. Cette représentation formelle permet à la fois de valider un cahier des charges mais aussi d'effectuer des preuves de propriétés. C'est l'aspect formel qui permet le calcul de l'espace d'états du système. L'espace d'états produit l'ensemble des états du système et de ses transitions qui peuvent être de type "évolution temporelle" ou bien encore correspondre à un évènement. Le calcul de l'espace d'états permet d'établir des invariants, d'invalider des scénarios d'exécution, ...

Nos travaux reposent sur l'utilisation d'un modèle formel. Plusieurs modèles existent et chaque modèle possède ses caractéristiques propres, plus ou moins pertinentes dans une conception spécifique. Le choix du modèle dépendra du système conçu et des propriétés à analyser. On peut citer les systèmes de transitions, les automates finis, les algèbres de processus, les réseaux de Petri, les langages synchrones, ...

Parmi l'ensemble des modèles existants, les réseaux de Petri (RdP) [69] possèdent un intérêt fondamental en fournissant des approches de modélisation basées sur un support graphique facilitant l'expression et la compréhension des mécanismes de base pour des systèmes communicants. Enfin, n'étant pas liés à un langage particulier, ils assurent l'indépendance de la modélisation vis-à-vis des implémentations. Une des raisons qui justifient le choix de ce formalisme est qu'il offre de nombreux outils d'analyse structurelle. Les réseaux de Petri ont trouvés des applications dans de très nombreux domaines. On peut citer les problèmes de partage des ressources, les protocoles de transmission, la gestion de la production, la synchronisation des tâches ... [73]. Le modèle de base des réseaux de Petri s'est révélé

[1] Par abus de langage, on utilisera le vocable système pour désigner la partie contrôle-commande d'un système temps réel.

néanmoins trop limité. Plusieurs extensions ont été introduites et sont soit plus compactes en terme de description ou soit plus puissantes en termes de pouvoir d'expression [62]. Nous pouvons citer les réseaux de Petri à arcs inhibiteurs [68], permettant le test à zéro d'une place, les réseaux de Petri avec des arcs de reset [44, 52] permettant la remise à zéro d'une place, les réseaux de Petri temporels [61] et temporisés [71] pour prendre en compte les contraintes temporelles quantitatives, ... Le comportement d'un réseau de Petri est souvent analysé en construisant son graphe d'états accessibles ou graphe des marquages accessibles [47] qui montre une représentation exhaustive des différents états atteignables par le système et permet de vérifier des propriétés tant génériques que spécifiques comme la bornitude, l'accessibilité, la vivacité, la terminaison, ... Cependant, la construction de l'espace d'états (même fini) d'un système n'est toujours pas possible à cause du problème d'explosion combinatoire [12] du nombre d'états due à la complexité du système, sa forte concurrence, la présence d'un comportement infini, ... Une façon de contenir cette explosion est basée sur l'idée de conserver des ordres partiels entre les évènements. Les techniques dites à «ordre partiel» [39] et plus particulièrement le dépliage (*unfolding*) [26], éliminent la représentation du parallélisme par entrelacement des actions et produisent des comportements, sous forme de succession d'évènements sans entrelacer tous les évènements concurrents. Outre le fait que le dépliage permet de contenir le problème d'explosion combinatoire, sa structure même présente des caractéristiques particulières qui font qu'il est très utilisé dans la communauté scientifique pour faire de la vérification (modèle checking) [28], pour faire du diagnostic [43] et aussi de la planification [40]. Pour un réseau de Petri, le nombre d'évènements du dépliage peut-être infini et les méthodes de vérification se basent plutôt sur le calcul d'une partie du dépliage appelée préfixe fini. Une des difficultés est de s'assurer qu'un tel préfixe fini est suffisamment grand pour permettre la vérification d'une propriété donnée. Il existe des algorithmes [58, 31] qui donnent une méthode de construction du préfixe fini et complet du dépliage. Le calcul du préfixe fini et complet du dépliage permet de capturer l'espace d'états et préserve l'accessibilité les réseaux de Petri bornés.

1.2 Objectifs de la thèse.

Le but de cette thèse est de contribuer à l'étude du dépliage des réseaux de Petri en général et plus particulièrement aux réseaux de Petri avec des arcs de reset. Aussi, elle vise une étude algébrique des processus de branchement

issus du dépliage des réseaux de Petri. Les contributions sont présentées dans la section suivante. L'intérêt de cette thèse est qu'elle s'attaque à un champ ouvert et propose un pont entre deux approches : les approches type systèmes à évènement discrets (réseaux de Petri) et les approches logiques (réduction sémantique et algébrique).

1.3 Contributions

Deux contributions sont présentées dans ce manuscrit.

La première contribution concerne le dépliage des réseaux de Petri avec des arcs de reset dans la Section 4.3. Nous montrons d'abord qu'il n'existe pas de traduction (ou transformation) d'un réseau de Petri avec des arcs de reset vers un réseau de Petri qui préserve à la fois la bisimulation et la causalité. Ainsi nous proposons un algorithme de construction du dépliage des réseaux de Petri avec des arcs de reset tout en conservation les arcs de reset. Une construction du préfixe fini et complet du dépliage d'un réseau de Petri avec des arcs de reset *borné* a également été proposée.

La deuxième contribution concerne l'étude algébrique des processus de branchement issus du dépliage d'un réseau de Petri. Un dépliage est constitué d'un ensemble de graphes acycliques et causaux qui explicitent une exécution possible du système. Ces graphes sont appelés *processus de branchement*. Les processus de branchement possèdent des caractéristiques singulières qui les distinguent fortement des processus classiques : *CSP (Communicating Sequential Processes)* de Hoare [41] ainsi que du *CCS* de Milner *(Calculus of Communicating Systems)* de Milner [63]. Nous proposons, dans la Section 5.5, une algèbre adaptée aux processus de branchement et comme dans l'algèbre de processus, nous chercherons à proposer des techniques de réduction, une forme canonique, des équivalences ainsi que des techniques d'extraction de sémantique.

1.4 Plan du document

Le manuscrit est organisé en sept chapitres. Le premier chapitre est la présente introduction. Le Chapitre 2 présente quelques généralités et notions préliminaires utilisées dans le reste du manuscrit. Dans le Chapitre 3, nous introduisons le formalisme par les réseaux de Petri et leurs sémantiques opérationnelles. Quelques extensions des réseaux de Petri ainsi que quelques équivalences comportementales (simulation, bisimulation, ...) y sont également abordées.

Nos travaux sont basés sur le dépliage des réseaux de Petri, le Chapitre 4 présente le dépliage des réseaux de Petri et le préfixe fini et complet du dépliage des réseaux de Petri. Dans ce chapitre, une section est consacrée au dépliage des réseaux de Petri avec des arcs de reset. Le Chapitre 5 aborde l'algèbre des processus de branchement issus du dépliage des réseaux de Petri. Le Chapitre 6 décrit deux outils développés pour automatiser des tâches. Le premier, dénommé *Penelope*, implémente l'algèbre de processus de branchement et permet de calculer la forme canonique de chaque dépliage. Le deuxième outil, *PN2A* pour *Petri Net to Arduino* permet d'embarquer un réseau de Petri T-temporel sur une carte Arduino.

Le dernier chapitre est consacré à la conclusion et aux perspectives de ces travaux.

2

Notations générales

Sommaire

Dans ce chapitre, nous présentons quelques notions préliminaires utilisées dans le reste du manuscrit.

2.1 Ensembles

$\mathbb{N}$, $\mathbb{Q}$ et $\mathbb{R}$ désignent respectivement l'ensemble des nombres entiers naturels, l'ensemble des nombres rationnels et l'ensemble des nombres réels. $\mathbb{B}$ est l'ensemble des valeurs booléennes (Vrai ou faux) et l'ensemble vide est représenté par $\emptyset$. Soit $K = \{\mathbb{N}, \mathbb{Q}, \mathbb{R}\}$, $K^{\infty} = K \cup \infty$

$\mathbb{N}^*$ représente l'ensemble des nombres naturels strictement positifs. $\mathbb{Q}^+$ (respectivement $\mathbb{R}^+$) est l'ensemble des nombres rationnels (respectivement réels) positifs ou nuls. $\mathbb{Q}_*^+$ (respectivement $\mathbb{R}_*^+$) est l'ensemble des nombres rationnels (respectivement réels) strictement positifs. Soit n un nombre entier naturel, $\mathbb{R}^n$ désigne l'espace réel à n dimension(s).

Si E désigne un ensemble fini alors $|E| = card(E)$ est le cardinal de E. 2^E est l'ensemble des sous-ensemble de E.

Un multi-ensemble sur un ensemble E est l'application de E dans $\mathbb{N}$ qui associe à chaque élément de E son nombre d'occurrence. Si de plus E est fini, un multi-ensemble sur E est un vecteur de $\mathbb{N}^n$. Soient x et y deux multi-ensembles sur E, $x \leq y$ (respectivement $x \geq y$) si et seulement pour tout élément e de E, $x(e) \leq y(e)$ (respectivement $x(e) \geq y(e)$.

$\wedge$ est l'opérateur logique *et*, $\vee$ l'opérateur logique *ou* et $\neg$ est le *non* logique. $\cap$ est l'opérateur d'intersection entre deux ensembles et $\cup$ celui de l'union.

2.2 Relations, relations d'équivalence

Soit E un ensemble. Une relation sur E est une partie R de E^2. On notera $(x,y) \in R$ ou xRy si x et y sont en relation par R. Une relation peut être :

- réflexive si et seulement si $\forall x \in E,\ xRx$;
- symétrique ou commutative si et seulement si $\forall x, y \in E,\ xRy \Rightarrow yRx$;
- transitive si et seulement si $\forall x, y, z \in E,\ (xRy \wedge yRz) \Rightarrow xRz$;
- antisymétrique si et seulement si $\forall x, y \in E,\ (xRy \wedge yRx) \Rightarrow x = y$;

Une *relation d'équivalence* sur un ensemble E est une relation binaire R qui est à la fois réflexive, symétrique et transitive.

2.3 Injection, surjection, bijection

Soit E, F deux ensembles et $f : E \to F$ une application :

- f est *injective* si pour tout $x, x' \in E$ avec $f(x) = f(x')$ alors $x = x'$.
- f est *surjective* si pour tout $y \in F$, il existe $x \in E$ tel que $y = f(x)$.
- f est *bijective* si elle est injective et surjective. Pour tout $y \in F$, il existe un unique $x \in E$ tel que $y = f(x)$.

2.4 Vecteurs et matrices

Vecteurs

Soit E un ensemble fini. Un vecteur de dimension E à coefficients dans $\mathbb{N}$ est une application de E dans $\mathbb{N}$. Pour tout $e \in E, v(e)$ désigne la composante en e de v. $\vec{0}$ est le vecteur nul. Soient x et y deux vecteurs :

$$x = (x_1, x_2, \ldots, x_n) \qquad\qquad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

- $x \leq y$ (respectivement $x \geq y$) si et seulement pour tout élément e de E, $x(e) \leq y(e)$ (respectivement $x(e) \geq y(e)$
- soit $z = x + y$ (respectivement $z = x - y$) la somme (respectivement la soustraction) des vecteurs x et y. Pour tout élément e de E, $z(e) = x(e) + y(e)$ (respectivement $z(e) = x(e) - y(e)$).
- le produit scalaire de x et de y est le scalaire définit par : $\sum_{i=1}^{|E|} x(e_i)y(e_i)$

Matrices

Soient $n, p \in \mathbb{N}^*$. On appelle matrice à n lignes et p colonnes à coefficients dans K (K est un ensemble de nombres. Par exemple $\mathbb{N}$ ou $\mathbb{Q}$ ou $\mathbb{R}$), un tableau à n lignes et p colonnes d'éléments de K. On note une telle matrice :

$$M = (a_{ij})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq p}} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1p} \\ a_{21} & a_{22} & \ldots & a_{2p} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{np} \end{pmatrix}$$

a_{ij} est le coefficient situé sur la $i^{ème}$ ligne et la $j^{ème}$ colonne de la matrice M.

Somme et soustraction de matrices

Soient $A = (a_{ij})_{\substack{1\leq i\leq n\\1\leq j\leq p}}$ et $B = (b_{ij})_{\substack{1\leq i\leq n\\1\leq j\leq p}}$ deux matrices de mêmes dimensions.

$$A+B = (c_{ij})_{\substack{1\leq i\leq n\\1\leq j\leq p}} \text{ avec } c_{ij} = a_{ij}+b_{ij}$$

$$A-B = (c_{ij})_{\substack{1\leq i\leq n\\1\leq j\leq p}} \text{ avec } c_{ij} = a_{ij}-b_{ij}$$

Produit matriciel

Soient $A = (a_{ij})_{\substack{1\leq i\leq m\\1\leq j\leq n}}$ une matrice de type (m,n) et $B = (b_{ij})_{\substack{1\leq i\leq n\\1\leq j\leq p}}$ une matrice de type (n,p).

$$AB = (c_{ij})_{\substack{1\leq i\leq m\\1\leq j\leq p}} \text{ avec } c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

Produit matriciel d'Hadamard

Soient $A = (a_{ij})_{\substack{1\leq i\leq n\\1\leq j\leq p}}$ et $B = (b_{ij})_{\substack{1\leq i\leq n\\1\leq j\leq p}}$ deux matrices de mêmes dimensions. Contrairement au produit matriciel, le produit matriciel d'Hadamard est un produit terme à terme entre les deux matrices A et B.

$$A \odot B = (c_{ij})_{\substack{1\leq i\leq n\\1\leq j\leq p}} \text{ avec } c_{ij} = a_{ij} \times b_{ij}$$

2.5 Les automates

Un automate est un outil fondamental en informatique où il intervient en compilation des langages, en théorie de la calculabilité, dans l'étude des langages formels ... C'est un graphe orienté dont les états sont ses sommets et les arêtes sont étiquetées par des transitions. Son comportement est commandé par un mot fourni en entrée. L'ensemble des mots de l'automate constitue son langage. L'automate est dit *fini* s'il possède un nombre fini d'états distincts. Dans la littérature, le terme de système de transitions étiquetées est utilisé pour désigner les automates.

Soit Σ un ensemble de caractères appelé *alphabet*.

Définition 1 (Système de transitions étiquetées)**.** *Un système de transitions étiqueté* $A = \langle Q, Q_0, \Sigma, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *est défini par :*

- Q *: l'ensemble des états ;*
- Q_0 *: l'ensemble des états initiaux ;*
- Σ *: l'alphabet des actions ;*
- $\{\xrightarrow{a}\}_{a\in\Sigma} \subseteq Q \times \Sigma \times Q$ *: la relation de transitions.*

Définition 2 (Séquence d'exécution)**.** *Soit* $A = \langle Q, Q_0, \Sigma, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *un système de transitions étiqueté. Un élément* $\rho \in \Sigma^*$ *noté* $\rho = a_1 \ \ldots \ a_n$ *est une séquence d'exécution ou séquence de transitions de S s'il existe* $(q_1, q_2, \ldots, q_n, q_{n+1}) \in Q^{n+1}$ *tels que* $q_1 \xrightarrow{a_1} q_2 \ldots q_n \xrightarrow{a_n} q_{n+1}$.

Si de plus q_1 est un état initial, ρ est un mot de S. On appelle *trace* d'une exécution, la séquence des étiquettes.

Définition 3 (Langage)**.** *Le langage* $\mathfrak{L}$ *d'un système de transitions étiqueté S est l'ensemble des mots acceptés par S.*

3

Formalisme par les Réseaux de Petri

Sommaire

3.1 Introduction

Les réseaux de Petri (RdP) ou réseaux de Petri place-transition ont été développés en tant que formalisme opérationnel pour la spécification des systèmes concurrents. Ils sont utilisés dans de nombreux domaines, notamment dans les milieux industriels, et sont adaptés à la modélisation des systèmes distribués ou répartis. Les applications sont diverses et couvrent les domaines des systèmes de production, de transports et de communication. De nombreuses extensions ont été ajoutées au modèle de base pour proposer des modèles plus compacts en terme de description et plus puissants en terme de pouvoir d'expression. Nous pouvons citer les réseaux de Petri avec des arcs inhibiteurs/lecteur, avec arc de resets/arcs de transfert, les extensions temporelles des réseaux de Petri, ... Ce chapitre introduit les réseaux de Petri et quelques-unes des leurs extensions ainsi que leurs sémantiques opérationnelles.

3.2 Définitions fondamentales

La base de tous les modèles de réseaux de Petri (RdP) est la définition de réseau proposée par Carl Adams Petri en 1962 [69]. Ils sont composés de deux types d'objets : les places et les transitions. Les places permettent de modéliser l'état du système, les transitions sont les événements dont les occurrences provoquent la modification ou l'évolution de l'état du système. Plus précisément, les places jouent le rôle de variables d'état du système et sont à valeurs entières. Le franchissement d'une transition dépend de la satisfaction de préconditions et produit des post-conditions.

Les réseaux de Petri offrent une représentation graphique simple des systèmes modélisés. Une place est représentée par un cercle, une transition par un rectangle, et les relations de causalité au sein du système sont représentées par la présence ou non d'arcs valués reliant les places aux transitions ou les transitions aux places. Par défaut, le poids d'un arc est égal à 1. Le marquage

d'un RdP est un vecteur à composantes entières positives ou nulles et dont la dimension est égale au nombre de places. La $p^{ième}$ composante de ce vecteur, représente le nombre de jetons dans la place p du réseau de Petri.

3.2.1 Réseau de Petri

Définition 4 (Réseau de Petri)**.** *Un réseau de Petri N est un triplet* $\langle P, T, W \rangle$ *avec :*

- P *un ensemble fini de places ;*
- T *un ensemble de transitions ;*
- $P \cap T = \emptyset$ *et* $P \cup T \neq \emptyset$ *;*
- $W : P \times T \cup T \times P \to \mathbb{N}$ *la fonction de valuation des arcs.*

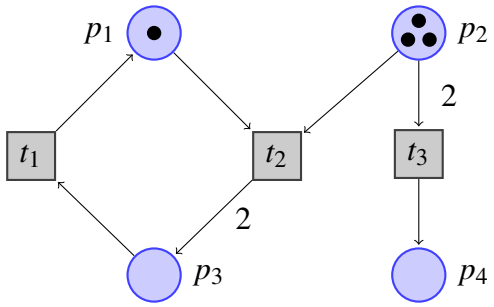


FIGURE 3.1 : Réseau de Petri

La Figure 3.1 est un réseau de Petri avec $P = \{p_1, p_2, p_3, p_4\}$, $T = \{t_1, t_2, t_3\}$ et $W = \{(p_1, t_2, 1), (t_2, p_3, 2), (p_3, t_1, 1), (t_1, p_1, 1), (p_2, t_2, 1), (p_2, t_3, 2), (t_3, p_4, 1)\}$.

Les places contiennent des *jetons* ou *marques* qui indiquent le nombre d'occurrences de ressources matérialisées dans la place. La fonction de valuation des arcs regroupe les pré-conditions et post-conditions. On peut alors définir les fonctions *Pre* et *Post* comme des restrictions de W respectivement à $P \times T$ et à $T \times P$:

- Pre : $P \times T \to \mathbb{N}$ est la restriction de W à $P \times T$. Elle exprime les pré-conditions nécessaires à la réalisation d'une action du système $Pre(p,t) = W(p,t)$. Par la suite, on utilisera $W(.,t)$ ou $Pre(t)$ pour désigner les pré-conditions de la transition t ;

- Post $T \times P \rightarrow \mathbb{N}$ est la restriction de W à $T \times P$. Elle exprime les post-conditions d'une action du système $Post(t,p) = W(t,p)$. Par la suite, on utilisera $W(t,.)$ ou $Post(t)$ pour désigner les post-conditions de la transition t ;

Pour tout nœud $x \in P \cup T$, on désigne par ${}^\bullet x = \{y \in P \cup T \mid W(y,x) > 0\}$ (respectivement par $x^\bullet = \{y \in P \cup T \mid W(x,y) > 0\}$) les nœuds précédents (respectivement suivants) de x. En particulier, pour tout $t \in T$, ${}^\bullet t = W(.,t) = Pre(t)$ et $t^\bullet = W(t,.) = Post(t)$.

Afin de décrire complètement le système, nous devons définir à un instant donné l'état global du système. Ceci est fait grâce à son marquage.

3.2.2 Marquage et RdP marqué

Définition 5 (Marquage)**.** *Le marquage d'un réseau de Petri $N = \langle P,T,W \rangle$ est la fonction $M : P \rightarrow \mathbb{N}$ qui associe à chaque place de N le nombre de jetons qu'elle contient.*

Un marquage peut être représenté soit par un vecteur $M = (m_i)_{1 \leq i \leq |P|}$ où m_i est le nombre de jeton(s) dans la place p_i ou soit par un multi-ensemble dans lequel une place apparait autant de fois que le nombre de jeton(s) qu'elle contient [1]. M_0 désigne le *marquage initial*.

Pour l'exemple de la Figure 3.1, le marquage initial peut être représenté par le vecteur $M_0 = \begin{bmatrix} 1 & 3 & 0 & 0 \end{bmatrix}^T$ ou par le multi-ensemble $M_0 = \{\{p_1, p_2, p_2, p_2\}\}$. Le marquage initial combiné à la Définition 4 permet de définir un réseau de Petri marqué.

Définition 6 (Réseau de Petri marqué)**.** *On appelle réseau de Petri marqué, tout couple $\langle N, M_0 \rangle$ où :*

- *N est un réseau de Petri (Définition 4) et*
- *M_0 son marquage initial.*

Par la suite, nous utiliserons indifféremment le terme de réseau de Petri et de réseau de Petri marqué.

3.2.3 RdP labellisé

On peut étendre la définition d'un réseau de Petri au réseau de Petri labellisé [54] dans lequel une transition peut être associée à plusieurs actions.

[1] Par la suite $M(p)$ représente le nombre de jeton(s) contenu dans la place p.

Définition 7 (Réseau de Petri labellisé)**.** *Un réseau de Petri labellisé (RdP-L) est un quadruplet* $N_L = \langle N, M_0, \Sigma, \lambda \rangle$ *avec :*

- (N, M_0) *est un RdP marqué (Définition 6) ;*
- Σ *un ensemble fini des événements ;*
- $\lambda : T \to \Sigma$ *la fonction de labellisation des transitions qui affecte un label à chaque transition.*

L'ensemble des événements $\Sigma = \Sigma_0 \uplus \{\varepsilon\}$. Σ_0 est l'ensemble des événements observables et toutes les transitions non-observables ont le label ε (les transitions non observables sont aussi notées par τ). Le même label peut être partagé par des transitions différentes. Un réseau de Petri peut être vu comme une sous classe particulière de réseau de Petri labellisé où $\Sigma = \Sigma_0$ et la fonction λ devenant alors bijective (Section 2.3).

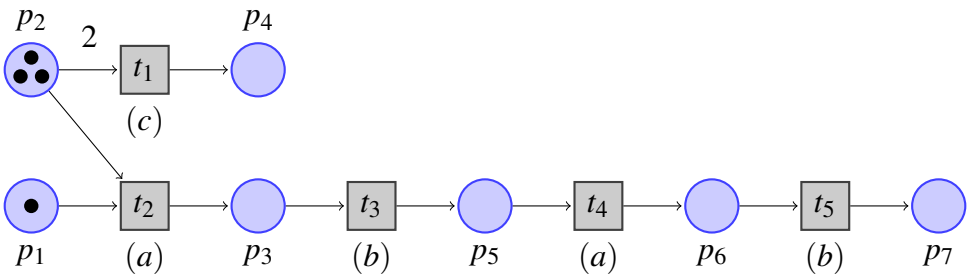


FIGURE 3.2 : Réseau de Petri labellisé

Un réseau de Petri labellisé est représenté sur la Figure 3.2 où $T = \{t_1, t_2, t_3, t_4, t_5\}$ et $\Sigma = \{a, b, c\}$. Le label a est associé aux transitions t_2 et t_4, b aux transitions t_3 et t_5, c à la transition t_1. Le réseau de la Figure 3.2 peut être vu comme une partie de l'exécution du réseau de Petri de la Figure 3.1 en identifiant les évènements a, b et c respectivement aux transitions t_2, t_3 et t_1.

3.2.4 Dynamique d'un réseau de Petri

Les sections précédentes montrent une représentation statique des systèmes. Mais dans la réalité, ils évoluent avec le temps et/ou avec le franchissement de transitions. Dans cette section, nous exprimerons la dynamique du réseau : comment un réseau de Petri peut évoluer et quels sont les résultats de ces évolutions. La définition suivante donne la condition nécessaire et suffisante pour qu'un évènement lié à une transition se produise dans le système.

Définition 8 (Sensibilisation d'une transition)**.** *Soit $N = \langle P, T, W, M_0 \rangle$ un réseau de Petri, $t \in T$ une transition et M un marquage du réseau. Une transition est sensibilisée (tirable, franchissable) depuis le marquage M si et seulement si : $\forall p \in P, M(p) \geq W(p,t)$ ($M \geq W(.,t)$). On le notera $M \xrightarrow{t}$.*

Avec la condition $M(p) \geq W(p,t)$, on vérifie que les ressources nécessaires sont disponibles. Sur la Figure 3.1, t_2 et t_3 sont sensibilisées par le marquage initial M_0. Par contre t_3 ne l'est pas car il manque une ressource dans la place p_3. La notion de franchissabilité exprime une condition d'exécution d'un événement dans le réseau. L'évolution du réseau est modélisée par le tir effectif d'une transition sensibilisée. A la terminologie de "tir (ou de franchissement) d'une transition" on associera le terme *d'évènement*. On notera par $enabled(M)$ l'ensemble des transitions sensibilisées par le marquage M.

Définition 9 (Tir effectif d'une transition)**.** *Si une transition t est sensibilisée par un marquage M ($M \xrightarrow{t}$) alors le tir effectif de t conduit le réseau du marquage M vers le marquage M' défini par :*

$$M' = M - W(.,t) + W(t,.) \tag{3.1}$$

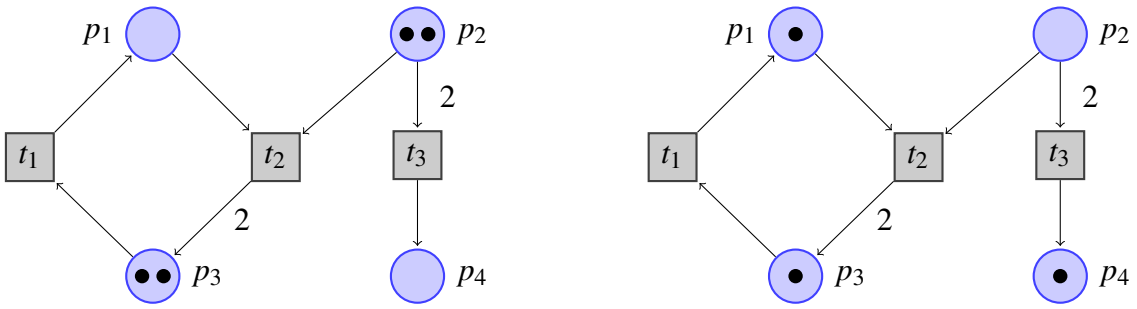
On le notera $M \xrightarrow{t} M'$.

Si on pose $C(t,.) = W(t,.) - W(.,t)$ alors l'équation 3.1 devient :

$$M' = M + C(t,.) \tag{3.2}$$

La matrice C est appelée matrice d'incidence du réseau de Petri.

Sur la Figure 3.1, t_2 est sensibilisée par le marquage $M_0 = \begin{bmatrix} 1 & 3 & 0 & 0 \end{bmatrix}^T$. Le tir effectif de t_2 conduit au marquage M_1 ($M_0 \xrightarrow{t_2} M_1$) avec $M_1 = \begin{bmatrix} 0 & 2 & 2 & 0 \end{bmatrix}^T$ (Figure 3.3.a.).



a. Tir de t_2 b. Tir de la séquence $\sigma = t_2 t_3 t_1$

FIGURE 3.3 : Tir de transitions

On peut étendre la notion de sensibilisation et de tir d'une transition à une séquence de transitions.

Définition 10 (Sensibilisation d'une séquence de transitions)**.** *Soit $N = \langle P, T, W, M_0 \rangle$ un réseau de Petri, $\sigma = t_1\ t_2\ \ldots\ t_n \in T^*$ une séquence de transitions et M un marquage du réseau. La séquence σ est franchissable depuis le marquage M si et seulement s'il existe des marquages M_1, M_2, ... M_n tels que : $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \ldots M_{n-1} \xrightarrow{t_n} M_n$. On le notera $M \xrightarrow{\sigma}$.*

Le tir effectif de σ, c'est-à-dire le tir effectif successif de chaque transition appartenant à σ, conduit le réseau du marquage M vers le marquage M' qu'on notera $M \xrightarrow{\sigma} M'$. La Figure 3.3.b. montre l'état du réseau de Petri de la Figure 3.1 après le tir de la séquence $\sigma = t_2\ t_3\ t_1$. Un marquage est accessible dans le réseau s'il existe une séquence de transitions σ telle que $M_0 \xrightarrow{\sigma} M$.

3.2.5 Relations entre les nœuds d'un réseau de Petri

Dans un réseau de Petri, il existe trois types de relations s'excluant mutuellement entre deux nœuds quelconques. Il s'agit de la causalité, du conflit ou de la concurrence.

Soient x et y deux nœuds d'un réseau de Petri $N = \langle P, T, W, M_0 \rangle$; $(x, y) \in (P \cup T)^2$.

3.2.5.1 Causalité

Définition 11 (Causalité)**.** *x et y sont dans une relation de causalité, que l'on notera* $x \prec y$*, s'il existe un chemin* [2] *entre x et y dans N. Plus généralement* $x \preccurlyeq y$ *si et seulement si* $x \prec y$ *ou* $x = y$*.*

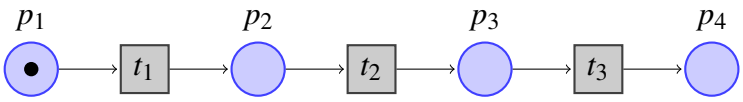


FIGURE 3.4 : Causalité

Il faut voir la causalité entre x et y comme une relation d'ordre entre x et y : l'occurrence de y précède celle de x et il existe alors une contrainte temporelle qualitative entre x et y à cause de cette précédence. Sur la Figure 3.4, $t_1 \prec t_2$, $t_1 \prec t_3$ et $t_2 \prec t_3$. La causalité permet de définir la sémantique d'ordre partiel entre les évènements.

3.2.5.2 Conflit

Définition 12 (Conflit)**.** *x et y sont en conflit, que l'on notera* $x \perp y$*, si et seulement s'il existe dans le réseau N deux chemins* $p\ t_1 \ldots x$ *et* $p\ t_2 \ldots y$ *partant de la même place p telle que* $x \neq y$ *;* $x \perp y \Leftrightarrow y \perp x$*.*

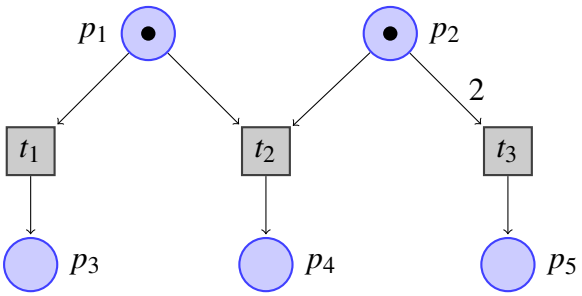


FIGURE 3.5 : Conflit

D'après la définition précédente, deux transitions sont en conflit si elles ont au moins une place précédente en commun. Ceci ne définit qu'un *conflit structurel*. Pour qu'il y ait un *conflit effectif*, il faut non seulement que les deux transitions t_1 et t_2 soient en conflit structurel mais aussi que, pour un marquage M, accessible de puis le marquage initial, et pour p la place commune, $M \geq W(p,t_1) \vee M \geq W(p,t_2)$. Ainsi sur la Figure 3.5, t_2 et t_3 sont en

[2] Tout comme en théorie des graphes, on peut définir la notion de chemin entre deux nœuds x et y d'un RdP. Un *chemin* d'origine x et d'extrémité y est défini par une suite finie d'arcs consécutifs reliant x à y.

conflit structurel mais pas en conflit effectif avec le marquage initial M_0. Par contre, avec le marquage initial M_0, t_1 et t_2 sont en conflit effectif.

3.2.5.3 Concurrence

Définition 13 (Concurrence)**.** *x et y sont concurrents ou parallèles, que l'on notera $x \wr y$, s'ils ne sont ni en causalité, ni en conflit : $x \wr y \Leftrightarrow ((x \prec y) \vee (y \prec x) \vee (x \perp y))$. $x \wr y \Leftrightarrow y \wr x$.*

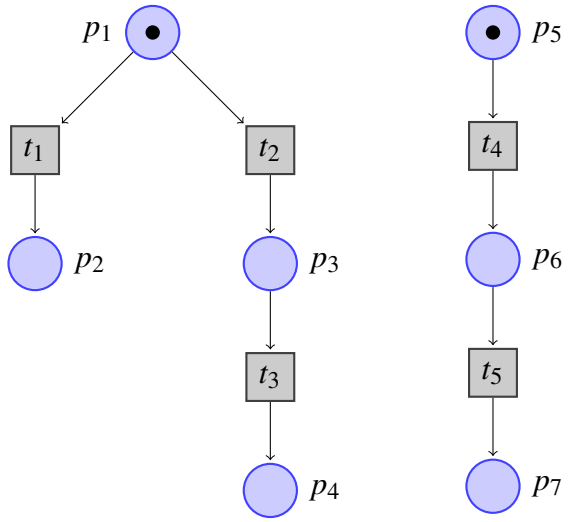


FIGURE 3.6 : Concurrence

Deux transitions sont donc en concurrence si elles peuvent être tirées indépendamment l'une de l'autre. C'est le cas des transitions t_1 et t_4 ou t_2 et t_4 sur la Figure 3.6. Tout comme pour le conflit, on peut avoir un parallélisme structurel ou effectif. Ainsi deux transitions t_1 et t_2 sont en parallélisme effectif si elles sont en parallélisme structurel (Définition 13) et si pour un marquage M accessible de puis le marquage initial, les deux transitions sont sensibilisées ($M \geq W(p,t_1) \vee M \geq W(p,t_2)$). Avec le marquage initial M_0, t_2 et t_4 sont en parallélisme effectif, tout comme t_1 et t_4. Par contre t_1 et t_5 ou t_3 et t_5 sont en parallélisme structurel.

La sémantique utilisée habituellement pour exprimer les comportements du réseau de Petri est une sémantique séquentielle dite *d'entrelacement*. Cette sémantique est très coûteuse en terme de représentation du parallélisme car elle impose une combinatoire, sous forme de séquences alternatives, de tous les évènements concurrents ou parallèles. Pour pallier à cette combinatoire, la sémantique dite *d'ordre partiel* décrit les comportements du système sous

forme d'une succession d'évènements. Cette sémantique évite l'entrelacement des évènements concurrents. Dans la sémantique d'ordre partiel, seule la causalité est représentée explicitement. Le conflict est implicitement représenté grâce à la causalité : s'il existe une place $p \in P$ telle que $p \prec t_1 \wedge p \prec t_2 \Rightarrow t_1 \perp t_2$. La concurrence n'est pas représentée et se déduit alors : s'il n'y a pas de relation entre deux transitions alors elles sont concurrentes.

La section suivante présente la sémantique d'entrelacement. Nous reviendrons sur celle d'ordre partiel dans le Chapitre 4.

3.3 Analyse comportementale des réseaux de Petri

La sémantique opérationnelle d'un système ou du réseau de Petri peut être perçue en observant soit les événements du système, soit ses états. On peut alors introduire deux ensembles à savoir :

- le langage du réseau : l'ensemble des séquences de transitions franchissables depuis le marquage initial ;
- l'ensemble d'accessibilité : l'ensemble de tous les états accessibles du système.

Définition 14 (Langage)**.** *Soit $N = \langle P, T, W, M_0 \rangle$ un réseau de Petri, l'ensemble $\mathscr{L}(N) = \{\sigma \in T^* \mid M_0 \xrightarrow{\sigma}\}$ est le langage du réseau N.*

Définition 15 (Ensemble d'accessibilité)**.** *Soit $N = \langle P, T, W, M_0 \rangle$ un réseau de Petri, l'ensemble $\mathscr{A}(N) = \{M \in \mathbb{N}^{|P|} \mid \exists\, \sigma \in \mathscr{L}(N),\ M_0 \xrightarrow{\sigma} M\}$ est l'ensemble d'accessibilité du réseau N.*

L'analyse du réseau de Petri ou du système repose sur l'étude des caractéristiques de ces deux ensembles. La construction du graphe des marquages (si on y arrive) permet d'avoir conjointement ces deux ensembles. On peut considérer une notion d'ordre sur les marquages en définissant la notion de couverture.

Définition 16 (Couverture)**.** *Un marquage M' couvre un marquage M ou tout simplement M' est supérieur à M si et seulement si :*

$$\forall p \in P,\ M'(p) > M(p) \tag{3.3}$$

3.3.1 Graphe des marquages

Définition 17 (Graphe des marquages)**.** *On appelle graphe des marquages (ou graphe d'accessibilité) du réseau $N = \langle P, T, W, M_0 \rangle$ le graphe $GDM(N)$ dont les nœuds sont les marquages accessibles de $\mathscr{A}(N)$ et dont les arcs sont étiquetés par les noms de transitions. Dans le graphe $GDM(N)$, un arc orienté étiqueté relie le nœud M au nœud M' s'il existe une transition t franchissable permettant de passer du marquage M au marquage M' ($M \xrightarrow{t} M'$).*

Il s'agit de construire de manière exhaustive le graphe des marquages accessibles. On démarre avec le seul marquage initial et on construit les différents arcs et nœuds progressivement. Pour chaque nouveau marquage ajouté, on vérifie s'il ne couvre pas un marquage déjà présent sur au moins une séquence entre M_0 et le nouveau marquage. Si tel est le cas, on abandonne la construction du graphe des marquages, le réseau n'est pas borné : le graphe des marquages n'est pas fini. Sinon, pour le nouveau marquage ajouté, on détermine l'ensemble des transitions franchissables et pour chaque transition de cet ensemble, on ajoute un arc vers le nouveau marquage. L'Algorithme 1 donne la procédure complète pour la construction du graphe des marquages [47] et le graphe des marquages du réseau de Petri de la Figure 3.1 est présenté sur la Figure 3.7.

Algorithme 1. *Soit $N = \langle P, T, W, M_0 \rangle$ un réseau de Petri marqué. Le graphe des marquages $GDM(N)$ de N est construit de la manière suivante :*

1. *Initialiser Accessibles et A_explorer à M_0 ;*
2. *Tant que A_explorer n'est pas vide :*
   (a) *Choisir un marquage M dans A_explorer ;*
   (b) *Pour chaque transition sensibilisée par M :*
      i. *Calculer M' tel que $M \xrightarrow{t} M'$;*
      ii. *Si $\exists\, M, M' \in \mathscr{A}(N)$ tel que $M \xrightarrow{\sigma} M'$ et M' couvre M alors le réseau n'est pas borné et l'algorithme s'arrête.*
      iii. *Sinon Si M' n'appartient pas encore à Accessibles alors l'ajouter à Accessibles et à A_explorer ;*
      iv. *Ajouter l'arc $\{(M, t, M')\}$ à $GDM(N)$*
3. *Si l'algorithme se termine sans abandon, $GDM(N)$ est le graphe des marquages.*

Nous allons prouver que le critère d'arrêt du point $2.(b).ii.$ de l'Algorithme 1 est nécessaire et suffisante pour son abandon car le réseau est non borné [76].

Preuve.

1. Monotonie : $\exists M, M' \mid (M \xrightarrow{\sigma} \wedge M' > M)$ alors $M' \xrightarrow{\sigma}$
   Lorsqu'une séquence de transitions est sensibilisée par un marquage M, elle reste sensibilisée par tout marquage M' supérieur à M.

2. Si M et M' sont deux marquages accessibles du réseau tels que $M \xrightarrow{\sigma} M'$ et $M' > M$ alors le réseau n'est pas borné. Ceci est une conséquence du point 1. La séquence σ peut être exécutée indéfiniment et à chaque fois, on accroît le contenu d'au moins une place du marquage M'.

3. Lemme de Karl et Miller [47] : Toute suite infinie de vecteurs formés d'entiers positifs ou nuls $v_1, v_2, \ldots, v_n, \ldots$ est telle qu'elle contient au moins deux éléments (en réalité, il s'agit d'une infinité de couples) v_i et v_j avec $i < j$ tels que $v_i \leq v_j$.

4. Soit une séquence infinie σ^∞ de transitions, la suite des marquages est une suite infinie de vecteurs d'entiers positifs ou nuls. Le lemme du point 3 implique l'existence d'une séquence σ' telle que $M \xrightarrow{\sigma'} M'$ et $M \leq M'$.

Les séquences de transitions du point 2 sont caractéristiques des séquences de transitions de longueur infinies non stationnaires [3]. Elles sont caractéristiques du fait que le réseau n'est pas borné c'est-à-dire que $\mathscr{A}(N)$ contient un nombre infini de marquages.

□

Pour un réseau de Petri identifié non borné par cet algorithme, une alternative est la construction du graphe de couverture [34].

Si cet algorithme est décidable, par contre, même pour les réseaux de Petri bornés, l'espace d'états calculé a une taille qui croît exponentiellement avec la taille du modèle [77], son calcul peut être long et consommer beaucoup de mémoire : c'est le problème d'explosion combinatoire [12].

[3] Une séquence de transitions non stationnaire est une séquence qui ne repasse jamais par les mêmes marquages

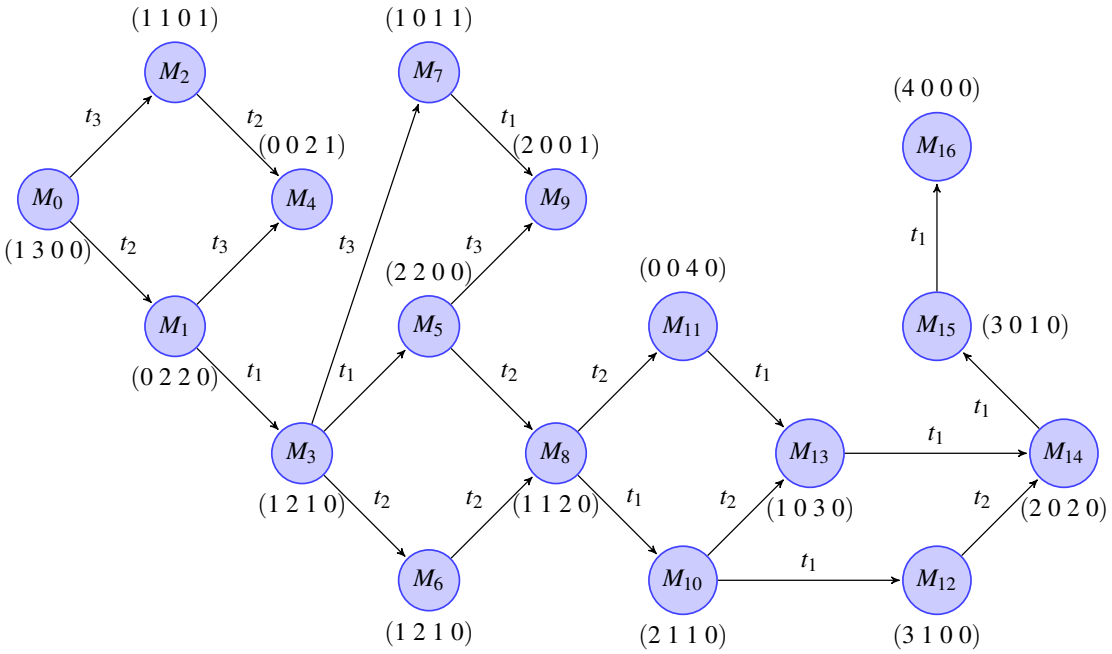


FIGURE 3.7 : Graphe des marquages du RdP de la Figure 3.1

3.3.2 Graphe ou arbre de couverture

Un graphe des marquages ne peut être construit quand le réseau est non borné c'est-à-dire quand le nombre de marquages accessibles est infini. On a recours au graphe de couverture ou arbre de couverture qui est un graphe fini mais qui contient moins d'informations que le graphe des marquages. Le mécanisme de construction est similaire à la construction du graphe des marquages. A ceci près que pour chaque nouveau marquage ajouté, on vérifie s'il ne couvre pas un marquage déjà présent sur une séquence entre M_0 et le nouveau marquage. Le cas échéant, toutes les composantes de place supérieures sont remplacées par w. Le symbole w matérialise le fait que la place en question n'est pas bornée. Les opérations sur w sont :

$$\forall n \in \mathbb{N} : \begin{cases} n < w \\ n + w = w + n = w + w = w \\ w - n = w \end{cases} \tag{3.4}$$

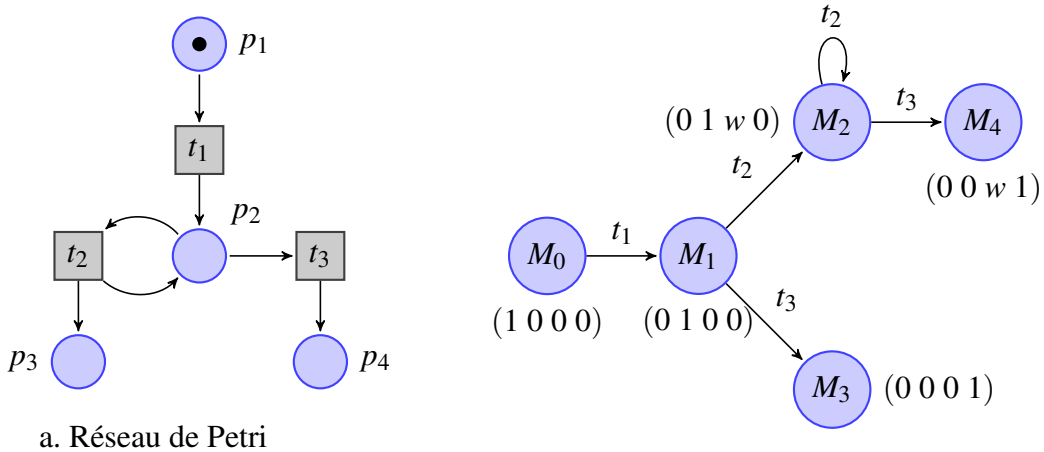
Finkel présente dans [34] un algorithme (Algorithme 2) pour la construction d'un arbre de couverture minimal.

Algorithme 2. *Soit $N = \langle P, T, W, M_0 \rangle$ un réseau de Petri marqué. L'arbre de couverture de N est construit de la manière suivante :*

1. *M_0 correspond à la racine de l'arbre de couverture ;*

2. *On ajoute pour chaque transition franchissable les marquages successeurs. Si un de ces marquages couvre M_0, on met w pour chacune des composantes supérieures aux composantes correspondantes de M_0 ;*
3. *Pour chaque nouveau marquage M_i :*
   (a) *S'il existe sur le chemin de M_0 à M_i (M_i exclu) un marquage $M_j = M_i$, alors M_i n'a pas de successeur ;*
   (b) *Sinon, on prolonge l'arbre en ajoutant tous les successeurs de M_i. Pour chaque successeur M_j de M_i :*
      i. *Une composante w de M_i reste une composante w de M_j ;*
      ii. *S'il existe un marquage M_k sur le chemin de M_0 à M_j couvert par M_j, alors on met w pour chacune des composantes supérieures aux composantes de M_k.*

La Figure 3.8 présente un réseau de Petri non borné (a) et son graphe de couverture (b).



a. Réseau de Petri

b. Graphe de couverture

FIGURE 3.8 : Réseau de Petri non borné et son graphe de couverture

3.4 Propriétés exprimables sur les Réseaux de Petri

Les réseaux de Petri permettent de modéliser des systèmes qui doivent obéir à des contraintes particulières. Toutes les propriétés que nous énoncerons dans cette section sont définies pour un réseau de Petri marqué $N = \langle P, T, W, M_0 \rangle$ donné (réseau de Petri et marquage initial donnés). Toutes ces propriétés sont décidables pour un réseau de Petri même si les établir peut être très coûteux.

3.4.1 Bornitude ou k-bornitude

Il s'agit d'exprimer le fait que le nombre de marquages accessibles ou le nombre d'états que peut prendre le système est fini.

Définition 18 (Place bornée)**.** *Une place $p \in P$ est bornée si et seulement si pour tout marquage accessible M, il existe un entier naturel k tel que $M(p) \leq k$.*

Définition 19 (Réseau de Petri borné)**.** *Un réseau de Petri marqué est borné si toutes les places sont bornées.*

Le nombre de marquages accessibles est fini. De plus, si k est la borne du réseau alors le réseau est dit *k-borné*. En particulier, un réseau 1-borné (chaque place contient au maximum un jeton) est dit *sauf*. Pour vérifier si un réseau de Petri est borné, il suffit de vérifier si son graphe de couverture ne contient aucune composante à w.

La construction du graphe de couverture part du marquage initial et chaque transition franchissable à partir du marquage initial donne naissance à une branche. On calcule les marquages obtenus par le franchissement des transitions et on reprend le processus avec les nouveaux marquages. On arrête le développement d'une branche lorsque :

- on trouve un marquage égal à un marquage déjà rencontré sur la branche en cours d'exploration et dont les successeurs ont été déjà calculés ou vont être calculés sur une autre branche pendante ;
- on trouve un marquage qui couvre un marquage rencontré sur la branche en cours d'exploration. Les composantes supérieures sont remplacées par w dans le marquage.

L'absence de composante w dans le graphe de couverture (dans ce cas, le graphe de couverture n'est rien d'autre que le graphe des marquages) assure l'absence de séquences infinies de transitions et l'énumération complète de tous les états accessibles du système : le réseau est borné.

3.4.2 Quasi-vivacité

Il s'agit de vérifier que toutes les actions spécifiées peuvent être exécutées au moins une fois à partir de M_0.

Définition 20 (Réseau quasi-vivant)**.** *Un réseau de Petri est quasi-vivant si et seulement si toute transition apparaît dans au moins une séquence de tirs.*

Un réseau est quasi-vivant si pour toute transition t il existe dans le graphe de marquages (ou dans le graphe de couverture) un arc étiqueté par t.

3.4.3 Vivacité

Il s'agit de vérifier si à tout instant, le système conserve la possibilité de reproduire toutes les transitions.

Définition 21 (Transition vivante)**.** *Une transition $t \in T$ est vivante si pour tout marquage accessible M, il existe une séquence de transitions σ qui contienne la transition t à partir de M. Autrement dit, quelle que soit l'évolution du système, il existera toujours une possibilité de franchir t.*

La vérification de la vivacité d'une transition revient à vérifier qu'elle apparaît dans toutes les composantes fortement connexes du graphe des marquages.

Définition 22 (Réseau vivant)**.** *Un réseau de Petri est vivant si et seulement si pour toute transition $t \in T$, t est vivante.*

3.4.4 Etat de blocage

Il s'agit de vérifier s'il existe un marquage du réseau à partir duquel plus aucune transition n'est franchissable.

Définition 23 (Etat de blocage)**.** *Un état accessible M est un état de blocage (ou état puits si et seulement si aucune transition n'est franchissable.*

Un réseau admet un état de blocage si le graphe des marquages contient un sommet sans successeur.

3.4.5 Etat d'accueil

Il s'agit de vérifier que le système peut toujours se replacer dans un état donné qui est appelé état d'accueil.

Définition 24 (Etat d'accueil)**.** *Un réseau de Petri a un état d'accueil M_a si pour tout marquage accessible M, il existe une séquence de transitions σ telle que*
$M \xrightarrow{\sigma} M_a$.

Un réseau de Petri admet un état d'accueil M_a si le graphe des marquages comporte une unique composante fortement connexe qui contient M_a.

Si M_0 est un état d'accueil du réseau, le réseau est dit *réinitialisable*.

3.5 Extensions des RdP

Bien que les réseaux de Petri permettent de modéliser simplement et de manière graphique les mécanismes de base tels que la synchronisation, le choix, la concurrence, les rendez-vous, ..., ils se sont montrés très limitatifs. En effet le simple jeu de consommation et de création de ressources (jetons) ne permet pas de modéliser les systèmes qui sont de plus en plus complexes. Plusieurs travaux visent à étendre le formalisme de référence par les réseaux de Petri pour proposer des modèles plus compacts et/ou plus expressifs. Il faudra toutefois trouver un compromis entre richesse du modèle par l'ajout des extensions et la perte au niveau de la décidabilité de certaines propriétés. La section suivante présente certaines de ces extensions.

3.5.1 Arc inhibiteur

En règle générale, les jetons permettent le tir d'une transition. On peut imaginer le cas d'une modélisation du test à zéro où ce sera plutôt l'absence de jetons dans une place qui induirait le tir d'une transition. Les arcs inhibiteurs [68] apportent cette expressivité au réseau de Petri de base. Un arc inhibiteur relie une place à une transition et l'une des conditions de franchissabilité de cette transition est l'absence de jetons dans la place. Comme on peut le voir, l'ajout des arcs inhibiteurs apporte une puissance d'expression et une certaine aisance dans la modélisation. Cependant, si la place reliée à l'arc inhibiteur est bornée, un réseau de Petri équivalent au réseau de Petri avec des arcs inhibiteurs peut être construit. Dans ce cas, les arcs inhibiteurs apportent une compacité mais aucune expressivité supplémentaire. L'accessibilité, le caractère borné, la vivacité, la couverture sont indécidables pour les réseaux de Petri avec des arcs inhibiteurs. Néanmoins l'accessibilité reste décidable dans le cas d'un seul arc inhibiteur [72].

Définition 25 (Réseaux de Petri avec des arcs inhibiteurs)**.** *Un réseau de Petri avec des arcs inhibiteurs est un tuple* $N = \langle P, T, W, Inh, M_0 \rangle$ *où* $\langle P, T, W, M_0 \rangle$ *est un réseau de Petri et* $Inh : P \times T \rightarrow \mathbb{N}_\infty^*$ [4] *est la relation d'inhibition.*

La définition suivante donne la règle de franchissement pour un réseau de Petri avec des arcs inhibiteurs.

Définition 26 (Règle de franchissement)**.** *Soit N un réseau de Petri avec des arcs inhibiteurs, soit M un marquage accessible et* $t \in T$ *une transition. t est franchissable à partir de M si et seulement si :*

$$\forall p \in P,\ W(p,t) \leq M(p) < Inh(p,t)$$

[4] $\mathbb{N}_\infty^* = \mathbb{N}^* \cup \{+\infty\}$

Le tir de la transition t conduit au marquage M' défini par :

$$\forall p \in P,\ M'(p) = M(p) - W(p,t) + W(t,p)$$

3.5.2 Arc de lecture

Dans le cas des arcs de lecture, on teste le contenu d'une place sans consommer les jetons de la place lors du tir de la transition reliée à l'arc de lecture. [79] voit cette spécification comme une transition qui consomme et crée le même nombre de jetons alors que [9] empêche ce mouvement de jetons. Vu sous l'angle de consommation et de création du même nombre de jetons, les arcs de lecture ne sont qu'une facilité d'écriture et n'apportent aucun pouvoir expressif. les propriétés décidables pour les réseaux de Petri restent décidables pour les réseaux de Petri avec des arcs de lecture.

Définition 27 (Réseaux de Petri avec des arcs de lecture)**.** *Un réseau de Petri avec des arcs de lecture est un tuple $N = \langle P,T,W,Read,M_0\rangle$ où $\langle P,T,W,M_0\rangle$ est un réseau de Petri et $Read : P \times T \to \mathbb{N}$ est l'ensemble des arcs de lecture.*

La définition suivante donne la règle de franchissement pour un réseau de Petri avec des arcs de lecture.

Définition 28 (Règle de franchissement)**.** *Soit N un réseau de Petri avec des arcs de lecture, soit M un marquage accessible et $t \in T$ une transition. t est franchissable à partir de M si et seulement si :*

$$\forall p \in P, M(p) \geq W(p,t) + Read(p,t)$$

Le tir de la transition t conduit au marquage M' défini par :

$$\forall p \in P, M'(p) = M(p) - W(p,t) + W(t,p)$$

3.5.3 Arc de reset

Pour pouvoir vider le contenu d'une place, les arcs de reset sont utilisés [44, 52]. Les règles de sensibilisation des transitions ne changent pas mais dès que la transition reliée à un arc de reset est tirée, la place liée est vidée. Les arcs de reset sont appelés également arcs de vidange. La couverture est décidable pour un réseau de Petri avec des arcs de reset [4], l'accessibilité [3] et la bornitude [35] sont indécidables.

Définition 29 (Réseau de Petri avec des arcs de reset)**.** *Un réseau de Petri avec des arcs de reset est un tuple $N = \langle P,T,W,R,M_0\rangle$ où $\langle P,T,W,M_0\rangle$ est un réseau de Petri et $R : P \times T \to \{0,1\}$ est l'ensemble des arcs de reset ($R(p,t) = 0$ s'il y a un arc de reset qui relie p à t, sinon $R(p,t) = 1$).*

Les arcs de reset ne changent pas les règles de sensibilisation des transitions [24]. Si $M \xrightarrow{t} M'$ alors $\forall p \in P$ tel que $R(p,t) = 0$, $M'(p) = 0$. Mais si $W(t,p) > 0$ alors $M'(p) = W(t,p)$. De façon générale M' est obtenu par :

$$M' = (M - W(.,t)) \odot {}^{5}R(t) + W(t,.) \tag{3.5}$$

3.5.4 Arc de transfert

Contrairement aux arcs de resets où les jetons sont perdus, les arcs de transfert [7], comme leurs noms l'indiquent, permettent de transférer les jetons d'une place vers une ou plusieurs autres place. Le caractère borné d'un réseau de Petri avec des arcs de transfert est décidable [35].

Définition 30 (Réseau de Petri avec des arcs de transfert)**.** *Un réseau de Petri avec des arcs de transfert est un tuple $N = \langle P, T, W, W_T, M_0 \rangle$ où $\langle P, T, W, M_0 \rangle$ est un réseau de Petri et $W_T \subseteq P \times T \times P$ est l'ensemble des arcs de transfert (avec la condition que pout tout $p, p' \in P$ et pour tout $t \in T$, il existe au plus un $(p,t,p') \in W_T$).*

Les arcs de transfert ne changent pas également les règles de sensibilisation des transitions [24]. Mais si $M \xrightarrow{t} M'$ alors M' est obtenu en :

- consommant les pré-conditions de t,
- transférant tous les jetons de la place p vers la place p' si $(p,t,p') \in W_T$
- créant les post-conditions de t comme dans le cas des réseaux de Petri.

3.5.5 Ajout du temps

Pour prendre en compte les contraintes temporelles quantitatives des systèmes temps réel, les réseaux de Petri ont été étendus avec les informations temporelles. Il existe deux grandes écoles, les réseaux de Petri *temporisés* [71] qui associent des durées minimales de tir à chaque transition et les réseaux de Petri *temporels* [61] par contre, ajoutent des intervalles temporels (date de tir au plus tôt et date de tir au plus tard) aux places, aux transitions ou aux arcs. Les modèles correspondants sont respectivement les réseaux de Petri *P-temporels* [49, 48], *A-temporels* [14, 27] ou *T-temporels* [61] qui correspondent respectivement à l'ajout des contraintes temporelles aux places, aux arcs et aux transitions. Les réseaux de Petri T-temporels sont les plus utilisés, nous proposons alors leurs sémantiques opérationnelles à la fin de cette section.

[5] $\odot$: produit matriciel de Hadamard (Section 2.4)

Réseaux de Petri P-temporel (P-TPN)

Les contraintes temporelles sont associées aux places : un intervalle statique I_s est associé à chaque place p. Un jeton contenu dans cette place pourra participer au tir des post-transitions de p (il est dit alors *disponible*) s'il a séjourné au minimum $Min(I_s)$ unité(s) de temps et si sa durée de séjour dans la place ne dépasse pas $Max(I_s)$ unité(s) de temps. Autrement, le jeton est *indisponible* ou *gelé* et au delà des $Max(I_s)$ unité de temps, il sera considéré comme *mort* et ne pourra plus participer au tir des post-transitions de p. Les P-TPN permettent de modéliser les contraintes de rendez-vous en un temps donné qui traduit la disponibilité des ressources.

Définition 31 (P-TPN)**.** *Un réseau de Petri P-temporel est défini par la paire* $\langle N, I_s \rangle$ *où :*

- *N est un réseau de Petri marqué ;*
- $I_s : P \to \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\})$ *est la fonction de temporisation qui associe à chaque place un intervalle statique des durées opératoires des jetons qui s'y trouvent.*

Réseau de Petri A-temporel (A-TPN)

Une autre possibilité d'ajout des contraintes temporelles à un réseau de Petri est de les associer aux arcs [80]. Le modèle a été amélioré avec les réseaux de Petri à flux temporel [23] pour la modélisation de la synchronisation dans les systèmes multimédia.

Définition 32 (A-TPN)**.** *Un réseau de Petri A-temporel est défini par la paire* $\langle N, I_s \rangle$ *où :*

- *N est un réseau de Petri marqué ;*
- $I_s : W|_{P \times T} \to \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\})$ *est la fonction de temporisation qui associe à chaque arc* (p,t) *de W un intervalle statique des durées de tir.*

Réseau de Petri T-temporel (T-TPN)

Les contraintes temporelles sont associées aux transitions : un intervalle statique I_s est associé à chaque transition t. La règle de sensibilisation d'une transition énoncée dans la Définition 8 reste valable pour les transitions d'un T-TPN. Par contre, le tir effectif d'une transition ne subvient au minimum qu'après $Min(I_s)$ unité(s) de temps et au plus tard $Max(I_s)$ unité(s) de temps

après sa sensibilisation. En temps dense, il existe deux sémantiques, une dite *faible* qui autorise une transition à ne pas être franchie même si on atteint sa date de tir au plus tard et l'autre dite *forte* qui force le tir de la transition qui atteint cette borne maximale. Les T-TPN permettent par exemple de modéliser l'urgence de certains évènements comme le mécanisme de *chien de garde* (Figure 3.9). Quelques résultats connus de la décidabilité des T-TPN sont : la bornitude et la vivacité sont indécidables pour les T-TPN [45], la k-bornitude est décidable [11], l'accessibilité est décidable pour les T-TPN bornés [11] et indécidable pour les T-TPN non bornés[45].

Définition 33 (T-TPN)**.** *Un réseau de Petri T-temporel est défini par la paire* $\langle N, I_s \rangle$ *où :*

- *N est un réseau de Petri marqué ;*
- $I : T \to \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\})$ *est la fonction de temporisation qui associe à chaque transition, un intervalle statique des durées de tir.*

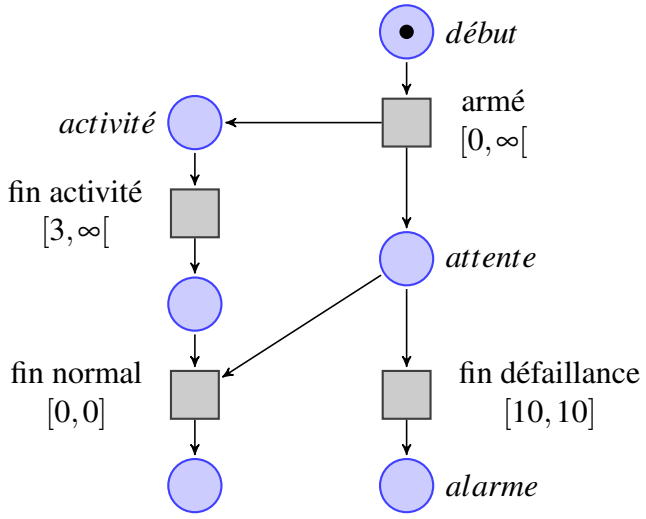


FIGURE 3.9 : Chien de garde [74]

Franchir une transition t sensibilisée par un marquage M ($M \geq W(.,t)$) n'est permis que dans l'intervalle de temps qui lui est affecté : dans cet intervalle, t est dit tirable. $Enabled(M)$ désigne également l'ensemble des transitions sensibilisées par un marquage. A la sensibilisation d'une transition et en fonction de l'évolution du temps, un intervalle dynamique lui est associé et indique l'intervalle de temps $I(t)$ dans lequel elle peut être tirée. Les bornes inférieure et supérieure de $I(t)$ représentent respectivement la date de tir au plus tôt, notée $DMin(t)$, et la date de tir au plus tard, notée $DMax(t)$.

Un état d'un T-TPN est alors caractérisé par un marquage et un intervalle de tir. Nous définissons l'état d'un T-TPN dans la Définition 34.

Définition 34 (Etat d'un T-TPN)**.** *Un état d'un réseau de Petri T-temporel est le couple $q = (M,I)$ où M est un marquage et I est une fonction qui associe un intervalle temporel à chaque transition sensibilisée par M.*

La définition de conflit structurel entre deux transitions pour un T-TPN est la même que celle énoncée dans la Section 3.2.5.2 . Par contre, deux transitions sont en conflit effectif si elles sont en conflit structurel et de plus il existe un état q pour lequel elles sont tirables. Une transition t' est nouvellement sensibilisée par le tir de t à partir du marquage M si t' est sensibilisée par le marquage $M' = M - {}^\bullet t + t^\bullet$ alors qu'elle ne l'était pas par M. $\uparrow Enabled(M)$ désigne l'ensemble des transitions nouvellement sensibilisées :

$$\uparrow Enabled(M) = ({}^\bullet t' \leq M - {}^\bullet t + t^\bullet) \wedge ((t' = t) \vee ({}^\bullet t' > M - {}^\bullet t))$$

Le tir d'une transition t, à un instant θ , depuis un état $q = (M,I)$ est possible si et seulement si :

1. t est sensibilisée par M : $M \geq W(.,t)$;
2. θ n'est pas inférieur à $Mmin(t)$: $\theta \geq DMin(t)$;
3. θ n'est pas supérieur à $DMax(t)$: $\theta \leq DMax(t)$;
4. en sémantique forte, θ n'est supérieur à la date de tir au plus tard d'aucune transition sensibilisée.

Le tir d'une transition sensibilisée t, à la date θ depuis l'état $q = (M,I)$ conduit le T-TPN dans un état $q' = (M',I')$ défini comme suit :

$$(M,I) \xrightarrow{t} (M',I') \Rightarrow \begin{cases} M' = M - W(.,t) + W(t,p) \\ \forall t' \in T, I'(t') = \begin{cases} \text{vide si } t' = t \text{ ou si t et t' sont en conflit} \\ [max(0,DMin(t') - \theta), Dmax(k) - \theta] \text{ ou} \\ ]max(0,DMin(t') - \theta), \infty] \text{ si } t' \in Enabled(M) \\ I_s(t') \text{ si } t' \in \uparrow Enabled(M) \end{cases} \end{cases} \tag{3.6}$$

3.6 Equivalence comportementale des réseaux de Petri

Un réseau de Petri est caractérisé par ses états globaux (ou marquages) et la relation de transitions entre ces états. Mais seulement, deux réseaux de

Petri non structurellement équivalents, peuvent néanmoins représenter des systèmes ayant des comportements que l'on jugera d'équivalents. Le but est de remplacer une partie du réseau de Petri ou tout le réseau de Petri par un autre réseau soit pour avoir un motif plus réduit ou soit pour vérifier une propriété qui est décidable dans le réseau équivalent ou dont la vérification est moins coûteuse. En fonction de ce que l'on veut faire avec le réseau équivalent, il existe plusieurs types d'équivalence entre réseaux de Petri et nous présenterons quelques-unes dans cette section.

3.6.1 Simulation

Définition 35 (Simulation)**.** *Soient* $N_1 = \langle P_1, T_1, W_1, M_{01} \rangle$ *et* $N_2 = \langle P_2, T_2, W_2, M_{02} \rangle$ *deux réseaux de Petri marqués. La relation binaire* $R \subseteq \mathbb{N}^{|P_1|} \times \mathbb{N}^{|P_2|}$ *est une relation de simulation entre* M_1 *et* M_2 *si et seulement si :*

$$\forall (M_1, M_2) \in R \Leftrightarrow \forall t \in T_2 \text{ et } M_2' \mid M_2 \xrightarrow{t}_{N_2} M_2' \Rightarrow \exists\, M_1' \mid M_1 \xrightarrow{t}_{N_1} M_1' \text{ et } (M_1', M_2') \in R \tag{3.7}$$

N_1 simule N_2 si pour tout marquage accessible M_2 du réseau N_2 il existe un marquage M_1 de N_1 tel que $(M_1, M_2) \in R$.

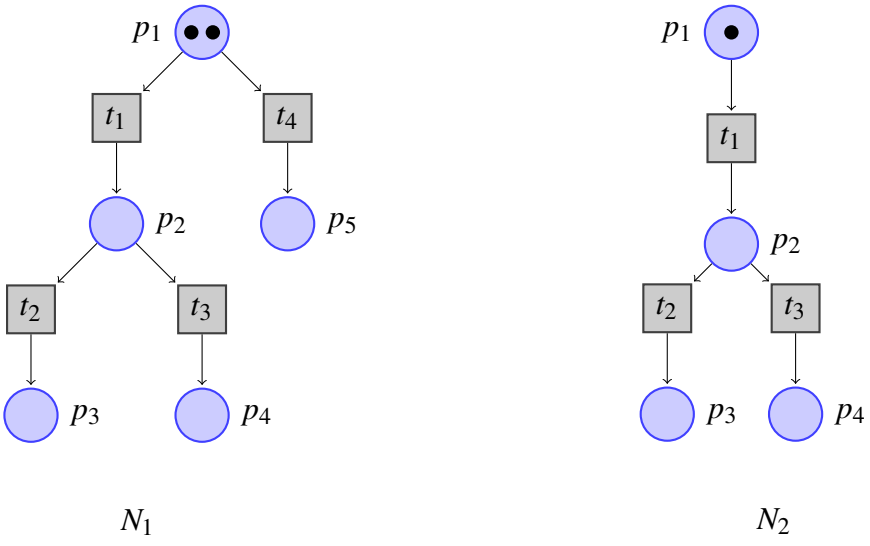


FIGURE 3.10 : N_1 simule N_2

On peut enrichir la définition de la simulation pour avoir une bisimulation entre les deux réseaux.

3.6.2 Bisimulation

La bisimulation a été introduite par Park [67] et est un faible raffinement de l'équivalence comportementale ou observationnelle [63]. La bisimulation définit une relation entre deux systèmes de transition d'états c'est-à-dire les systèmes qui se comportent de la même manière au point où l'un simule l'autre et vice versa.

Définition 36 (Bisimulation)**.** *Soient* $N_1 = \langle P_1, T_1, W_1, M_{01}\rangle$ *et* $N_2 = \langle P_2, T_2, W_2, M_{02}\rangle$ *deux réseaux de Petri marqués. Soit* M_1 *un marquage de* N_1 *et* M_2 *un marquage de* N_2*. La relation binaire* $R \subseteq \mathbb{N}^{|P_1|} \times \mathbb{N}^{|P_2|}$ *est une bisimulation (forte) si et seulement si :*

$$\forall (M_1, M_2) \in R \Leftrightarrow \begin{cases} \forall t \in T_1 \mid M_1 \xrightarrow{t}_{N_1} M_1' \Rightarrow \exists\, M_2' \mid M_2 \xrightarrow{t}_{N_2} M_2' \text{ et } (M_1', M_2') \in R \\ \forall t \in T_2 \mid M_2 \xrightarrow{t}_{N_2} M_2' \Rightarrow \exists\, M_1' \mid M_1 \xrightarrow{t}_{N_1} M_1' \text{ et } (M_1', M_2') \in R \end{cases} \tag{3.8}$$

N_1 est bisimilaire à N_2 (et vice versa) si pour tout marquage accessible M de N_1, il existe un marquage accessible M' de N_2 tel que $(M, M') \in R$.

La définition précédente définit la *bisimulation forte*. Il arrive que le système de transitions d'états inclut la notion de transition non-observable. Dans ce cas, la bisimulation est affaiblie et devient une *bisimulation faible*. Dans le cas de la bisimilation faible, l'équation 3.8 devient :

$$\forall (M_1, M_2) \in R \Leftrightarrow \begin{cases} \forall t \in T_1 \mid M_1 \xrightarrow{t}_{N_1} M_1' \Rightarrow \exists\, M_2' \mid M_2 \xrightarrow{\tau^* t \tau^*}_{N_2} M_2' \text{ et } (M_1', M_2') \in R \\ \forall t \in T_2 \mid M_2 \xrightarrow{t}_{N_2} M_2' \Rightarrow \exists\, M_1' \mid M_1 \xrightarrow{\tau^* t \tau^*}_{N_1} M_1' \text{ et } (M_1', M_2') \in R \end{cases} \tag{3.9}$$

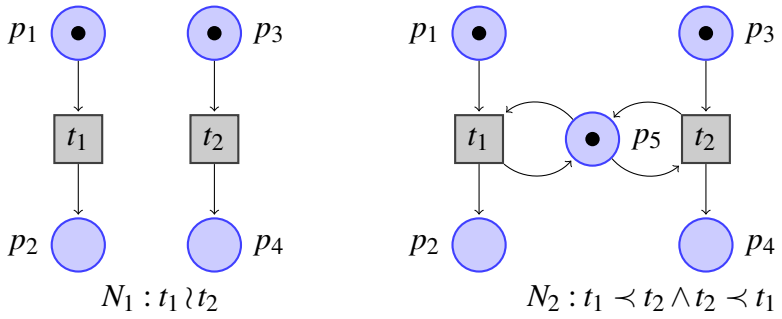


FIGURE 3.11 : N_1 et N_2 sont bisimilaires

La bisimulation est définie en terme de séquences d'exécution. Dans ce cas, la bisimulation ne fait pas de différence entre les exécutions concurrentes et séquentielles c'est-à-dire l'ordre partiel entre les événements [13].

Par exemple, sur la Figure 3.11, N_1 et N_2 sont bisimilaires mais t_1 et t_2 n'ont pas les mêmes ordres partiels. Les transitons t_1 et t_2 sont concurrentes dans N_1 et dans N_2, elles sont dans une relation de causalité. Pour préserver l'ordre partiel, Glabbeek et Vaandrager [78] proposent une définition de la bisimulation basée sur l'ordre partiel appelée *pomset bisimulation*.

3.6.3 Pomset bisimulation

Pomset

Définition 37 (Pomset)**.** *Un pomset [70] sur un réseau de Petri* $N = \langle P, T, W, M_0 \rangle$ *est le triplet* $(E, \preccurlyeq, l)$ *avec :*

- E *un ensemble d'événements ;*
- $\preccurlyeq$ *une relation d'ordre partielle et* l
- *la fonction de labellisation de* E *vers* T.

Un *pomset* est une abstraction de l'exécution d'un système concurrent [32]. Soit e un élément de E, $l(e)$ correspond à une transition du système ; plusieurs éléments de E peuvent alors être labellisés par la même transition. Un pomset représente un multi-ensemble d'événements qui se produisent suivant un ordre indiqué. $\emptyset$ appartient à tout pomset. Les exécutions possibles du réseau N_1 de la Figure 3.12.a. sont représentées par les pomsets $\emptyset$, a, b , $\begin{matrix} a \\ b \end{matrix}$, $a \to b$, $a \to b \to c$.

Pomset bisimulation

Définition 38 (Pomset bisimulation)**.** *Soient deux réseaux de Petri labellisés,* $N_1 = \langle P_1, T_1, W_1, A_1, l_1, M_{01} \rangle$ *et* $N_2 = \langle P_2, T_2, W_2, A_2, l_2, M_{02} \rangle$. *Soit* M_1 *un marquage accessible de* N_1 *et* M_2 *celui de* N_2. *Soit u un pomset. La relation binaire* $R \subseteq \mathbb{N}^{|P_1|} \times \mathbb{N}^{|P_2|}$ *est une pomset bisimulation si et seulement si :*

$$\forall (M_1, M_2) \in R \Leftrightarrow \begin{cases} \forall u \mid M_1 \xrightarrow{u}_{N_1} M_1' \Rightarrow \exists M_2' \mid M_2 \xrightarrow{u}_{N_2} M_2' \text{ et } (M_1', M_2') \in R \\ \forall u \mid M_2 \xrightarrow{u}_{N_2} M_2' \Rightarrow \exists M_1' \mid M_1 \xrightarrow{u}_{N_1} M_1' \text{ et } (M_1', M_2') \in R \end{cases} \tag{3.10}$$

N_1 et N_2 sont pomset bisimilaires si pour tout marquage accessible M de N_1, il existe un marquage accessible M' de N_2 tel que $(M, M') \in R$. Sur la Figure 3.12, N_1 et N_2 sont pomset bisimilaires

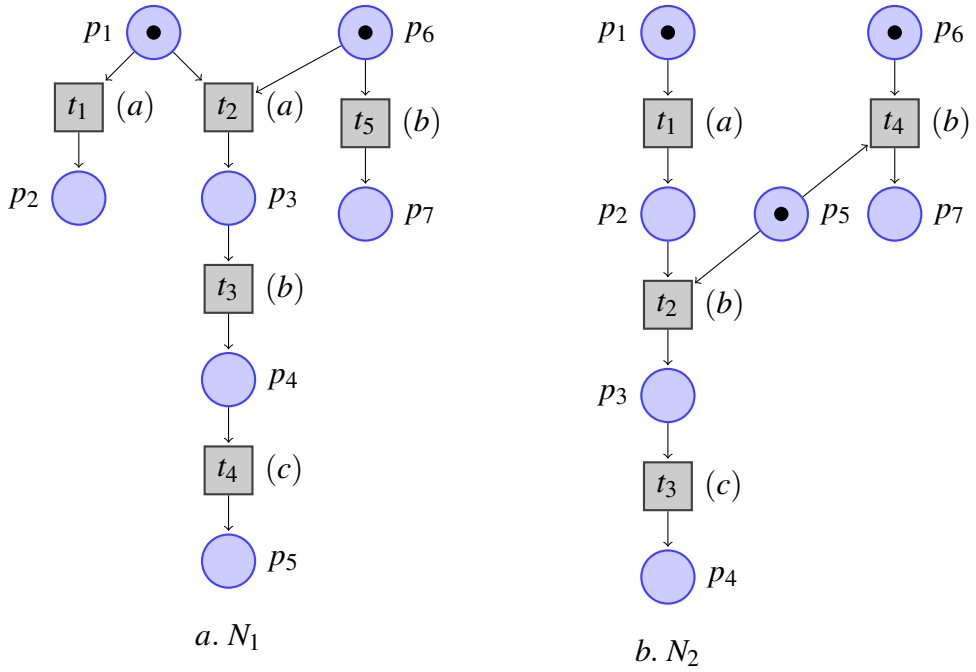


FIGURE 3.12 : N_1 et N_2 sont pomset bisimilaires

3.7 Conclusion

Dans ce chapitre, nous avons présenté les réseaux de Petri qui constituent un bon outil de modélisation pour la spécification des systèmes temps réel. Ils expriment aisément les conflits, le parallélisme vrai, les mécanismes de synchronisation et de communication inhérents au fonctionnement des systèmes concurrents. Des extensions au modèle de base ont été proposées afin d'étendre leurs expressivités ou leurs représentations.

Nous avons montré aussi que certaines propriétés sont exprimables sur les réseaux de Petri telles que la bornitude, le blocage, la terminaison, la vivacité, ... Les méthodes de vérification de ces propriétés sont basées sur la construction du graphe d'états ou graphe des marquages accessibles. Par exemple, si la construction du graphe d'états se termine sans abandon, alors le réseau est borné. Ou par exemple, si le graphe possède un circuit, alors le réseau admet une séquence infinie ou encore si le graphe d'états contient un sommet sans successeur alors il existe un blocage dans le réseau, ...

Par contre, la construction du graphe d'états n'est toujours pas possible. On ne peut pas construire un graphe d'états pour un réseau de Petri non borné. Même pour les réseaux de Petri bornés, l'espace d'états croît exponentiellement et la construction du graphe d'états est confrontée au problème d'explosion combinatoire [12]. Le problème d'explosion combinatoire est

causé par la sémantique d'entrelacement qui est une technique très coûteuse de représentation du parallélisme. Toutes les séquences alternatives de tous les événements concurrents sont représentées afin d'avoir l'espace d'états. Une façon de contenir cette explosion est basée sur l'idée de conserver les ordres partiels entre les événements. Les techniques d'ordre partiel et plus particulièrement le dépliage [26], éliminent la représentation du parallélisme par l'entrelacement des actions et produisent des comportements, sous forme de succession d'événements sans entrelacer tous les événements concurrents.

Dans le chapitre suivant, nous présentons le dépliage des réseaux de Petri et notre contribution au dépliage des réseaux de Petri avec arc de resets.

4

Méthode d'ordre partiel : dépliage des réseaux de Petri

Sommaire

4.1 Introduction

A la fin du chapitre précédent, nous introduisions le dépliage comme une alternative moins coûteuse, à la construction du graphe d'états. Outre le fait, qu'il réduit la combinatoire en supprimant les entrelaçages de parallélisme, sa structure même, présente d'autres avantages. C'est un graphe acyclique dans lequel chaque condition a au plus un évènement précédent, ce qui le rend adapté au test et au diagnostic dans les grands systèmes [43, 10]. Il est aussi utilisé en vérification [28] et en planification [40]. Les ordres partiels, permettent de plus, d'expliciter les exécutions concurrentes, ce qui permet d'exécuter le système en parallèle [46]. La Section 4.2 présente le dépliage des réseaux de Petri et la Section 4.3 est consacrée au dépliage des réseaux de Petri avec des arcs de reset.

4.2 Dépliage des réseaux de Petri

Le dépliage de réseaux de Petri fait partie des méthodes dites *d'ordre partiel* qui sont une alternative au calcul de l'espace d'états du réseaux de Petri. L'objectif du dépliage est de représenter explicitement l'ordre partiel des évènements du système étudié par un réseau de Petri particulier. Cette catégorie correspond aux réseaux d'occurrences [66, 17] et utilise une fonction d'homomorphisme qui associe aux évènements (respectivement aux conditions) du réseau d'occurrence les transitions (respectivement les places) du réseau étudié [26].

Pour un réseau de Petri, le nombre d'évènements du dépliage peut être infini et les méthodes de vérification se basent plutôt sur le calcul d'une partie du dépliage appelée *préfixe "fini"*. Une des difficultés est de s'assurer qu'un préfixe est suffisamment grand pour permettre la vérification d'une propriété donnée. Il existe des algorithmes (dont le plus connu est celui de McMillan [60] et amélioré par Esparza, Römer et Vogler [31]) qui donnent une méthode de construction du préfixe fini et complet du dépliage.

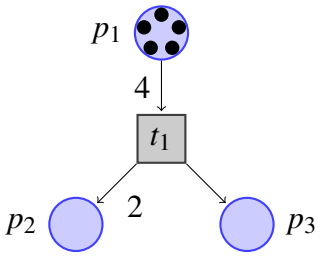
4.2.1 Homomorphisme de réseaux

Le dépliage est basé sur l'ordre partiel des événements. On a besoin de définir la manière dont un réseau peut être partiellement simulé par un autre tout en préservant les nœuds (les places et les transitions). La définition suivante formalise la notion d'*homomorphisme de réseaux*.

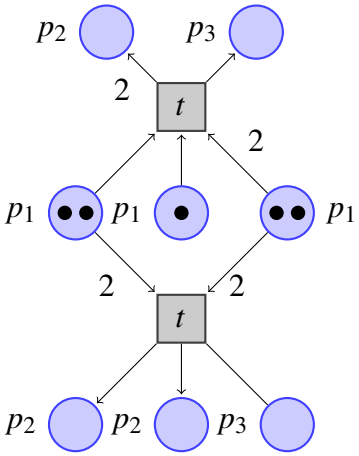
Définition 39 (Homomorphisme de réseaux)**.** *Soient* $N_1 = \langle P_1, T_1, W_1, M_{01} \rangle$ *et* $N_2 = \langle P_2, T_2, W_2, M_{02} \rangle$ *deux réseaux de Petri. Soit* λ *une application* $\lambda : P_1 \cup T_1 \to P_2 \cup T_2$ *telle que* $\lambda(P_1) \subseteq P_2$, $\lambda(T_1) \subseteq T_2$. λ *est un homomorphisme de réseaux de* N_1 *vers* N_2 *si :*

- $\forall\, t \in T_1 : W_2(.,\lambda(t)) = \lambda(W_1(.,t))$;
- $\forall\, t \in T_1 : W_2(\lambda(t),.) = \lambda(W_1(t),.)$;
- $M_{02} = \lambda(M_{01})$.

Les deux premiers items de la définition assurent que l'environnement des transitions est préservé par l'application λ et le troisième impose à λ que les marquages initiaux des deux réseaux soient les mêmes.



a. N : Réseau de Petri



b. Un homomorphisme du réseau N

FIGURE 4.1 : Réseau de Petri et un homomorphisme de ce réseau [19]

La Figure 4.1 illustre l'exemple d'un réseau de Petri et un homomorphisme de ce réseau. La proposition suivante donne la relation entre les marquages accessibles (ou comportements) entre les réseaux par un homomorphisme.

Proposition 1. *Soient deux réseaux de Petri* $N_1 = \langle P_1, T_1, W_1, M_{01} \rangle$ *et* $N_2 = \langle P_2, T_2, W_2, M_{02} \rangle$. *Soit* λ *un homomorphisme de* N_1 *vers* N_2. *S'il existe dans* N_1 *une transition* t *et deux marquages* M *et* M' *tels que* $M \xrightarrow{t}_{N_1} M'$ *alors, dans* N_2, $\lambda(M) \xrightarrow{\lambda(t)}_{N_2} \lambda(M')$.

Preuve. D'après les règles de franchissement d'une transition (Définition 8) et de tir effectif d'une transition (Définition 9), $M \xrightarrow{t}_{N_1} M'$ implique que $M \geq W_1(.,t)$ et $M' = M - W_1(.,t) + W_1(t,.)$

D'après la définition d'un homomorphisme, $W_2(.,\lambda(t)) = \lambda(W_1(.,t))$ et $W_2(\lambda(t),.) = \lambda(W_1(t),.)$. Ainsi :

$$\begin{array}{rcl} M \geq W_1(.,t) & \Rightarrow & \lambda(M) \geq \lambda(W_1(.,t)) \\ & \Rightarrow & \lambda(M) \geq W_2(.,\lambda(t)) \ (1) \end{array}$$

$$\begin{array}{rcl} M' = M - W_1(.,t) + W_1(t,.) & \Rightarrow & \lambda(M') = \lambda(M) - \lambda(W_1(.,t)) + \lambda((W_1(t,.)) \\ & \Rightarrow & \lambda(M) - W_2(.,\lambda(t)) + W_1(\lambda(t),.) \ (2) \end{array}$$

De (1) et (2), $\lambda(M) \stackrel{\lambda(t)}{\rightarrow}_{N_2} \lambda(M')$.

□

De cette proposition qui peut être étendue à une séquence de transitions, on déduit le corollaire suivant.

Corollaire 1. *S'il existe un homomorphisme de réseaux de N_1 vers N_2 alors il existe une relation d'inclusion entre l'ensemble des états accessibles des deux réseaux N_1 et N_2 :*

$$\lambda(\mathscr{A}(N_1)) \subseteq \mathscr{A}(N_2) \tag{4.1}$$

4.2.2 Réseau d'occurrence

Les processus de branchement et le dépliage de réseaux de Petri sont représentés par une classe particulière de réseaux de Petri appelée *réseaux d'occurrences* introduits par Nielsen, Plotkin et Winskel [66].

Un réseau d'occurrence est un réseau de Petri sauf, sans boucle dont les places sont appelées *conditions* et les transitions, *évènements*. Comme dans tout réseau de Petri on peut définir les pré-conditions ${}^\bullet x$ d'un nœud x et ses post-conditions $x^\bullet$. La définition suivante est celle de la notion de réseau d'occurrence.

Définition 40 (Réseau d'occurrence)**.** *Un réseau d'occurrences est un réseau de Petri $O = \langle B, E, F \rangle$ tel que :*

- $|{}^\bullet b| \leq 1, \forall b \in B$ *;*
- *O est acyclique ;*
- *O est fini par précédence, c'est-à-dire $\forall x \in B \cup E$, $\{y \in B \cup E | y \prec x\}$ est fini ;*
- *aucun élément n'est en conflit avec lui-même.*

On désigne par $Min(O)$ les conditions minimales de O et $Max(O)$, les conditions maximales.

- $Min(O) = \{b \in B \text{ tel que } |{}^\bullet b| = 0\}$
- $Max(O) = \{x \in B \cup E \text{ tel que } |x^\bullet| = 0\}$

Dans un réseau d'occurrence, les trois types de relations (la causalité, le conflit et la concurrence), mutuellement exclusives, sont définies entre deux nœuds quelconques (Section 3.2.5).

4.2.3 Processus de branchement ou processus arborescents

Les réseaux d'occurrence associés aux homomorphismes de réseaux, permettent de définir la notion de *processus de branchement* [26].

Définition 41 (Processus arborescent)**.** *Soient $N = \langle P, T, W, M_0 \rangle$ un réseau de Petri, $O = \langle B, E, F \rangle$ un réseau d'occurrence et λ un homomorphisme de réseau de O dans N. La paire $\langle O, h \rangle$ est un processus arborescent de $\langle N, M_0 \rangle$ si :*

$$\forall e_1, e_2 \in E : ((Pre(e_1) = Pre(e_2)) \wedge (h(e_2) = h(e_1))) \Rightarrow e_1 = e_2$$

La définition impose qu'un même évènement ne soit représenté au sein d'un même processus qu'une fois. Un processus arborescent est plus petit qu'un autre s'il peut être obtenu à partir du second par élimination de certains évènements et/ou de certaines conditions. On peut établir un ordre partiel entre les processus arborescents :

Définition 42 (Ordre partiel entre processus de branchement)**.** *Soient β_1 et β_2 deux processus arborescents d'un réseau de Petri. β_1 est plus petit que β_2 (noté $\beta_1 \sqsubseteq \beta_2$) s'il existe un homomorphisme injectif de β_1 vers β_2.*

Chaque processus de branchement fournit un moyen de représenter une exécution finie d'un réseau de Petri. C'est donc un ensemble d'évènements partiellement ordonné (*ordre partiel*) par la relation de causalité. Dans un processus de branchement, si deux évènements ne sont pas en causalité, ils sont explicitement en concurrence. Il n'y a pas de conflit dans un processus de branchement. La notion de conflit peut être re-définie par rapport à la notion de processus de branchement comme suit.

Définition 43 (Conflit)**.** *Deux évènements e_1 et e_2 sont en conflit, si et seulement s'ils respectent l'une des propositions suivantes :*

- *Il n'existe aucun processus de branchement qui contient à la fois* e_1 *et* e_2 *;*
- $\lceil e_1 \rceil \cup \lceil e_2 \rceil$ *n'est pas un processus de branchement* [1] *;*
- *il existe* $e'_1 \in \lceil e_1 \rceil$ *et* $e'_2 \in \lceil e_2 \rceil$ *tel que* $e'_1 \neq e'_2$ *et* ${}^\bullet e'_1 \cap {}^\bullet e'_2 \neq \emptyset$.

La notion de configuration permet de définir les comportements possibles d'un réseau d'occurrence.

Définition 44 (Configuration)**.** *La configuration C d'un réseau d'occurrence est un ensemble d'évènements satisfaisant les deux conditions suivantes :*

- $e \in C \Rightarrow \forall e' \preccurlyeq e,\ e' \in C$
- $\forall e, e' \in C,\ \neg(e \perp e')$

Le "tir" d'une configuration C aboutit à un ensemble de conditions, noté $C^\bullet$, qui sont les post-conditions obtenues après les occurrences de tous les évènements contenus dans C. Les pré-conditions de la configuration (notée ${}^\bullet C$) correspondent à toutes les conditions b qui appartiennent aussi à $Min(O)$ contribuant à produire un évènement de C.

Définition 45 (Pré-conditions d'une configuration)**.** *Soit C une configuration d'un réseau d'occurrence* $O = \langle B, E, F \rangle$*. L'ensemble* ${}^\bullet C$ *des pré-conditions de C est l'ensemble des conditions* $b \in B$ *qui vérifie :*

- $b \in Min(O)$
- $\exists e \in C \mid b \in {}^\bullet e$

Définition 46 (Post-conditions d'une configuration)**.** *Soit C une configuration C d'un réseau d'occurrence* $O = \langle B, E, F \rangle$*. L'ensemble* $C^\bullet$ *des post-conditions de C est défini par :*

$$C^\bullet = \{b \in B \mid {}^\bullet b \in C \wedge b^\bullet \notin C\}$$

Une configuration est un processus de branchement auquel on a retiré des conditions. Les notions suivantes de *co-set* et de *cut* vont nous permettre de retrouver les marquages accessibles du réseau de Petri.

Définition 47 (co-set)**.** *Un co-set, dans un processus de branchement, est un ensemble de conditions* $B' \subseteq B$*, contenant toutes les paires de conditions du processus qui sont en relation de concurrence.*

[1] $\lceil e \rceil$ est la configuration locale de l'évènement e (Définition 49).

Définition 48 (cut)**.** *Un cut est un co-set maximal au sens de l'inclusion.*

A un *cut*, est associé un marquage accessible du réseau de Petri [25, 26]. Pour toutes les places $p \in P, M(p)$ correspond aux conditions contenues dans B'. On notera $\lambda(B')$ ce marquage. Pour une configuration C, le *co-set* $Cut(C)$ est le marquage atteint par le franchissement des évènements de C à partir du marquage initial.

$$Cut(C) = Min(O) + C^{\bullet} - {}^{\bullet}C \tag{4.2}$$

On notera $Mark(C)$ le marquage accessible atteint par le franchissement des évènements de C. On désigne par $Conf(O)$ l'ensemble des configurations du réseau d'occurrence O.

Dans la pratique, on considère que les préfixes de dépliages et la notion de configuration est remplacée par celle de $configuration\ locale$. La configuration locale d'un évènement e correspond à tout le passé causal de e . La configuration locale de e est tout simplement la plus petite configuration contenant e.

Définition 49 (Configuration locale)**.** *La configuration locale d'un événement e est notée* $\lceil e \rceil$ *et définie par :*

$$\forall e \in E, \lceil e \rceil = \{e' \in E \mid e' \preccurlyeq e\}$$

On peut imaginer une structure qui représente l'ensemble des processus de branchement. Cette structure, considérée comme le plus grand processus de branchement (dans le sens de $\sqsubseteq$) est appelé le dépliage [66, 26] et consiste tout simplement en la superposition de tous les processus de branchement.

4.2.4 Dépliage d'un réseau de Petri

Il est possible de superposer tous les processus de branchement β_i d'un réseau de Petri N au sein d'un seul processus de branchement. Le résultat est le dépliage Unf_N d'un réseau de Petri N. Nous le noterons Unf_N, et si il n'y a pas de confusion, on le notera simplement Unf.

$$Unf \stackrel{def}{=} \bigcup \beta_i$$

Définition 50 (Dépliage)**.** *Soit* $N = \langle P, T, W, M_0 \rangle$ *un réseau de Petri. Il existe un unique (à une équivalence près) plus grand (au sens de* $\sqsubseteq$*) processus arborescent de N. Ce processus arborescent est appelé le dépliage de N et nous le notons* $Unf = \langle O, \lambda \rangle$ *avec :*

- $O = \langle B, E, F \rangle$ *un réseau d'occurrence et*
- λ *un homomorphisme de O vers N.*

Nous abordons maintenant la construction pratique du dépliage d'un réseau de Petri. Nous adoptons la même structure de données que les auteurs de [30] pour représenter les conditions et les évènements. Une condition est représentée par deux éléments : la place $p \in P$ à laquelle elle est liée et l'évènement produisant cette condition ($\emptyset$ pour les conditions initiales). De même, un évènement est représenté par la transition $t \in T$ à laquelle il est lié et un ensemble X formé par les pré-conditions de l'évènement : $X \neq \emptyset$.

La Définition 51 donne les Possibles Extensions (PE) d'un processus de branchement. Le dépliage commence par la création des conditions liées au marquage initial. Un nouvel évènement appartenant aux extensions possibles est ajouté au dépliage ainsi que ses post-conditions. A partir des post-conditions, les nouvelles extensions possibles sont calculées et l'opération recommence jusqu'à ce qu'il n'y ait plus d'extensions possibles.

Définition 51 (Possibles extensions)**.** *Soit β un processus de branchement d'un réseau de Petri N. Les possibles extensions de β : $PE(\beta)$ sont les évènements de la forme (t, X) où X est un cut et t une transition de N telle que : $\lambda(X) =^\bullet t$ et (t, X) n'appartient pas déjà à β.*

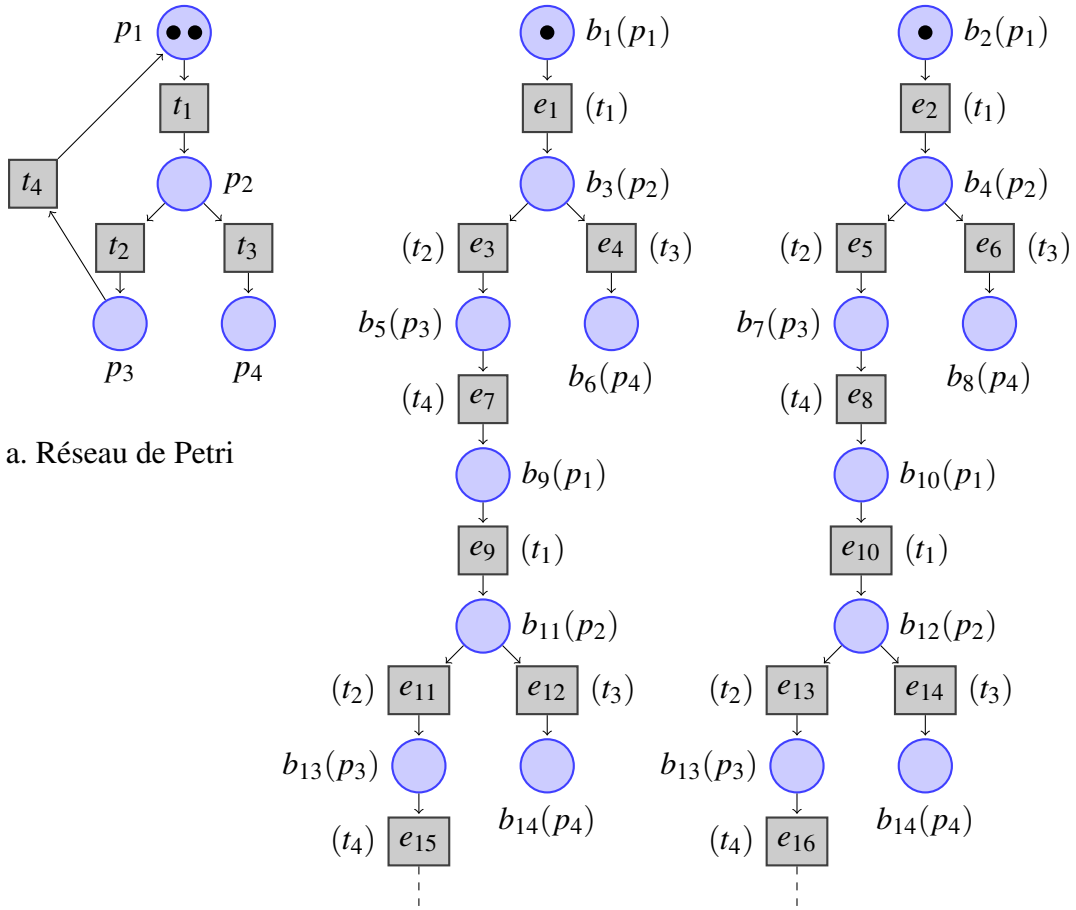
L'Algorithme 3 donne la procédure complète de construction du dépliage d'un réseau de Petri.

Algorithme 3 (Algorithme de construction du dépliage d'un réseau de Petri)**.**

Soit $N = \langle P, T, W, M_0 \rangle$ un réseau de Petri avec $M_0 = \{p_1, ..., p_k\}$. Le dépliage Unf de N est obtenu par :

1. *Initialisation : $Unf := \{(p_1, \emptyset), ..., (p_k, \emptyset)\}$ et $pe := PE(Unf)$*
2. *Tant que $pe \neq \emptyset$ faire :*
   (a) *ajouter à Unf un évènement $e = (t, X)$ de pe et la condition (p, e) pour chaque post-condition p de t*
   (b) *$pe := pe \backslash \{e\}$*
   (c) *$pe := PE(Unf)$*

La Figure 4.2 donne en a. un exemple de réseau de Petri et en b. le début de son dépliage. Le processus de construction du dépliage peut s'exécuter indéfiniment si le réseau de Petri N permet un ou plusieurs comportements infinis. La solution pour conduire alors des vérifications, serait de considérer un préfixe fini (et idéalement complet) du dépliage du réseau.



a. Réseau de Petri

b. Dépliage du réseau de Petri

FIGURE 4.2 : Exemple du dépliage d'un réseau de Petri

4.2.5 Préfixe fini et complet du dépliage d'un réseau de Petri

McMillan [60] a été le premier à montrer qu'il existe un préfixe fini et complet du dépliage d'un réseau de Petri borné qui contient suffisamment d'informations pour retrouver l'espace d'états du système. Esparza, Römer et Vogler ont proposé dans [31] une amélioration de l'algorithme de McMillan. La construction de ce préfixe est basée sur l'identification d'évènements cut-off. Un évènement cut-off est un évènement déjà arrivé dans la construction du dépliage : il est inutile de le représenter à nouveau. Pour qualifier un évènement de cut-off, Esparza, Römer et Vogler [31] explorent sa configuration locale alors que Khomenko, Koutny et Vogler [50] explorent la configuration globale. Ce qui conduit à un calcul plus lent mais qui donne des préfixes plus court au sens de l'inclusion. L'Algorithme 4 donne la procédure complète pour construire le préfixe fini et complet d'un réseau de Petri borné. Cet algorithme est presque similaire à l'Algorithme 3. A la différence de ce der-

nier, le nouvel algorithme, vérifie d'abord si un évènement est cut-off avant de l'ajouter. Ce préfixe fini et complet existe pour tout réseau de Petri sauf [59]. De plus, il est prouvé dans [31] que ce préfixe existe pour tout réseau de Petri non sauf mais borné, qu'il est unique et qu'en plus d'être fini, il est complet c'est-à-dire que pour tout marquage global accessible M du réseau de Petri, il existe une configuration C dans le préfixe tel que $Mark(C) = M$. Le préfixe fini et complet sera noté Fin par la suite. La complétude de ce préfixe est assurée par une classe d'ordres partiels sur les configurations du dépliage appelée ordre adéquat.

Définition 52 (Ordre adéquat)**.** *Un ordre partiel $\preccurlyeq$ sur les configurations d'un dépliage $\langle O, \lambda \rangle$ est un ordre adéquat [31] si et seulement si :*

- *$\preccurlyeq$ est bien fondé (c'est-à-dire qu'il n'existe pas de suite infinie strictement décroissante) ;*
- *$C_1 \subset C_2$ implique que $C_1 \preccurlyeq C_2$;*
- *$\preccurlyeq$ est préservé par les extensions finies c'est-à-dire pour tout évènement e et pour toutes configurations C_1 et C_2 de O telles que $Mark(C_1) = Mark(C_2)$:*

$$C_1 \preccurlyeq C_2 \Rightarrow \forall C_2' \in C_2 \oplus e^2, \exists C_1' \in C_1 \oplus e \text{ tel que } C_1' \preccurlyeq C_2'$$

La finitude du préfixe est garantie par l'identification des évènements cut-off.

Définition 53 (Cut-off)**.** *Un évènement e est un évènement cut-off de Unf si et seulement s'il existe un évènement $e' \in Unf$ tel que :*

- *$Mark(\lceil e \rceil) = Mark(\lceil e' \rceil)$ et*
- *$\lceil e' \rceil \preccurlyeq \lceil e \rceil$.*

L'Algorithme 4 présente la procédure pour construire le préfixe fini et complet du dépliage des réseaux de Petri *bornés*. La Figure 4.3 donne le préfixe fini et complet du dépliage du réseau de Petri de la Figure 4.2 a. Les évènements e_9 et e_{10} sont cut-off car ils sont identifiés, respectivement, aux évènements e_1 et e_2.

Algorithme 4 (Algorithme de construction du préfixe fini et complet du dépliage d'un réseau de Petri)**.** *Soit $N = \langle P, T, W, M_0 \rangle$ un réseau de Petri borné avec $M_0 = \{p_1, ..., p_k\}$. Le préfixe fini et complet du dépliage Fin de N est obtenu par :*

[2] Pour une configuration C, $C \oplus e$ est l'extension de C avec l'évènement e

1. *Initialisation :* $Fin := \{(p_1,\emptyset),...,(p_k,\emptyset)\}$, $pe := PE(Fin)$ *et* $co := \emptyset$ *(l'ensemble des évènements cut-off).*
2. *Tant que* $pe \neq \emptyset$ *faire :*
   (a) *Choisir un évènement* $e = (t,X)$ *de* pe *tel que* $\lceil e \rceil$ *soit minimale ;*
      i. *Si* e *est cut-off alors ajouter* e *à* co
      ii. *Sinon ajouter* e *à* Fin *et la condition* (p,e) *pour chaque post-condition* p *de* $\lambda(e)$
   (b) $pe := pe \backslash \{e\}$
   (c) $pe := PE(Fin)$

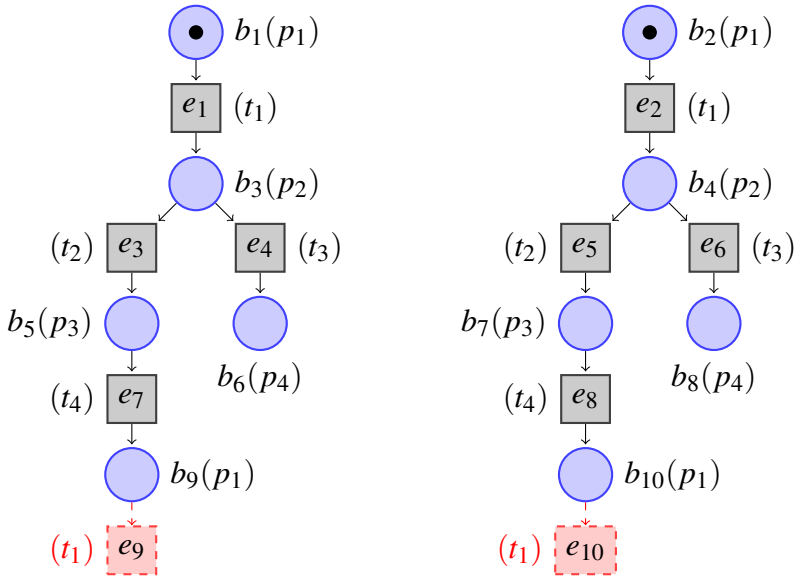


FIGURE 4.3 : Préfixe fini et complet du RdP de la Figure 4.2 a.

Les techniques du dépliage et du préfixe du dépliage ont été appliquées à d'autres classes de réseaux de Petri. Pour les réseaux de Petri non bornés, [1, 22] proposent une technique de construction d'un préfixe fini du dépliage. Ce préfixe n'est pas nécessairement complet. Vogler, Semenov et Yakovlev [79] donnent une procédure pour construire le dépliage et le préfixe fini du dépliage pour les réseaux de Petri avec des arcs de lecture. Ceci a été repris par Baldan et *all* [8] et étendu aux réseaux de Petri à arcs inhibiteurs.

Sur la base du dépliage des réseaux de Petri, des algorithmes ont été proposés pour la construction du dépliage des réseaux de Petri temporels. On peut citer entre autres les travaux de Chatain [16], celui de Traonouez [55] de Sogbohossou [65], ...

Dans la section qui suit, nous proposons la construction du dépliage des réseaux de Petri avec des arcs de reset et plus particulièrement un algorithme pour la construction du préfixe fini et complet des réseaux de Petri avec des arcs de reset bornés.

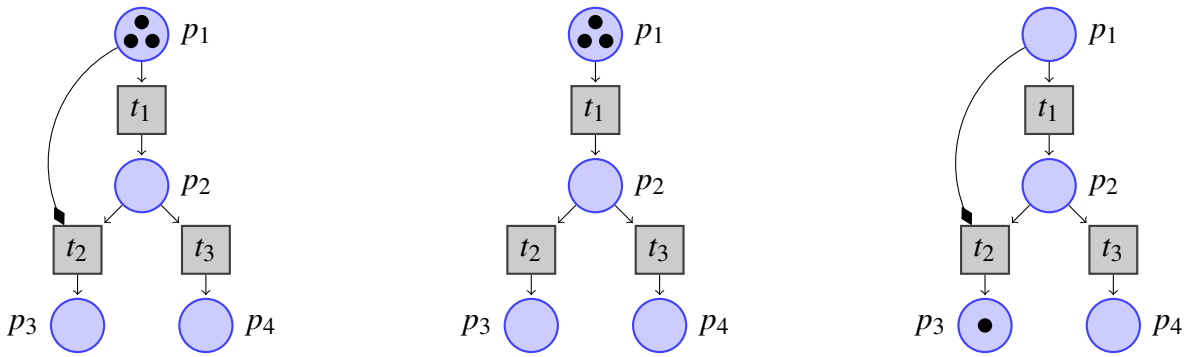
4.3 Dépliage des réseaux de Petri avec des arcs de reset

Nous démontrons avec la Proposition 3 de la Section 4.3.2 qu'il n'existe pas de réseau de Petri équivalent à un réseau de Petri avec des arcs de reset (Section 3.5.3) qui soit à la fois bisimilaire et respectant l'ordre partiel entre les évènements. Ceci est lié à l'expressivité qu'introduisent les arcs de reset : Nous montrerons dans cette même proposition que la traduction d'un *arc de reset engendre des causalités entre deux transitions initialement en concurrence* dans le réseau de Petri sous-jacent.

Et donc, pour préserver les causalités et les ordres partiels, il nous est apparu indispensable d'introduire des arcs de reset *a posteriori* dans le dépliage des réseaux de Petri avec des arcs de reset. Ce nouveau dépliage devra en effet aboutir, lui-même, à un réseau de Petri sauf avec des arcs de reset.

4.3.1 Réseau de Petri sous jacent d'un réseau de Petri avec des arcs de reset

Définition 54 (réseau de Petri sous-jacent d'un réseau de Petri avec des arcs de reset)**.** *Le réseau de Petri sous-jacent d'un réseau de Petri avec des arcs de reset $N_R = \langle P, T, W, R, M_0 \rangle$ est défini par $N = \langle P, T, W, M_0 \rangle$. C'est le réseau de Petri obtenu en supprimant tous les arcs de reset.*



a. RdP avec des arcs de reset b. RdP sous-jacent c. Tir de la séquence $t_1 t_2$

FIGURE 4.4 : RdP avec des arcs de reset et son RdP sous-jacent

La Figure 4.4.a présente un réseau de Petri avec un arc de reset et son réseau de Petri sous-jacent (Figure 4.4.b). A partir du marquage initial $M_0 = (3\ 0\ 0\ 0)$, le tir de la séquence t_1t_2 conduit au marquage $M_2 = (0\ 0\ 1\ 0)$ (Figure 4.4.c).

Proposition 2. *Soient $N_R = \langle P,T,W,R,M_0\rangle$ un réseau de Petri avec des arcs de reset et $N = \langle P,T,W,M_0\rangle$ son réseau de Petri sous-jacent. N simule N_R.*

Preuve. Soient M_R un marquage de N_R et M un marquage de N. Soit R une relation entre N_R et N telle que $(M_R,M) \in R$ si et seulement si $M \geq M_R$. Démontrons que R est une relation de simulation :

Soit $(M_R,M) \in R$ alors $M \geq M_R$

1. $\forall t \in T$ tel que $M_R \geq W(p,t)$ alors il existe un marquage M'_R dans N_R tel que $M_R \xrightarrow{t}_{N_R} M'_R$ et $M'_R = (M_R - W(p,t)) \odot R(t) + W(t,p)$;

2. $\forall t \in T$ tel que $M_R \geq W(p,t)$, nous avons aussi $M \geq W(p,t)$ (parce que $M \geq M_R$) ;

3. Finalement, $M \xrightarrow{t}_N M' = M - W(p,t) + W(t,p)$ et $M' \geq M'_R$.
   En effet, comme $R(t)$ est à valeur dans {0,1}, M' est au moins égal à M'_R.

Par conséqent $\forall t \in T$ tel que $M_R \xrightarrow{t}_{N_R} M'_R$, $\exists M'$ tel que $M \xrightarrow{t}_N M'$ et $(M'_R,M') \in R$. De plus, N_R et N ont le même marquage initial M_0 et évidemment $(M_0,M_0) \in R$, alors R est une relation de simulation et N simule N_R.

□

On peut remarquer que les arcs de reset ne rajoutent pas de comportements au réseau de Petri sous-jacent, *ils en enlèvent*. Ce qui nous permet de déduire le corollaire suivant :

Corollaire 2. *Le langage de N_R est inclus dans le langage de N.*

4.3.2 Expressivité des arcs de reset

Dans cette sous-section, nous essaierons d'obtenir un motif, ou de définir une transformation permettant de représenter l'expressivité qu'introduisent les arcs de reset. Par la même occasion, nous répondrons à la question suivante : Existe-t-il un réseau de Petri équivalent à tout réseau de Petri avec des arcs de reset ?

Expressivité au regard de la bisimulation

Pour les réseaux non bornés, les arcs de reset augmentent strictement l'expressivité des réseaux de Petri. En effet, il a été prouvé que l'accessibilité est décidable pour les réseaux de Petri non bornés [56, 53]. Une traduction d'un réseau de Petri avec des arcs de reset vers un réseau de Petri qui préserve la bisimulation donnerait un moyen de rendre décidable l'accessibilité pour les réseaux de Petri avec des arcs de reset non bornés. Or cette propriété est indécidable pour les réseaux de Petri avec des arcs de reset non bornés [3]. On en conclut qu'il n'existe pas de transformation d'un réseau de Petri avec des arcs de reset non borné vers un réseau de Petri qui préserve la bisimulation.

Pour les réseaux bornés, il est simple de voir que les arcs de reset n'ajoutent aucune expressivité du point de vue de la bisimulation. En effet, si un réseau de Petri avec des arcs de reset est borné alors il est possible de construire son graphe des marquages. Tout graphe des marquages est un réseau de Petri sauf dans lequel, les nœuds sont les places et les arcs, les transitions. Ce graphe des marquages est bisimilaire au réseau de Petri avec des arcs de reset. Du point de vue de la bisimulation, les arcs de reset n'ajoutent donc pas d'expressivité.

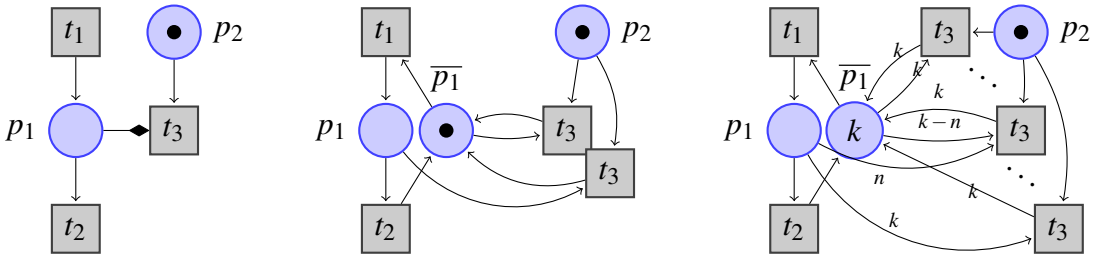
Mais, la bisimulation ne préserve que la sémantique d'entrelacement et non la sémantique d'ordre partiel. Examinons maintenant si les arcs de reset ajoutent de l'expressivité au réseau borné en respectant à la fois la bisimulation et l'ordre partiel.

Expressivité au regard de la bisimulation et de l'ordre partiel

La définition de la causalité (Définition 11) stipule que la transition x cause y si l'occurrence de y dépend de l'occurence de x. Dans le réseau N_1 de la Figure 4.5, les occurrences de la transition t_1 qui approvisionnent la place p_1 en jetons, et celles de la transition t_3 qui remettent à zéro la place p_1 ne sont pas en relation de causalité : elles sont en concurrence.

Nous allons maintenant introduire une transformation structurelle d'un réseau de Petri avec un arc de reset vers un réseau de Petri, qui préserve la bisimulation. Cette transformation donne le réseau de Petri N_2 (Figure 4.5) si N_1 est sauf, et N_3 s'il est k-borné. Par souci de concision, nous allons considérer le cas sauf. Nous allons montrer que toute transformation structurelle effectuée sur N_1 vers un réseau de Petri sans arc de reset, préservant la bisimulation, introduit une causalité entre t_1 et t_3.

Algorithme 5 (Transformation structurelle d'un réseau de Petri avec des arcs de reset vers un réseau de Petri)**.** *Soit N_R un réseau de Petri avec des arcs*



RdP $N_1 : t_1 \wr t_3$ RdP sauf $N_2 : t_1 \prec t_3$ RdP k-borné $N_3 : t_1 \prec t_3$ $n = 1 \ldots k$

FIGURE 4.5 : Causalité entre t_1 et t_3

de reset sauf. La transformation structurelle de N_R vers un réseau de Petri N_{struct} est obtenue de la manière suivante : Pour toute place p et pour toute transition t telle qu'il existe un arc de reset de p vers t, supprimer l'arc de reset et :

1. *créer une place $\overline{p}$ duale de p telle que si p n'est pas marquée, $\overline{p}$ l'est et vice versa ;*
2. *pour toute transition t' telle qu'il existe un arc de t' vers p (respectivement de p vers t'), ajouter un arc de $\overline{p}$ vers t' (respectivement de t' vers $\overline{p}$) ;*
3. *dupliquer la transition t : l'une des copies a un arc qui part de p et l'autre copie a un arc qui part de $\overline{p}$, les deux ont un arc conduisant à $\overline{p}$.*

N_R et N_{struct} sont bisimilaires. Cependant, comme on peut le voir avec les transformations structurelles N_2 et N_3 de la Figure 4.5, la causalité n'est pas préservée. Dans les deux cas, la transformation structurelle rajoute de la causalité entre les transitions t_1 et t_3. Nous allons généraliser maintenant ce résultat.

Considérons le réseau de Petri avec des arcs de reset borné N'_R de la Figure 4.6. N'_R est basé sur le motif N_1 de la Figure 4.5 dans lequel les transitions t_1 et t_3 peuvent être tirées infiniment souvent et ceci dans n'importe quel ordre. N'_R est sauf car t_1 crée un jeton dans la place p_1 qui est immédiatement vidée grâce aux deux arcs de reset. Dans N'_R, t_1 et t_3 sont concurrentes.

Proposition 3. *Tous les réseaux de Petri bisimilaires (même faiblement) au réseau de Petri avec des arcs de reset borné N'_R de la Figure 4.6 créent une causalité entre t_1 et t_3.*

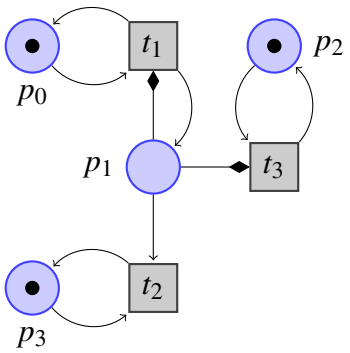


FIGURE 4.6 : RdP avec des arcs de reset borné N'_R

Preuve. Supposons qu'il existe un réseau de Petri N bisimilaire au réseau N'_R sans causalité entre t_1 (ou une copie de t_1) et t_3 (ou une copie de t_3). Pour être complet, autorisons que N contienne des transitions invisibles ou non-observables (Définition 3.2.3). N bisimilaire à N'_R alors le tir de t_3 empêche celui de t_2 dans N : c'est l'hypothèse de départ.

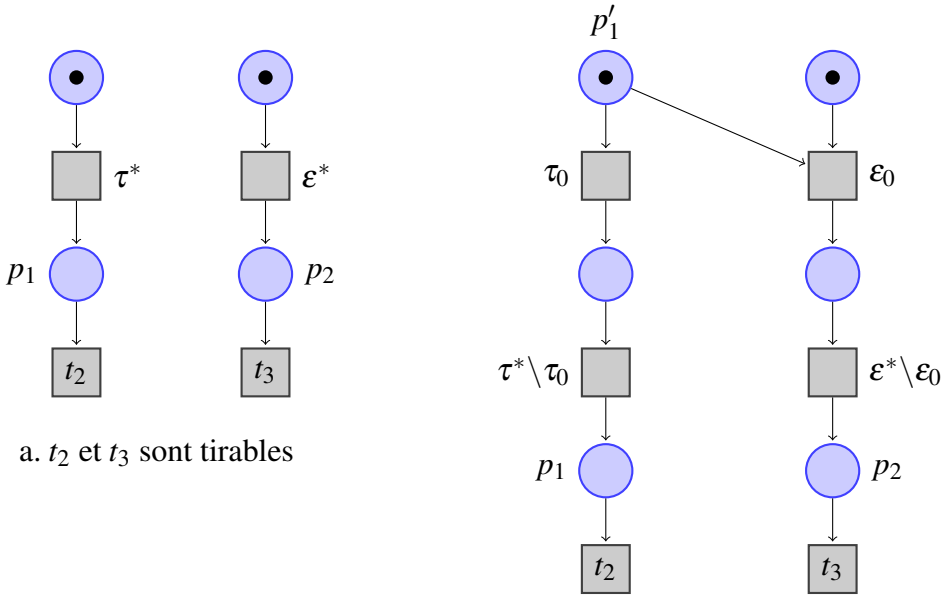
Soient τ^* une séquence de transitions invisibles conduisant à t_2 et ε^* une séquence de transitions invisibles conduisant à t_3 (Figure 4.7.a.). τ^* et ε^* ne sont pas forcément disjoints. Considérons un marquage M_R de N'_R dans lequel les places p_1 et p_2 sont marquées. Depuis ce marquage, t_2 et t_3 sont tirables. N est bisimilaire à N'_R alors il existe un marquage M bisimilaire à M_R dans lequel les deux séquences $\tau^* t_2$ et $\varepsilon^* t_3$ sont tirables (Figure 4.7.a.).

Le tir de t_3 empêche celui de t_2 dans le marquage M_R (t_3 et t_2 sont dans un conflit direct). On peut en conclure que dans le marquage M, les deux séquences $\tau^* t_2$ et $\varepsilon^* t_3$ sont nécessairement dans un conflit direct [3] sinon la séquence $\varepsilon^* t_3 \tau^* t_2$ deviendrait tirable comme le montre la Figure 4.7.a. La transformation de la Figure 4.7.a. n'est pas bisimilaire à N'_R car t_2 reste tirable après le tir de t_3.

Dénommons τ_0 le préfixe de $\tau^* t_2$ et ρ_{t_2} le suffixe de cette séquence ainsi que ε_0 le préfixe de $\varepsilon^* t_3$ et ρ_{t_3} son suffixe. Analysons maintenant les différentes relations entre les transitions des deux séquences $\tau_0 \rho_{t_2}$ et $\varepsilon_0 \rho_{t_3}$:

1. L'occurrence de la transition t_3 dans le marquage M_R du réseau N'_R inhibe celle de la transition t_2. On en déduit donc que dans le marquage M du réseau N, il doit exister une transition ε_0 en conflit direct avec une transition τ_0 de la séquence amenant à t_2. Le franchissement de ε_0 em-

[3] On peut étendre la notion de transitions en conflit direct vue dans la Section 3.2.5.2 à des séquences de transitions. Deux séquences sont en conflit direct s'il existe dans les deux séquences, deux transitions en conflit direct.

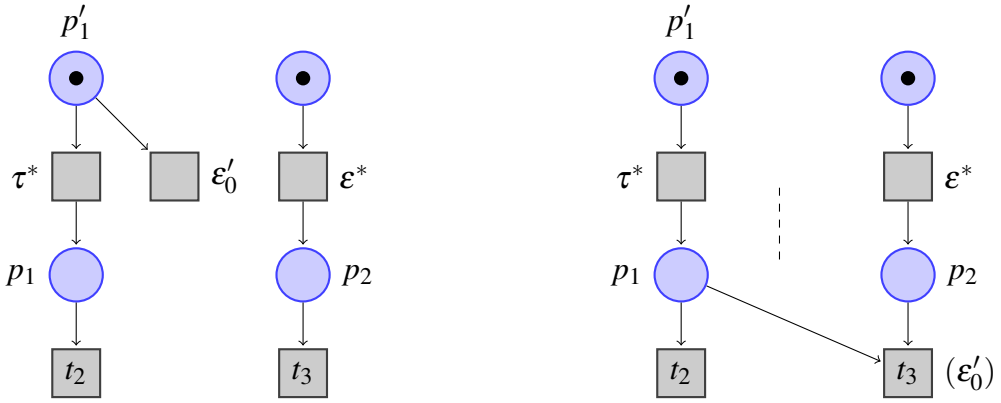


a. t_2 et t_3 sont tirables

b. $\tau_0 \perp \varepsilon_0$

FIGURE 4.7 : Etape 1 de la preuve

pêche le tir de t_2, et permet la séquence ρ_{t_3} (Figure 4.7.b.). Désignons parp'_1 la place qui sensibilise à la fois ε_0 et τ_0. On peut constater que $\varepsilon_0 \preccurlyeq t_3$.



a. $\varepsilon'_0 \perp t_2$, $p'_1 \prec t_2$ et $(\varepsilon'_0 \wr \varepsilon^* t_3)$

b. $\rho_{t_2} = t_2$ et $\varepsilon'_0 = t_3$

FIGURE 4.8 : Etape 2 de la preuve

2. Considérons maintenant une transition en conflit avec ρ_{t_2} qui ne cause pas t_3, que nous renommons ε'_O : $\varepsilon'_0 \not\preccurlyeq t_3$ comme sur la Figure 4.8.a. La séquence ρ_{t_3}, et la transition ε'_0 sont en concurrence, et donc ρ_{t_3} est tirable depuis le marquage M et :

- soit ρ_{t_2} et ρ_{t_3} sont dans un conflit direct et nous nous ramenons au point 1 en utilisant ce conflit pour définir la nouvelle transition ε_0 (ainsi de suite jusqu'à ce que $\rho_{t_2} = t_2$ (Figure 4.8.b.) ou que ρ_{t_2} et ρ_{t_3} ne soient plus dans un conflit direct) ;
- soit ρ_{t_2} et ρ_{t_3} ne sont pas dans un conflit direct et, dans ce cas, la séquence $\tau^* t_2 \varepsilon^* t_3$ est tirable depuis le marquage M, ce qui contredit l'hypothèse de départ.

Ainsi, pour que N soit bisimilaire à N'_R, un cas singulier est $\varepsilon'_0 = t_3$ (Figure 4.8.b) ou bien on peut aussi envisager un conflit indirect entre t_2 et t_3 (Figure 4.9) : après le tir de ε'_0, t_3 est tirable à partir d'une séquence qui n'était pas possible sans le tir de ε'_0. Il existe donc forcement un chemin de ε'_0 vers t_3, ce qui implique que ε'_0 cause t_3 ($\varepsilon'_0 \preccurlyeq t_3$).

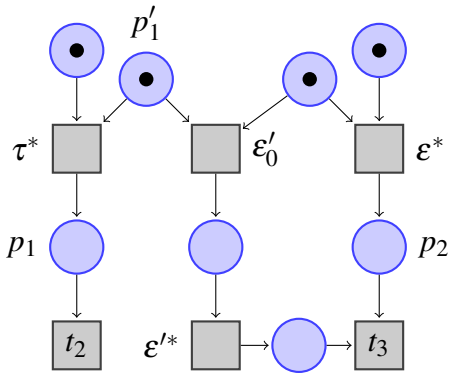


FIGURE 4.9 : Etape 3 de la preuve ($\varepsilon'_0 \preccurlyeq t_3$)

Dans le conflit indirect de la Figure 4.9 nous avons $p'_1 \prec t_2$ et $p'_1 \prec t_3$, ou plutôt il existe un ensemble S_{p_1} de places p'_1 respectant la même règle $p'_1 \prec t_2$ et $p'_1 \prec t_3$. Dans le premier réseau N'_R de la Figure 4.6 qui modélise l'arc de reset, les transitions t_1, t_2 et t_3 peuvent être tirées infiniment souvent ; dans N_{struct} de la Figure 4.9 également. Le nombre de places du réseau N étant fini [4], alors l'ensemble S_{p_1} est fini et il existe au moins une place $p'_1 \in S_{p_1}$ qui n'est pas une *one shot place* [5] (en effet comme les transitions t_1, t_2 et t_3 peuvent être tirées infiniment souvent, je peux vouloir tirer t_2 après une séquence $t_1 t_3$ par exemple). Par conséquent, après le tir de t_1, la place p'_1 doit être à nouveau marquée pour rendre la séquence conduisant à t_2 tirable à nouveau. Par suite, il existe un chemin de t_1 vers p'_1 et par transitivité il

[4] N est fini, découle de la définition d'un réseau de Petri. Le nombre de places d'un réseau de Petri est fini.

[5] c'est-à-dire une place initialement marquée sans transition source : dont le jeton est définitivement perdu après le tir de la transition ε_0

existe un chemin de t_1 vers t_3 qui créé alors une causalité entre t_1 et t_3, d'où la Proposition 3.

□

Illustrons le résultat de la Proposition 3 sur les traductions structurelles intuitives N_2 et N_3 du réseau N_1 que nous avons présentées sur la Figure 4.5. N_2 et N_3 sont bisimilaires à N_1 mais créent une causalité entre t_1 et t_3 qui n'existait pas dans N_1.

De la Proposition 3, on peut déduire les deux corollaires suivants.

Corollaire 3. *Les arcs de reset augmentent strictement l'expressivité des réseaux de Petri du point de vue de la bisimilation et de la causalité pour les réseaux de Petri avec des arcs de reset bornés.*

Corollaire 4. *Pour préserver la causalité, le dépliage ou préfixe fini du dépliage d'un réseau de Petri avec des arcs de reset doit être lui même un réseau de Petri avec des arcs de reset ou un modèle plus expressif.*

4.3.3 Dépliage des réseaux de Petri avec des arcs de reset

Dans cette section, nous proposons un algorithme de construction du dépliage des réseaux de Petri avec des arcs de reset. Le Corollaire 4 démontre de la nécessité d'utiliser les arcs de reset dans le dépliage des réseaux de Petri avec des arcs de reset.

Soient $N_R = \langle P, T, W, R, M_0 \rangle$ un réseau de Petri avec des arcs de reset et $N = \langle P, T, W, M_0 \rangle$ son réseau de Petri sous-jacent. Soient $Unf = \langle O, \lambda \rangle$ le dépliage de N et Unf_R celui de N_R. Notre approche de construction est en deux étapes. La première étape est de construire le dépliage du réseau de Petri sous-jacent. Ce dépliage simule le dépliage du réseau de Petri avec des arcs de reset (Proposition 4). La deuxième étape est de rajouter, de façon adéquate, les arcs de reset. La procédure d'ajout des arcs de reset est donnée par la Définition 58. L'ajout des arcs de reset enlève les comportements représentés dans le dépliage sous-jacent mais qui ne le sont pas dans le réseau de Petri avec des arcs de reset et par conséquent ne doivent pas apparaître dans le dépliage du réseau de Petri avec des arcs de reset. Le résultat obtenu à partir de ces deux étapes est le dépliage du réseau de Petri avec des arcs de reset (Proposition 4).

La construction du dépliage du réseau de Petri avec des arcs de reset est précisée par la Définition 59. Elle nécessite au préalable, de redéfinir la notion de réseau d'occurrence. Nous introduisons ici les R-réseaux d'occurrence (Définition 55) qui étendent les réseaux d'occurrence aux arcs de reset.

Définition 55 (R-réseau d'occurrence)**.** *Un R-réseau d'occurrence est un réseau* $O_R = \langle O, F_R \rangle$ *tel que :*

- $O = \langle B, E, F \rangle$ *est un réseau d'occurrence (Définition 40) ;*
- $F_R : B \times E \to \{0,1\}$ *est l'ensemble des arcs de reset (*$F_R(b,e) = 0$ *s'il y a un arc de reset qui relie b à e, sinon* $F_R(b,e) = 1$*).*

A part les trois relations (causalité, conflit et concurrence) mutuellement exclusives qui existent entre deux évènements d'un réseau d'occurrence, nous introduisons une nouvelle relation entre les évènements d'un R-réseau d'occurrence que nous nommons *relation de reset*. C'est une relation non commutative, qui permet à un évènement e_1 d'empêcher l'occurrence d'un évènement e_2 de O_R à cause d'un arc de reset. Ces deux évènements ne peuvent pas appartenir à une même configuration.

Définition 56 (Relation de reset)**.** *Soient deux évènements* e_1 *et* e_2 *d'un R-réseau d'occurrence.* e_1 *et* e_2 *sont dans une relation de reset (notée* $e_1 \multimap e_2$*) si et seulement si* $\exists\, b$ *et e tel que* $b \in {}^\bullet e \wedge e \preccurlyeq e_2 \wedge F_R(b, e_1) = 0$.

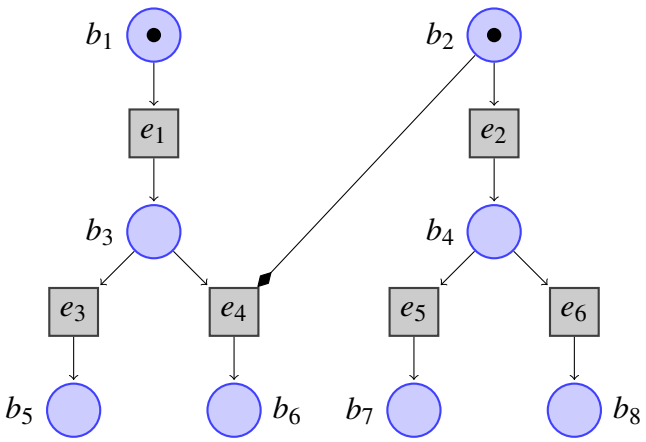


FIGURE 4.10 : R-réseau d'occurrence et configuration

Dans le R-réseau d'occurrence de la Figure 4.10, la séquence $\sigma_1 = e_1 e_2 e_4 e_5$ est tirable alors que la séquence $\sigma_2 = e_1 e_4 e_2 e_5$ n'est pas tirable. La fermeture causale et l'absence de conflit d'une séquence ne suffisent pas pour caractériser l'effectivité de son tir dans le contexte des R-réseaux d'occurrence. Le tir d'une séquence est aussi soumis à l'ordre d'apparition des évènements en relation de reset.

Soit σ une séquence d'évènements d'un R-réseau d'occurrence. Pour $e \in \sigma$, notons $indice(e)$ la position de l'évènement e dans σ. La définition suivante donne trois conditions pour que σ soit une séquence effective pour un R-réseau d'occurrence.

Définition 57. *σ est une séquence effective si elle respecte les trois conditions suivantes :*

- $e \in \sigma \Rightarrow \forall e' \preccurlyeq e,\ e' \in \sigma$ *(σ est causalement fermée) ;*
- $\forall e, e' \in \sigma,\ \neg(e \perp e')$ *(σ est sans conflit) ;*
- $\forall e, e' \in \sigma,$ ***Si*** $(e \multimap e')$ ***alors*** $(indice(e) > indice(e'))$
  σ ne contient pas de relation de reset qui empêcherait le tir d'un évènement d'indice supérieur.

Définition 58 (Procédure d'ajout des arcs de reset)**.** $\forall (b,e) \in B \times E$ *tel que* $R(\lambda(b), \lambda(e)) = 0$ *alors créer un arc de reset* (b,e) *et l'ajouter à* F_R *i.e.* $F_R(b,e) = 0$*, sinon* $F_R(b,e) = 1$.

Définition 59 (Dépliage des réseaux de Petri avec des arcs de reset)**.** *Soit N_R un réseau de Petri avec des arcs de reset. N le réseau de Petri sous-jacent de N_R, et Unf le dépliage de N. Le dépliage $Unf_R = \langle O_R, \lambda \rangle$ de N_R est le dépliage $Unf = \langle O, \lambda \rangle$ de N sur lequel on applique la procédure d'ajout des arcs de reset (Définition 58).*

Proposition 4. *N_R et Unf_R sont bisimilaires.*

Preuve. Dans la Proposition 3.7 de [29], les auteurs prouvent qu'un réseau de Petri N et son dépliage Unf sont bisimilaires. La Proposition 3 montre par ailleurs que N simule N_R, on en déduit alors que Unf simule N_R. Tous les comportements enlevés par les arcs de reset en passant de N à N_R sont aussi sémantiquement enlevés en passant de Unf à Unf_R en appliquant la procédure d'ajout des arcs de reset de la Définition 58. Et donc Unf simule Unf_R.

Notons que s'il existe dans Unf, une condition b telle que la place $\lambda(b)$ est marquée dans N mais ne l'est pas dans N_R, alors b apparaît bien dans Unf_R mais le marquage tel que $\lambda(b)$ est marqué, n'est pas accessible dans Unf_R. N_R et Unf_R sont bisimilaires.

□

4.3.4 Configuration d'un R-réseau d'occurrence

Dans la sémantique des réseaux de Petri avec des arcs de reset, les arcs de reset n'introduisent pas de causalité. La construction du dépliage Unf_R du réseau de Petri avec des arcs de reset N_R à partir de son réseau sous-jacent N n'ajoute pas de la causalité. Par contre la complexité de l'analyse des configurations dans un R-réseau d'occurrence est plus grande que dans un réseau

d'occurrence. Une configuration représente un ensemble d'évènements qui forme une exécution possible du modèle. Dans un réseau d'occurence, pour vérifier si un ensemble E est une configuration, il faut simplement vérifier qu'il est causalement fermé et sans conflit. La décidabilité est linéaire. Par contre pour les R-réseaux d'occurrence, elle est NP-complet [6]. Nous avons la proposition suivante.

Proposition 5. *Le problème de décider si un ensemble d'évènements E d'un R-réseau d'occurrence est une exécution possible (une configuration) est NP-complet.*

Preuve. Afin de vérifier si E est une exécution possible (une configuration), il suffit de trouver une séquence de transitions (de taille $|E|$) contenant tous les évènements de E. Pour le R-réseau d'occurrence de la Figure 4.10, l'ensemble $\{e_5, e_2, e_4, e_1\}$ est une configuration car la séquence $\sigma = e_1e_2e_4e_5$ est tirable.

Pour démontrer qu'un nouveau problème Π est NP-complet, il suffit de trouver un problème Π' connu pour être NP-complet qui se réduit à Π [18] et trouver un algorithme efficace (en temps polynomial) pour le résoudre. Tous les problèmes de la classe NP se ramènent à Π par une réduction polynomiale, cela signifie que le problème est au moins aussi difficile que tous les autres problèmes de la classe NP. Un problème NP-difficile est un problème qui remplit la dernière condition et est dans une classe de problèmes plus larges et donc plus difficiles que la classe NP.

Le problème de k-coloration de graphe avec $k \geq 3$ est connu être NP-difficile [38]. Nous allons montrer que le problème de 3-coloration de graphe se réduit à la décidabilité d'un ensemble d'évènements d'un R-réseau d'occurrence est une configuration. Construisons le graphe 3-colorés correspondant aux événements effectifs du R-réseau d'occurrence de la figure 4.12. La construction de ce graphe se retrouve dans la partie gauche de Figure 4.11. C'est un graphe composé de deux sommets v_1 et v_2 connectés par un arc.

[6] En théorie de la complexité, un problème NP-complet [18] est un problème complet pour la classe NP où NP signifie Non-déterministe Polynomial. Un problème NP-complet vérifie les deux propriétés suivantes : i) il est possible de trouver une solution en temps polynomial (par une machine de Turing non déterministe) et ii) tous les problèmes de la classe NP se ramènent à celui-ci par une réduction polynomiale.

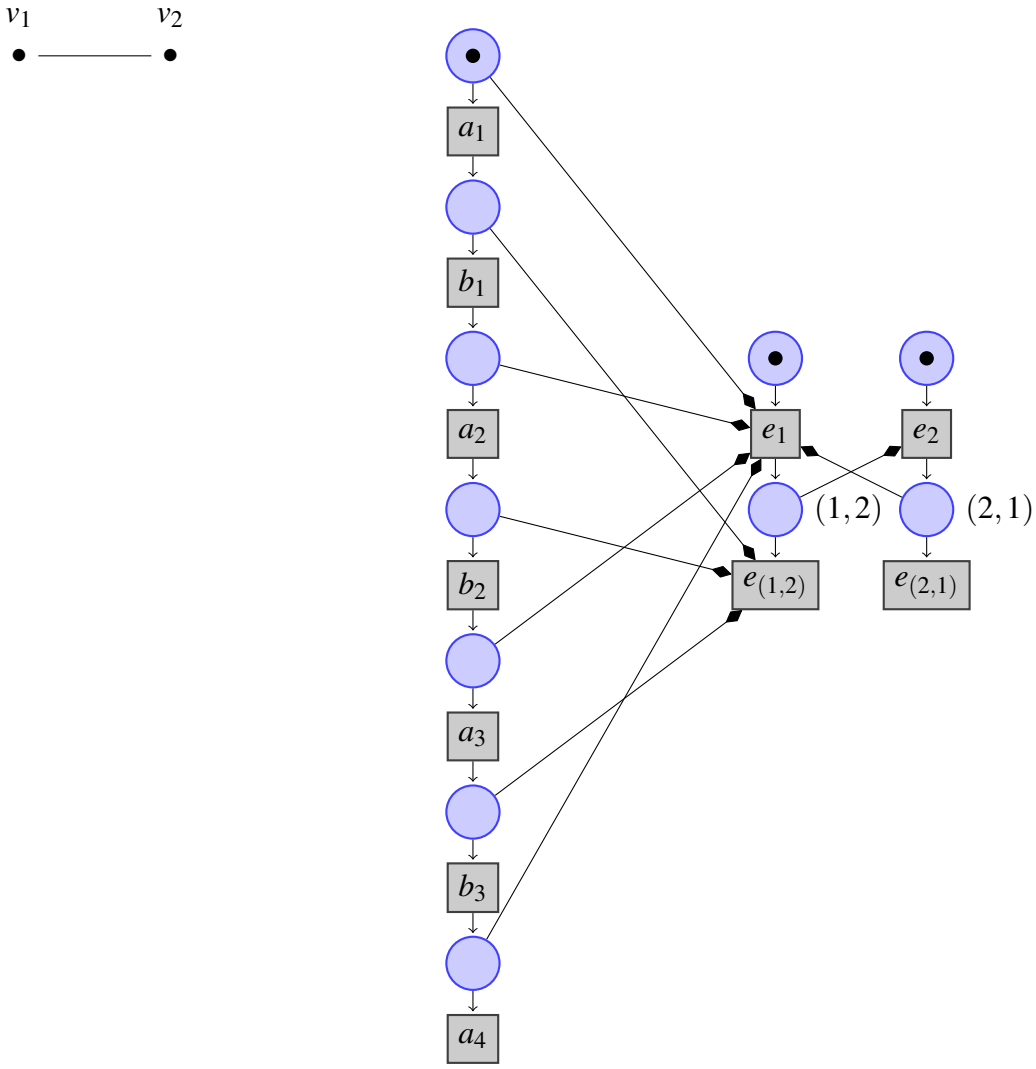


FIGURE 4.11 : Passage de la 3-coloration d'un graphe à l'exécution d'un R-réseau d'occurrence. Les évènements e_1 et $e_{(1,2)}$ vident les places de la séquence de gauche. De même que les évènements e_2 et $e_{(2,1)}$ mais les arcs de reset ne sont pas représentés sur la figure.

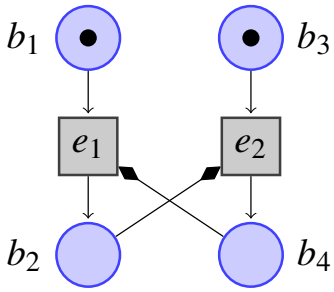


FIGURE 4.12 : Un R-réseau d'occurrence

L'idée de la réduction est de construire un R-réseau d'occurrence où chaque sommet v_i du graphe est représenté par un évènement e_i : tous les évènements e_i sont alors concurrents. Nous utilisons des évènements supplémentaires $a_1 \prec b_1 \prec a_2 \prec b_2 \prec a_3 \prec b_3$ comme des bornes de manière à ce que, pour chaque séquence de tir contenant tous les évènements du R-réseau d'occurrence construit, le tir d'un évènement e_i se fait dans trois *slots* distincts (entre les tirs de a_1 et b_1, celui de a_2 et b_2 ou de a_3 et b_3). En effet, si le tir d'un évènement e_i se produit en dehors de ces *slots*, les arcs de reset de l'évènement vont consommer les jetons de la partie gauche de la Figure 4.11 et empêcher le tirs des autres évènements suivant de la séquence.

Ces trois *slots* représentent les trois couleurs de la coloration. Chaque exécution possible du R-réseau d'occurrence assigne un *slot* à chaque évènement e_i et ce slot représente la couleur du sommet correspondant.

Il ne reste qu'à représenter les arcs du graphe coloré, c'est-à-dire pour chaque arc $\{v_i, v_j\}$, on va forcer les évènements e_i et e_j à être tirés dans des *slots* différents. Cela se fait en utilisant deux conditions (i,j) et (j,i) et deux évènements $e_{(i,j)}$ et $e_{(j,i)}$ tels que $(i,j) \in e_i^\bullet$ et ${}^\bullet e_{(i,j)} = \{(i,j)\}$ (de façon symétrique $(j,i) \in e_j^\bullet$ et ${}^\bullet e_{(j,i)} = \{(j,i)\}$). De plus, $e_{(i,j)}$ et $e_{(j,i)}$ ont des arcs de reset liés aux conditions dans la séquence à gauche, qui forcent leurs occurrences en dehors des trois *slots*. Dans une exécution comprenant le tir de tous les évènements, supposons, sans perte de généralité, que le tir de e_i vienne avant celui de e_j. Alors $e_{(i,j)}$ doit être tiré entre les tirs de e_i et e_j (tant qu'il y a un jeton dans la place (i,j)). Ce qui n'est possible finalement, que si les tirs de e_i et de e_j sont dans deux *slots* différents.

□

On peut remarquer que la notion de *cut* pour un R-réseau d'occurrence n'est pas claire. En effet, l'ensemble des conditions marquées après le tir d'un ensemble d'évènements dépend en général de l'ordre dans lequel les évènements ont été tirés. A titre d'exemple, considérons le R-réseau d'occurrence de la Figure 4.12, en fonction de l'ordre dans lequel e_1 et e_2 sont

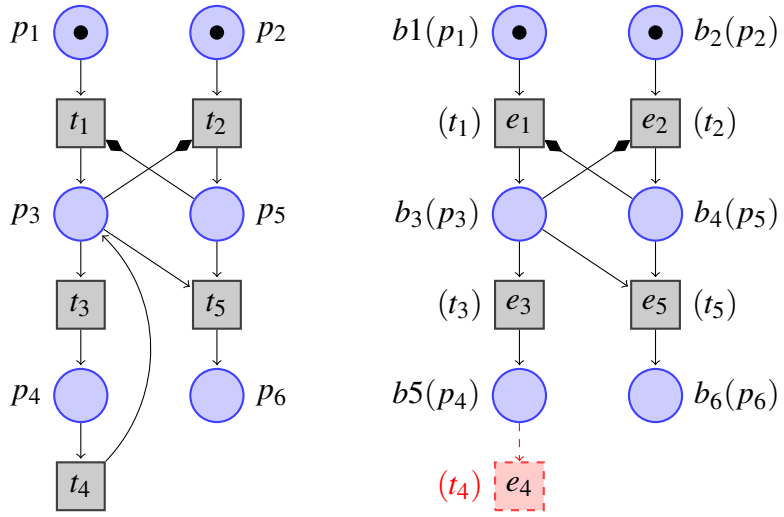
tirés, soit b_2 ou soit b_4 restent marqués à la fin.

4.3.5 Préfixe fini et complet du dépliage dans le cas borné (Fin_R)

Dans cette section, N_R désigne un réseau de Petri avec arcs de reset borné. Nous nous proposons de construire un préfixe fini et complet Fin_R du dépliage de N_R. Fin_R doit préserver les états accessibles de N_R. Commençons par montrer que le préfixe fini et complet Fin du réseau de Petri sous-jacent N ne permet pas d'obtenir Fin_R.

Proposition 6. *Soit Unf le dépliage du réseau de Petri sous-jacent N de N_R et Fin son préfixe fini et complet [59, 31]. Soit Fin' le préfixe obtenu en appliquant à Fin la procédure d'ajout des arcs de reset de la Definition 58. Fin' ne préserve pas les états accessibles de N_R.*

Preuve. Considérons le réseau de Petri avec des arcs de reset N_4 de la Figure 4.13.a. En appliquant les évènements cut-off de [59] ou de [31] à Unf, on obtient le préfixe Fin de la Figure 4.13.b (sans les arcs de reset) parce que l'évènement e_4 est identifié comme évènement cut-off. En utilisant les même cut-off pour Unf_R, Fin'_R ne préserve pas les marquages accessibles. Le marquage $\{p_6\}$ n'est jamais atteint dans Fin' alors qu'il est atteignable en exécutant par exemple la séquence de transitions $\sigma = t_1t_3t_2t_4t_5$.



a. N_4 b. Fin' de N_4

FIGURE 4.13 : Réseau de Petri avec arcs de reset N_4 et le préfixe Fin' de N_4

□

C'est l'identification de e_4 comme évènement cut-off dans le calcul du préfixe fini et complet Fin du réseau de Petri sous-jacent à N_4 qui rend le

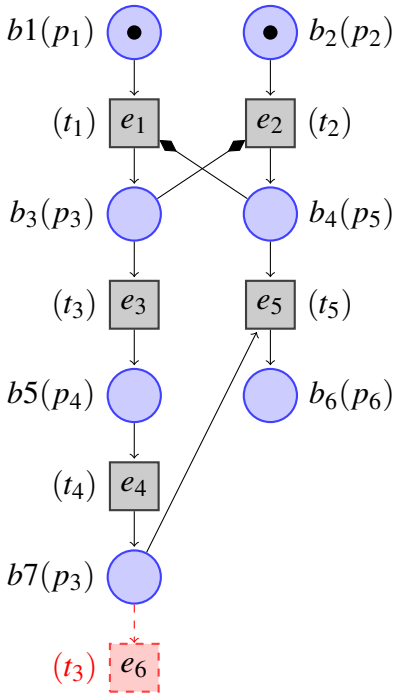


FIGURE 4.14 : Préfixe fini et complet de N_4

préfixe Fin'_R incomplet. Un calcul plus en profondeur de Fin (Figure 4.14) nous aurait permis d'avoir le préfixe fini et complet Fin_R de N_4 en identification plutôt e_6 comme cut-off.

Nous proposons maintenant de calculer les évènements cut-off pour Unf_R en nous basant sur la transformation structurelle de N_R (Algorithme 5). La Figure 4.15 présente la transformation structurelle du réseau N_4 de la Figure 4.13.a. Nous noterons N_{struct} cette transformation et Fin_{struct} (Figure 4.16) le préfixe fini et complet de son dépliage. Mise à part la causalité engendrée, Fin_{struct} et Fin_R sont parfaitement bisimilaires.

La Proposition 2 stipule que N simule N_R, on peut déduire facilement que le préfixe Fin du réseau sous-jacent N simule Fin_R et par conséquent Fin_{struct}. C'est ce qui justifie l'algorithme suivant :

Algorithme 6 (Préfixe fini et complet Fin_R de N_R)**.** *Soient N_R un réseau de Petri avec des arcs de reset, N son réseau de Petri sous-jacent et N_{struct} la transformation structurelle de N_R. Soient Unf le dépliage de N et Fin_{struct} le préfixe fini et complet de N_{struct}. Le préfixe fini et complet Fin_R du dépliage de N_R peut être obtenu comme suit :*

1. *Construire le préfixe fini et complet Fin_{struct} de N_{struct} en utilisant l'algorithme de Esparza-Römer-Vogler [31] ;*
2. *Construire le plus petit préfixe Fin de Unf qui simule Fin_{struct} (cela*

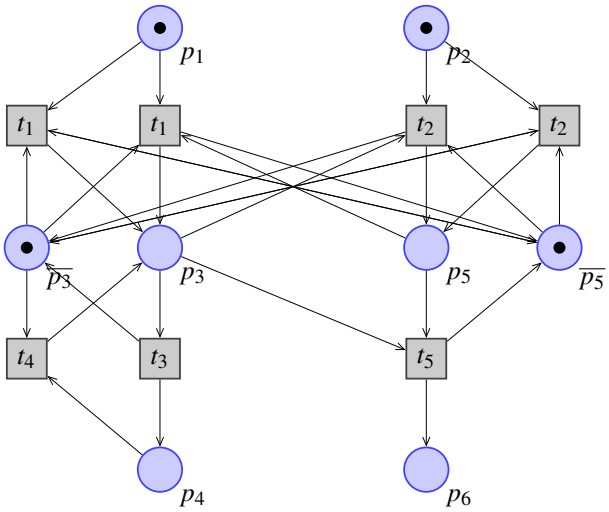


FIGURE 4.15 : Transformation structurelle N_{struct} du réseau N_4 de la Figure 4.13.a.

signifie que la profondeur de chaque branche de Fin est donnée par la profondeur de la branche correspondante dans Fin_{struct}) ;

3. *Construire Fin_R en appliquant à Fin la procédure d'ajout des arcs de reset (Définition 58).*

L'Algorithme 6 utilise Fin_{struct} pour déterminer la profondeur nécessaire au préfixe du dépliage Unf du sous-jacent afin de préserver tous les marquages accessibles. Cet algorithme appliqué au réseau N_4 de la Figure 4.13.a, nous permet d'avoir exactement le même préfixe fini et complet Fin_R que celui de la Figure 4.14. L'inconvénient de l'Algorithme 6 est qu'il calcule deux dépliages. Il calcule d'abord le préfixe fini et complet du dépliage Fin_{struct} de N_{struct} puis le préfixe fini du réseau sous-jacent en regardant Fin_{struct}.

Nous proposons dans l'Algorithme 7 de supprimer les causalités créées dans Fin_{struct} pour avoir directement le préfixe fini et complet Fin_R du réseau de Petri avec des arcs de reset N_R. Plus précisément, il s'agit de supprimer toutes les places $\overline{p}$ créées, de fusionner les transitions dupliquées et de remplacer les arcs qui représentaient les arcs de reset par des arcs de reset.

Algorithme 7 (Construction directe de Fin_R à partir de Fin_{struct})**.** *Soient N_R un réseau de Petri avec des arcs de reset, N_{struct} la transformation structurelle de N_R et Fin_{struct} le préfixe fini et complet de N_{struct}. Le préfixe fini et complet Fin_R du dépliage de N_R est obtenu à partir de Fin_{struct} de la manière suivante :*

1. *Supprimer toutes les places $\overline{p}$ créées par la transformation structurelle (Algorithme5) : $\forall b$ s.t. $\lambda(b) = \overline{p}$:*

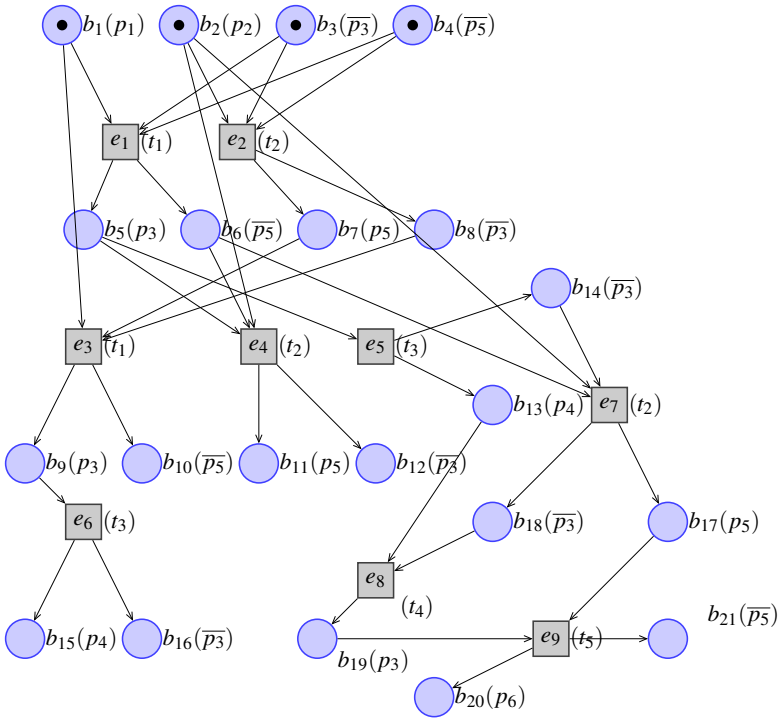


FIGURE 4.16 : Préfixe fini et complet Fin_{struct} du dépliage de N_{struct}

   (a) *supprimer b ;*

   (b) *supprimer tous les arcs arrivant et partant de b.*

2. *Supprimer les causalités : pour tout arc (b,e) tel que $(\lambda(b),\lambda(e))$ est un arc de reset dans N_R alors supprimer l'arc (b,e) dans Fin_{struct}.*

3. *Fusionner les transitions dupliquées : tant qu'il existe un couple d'évènements $(e,e') \in Fin_{struct}$ tel que $(\lambda(e) = t) \wedge (\lambda(e') = t) \wedge ({}^\bullet e = {}^\bullet e')$ faire :*

   (a) *fusionner (e,e') en e_m et $e_m^\bullet = e^\bullet \cup e'^\bullet$*

   (b) *fusionner les postconditions de e_m qui se correspondent : $\forall b \in e_m^\bullet$, if $\exists b' \in e_m^\bullet \backslash \{b\}$ tel que $\lambda(b) = \lambda(b')$ alors $fusionner(b,b')$ en b*

4. *Rajouter les arcs de reset avec la procédure de la Définition 58 .*

L'Algorithme 7 appliqué à Fin_{struct} de la Figure 4.16 , nous donne exactement le même préfixe fini et complet Fin_R de la Figure 4.14.

Nous prouverons à travers les propositions 7 et 8 de la Section 4.3.6 que Fin_R est fini et complet.

4.3.6 Finitude et complétude de Fin_R

Le préfixe Fin_{struct} est fini et complet parce qu'il est calculé par l'algorithme de Esparza-Römer-Vogler [31] sur le réseau de Petri borné N_{struct}.

Proposition 7 (Finitude)**.** *Fin_R est fini.*

Preuve.

- Algorithme 6 : Fin, Fin_{struct} et Fin_R ont la même profondeur alors si Fin_{struct} (ce qui est le cas) est fini alors Fin_R est aussi fini.
- Algorithme 7 : Fin_R est construit à base de Fin_{struct} en fusionnant soit des conditions ou soit des évènements. Aucun évènement ou aucune condition n'est rajouté, Fin_R contient au plus le même nombre d'évènements et de conditions que Fin_{struct}. Fin_{struct} est fini alors Fin_R est fini.

□

Proposition 8 (Complétude)**.** *Fin_R est complet.*

Preuve. Fin simule Fin_{struct} alors toutes les séquences de transitions (modulo λ) de Fin_{struct} appartiennent aussi à Fin. La procédure d'ajout des arcs de reset (Définition 58) enlève sémantiquement de Fin seulement les séquences non exprimées (c'est-à-dire les séquences qui sont possibles dans Fin mais qui ne le sont pas dans Fin_{struct}). Par suite Fin_R et Fin_{struct} ont les même états accessibles et Fin_R est complet.

□

4.3.7 Etude de cas : Distributeur Automatique de Billets (DAB)

Nous proposons en application des réseaux de Petri à arcs de reset, la modélisation d'un Distributeur Automatique de Billets. Le DAB permet aux détenteurs d'une carte de la banque de consulter le solde de leur compte ou de retirer de l'argent. De plus, toutes ces transactions sont sécurisées et nécessitent la saisie d'un code de sécurité. Au démarrage, le client dispose de trois tentatives ($NbTries$) pour saisir son code. A la saisie de trois codes erronés, la carte est avalée par le DAB pour des raisons de sécurité. On ne peut pas consulter son solde, ni retirer de l'argent tant que le code entré n'est pas correct. Ainsi, on ne peut avoir dans un processus, un retrait d'argent ($GetCash$) sans avoir saisi un bon code ($Good$). Nous proposons de construire le dépliage d'un DAB et de vérifier cette propriété. Nous pouvons aussi vérifier

qu'après trois codes erronés, celle ci est avalée (*Swallow*). L'interêt des arcs de reset dans cet exemple, est de permettre de remettre à zéro la place modélisant le nombre de saisie du code secret restant, dès qu'un code correct a été saisie.

Modélisation du DAB

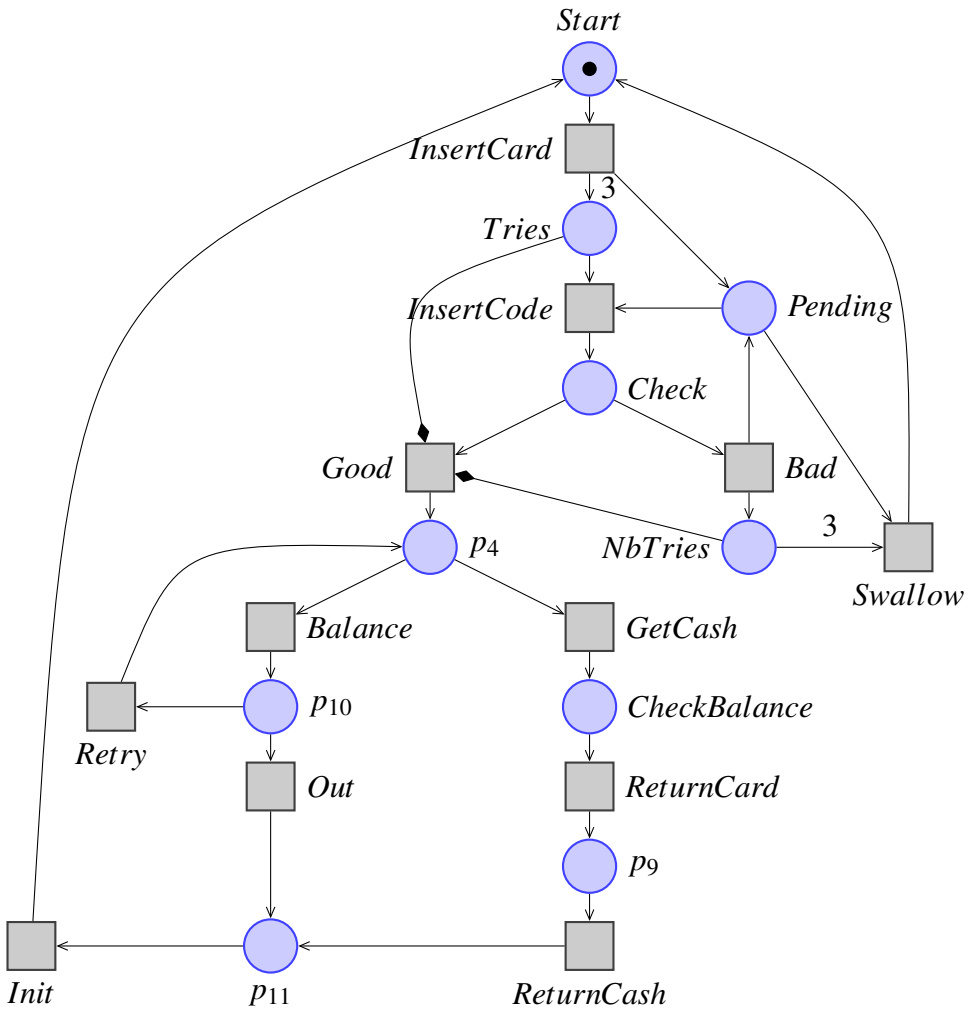


FIGURE 4.17 : Distributeur Automatique de Billets

Nous proposons sur la Figure 4.17 la modélisation d'un distributeur automatique de billets. Sur ce modèle, deux arcs de reset sont utilisés. Les deux arcs de reset sont liés à la transition *Good* (un bon code a été saisi) et partent respectivement des places *Tries* (le nombre d'essais restant) et *NbTries* (le nombre de fois qu'un code erroné a été saisi). Ainsi dès qu'un bon code est saisi, on vide les places *Tries* et *NbTries*.

La construction du dépliage du DAB passe par la construction de sa transformation structurelle.

Transformation structurelle du modèle

La Figure 4.18 donne la transformation structurelle du DAB de la Figure 4.17.

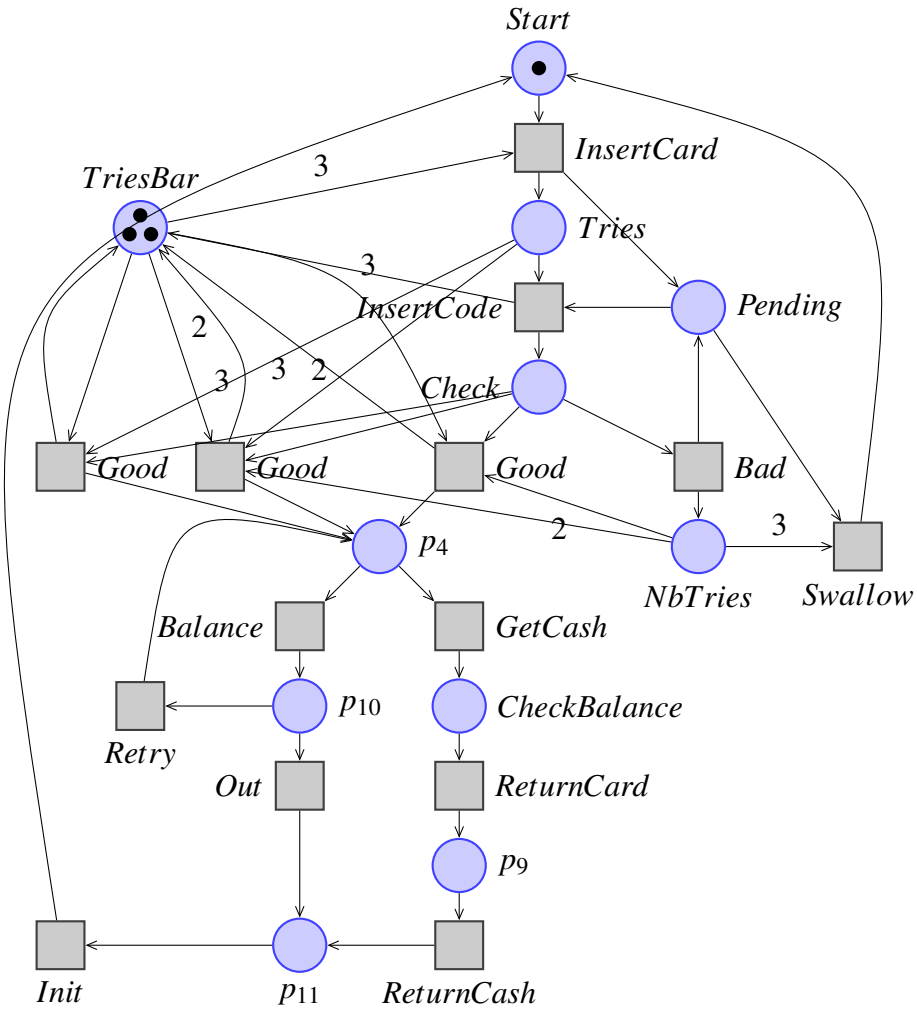


FIGURE 4.18 : Transformation structurelle du DAB de la Figure 4.18

Préfixe fini et complet du dépliage de la transformation structurelle du modèle

La première étape pour avoir le préfixe fini et complet dépliage du DAB est de construire celui de la transformation structurelle.

Le préfixe fini et complet du dépliage de la transformation structurelle se trouve en Annexe A. Le tableau suivant présente les grandes lignes du préfixe.

Résultats	Nombres
Nombre de conditions	92
Nombre d'évènements	73
Nombre d'évènements cut-off	27

TABLE 4.1 : Résultat du dépliage de la Figure 4.18

Préfixe fini et complet du dépliage du DAB

Nous appliquerons l'Algorithme 7 sur le préfixe fini et complet du dépliage de la transformation structurelle. Le résultat est le préfixe fini et complet du dépliage du DAB (Annexe B).

Résultats	Nombres
Nombre de conditions	61
Nombre d'évènements	44
Nombre d'arcs de reset	18

TABLE 4.2 : Résultat du préfixe fini et complet du dépliage du DAB de la Figure 4.17

Le dépliage nous permet de vérifier formellement qu'on ne peut retirer de l'argent sans avoir fait une saisie correcte de son code bancaire. Pour cela, il suffit d'examiner tous les processus de branchements contenant *Getcash* et de vérifier qu'ils ont toujours *Good* dans leur passé causal. Le chapitre suivant propose que la démonstration de ces propriétés se fasse à partir d'une algèbre de processus de branchements.

4.4 Conclusion

Dans ce chapitre, nous avons présenté le dépliage des réseaux de Petri et proposé un algorithme pour la construction du dépliage des réseaux de Petri avec des arcs de reset ainsi que la construction du préfixe fini et complet du dépliage des réseaux de Petri avec des arcs de reset bornés.

Un dépliage est une structure formée par toutes les exécutions possibles du système (processus). Outre le fait qu'il conserve l'ordre partiel entre les évènements, les processus qu'il contient (*processus de branchement*) possèdent des caractéristiques singulières qui les distinguent fortement des processus classiques : *CSP (Communicating Sequential Processes)* de Hoare [41] ainsi que du *CCS (Calculus of Communicating Systems)* de Milner [63]. On va donc s'attacher dans le chapitre suivant de proposer une algèbre adaptée aux processus de branchement issus du dépliage des réseaux de Petri. L'algèbre des processus communicants de Milner sera une bonne base pour cette nouvelle algèbre.

5

Analyse algébrique du dépliage d'un réseau de Petri

Sommaire

5.1 Introduction

Une algèbre de processus permet d'exprimer des comportements asynchrones de systèmes concurrents. Les algèbres de processus présentent généralement les caractéristiques suivantes :

- une expressivité, couplée à une relative simplicité de représentation [37] ;
- la capacité de modéliser divers types d'applications ;
- la possibilité de générer automatiquement des implémentations ;
- une bonne efficacité de vérification logique. Le point fort des algèbres de processus est donc notamment de permettre :
  - la réduction d'expressions algébriques ;
  - d'établir des relations d'équivalence, de bisimulation, ou de congruence ;
  - d'établir des démonstrations de théorèmes.

Par contre, le point faible des approches algébriques est la difficulté de modéliser les aspects quantitatifs de ressource et notamment les notions de consommation ou de production de ressources. Cette "arithmétique" (addition ou soustraction) de ressources, n'est pas au coeur du modèle comme dans les réseaux de Petri et est, par conséquent, un concept coûteux à exprimer dans une approche algébrique.

Un dépliage produit des processus où l'aspect quantitatif de ressources à disparu : un dépliage est un *réseau de Petri sauf*. Seule, la notion de jeton qui passe d'une place à l'autre, subsiste et algébriquement c'est un concept plus aisé à représenter avec la notion de règles de réduction.

L'idée développée dans ce chapitre est donc que le dépliage d'un réseau de Petri permet de lever la notion de ressource quantitative qui est l'un des désavantages majeurs des approches algébriques et qu'il est sans doute plus efficace après un dépliage, de considérer un type d'approche algébrique. Nous utiliserons cette approche algébrique pour calculer les processus de branchements maximaux. De plus, nous allons définir une équivalence qui préserve les conflit.

S'il existe plusieurs algèbres de processus dont CSP [41], CCS [63, 64], LOTOS [42], ... pour des besoins d'implémentation, nous avons opté pour Lisp (Racket étant une dernière évolution) [57, 36] qui est un langage fonctionnel largement utilisé en intelligence artificielle. De plus, racket dispose d'un module Redex qui permet de faire de la réduction d'expressions [33].

5.2 Processus

Pour [5], un processus est un système de transitions déterministe, partiellement ordonné, qui décrit une exécution concurrente. Il correspond à un ensemble de séquences de transitions : un ordre partiel existant entre les évènements.

Définition 60 (Processus)**.** *Un processus est un tuple* $\mathscr{P} = \langle E, \Sigma, Q, Q_0, \prec, \lambda \rangle$ *où :*

- E *est un ensemble fini d'évènements ;*
- Σ *un ensemble fini non vide de labels ;*
- Q *l'ensemble des états du processus*
- $Q_0 \in Q$*, l'état initial du processus*
- $\prec : Q \times E \to Q$ *est la relation d'ordre partiel qui définit les transitions du processus*
- $\lambda : Q \to \Sigma$ *est la fonction de labellisation.*

Remarque 1. *On parle d'ordre partiel car la relation* $\prec$ *n'ordonne pas forcément tous les éléments de* E*.*

5.3 Algèbre de processus de Millner (CCS)

5.3.1 Alphabet

Toutes les algèbres de processus ont la même base fondamentale : il en est de même pour *CCS*. Ils commencent par un ensemble d'actions atomiques à partir de laquelle les processus peuvent être construits. Ces actions forment l'alphabet de l'algèbre que nous désignons par $\mathbb{A}$. Les processus peuvent être composés à partir de cet alphabet à l'aide d'opérateurs pour créer des comportements plus complexes, les opérateurs eux-mêmes obéissent à des lois algébriques.

5.3.2 CCS : algèbre de processus

CCS [63, 64] est une algèbre de processus représentant de façon abstraite un processus. En CCS, un processus possède des "capacités" qui correspondent à des demandes de communication (émission et réception) sur des canaux.

A la différence de la définition d'Arnold précédemment évoquée, les interactions y sont perçues de façon entrelacée, et la notion de parallélisme vrai ou effectif (Section 13) n'est pas dans le modèle. La Figure 5.1 présente sa syntaxe complète :

Capacité	α	:=		a	Demi-action
			|	τ	Action interne
	a	:=		$\bar{x},\bar{y},\bar{z},...$	Emission
			|	$x,y,z,...$	Réception
Processus	p	::=		$\alpha.p$	Préfixe
			|	$p\|\|q$	Composition parallèle
			|	$p+q$	Choix
			|	$D(\tilde{x})$	Définition (récursive)
			|	$p\backslash x$	Restriction
			|	0	Processus terminé

FIGURE 5.1 : Syntaxe des processus CCS [51].

Une capacité est une demi-action ou une action interne (τ). Une demi-action est soit une émission ($\bar{x},\bar{y},\bar{z},...$) ou soit une réception ($x,y,z,...$). 0 est le processus nul ou terminé. Un processus $\alpha.p$ est dit préfixé par une capacité α et de continuation p. La notation préfixée $p := \alpha.q$ correspond à la relation de transition $p \xrightarrow{\alpha} q$. Le processus $p+q$ exprime le choix ou le conflit, tandis que le processus $\alpha_1.p+\alpha_2.q$ peut, soit réaliser la capacité d'action α_1 et passer dans l'état p, soit la capacité d'action α_2 et passer dans l'état q. Le parallélisme est explicitement exprimé dans CCS. $p||q$ exprime la concurrence entre les exécutions de p et q. $D(\tilde{x})$ est une définition récursive où $\tilde{x}$ est un vecteur de capacité d'actions. Le processus $P(x,y) := \alpha_1.\ P(x,y)+\alpha_2.0$ attend un nombre arbitraire de capacité d'actions et peut se terminer à tout moment. Le processus $p\backslash x$ est une restriction, c'est-à-dire, seul le processus p a accès au nom d'action x.

Dans [64] trois sortes d'équivalence ont été introduites : l'équivalence forte ou bisimulation forte, l'équivalence observationnelle ou bisimulation faible et la congruence observationnelle.

Définition 61 (Bisimulation forte, équivalence forte)**.** *Soient P et Q deux expressions CCS. La relation binaire R est une bisimulation forte si et seulement si :*

$$\forall (P,Q) \in R \Leftrightarrow \begin{cases} \forall a \in \mathbb{A}, \forall P' \mid P \xrightarrow{a} P' \Rightarrow \exists Q' \mid Q \xrightarrow{a} Q' \text{ et } (P',Q') \in R \\ \forall a \in \mathbb{A}, \forall Q' \mid Q \xrightarrow{a} Q' \Rightarrow \exists P' \mid P \xrightarrow{a} P' \text{ et } (P',Q') \in R \end{cases} \tag{5.1}$$

La bisimulation faible ou équivalence observationnelle est obtenue en autorisant dans la définition précédente les actions internes ou non-observables τ :

Définition 62 (Bisimulation faible, équivalence observationnelle)**.** *Soient P et Q deux expressions CCS. La relation binaire R est une bisimulation faible si et seulement si :*

$$\forall (P,Q) \in R \Leftrightarrow \begin{cases} \forall a \in \mathbb{A}, \forall P' \mid P \xrightarrow{\tau^* a \tau^*} P' \Rightarrow \exists Q' \mid Q \xrightarrow{\tau^* a \tau^*} Q' \text{ et } (P',Q') \in R \\ \forall a \in \mathbb{A}, \forall Q' \mid Q \xrightarrow{\tau^* a \tau^*} Q' \Rightarrow \exists P' \mid P \xrightarrow{\tau^* a \tau^*} P' \text{ et } (P',Q') \in R \end{cases} \tag{5.2}$$

Le problème de savoir si deux processus sont équivalents observationnellement est un problème d'existence. Il s'agit d'exhiber une bonne instance de R qui respecte les conditions indiquées. L'équivalence observationnelle n'est donc pas une congruence à cause de l'opérateur de choix non déterministe "+" : c'est pourquoi, la congruence observationnelle ou égalité est introduite :

Définition 63 (Congruence observationnelle ou égalité)**.** *Soient P et Q deux expressions CCS. P et Q sont observationnellement congruentes si et seulement si :*

$$\begin{cases} \forall a \in \mathbb{A}, \forall P' \mid P \xrightarrow{a} P' \Rightarrow \exists Q' \mid Q \xrightarrow{a} Q' \text{ et } P' \approx Q' \\ \forall a \in \mathbb{A}, \forall Q' \mid Q \xrightarrow{a} Q' \Rightarrow \exists P' \mid P \xrightarrow{a} P' \text{ et } P' \approx Q' \end{cases} \tag{5.3}$$

5.4 CCS vs Algèbre de processus de branchement

Contrairement aux réseaux de Petri et par conséquent aux processus de branchement, la syntaxe de *CCS* confère une bonne puissance d'expression combinée à une certaine compacité. La contrepartie est que cette syntaxe est moins intuitive que le support graphique associé aux réseaux de Petri.

Une remarque plus significative concerne l'équivalence observationnelle. En *CCS* l'opérateur de séquence n'est pas distributif sur l'opérateur de choix. Ainsi, sur la Figure 5.2 on a deux processus qui ne sont, en effet, pas en bisimulation observationnelle au sens des Définitions 61 et 62. Le concept d'observabilité est lié à la notion de "futur possible" : pour $a.(b+c)$, après le tir de a les futurs possibles sont b ou c. Par contre pour le processus $(a.b+a.c)$ après le tir de a, on n'a plus qu'un seul futur possible.

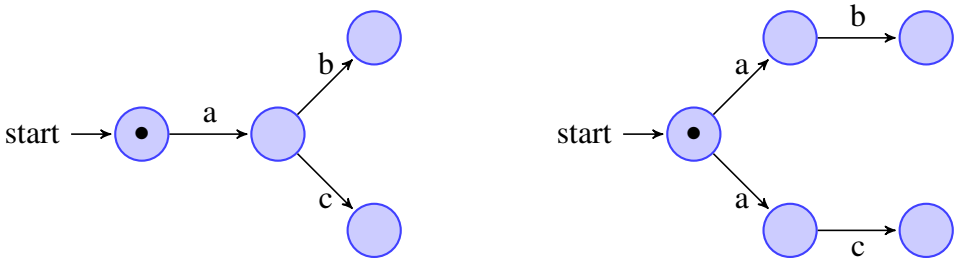
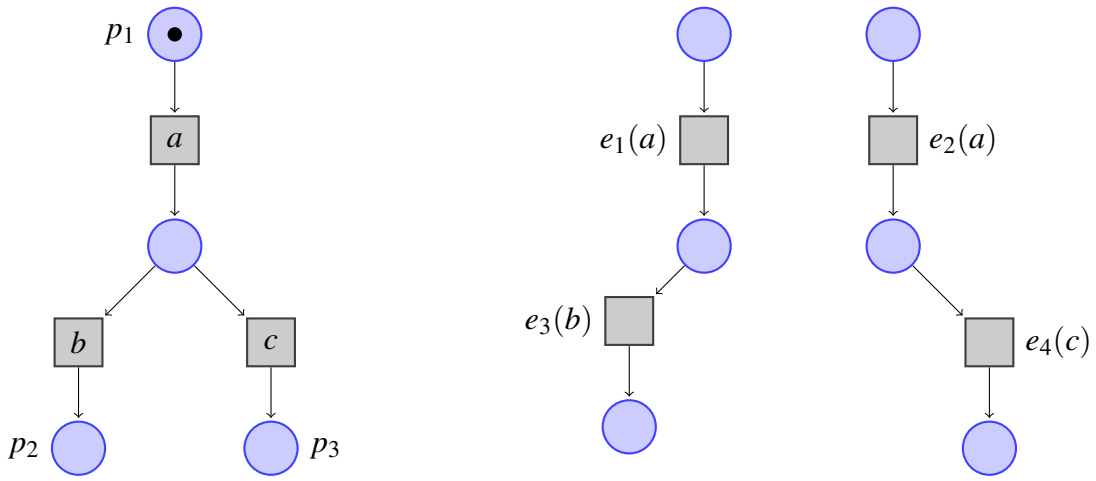


FIGURE 5.2 : CCS : Rejet de la distributivité de la séquence sur le choix.

Un dépliage produit des évènements (et des conditions) liés par des opérateurs semblables à CCS : séquence ($\prec$), choix($\perp$) et concurrence ($\wr$). Les évènements correspondent à des transitions de réseaux de Petri qui ont été franchies. La notion de "transition franchie" fait bien comprendre qu'à la notion de futur possible de *CCS* se substitue la notion de "passé possible". Un dépliage représente plutôt un ensemble arborescent de "passés".

Dans ce contexte, contrairement à *CCS*, la séquence *est* distributive sur le choix. La Figure 5.11.a. fait apparaitre un réseau et les processus branchements issus du dépliage du réseau (Figure 5.11.b). Dans cet ensemble composé de deux processus de branchements, après a, on a soit b, soit c.



a. Réseau de Petri

b. Processus de branchement

FIGURE 5.3 : La causalité est distributive sur le conflit.

5.5 Algèbre de processus de branchement

Dans cette section, nous définissons complètement l'algèbre de processus de branchement ([20] et [21]) .

5.5.1 Définition de l'algèbre

Comme on l'a vu dans le précédent chapitre, un dépliage est un graphe acyclique fini, non nécessairement connexe, dont les sommets sont des conditions et les arcs sont étiquetés par des évènements. Au niveau implémentation, ce graphe peut être stocké sous la forme d'une table donnant toutes les relations binaires entre chaque élément : évènement ou condition.

Ces relations peuvent être partitionnées en trois sous-ensembles disjoints : causalité, concurrence ou conflit. La figure (Figure 5.4) illustre ces trois relations et met en relief la dualité qui peut exister entre les conflits d'une part et d'autre part, l'ensemble constitué des relations de causalité et de concurrence.

Pour illustrer cette dualité, nous introduisons l'opérateur $\oplus$: Si $e_1 \oplus e_2$ alors cela signifie que e_1 et e_2 sont dans le même processus. Cet opérateur de "processus" agrège la causalité et la concurrence.

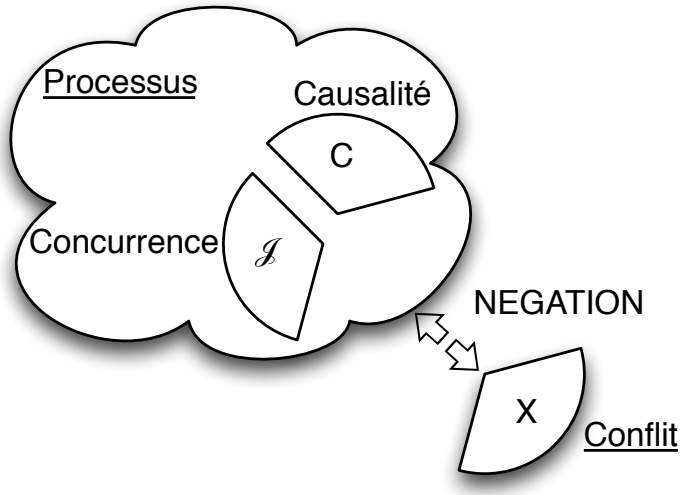


FIGURE 5.4 : Partitionnement des événements d'un dépliage

Au vu des différences exprimées avec les processus de CCS, nous définissons une algèbre dédiée aux processus de branchement *APB* :

$$APB = \{\mathscr{U}, \prec, \wr, \perp, \oplus, \neg\} \tag{5.4}$$

L'algèbre est définie sur l'alphabet $\mathscr{U}$ formé par les événements et les conditions du dépliage $Unf = \langle B, E, F, \lambda \rangle$ d'un réseau de Petri et est munie des opérateurs n-aires $\wr$, $\perp$, $\oplus$, de l'opérateur binaire $\prec$ et de l'opérateur unaire $\neg$. Soient $* \in \{\oplus, \prec, \wr, \perp\}$, $\#t$ le processus interne (non-observable) et $\#f$ le processus irréalisable. La signature formelle de APB est la suivante :

- $\forall n \in \mathscr{U}, n \in APB$
- $\#t \in APB, \#f \in APB$
- $\forall e \in APB, \neg e \in APB$
- $\forall (e_1, e_2) \in APB^2, e_1 * e_2 \in APB$ avec $* \in \{\prec, \perp, \wr, \oplus\}$

5.5.2 Définition des opérateurs

Notons e_i un évènement ou une condition produite par le dépliage. Par la suite, $\equiv$ désigne une congruence. Pour rappel, une congruence est une relation d'équivalence qui préserve les opérations de la structure algébrique du dépliage d'un réseau de Petri.

Opérateur de causalité ($\prec$)

Deux évènements sont en relation de causalité s'il existe un chemin entre les nœuds du dépliage. On peut aussi définir la notion de causalité en utilisant la notion de configuration locale d'un événement définie au chapitre précédent (Définition 49). $e_1 \prec e_2$ si et seulement si e_1 est dans la configuration locale de e_2.

$$e_1 \prec e_2 \Leftrightarrow e_1 \in \lceil e_2 \rceil \tag{5.5}$$

Enonçons les propriétés de l'opérateur de causalité. Les exemples sont donnés sur le dépliage de la Figure 5.5.

- $\prec$ est associative : $e_1 \prec (e_3 \prec e_5) \equiv (e_1 \prec e_3) \prec e_5$;
- $\prec$ est transitive : $(e_1 \prec e_3) \vee (e_3 \prec e_5) \equiv e_1 \prec e_5$;
- $\prec$ n'est pas commutative : $e_1 \prec e_3$ mais $e_3 \not\prec e_1$;
- $\#t$ est un élément neutre pour $\prec$: $\#t \prec e \equiv e$;

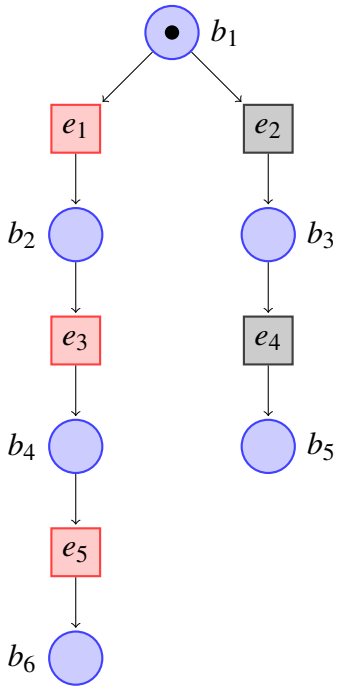


FIGURE 5.5 : Chaîne de causalité

- $\prec$ est idempotent : $e \prec e \equiv e$;
- Chaque événement de E admet un opposé : $\#f \prec e \equiv \neg e$ et $e \prec \neg e \equiv \#f$

Opérateur de conflit ($\perp$)

Deux événements e_1 et e_2 sont en conflit s'il existe une condition b et deux chemins $b\ e_i\ \ldots\ e_1$ et $b\ e_j\ \ldots\ e_2$ partant de la même condition et telle que $e_1 \neq e_2$. On peut formaliser le conflit de la sorte :

$$e_1 \perp e_2 \Leftrightarrow ({}^\bullet e_1 \cap {}^\bullet e_2 \neq \emptyset) \vee (\exists e_i,\ e_i \prec e_2 \ et\ e_1 \perp e_i) \tag{5.6}$$

Les propriétés suivantes sont établies pour l'opérateur de conflit. Les exemples sont donnés pour le dépliage de la Figure 5.6.

- $\perp$ est commutatif : $e_1 \perp e_2 \equiv e_2 \perp e_1$;
- $\perp$ est associatif : $e_1 \perp (e_2 \perp e_3) \equiv (e_1 \perp e_2) \perp e_3$;
- $\perp$ n'est pas transitif : Si $(e_1 \perp e_2)$ et $(e_2 \perp e_3)$, on n'a pas forcement $(e_1 \perp e_3)$ (Figure 5.7) ;
- $\#f$ est un élément neutre pour $\perp$: $e \perp \#f \equiv e$;

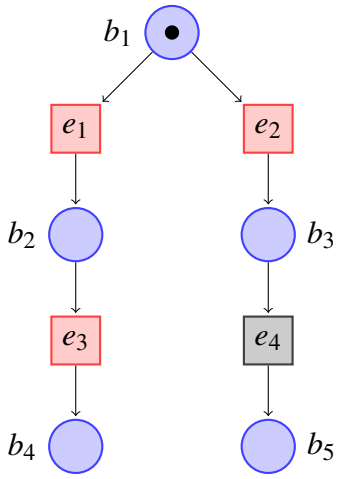


FIGURE 5.6 : Conflit : $(b_4 \perp e_2) \wedge (e_1 \perp e_2)$

- $\#t$ est un élément absorbant de $\perp$: $e \perp \#t \equiv \#t$.

La Figure 5.7 montre la non-transitivité de l'opérateur $\perp$

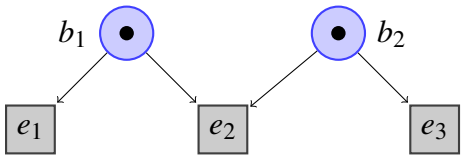


FIGURE 5.7 : Non transitivité de $\perp$: $(e_1 \perp e_2) \wedge (e_2 \perp e_3)$ pourtant $e_1 \wr e_3$

Opérateur de concurrence ($\wr$)

Deux événements sont en concurrence s'ils ne sont ni en causalité, ni en conflit. Ils sont vus comme deux événements indépendants.

$$e_1 \wr e_2 \Leftrightarrow \neg((e_1 \prec e_2) \vee (e_2 \prec e_1) \vee (e_1 \perp e_2)) \tag{5.7}$$

Nous formulons les propriétés suivantes pour l'opérateur de concurrence. Les exemples sont donnés pour le dépliage de la Figure 5.8.

- $\wr$ est commutative : $e_1 \wr e_3 \equiv e_3 \wr e_1$;
- $\wr$ est associative : $e_1 \wr (e_3 \wr e_7) \equiv (e_1 \wr e_3) \wr e_7$;
- $\wr$ n'est pas transitive : $(e_1 \wr e_2) \wedge (e_2 \wr e_3)$ mais on n' a pas forcement $e_1 \wr e_3$;

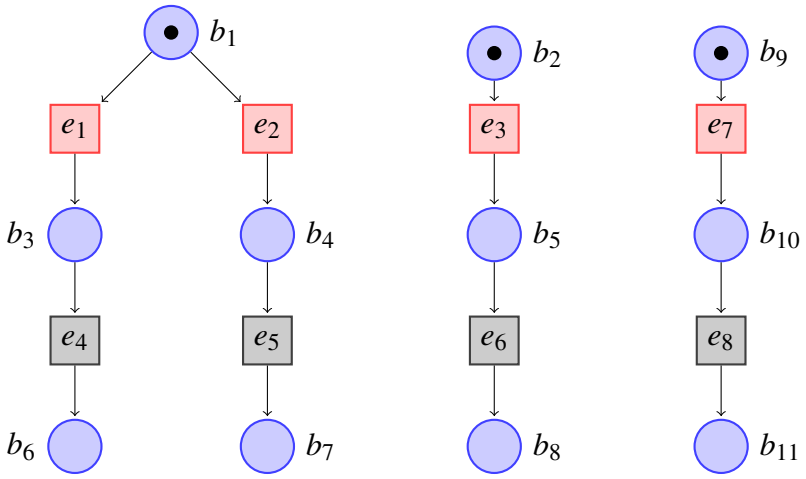


FIGURE 5.8 : Chaîne de concurrence

- $\#t$ est un élément neutre pour $\wr$: $e \wr \#t \equiv e$;
- $\#f$ est un élément absorbant pour $\wr$: $e \wr \#f \equiv \#f$.

La Figure suivante (5.9) est un exemple montrant que $\wr$ n'est pas transitif

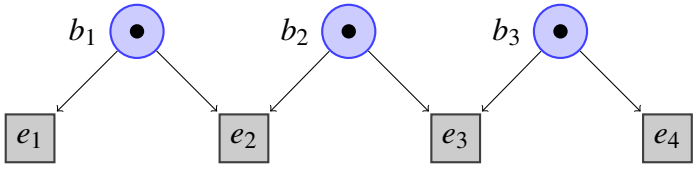


FIGURE 5.9 : Conflit : $(e_1 \wr e_4) \wedge (e_4 \wr e_2)$ pourtant $e_1 \perp e_2$

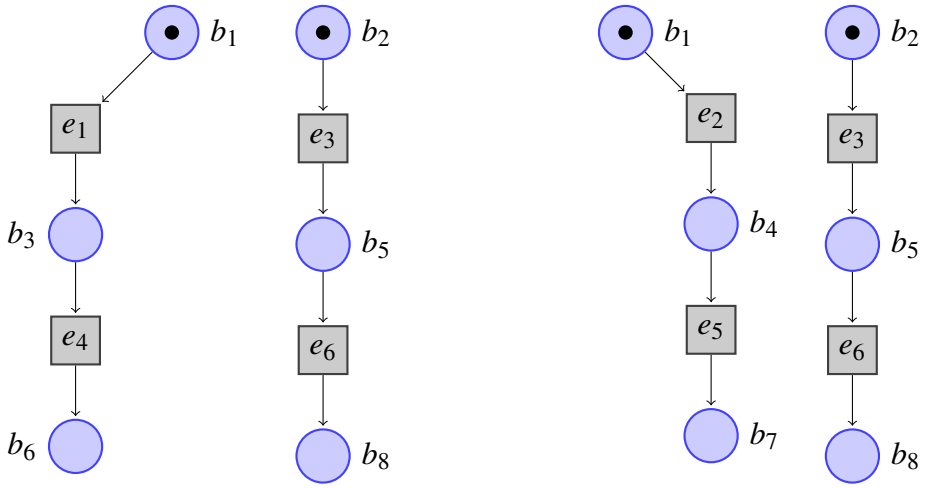
Opérateur de processus ($\oplus$)

Deux événements sont dans un même processus s'ils sont en causalité ou en concurrence.

$$e_1 \oplus e_2 \Leftrightarrow (e_1 \prec e_2) \vee (e_2 \prec e_1) \vee (e_1 \wr e_2) \tag{5.8}$$

Les propriétés suivantes sont établies pour l'opérateur de processus. Les exemples sont donnés pour la Figure 5.10.

- $\oplus$ est commutatif, associatif, et transitif ;
- $\oplus$ est idempotent : $e \oplus e \equiv e$
- $\#t$ est un élément neutre pour $\oplus$: $e \oplus \#t \equiv e$



a) Processus 1 b) Processus 2

FIGURE 5.10 : Processus de branchement

- $\#f$ est un élément absorbant pour $\oplus$: $e \oplus \#f \equiv \#f$

Opérateur de négation ($\neg$)

L'opérateur $\neg$ est un opérateur unaire qui permet d'exprimer l'événement contraire d'un événement. Si un événement e appartient à un processus, son contraire ou son opposé ne l'est pas : dans un processus, on ne peut y avoir un événement et son contraire.

- $e \oplus (\neg e) \equiv \#f$
- $\neg(\neg e) \equiv e$

5.5.3 Axiomes

Les axiomes qui vont suivre découlent soit de la définition des opérateurs, soit de l'hypothèse de base de l'algèbre. Nous nous baserons sur ces axiomes pour faire par la suite des preuves.

Axiome 1 (Distributivité de $\prec$)**.**

$$e \prec (e_1 \perp e_2) \equiv (e \prec e_1) \perp (e \prec e_2)$$

C'est le point de départ de l'algèbre évoquée au début du chapitre : la distributivité de la causalité sur le conflit. Si un événement e cause deux événements e_1 et e_2 en conflit alors les processus $e \prec e_1$ et $e \prec e_2$ sont en conflits. En terme de processus, e *et* e_1 sont dans un même processus en conflit avec le processus formé par e *et* e_2. D'où le corollaire suivant :

Corollaire 5.

$$e \prec (e_1 \perp e_2) \equiv (e \oplus e_1) \perp (e \oplus e_2)$$

Axiome 2 (Définition de $\oplus$)**.**

$$e_1 \oplus e_2 \equiv (e_1 \prec e_2) \perp (e_2 \prec e_1) \perp (e_1 \wr e_2)$$

Axiome 3 (Causalité)**.**

$$e_1 \prec e_2 \equiv \neg e_1 \perp (e_1 \oplus e_2)$$

La causalité peut être vue comme deux processus en conflit. Si $e_1 \prec e_2$ alors l'occurrence de e_1 permet celle de e_2, ainsi $e_1 \oplus e_2$. Mais si e_1 ne se produit pas, e_2 ne peut non plus se produire.

Axiome 4 (Dualité entre $\oplus$ et $\perp$)**.**

$$e_1 \oplus e_2 \equiv e_1 \neg\perp e_2 \qquad e_1 \neg\oplus e_2 \equiv e_1 \perp e_2$$

Cet axiome découle aussi de la définition de l'opérateur $\oplus$. e_1 et e_2 sont dans un même processus s'il ne sont pas en conflit ou *vice versa*.

Axiome 5 (Conflit)**.**

$$e_1 \perp e_2 \equiv (\neg e_1 \oplus e_2) \perp (e_1 \oplus \neg e_2)$$

Le conflit peut être aussi considéré comme deux processus en conflit. $e_1 \perp e_2$, l'occurrence de e_1 inhibe celle de e_2 ou le contraire.

Axiome 6 (Les "non-événements")**.**

$$e_1 \oplus \neg e_2 \equiv e_1$$

On peut supprimer ou ajouter une "non occurrence" d'évènement à un processus.

5.5.4 Règles de dérivation

Dans cette section, nous donnons un ensemble de règles qui seront utilisées dans la Section 5.6 pour faire des transformations des dépliages pour avoir leurs formes canoniques.

Pour la suite, b désigne une condition, e_i un événement et E une expression bien formulée sur l'algèbre de processus de branchement (Section 5.5.1).

Réduction de $\prec$ et de $\wr$

$$\frac{\vdash e_1 \prec e_2}{\vdash e_1 \oplus e_2}\, P_1 \qquad \frac{\vdash e_1 \wr e_2}{\vdash e_1 \oplus e_2}\, P_2$$

Cette règle de réduction découle de la définition de l'opérateur $\oplus$.

Modus Ponens (suppression des b_i)

$$\frac{\vdash b \quad \vdash b \prec e}{\vdash e}\, MP_1 \qquad \frac{\vdash e \quad \vdash e \prec b}{\vdash e \oplus b}\, MP_2$$

Forme duale

$$\frac{\vdash \neg e_1 \quad \vdash e_1 \prec e_2}{\vdash \neg e_1 \oplus \neg e_2}\, FD$$

Simplifications

$$\frac{\vdash \neg e_1 \oplus E}{\vdash E}\, S_1 \qquad \frac{\vdash b \oplus E}{\vdash E}\, S_2$$

5.5.5 Distributivité

Nous établissons dans cette section, les distributivités des opérateurs à partir des axiomes. Nous utiliserons la distributivité sur le conflit ($\perp$) pour pouvoir calculer la forme canonique réduite d'un dépliage dans la Section 5.6. Les autres distributivités seront utilisées pour faire de la réduction de processus.

1. $\prec$ est distributive sur $\wr$:

$$e \prec (e_1 \wr e_2) \equiv (e \prec e_1) \wr (e \prec e_2)$$

Preuve.
Si e cause le processus $(e_1 \wr e_2)$ alors on peut déduire trois énoncés : e cause e_1,e cause e_2 et e_1 est concurrent à e_2. Formellement la preuve s'écrit sous la forme suivante :

$$e \prec (e_1 \wr e_2) \equiv \begin{cases} (e \prec e_1) \\ (e \prec e_2) \\ (e_1 \wr e_2) \end{cases} \equiv \begin{cases} (e \prec e_1) \\ (e \prec e_2) \\ ((e_1 \prec e_1) \wr (e_2 \prec e_2)) \end{cases} \begin{matrix} \\ \\ Indempotence \\ de \prec \end{matrix}$$

$$e \prec (e_1 \wr e_2) \equiv ((e \prec e_1) \wr (e \prec e_2)) \quad Transitivité\ de\ \prec$$

Notons b_1 et b_2 les ressources produites par e sensibilisant les tirs concurrents de e_1 et de e_2. On a donc $b_1 \prec e_1$ et $b_2 \prec e_2$. Cette figure illustre

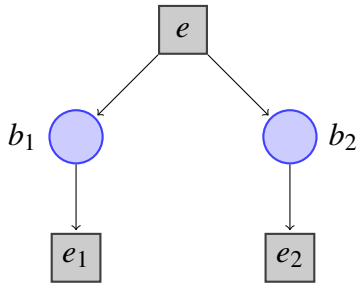


FIGURE 5.11 : Distributivité de $\prec$ sur $\wr$

la preuve, on y retrouve les deux processus concurrents : $(e \prec b_1 \prec e_1)$ et $(e \prec b_1 \prec e_1)$ qui sont réduits par la transitivité de $\prec$.

□

2. $\oplus$ est distributif sur $\wr$:

$$e \oplus (e_1 \wr e_2) \equiv (e \oplus e_1) \wr (e \oplus e_2)$$

Preuve.
Si e est dans le même processus que $(e_1 \wr e_2)$ alors on peut déduire trois énoncés : e est dans le même processus que e_1 et que e est dans le même processus que e_2, ainsi que e_1 est concurrent à e_2. Formellement la preuve s'écrit sous la forme suivante :

$$e \oplus (e_1 \wr e_2) \equiv \begin{cases} (e \oplus e_1) \\ (e \oplus e_2) \\ (e_1 \wr e_2) \end{cases} \equiv \begin{cases} (e \oplus e_1) \\ (e \oplus e_2) \\ ((e_1 \oplus e_1) \wr (e_2 \oplus e_2)) \end{cases} \quad \begin{matrix} \\ \\ \textit{Indempotence} \\ \textit{de } \oplus \end{matrix}$$

$$e \oplus (e_1 \wr e_2) \equiv ((e \oplus e_1) \wr (e \oplus e_2)) \quad \textit{Transitivité de } \oplus$$

□

3. $\prec$ est distributif sur $\perp$ (Axiome 1) :

$$e \prec (e_1 \perp e_2) \equiv (e \prec e_1) \perp (e \prec e_2)$$

4. $\oplus$ est distributif sur $\perp$:

$$e \oplus (e_1 \perp e_2) \equiv (e \oplus e_1) \perp (e \oplus e_2)$$

Preuve.
Avec les items précédents, nous avons montré la distributivité de la causalité sur le conflit : $e \prec (e_1 \perp e_2) \equiv (e \prec e_1) \perp (e \prec e_2)$ et la distributivité de la concurrence sur le conflit : $e \wr (e_1 \perp e_2) \equiv (e \wr e_1) \perp (e \wr e_2)$. Par

définition de l'opérateur de processus $\oplus$, nous pouvons remplacer soit l'opérateur $\prec$ par $\oplus$ dans la première équivalence ou soit l'opérateur $\wr$ par $\oplus$ dans la seconde équivalence. Nous obtenons bien $e \oplus (e_1 \perp e_2) \equiv (e \oplus e_1) \perp (e \oplus e_2)$.

□

5. $\wr$ est distributive sur $\oplus$:

$$e \wr (e_1 \oplus e_2) \equiv (e \oplus e_1) \wr (e \oplus e_2)$$

Preuve.
Il n'y a aucun conflit dans le dépliage $e \wr (e_1 \oplus e_2)$. Les évènements e et e_1 sont dans un même processus indépendamment de e et e_2 : $e \wr (e_1 \oplus e_2) \equiv (e \oplus e_1) \wr (e \oplus e_2)$.

□

5.5.6 Théorèmes

Théorème 1. $e_1 \prec (e_2 \perp e_3) \equiv (e_1 \prec (e_2 \oplus \neg e_3)) \perp (e_1 \prec (\neg e_2 \oplus e_3))$

Preuve.

$$\begin{array}{rcl} e_1 \prec (e_2 \perp e_3) & \equiv & e_1 \prec ((e_2 \oplus \neg e_3) \perp (\neg e_2 \oplus e_3)) \\ & \equiv & (e_1 \prec (e_2 \oplus \neg e_3)) \perp (e_1 \prec (\neg e_2 \oplus e_3)) \end{array}$$

□

Théorème 2. *Soient E et F deux processus : $E \perp (E \oplus F) \equiv E \oplus F$*

Preuve.

$$\begin{array}{rcl} E \perp (E \oplus F) & \equiv & (E \oplus \#t) \perp (E \oplus F) \\ & \equiv & E \oplus (\#t \perp F) \\ & \equiv & E \oplus F \end{array}$$

□

5.6 Forme canonique d'un dépliage et équivalences de conflits

5.6.1 Définitions

La forme canonique d'un dépliage est une relation permettant de représenter algébriquement un dépliage de façon unique. Cette representation est basée sur les éléments de E ainsi que les opérateurs $\oplus$ et $\perp$.

Théorème 3 (Forme canonique)**.** *Soit Unf le dépliage d'un réseau de Petri. Il existe une forme réduite et unique de Unf sous la forme suivante :*

$$Unf \;=\; (\perp\; P_1\; P_2\; ...\; P_n) \; avec \; P_i = (\oplus\; e_{i_1}\; e_{i_2} ... \; e_{i_n}) \tag{5.9}$$

Démonstration. L'unicité est basée sur le fait que les symboles (événements e_i) sont ordonnés de façon alphanumérique. Comme on l'a évoqué dans la Section 5.5.4, des règles de réduction permettent de transformer chaque relation de causalité et de concurrence en $\oplus$. De plus, $\oplus$ étant $\perp$ (Section 5.5.5) mutuellement distributives, on pourra toujours remonter l'opérateur de conflit au plus haut niveau. □

Cette forme exhibe tous les processus de branchement maximaux du dépliage. Chaque dépliage peut être exprimé sous sa forme canonique. Nous pouvons alors comparer des dépliages en nous basant sur leurs formes canoniques. Cette comparaison ne conservera que les conflits. Nous définissons l'équivalence de conflits dans la Définition 64.

Définition 64 (Equivalence de conflits)**.** *Soient Unf_1 et Unf_2 deux dépliages de deux réseaux de Petri N_1 et N_2 respectifs. N_1 est équivalent à N_2 en terme de conflit (noté $N_1 \approx_{conf} N_2$) si et seulement si $\lambda(Unf_1) = \lambda(Unf_2)$.*

Pour avoir la forme canonique du dépliage, la relation $\oplus$ agrège les relations $\prec$ et $\wr$ qui sont alors perdues. L'équivalence de conflit est moins forte que l'équivalence de trace mais est similaire à la *pomset* équivalence [78].

5.6.2 Exemples

Exemple 1

Soient les réseaux de Petri N_1 et N_2 de la Figure 5.12. Désignons par Unf_1 et Unf_2 respectivement le dépliage de N_1 et de N_2. Exprimons la forme canonique de Unf_1 et de Unf_2.

$$\begin{aligned} Unf_1 &= (b_1 \prec e_1 \prec b_3) \wr (b_2 \prec e_2 \prec b_4) \\ &= (b_1 \oplus e_1 \oplus b_3) \oplus (b_2 \oplus e_2 \oplus b_4) \\ &= (e_1 \oplus e_2) \\ &= (\oplus\; e_1\; e_2) \end{aligned}$$

$$\lambda(Unf_1) = (\oplus\; t_1\; t_2)$$

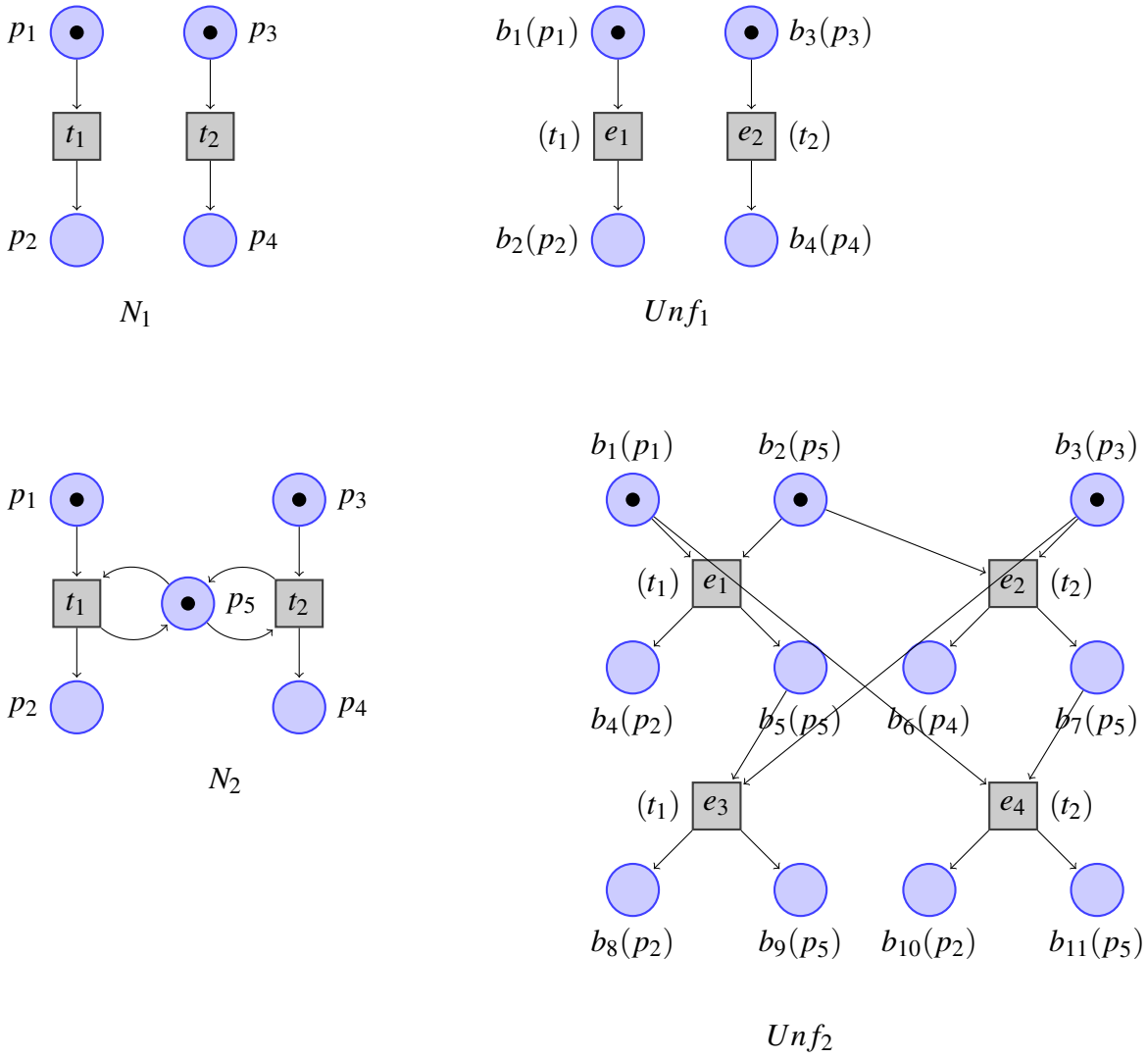


FIGURE 5.12 : Exemple 1

$$
\begin{aligned}
Unf_2 &= \left\{ \begin{array}{l} (e_1 \perp e_2) \\ (e_1 \prec e_3) \\ (e_2 \prec e_4) \end{array} \right. \\
&= (e_1 \prec e_3) \perp (e_2 \prec e_4) \\
&= (\perp\ (\oplus\, e_1\ e_3)\ (\oplus\, e_2\ e_4))
\end{aligned}
$$

$$\lambda(Unf_2) = (\perp\ (\oplus\, t_1\ t_2)\ (\oplus\, t_1\ t_2)) = (\oplus\, t_1\ t_2)$$

$\lambda(Unf_1) = \lambda(Unf_2)$ alors les deux réseaux de Petri N_1 et N_2 sont équivalents en terme de conflit : $N_1 \approx_{conf} N_2$.

Exemple 2

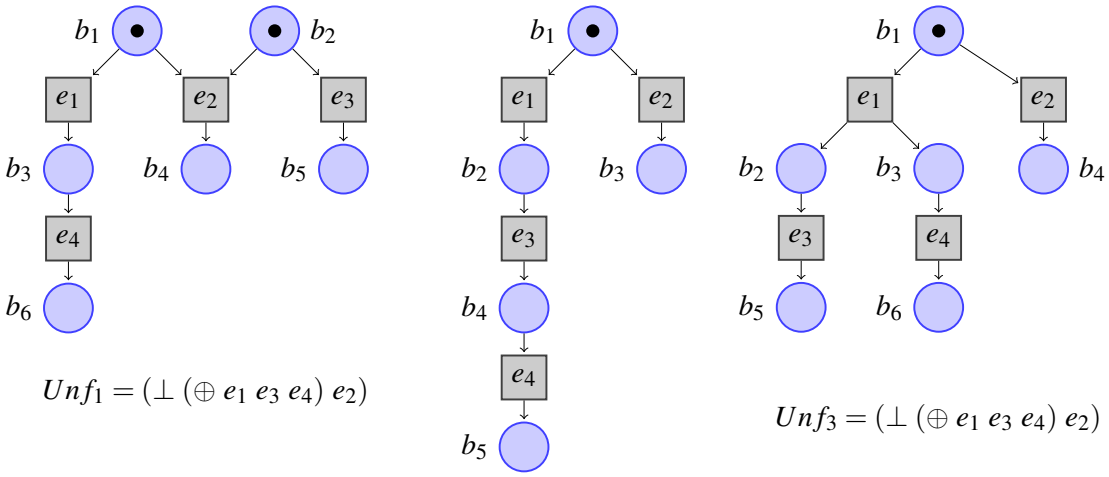


FIGURE 5.13 : Exemple 2

$$\lambda(Unf_1) = \lambda(Unf_2) = \lambda(Unf_3) \Longleftrightarrow N_1 \approx_{conf} N_2 \approx_{conf} N_3$$

5.6.3 Chaîne de conflits

Une chaîne de conflits est la forme généralisée du problème des philosophes. Dans le problème de philosophes on a $e_n \equiv e_1$ (figure 5.14). Et cette section présente un algorithme qui calcule tous les processus de branchement d'un dépliage composé d'une chaîne de conflit.

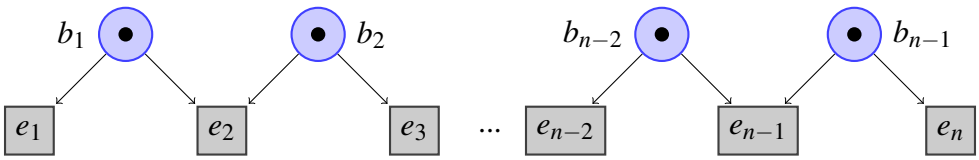


FIGURE 5.14 : Chaîne de conflit

La représentation algébrique d'une chaîne de conflit est :

$$Unf = ((\oplus\ b_1\ b_2\ ...\ b_{n-1})\ (b_2 \prec (e_1 \perp e_2))\ (b_1 \prec (e_2 \perp e_3))\ ...\ (b_{n-1} \prec (e_{n-1} \perp e_n)))$$

Après réduction on obtiendra :

$$Unf = (e_1 \perp e_2 \perp ... \perp e_n)$$

Forme canonique d'une chaîne de conflits

Proposition 9. *Soit $l = (e_1, e_2, \ldots e_n)$ une chaîne de conflit. Pour tout événement e_i, l'événement immédiatement suivant (respectivement immédiatement précédent) qui se trouve dans le même processus de branchement que e_i est soit e_{i+2} ou soit e_{i+3} (respectivement e_{i-2} ou soit e_{i-3})*

Preuve.

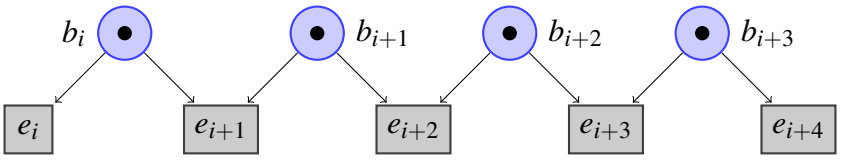


FIGURE 5.15 : Preuve

Soit le dépliage représenté sur la Figure 5.15. $(e_i \perp e_{i+1})$ alors e_i et e_{i+1} ne sont pas dans le même processus. L'occurrence de e_i laisse possible celle de tous les événements à partir de e_{i+2}. $(e_{i+2} \perp e_{i+3})$ alors un seul des événements appartient au même processus. On en déduit que soit $(e_i \oplus e_{i+2})$ ou soit $(e_i \oplus e_{i+3})$. Au delà de e_{i+3}, l'occurrence de n'importe quel événement laisse toujours possible soit e_{i+2} ou soit e_{i+3}.

On conclut que l'événement immédiatement suivant à e_i et qui se trouve dans le même processus de branchement que e_i est soit e_{i+2} ou soit e_{i+3}.

Le raisonnement est similaire à l'événement immédiatement précédent. □

A partir de la Proposition 9, on déduit le corollaire suivant.

Corollaire 6. *Soit e_i un événement d'une liste $l = (e_1, e_2, e_3, \ldots, e_n)$ qui forme une chaîne de conflit. L'ensemble des événements de l qui sont dans le même processus de branchement que e_i est le chemin d'une feuille[1] d'un arbre de racine e_i construit de la manière suivante :*

Algorithme 8 (Calcul de tous les processus maximaux d'une chaîne de conflit)**.** *Soit $Unf = (e_1 \perp e_2 \perp \ldots \perp e_n)$ la forme algébrique du dépliage d'une chaîne de conflit. L'ensemble des processus maximaux du dépliage est construit de la manière suivante :*

- *Construire grâce au Corollaire 6, deux arbres A_1 et A_2 de racines respectives e_1 et e_2 ;*

[1] Pour un arbre, une feuille est un nœud n'ayant aucun fils. Le chemin d'une feuille est le chemin de l'arbre qui part de la racine vers cette feuille.

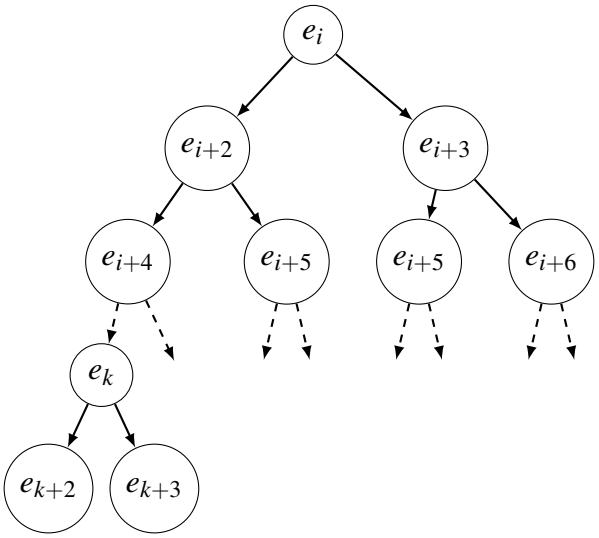


FIGURE 5.16 : Arbre de racine e_i

- *L'ensemble des processus maximaux L_p est l'union des chemins des feuilles des deux arbres A_1 et A_2.*

Les Propositions 10 et 11 montrent que l'Algorithme 8 est exact et est complet.

Proposition 10. *Les processus construits par l'Algorithme 8 sont exacts.*

Preuve.
Soit $q = (\oplus\ e_{q_1}\ e_{q_2}\ ...\ e_{q_p})$ un processus incorrect. q est incorrect si et seulement s'il existe dans q, deux événements successifs e_{q_i} et $e_{q_{i+1}}$ en conflit. Ceci contredit la construction des deux arbres grâce au Corollaire 6 qui ajoute à e_{q_i} soit l'événement $e_{q_{i+2}}$ ou soit l'événement $e_{q_{i+3}}$. □

Proposition 11. *L'Algorithme 8 est complet c'est-à-dire qu'il donne tous les processus maximaux.*

Preuve.
Soit $l = (e_1, e_2, e_3, ..., e_n)$ une chaîne de conflit. Soit $q = (\oplus\ e_{q_1}\ e_{q_2}\ ...\ e_{q_p})$ un processus maximal qui n'est pas inclus dans L_p. Désignons par $indice(e_{q_i})$, l'indice de l'événement e_{q_i} dans la liste l. q n'est pas inclus dans L_p alors il existe dans q deux événements successifs e_{q_i} et e_{q_j} tels que $n = indice(e_{q_j}) - indice(e_{q_i}) > 3$. $n > 3$ alors il existe au moins un événement $e_{q_{i2}}$ d'indice $indice(e_{q_j}) + 2$ qui ne soit ni en conflit avec e_{q_i}, ni avec e_{q_j}. $e_{q_{i2}}$ peut appartenir à q. Ceci contredit le fait que q soit maximal.

□

La forme canonique du dépliage $Unf = (e_1 \perp e_2 \perp ... \perp e_n)$ d'une chaîne de conflit est donnée par :

$$Unf = (\perp L_{p1}\ L_{p2}\ ....\ L_{pn}) \text{ où } L_p \text{ est l'ensemble des processus maximaux.} \tag{5.10}$$

Exemple

Soit la chaîne de conflit de sept événements de la Figure 5.17. Nous donnons la construction des deux arbres de racine e_1 (Figure 5.18) et e_2 (Figure 5.19) puis la forme canonique du dépliage.

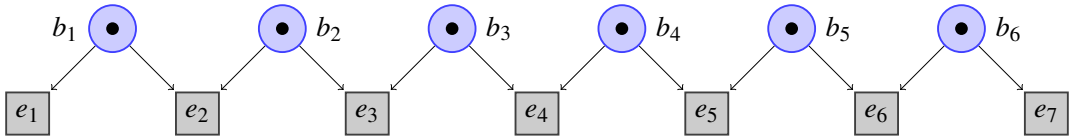


FIGURE 5.17 : Chaîne de conflit à sept événements

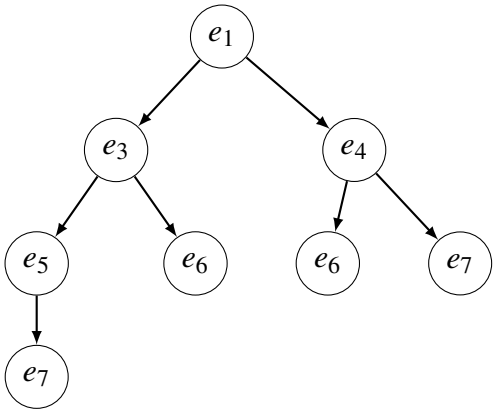


FIGURE 5.18 : Processus commençant par e_1

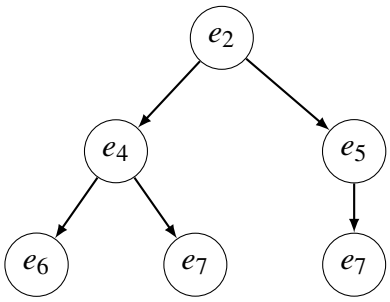


FIGURE 5.19 : Processus conmmençant par e_2

$$Unf = (\perp (\oplus\ e_1\ e_3\ e_5\ e_7)(\oplus\ e_1\ e_3\ e_6)(\oplus\ e_1\ e_4\ e_6)(\oplus\ e_1\ e_4\ e_7) \\ (\oplus\ e_2\ e_4\ e_6)(\oplus\ e_2\ e_4\ e_7)(\oplus\ e_2\ e_5\ e_7))$$

5.7 Relation de reset et algèbre de processus de branchement

Dans la Section 4.3, pour garder l'expressivité des arcs de reset, le dépliage d'un réseau de Petri avec des arcs de reset est lui-même un réseau de Petri avec des arcs de reset. Nous avions aussi énoncé, dans la Définition 56, la relation de reset. La relation de reset peut empêcher deux événements, même

n'étant pas en conflit, de ne pas appartenir à un même processus de branchement.

Pour rappel, e_1 et e_2 sont dans une relation de reset directe (notée $e_1 \multimap e_2$) si et seulement si $\exists\, b \in {}^\bullet e_2$ tel que $(b, e_1) \in F_R$. De façon générale, $e_1 \multimap e_2$ si et seulement si $\exists\, e \in E$ tel que $((e \prec e_2) \wedge (e_1 \multimap e))$. Les propriétés de la relation sont de reset sont :

- $\multimap$ n'est pas commutative ;
- $\multimap$ est transitive par causalité : si $(e_1 \multimap e_2)$ et $(e_2 \prec e_3)$ alors $(e_1 \multimap e_3)$.

La relation de reset entre deux événements e_1 et e_2 conditionne l'occurrence de l'événement e_2 à la non-observation de l'occurrence de e_1. Ceci nous permet d'énoncer l'axiome suivant pour prendre les arcs de reset dans l'algèbre de processus de branchement.

Axiome 7. $e_1 \multimap e_2 \equiv \neg e_1 \prec e_2$

Etude de cas

Sur le dépliage du réseau de Petri avec des arcs de reset de la Figure 5.20, il y a deux arcs de reset qui induisent plusieurs relations de reset. On a par exemple $e_4 \multimap e_2$, $e_4 \multimap e_5$, $e_4 \multimap e_6$ ...

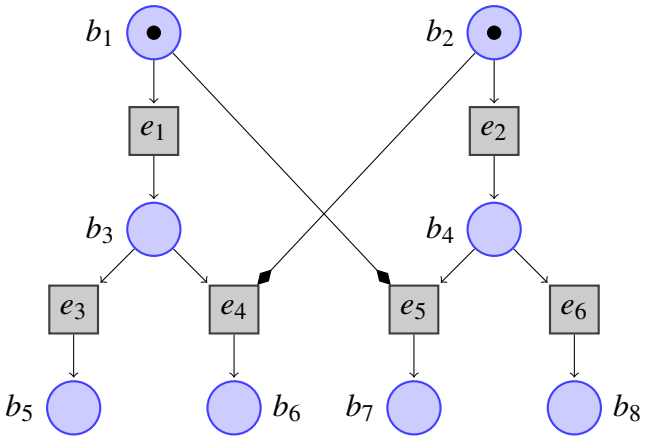


FIGURE 5.20 : Dépliage d'un réseau de Petri avec des arcs de reset

Exprimons ce dépliage sous sa forme canonique. On peut mettre le dépliage Unf sous sa forme algébrique en supprimant préalablement les conditions suivant les règles de réduction énoncées dans la Section 5.5.4.

$$\begin{aligned} Unf = \quad & ((e_1 \prec (e_3 \perp e_4)) \wr (e_2 \prec (e_5 \perp e_6)) \\ & (e_4 \multimap e_2)\ (e_4 \multimap e_5)\ (e_4 \multimap e_6) \\ & (e_5 \multimap e_1)\ (e_5 \multimap e_3)\ (e_5 \multimap e_4)) \end{aligned}$$

En remplaçant la relation $\multimap$ par celle équivalente donnée par l'Axiome 7 on obtient :

$$\begin{aligned} Unf = \quad & ((e_1 \prec (e_3 \perp e_4)) \wr (e_2 \prec (e_5 \perp e_6)) \\ & (\neg e_4 \prec e_2)\ (\neg e_4 \prec e_5)\ (\neg e_4 \prec e_6) \\ & (\neg e_5 \prec e_1)\ (\neg e_5 \prec e_3)\ (\neg e_5 \prec e_4)) \end{aligned}$$

En utilisant la distributivité de $\prec$ sur le $\perp$ puis celle de $\wr$ sur $\perp$ on a :

$$\begin{aligned} Unf \quad = \quad & (((e_1 \prec e_3) \perp (e_1 \prec e_4)) \wr ((e_2 \prec e_5) \perp (e_2 \prec e_6)) \\ & (\neg e_4 \prec e_2)\ (\neg e_4 \prec e_5)\ (\neg e_4 \prec e_6) \\ & (\neg e_5 \prec e_1)\ (\neg e_5 \prec e_3)\ (\neg e_5 \prec e_4)) \\ \\ = \quad & (((e_1 \prec e_3) \wr (e_2 \prec e_5)) \perp ((e_1 \prec e_3) \wr (e_2 \prec e_6)) \perp \\ & ((e_1 \prec e_4) \wr (e_2 \prec e_5)) \perp ((e_1 \prec e_4) \wr (e_2 \prec e_6)) \\ & (\neg e_4 \prec e_2)\ (\neg e_4 \prec e_5)\ (\neg e_4 \prec e_6) \\ & (\neg e_5 \prec e_1)\ (\neg e_5 \prec e_3)\ (\neg e_5 \prec e_4)) \end{aligned}$$

Remplaçons les $\prec$ et les $\wr$ par $\oplus$ sur la première ligne de l'équation :

$$\begin{aligned} Unf \quad = \quad & (((\oplus\ e_1\ e_3\ e_2\ e_5) \perp (\oplus\ e_1\ e_3\ e_2\ e_6) \perp (\oplus\ e_1\ e_4\ e_2\ e_5) \perp (\oplus\ e_1\ e_4\ e_2\ e_6)) \\ & (\neg e_4 \prec e_2)\ (\neg e_4 \prec e_5)\ (\neg e_4 \prec e_6) \\ & (\neg e_5 \prec e_1)\ (\neg e_5 \prec e_3)\ (\neg e_5 \prec e_4)) \end{aligned}$$

En combinant la première ligne et les deux dernières on obtient finalement :

$$\begin{aligned} Unf \quad = \quad & (\perp (\oplus\ e_1\ e_3\ e_2\ e_5)\ \ (\oplus\ e_1\ e_3\ e_2\ e_6)\ \ (\oplus\ e_1\ e_2\ e_4\ e_5)\ (\oplus\ e_1\ e_2\ e_4\ e_6) \\ & (\oplus\ e_1\ e_4)\ \ (\oplus\ e_2\ e_5)) \end{aligned}$$

5.8 Conclusion

Dans ce chapitre, nous avons présenté l'algèbre de processus et plus précisément celle des processus de branchement. Les opérateurs de l'algèbre

ont été définis ainsi que les axiomes, théorèmes et règles de réduction. Finalement nous pouvons exprimer chaque dépilage sous sa forme canonique en préservant ses conflits. Ceci nous a permis de comparer des dépliages et donc des réseaux de Petri en établissant des équivalences de conflit entre eux. Cette équivalence est très faible parce que la causalité et la concurrence sont agrégées par l'opérateur de processus $\oplus$ (on perd de la sémantique) mais converse les conflits. Pour les grosses structures, c'est en revanche, un processus rapide (simplicité des règles) de vérification des conflits, c'est-à-dire qui permet de s'assurer que certains évènements n'appartiennent pas à un même processus ou qu'un couple d'événements appartiennent à deux processus différents.

Dans le chapitre suivant qui traite des outils développés, nous présenterons *Penelope*, un outil qui implémente l'algèbre et le calcul de la forme canonique d'un dépliage.

6

Outils

Sommaire

6.1 Introduction

Nous présentons dans ce chapitre deux outils (Penelope et PN2A) développés dans le cadre de cette thèse.

- Penelope implémente l'algèbre de processus de branchement présentée à la Section 5.5. Il implémente les propriétés des opérateurs (commutativité, distributivité, ...), les règles de dérivation et de réduction, les théorèmes pour automatiser le calcul de la forme canonique d'un dépliage ;
- PN2A (Petri Net to Arduino) permet d'embarquer des réseaux de Petri T-temporels édités à partir des outils open-source (Roméo [1] et Tina [2]) sur des cartes Arduino. PN2A génère un *croquis* (sketch) Arduino qui peut être compilé et téléchargé sur un microcontrôleur présélectionné.

Les deux outils ont été réalisés grâce à l'environnement de développement intégré (IDE) DrRacket [3] basé sur le langage de programmation Scheme. Scheme est un dialecte du langage fonctionnel Lisp [57], créé dans les années 1970. L'objectif de ce nouveau langage était d'épurer Lisp en conservant les aspects essentiels, la flexibilité et la puissance expressive. Scheme a une syntaxe extrêmement simple, avec un nombre très limité de mots-clés. Comme en Lisp, Scheme utilise la notation préfixée (ou notation polonaise). Une documentation complète sur DrRacket pour sa prise en main existe au sein de l'application et est également disponible en ligne[4].

6.2 Penelope

Penelope est composé de trois modules (Figure 6.1) : le parsing, celui du dépliage et enfin Redex. Le module de parsing prend en entrée un fichier *.xml* (réseau de Petri borné) issue de Roméo et le convertit en une structure de données utilisable directement par Penelope. Il s'agit de la liste des places, des transitions, des prédécesseurs et des successeurs des transitions, du délai au plus tôt et au plus tard des transitions, le marquage initial. Ce module produit un fichier *.par* réutilisable par Penelope. Le module de dépliage calcule grâce à l'algorithme de Esparza, Römer et Vogler [31] le dépliage du réseau fourni en entrée. Ce module produit un fichier $*\_unf.xml$ visualisable par

[1] http ://romeo.rts-software.org/
[2] http ://projects.laas.fr/tina/
[3] http ://racket-lang.org/
[4] http ://docs.racket-lang.org

Roméo. Le dernier module fait de la réduction d'expressions pour fournir la forme canonique du dépliage.

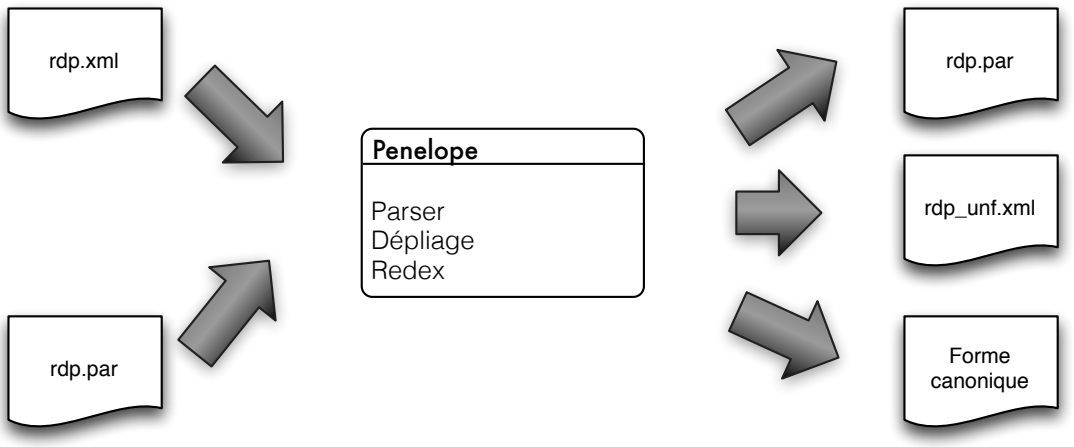


FIGURE 6.1 : Architecture de Penelope

Nous n'aborderons que la mise en œuvre du troisième module qui concerne directement l'algèbre de processus de branchement proposée. Le développement de ce module à été possible grâce au package *PLT/Redex* [33] de DrRacket.

6.2.1 Définition d'un langage en PLT/Redex

L'importation du package *redex* se fait en mettant en entête les deux lignes suivantes :

```
#lang racket
(require redex)
```

Le mot clé *define-language* spécifie une syntaxe abstraite d'un langage. La syntaxe générale se présente comme suit :

```
(define-language nom-du-langage
[
  ;définition de la grammaire
])
```

Reconnaissance d'expression

Redex offre un moyen de vérifier si une expression constitue un mot d'un langage. Ceci est réalisé grâce au mot clé *redex-match*. Sa syntaxe est la suivante :

```
(redex-match nom-du-lang
   motif
   expression)
```

- nom-du-lang, le nom du langage ;
- motif, un motif visé dans le langage ;
- expression, une expression dont on vérifie l'appartenance au langage.

redex-match retourne #*f* lorsque l'expression n'est pas un mot du langage mais retourne les différentes combinaisons utilisées par Redex pour déterminer que l'expression est un mot du langage.

Réduction

Une fois la syntaxe du langage définie, on peut procéder à la spécification d'un système de réduction d'expressions. Le mot clé en Redex qui permet de définir cette réduction est *reduction-relation*. Il prend en argumant le nom du langage et une série de règles qui spécifie les différentes réductions. Chaque règle commence par l'opérateur (- ->), puis un motif (type d'expression à réduire), ensuite le résultat (après réduction) et enfin un nom attribué à la réduction.

La réduction se fait en redex dans un contexte d'évaluation. Un contexte d'évaluation est un terme avec un trou (un trou capture le contexte) à l'endroit où la prochaine étape de réduction se fera. En redex, ce trou correspond au terme *hole*. Une relation de réduction est définie comme une série de règles de la forme (- -> *terme*1 *terme*2) où toute expression correspondant au terme1 se transforme en terme2. La définition d'un contexte de réduction se fait avec *in-hole*.

Soit le langage qui définit l'ensemble des expressions en logique disjonctive, il est défini de la façon suivante :

```
(define-language ens-bool
  [B true
     false
     (V B B)
   ])
```

Une réduction qui transforme toute expression de la forme *(V true B)* en *true* s'exprime de la façon suivante :

```
(--> (in-hole B (V true B))
        (in-hole B true)
        V-true)
```

Traces

Redex ne se contente pas seulement de donner le motif réduit, mais indique également tous les chemins possibles afin d'obtenir cette réduction. La fonction qui permet de construire cette trace de réduction est *traces*. Elle combine la définition du langage, la définition des règles de réduction et l'expression à réduire. La syntaxe pour construire les traces d'une réduction est la suivante : toutes les traces conduisent au même résultat.

```
(traces nom-du-lang
       expression)
```

La réduction de l'expression *(V (V true false) (V true true))* donne la trace de la Figure 6.2 :

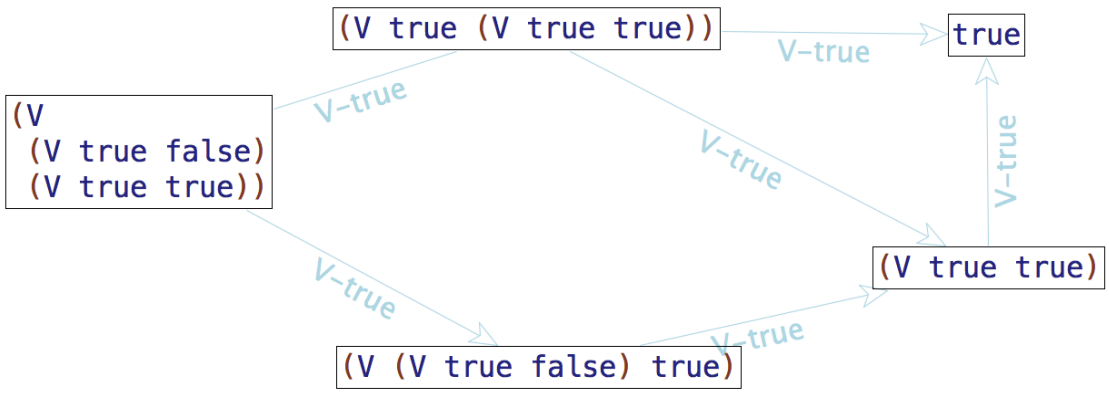


FIGURE 6.2 : Réduction de (V (V true false) (V true true))

6.2.2 Implémentation de l'algèbre de processus de branchement

L'algèbre de processus de branchement est définie en Racket comme suit :

Définition du langage

Le langage porte le nom de $Process - lang$.

```
(define-language Process-lang
 ...
)
```

Le langage comprend les nœuds qui sont soit une variable booléenne (t ou f), soit une condition b ou soit un évènement e. Il s'agit aussi du contraire d'un nœud.

```
[n variable bool b e ¬n]
[bool t f]
[e variable]
[b variable]
```

Les opérateurs de l'algèbre de branchement sont ceux de processus $\oplus$, du conflit $\perp$, de la concurrence $\wr$ et de la causalité $\prec$.

```
[on ⊕ ⊥ ≀]
[o2 ≺] ; binary operators
```

Un processus est un ensemble de nœuds liés par l'opérateur de processus $\oplus$. De façon plus large, il désigne également un ensemble de processus relié par le même opérateur.

```
[P variable (⊕ Q ... )]
[Q variable n (¬ n)] ; essayer de supprimer P
[C-P (⊕ C-P P) (⊕ P C-P) hole]
```

Le conflit est le même que le processus en remplaçant l'opérateur $\oplus$ par $\perp$.

```
[X variable (⊥ Y ... )]
[Y variable X P n (¬ n)]
[C-X (⊥ C-X X) (⊥ X C-X) hole]
```

Un mot du langage ou une expression est défini conformément à la signature du langage (Section 5.5.1).

```
[E variable (on F ...) (e o2 b) (P o2 X) (P o2 e) (¬ n)]
[F variable E P X]
[C-E (on C-E E) (on E C-E) (E o2 C-E) (C-E o2 E) hole]
```

Implémentation des règles de réduction

Les différentes règles de réduction sont implémentées au sein de *Process-red*

```
(define Process-red
  (reduction-relation
   Process-lang
   ...
   )
 )
```

Il s'agit d'un ensemble de règles de réduction aboutissant à la forme canonique du dépliage d'un réseau de Petri. Par exemple la distributivité de l'opérateur $\oplus$ sur l'opérateur $\bot$ est réalisée comme suit :

```
(--> (in-hole C-P (⊕ E_0 ... (⊥ E_1 E_2) E_3 ...))
        (in-hole C-P (⊥ (⊕ E_0 ... E_1 E_3 ... ) (⊕ E_0
           ... E_2 E_3 ...) )) "⊕→⊥1")
```

L'outil Penelope contenant toutes les règles de réduction et de dérivation se trouve à l'adresse http ://penelope.rts-software.org/

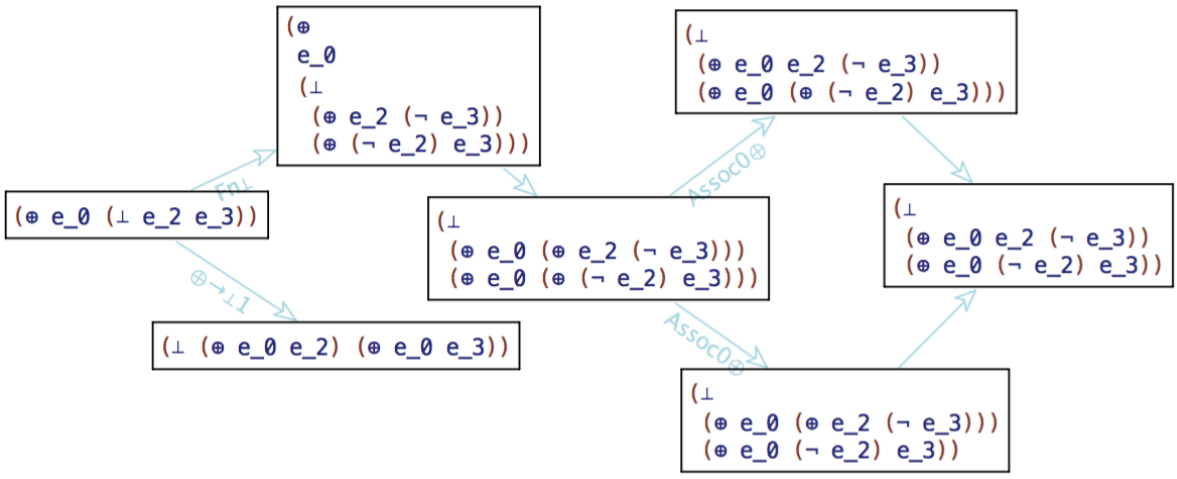


FIGURE 6.3 : Exemple de réduction avec Penelope

6.3 Petri Net To Arduino (PN2A)

PN2A permet d'embarquer un réseau de Petri T-temporel sur un microcontrôleur. Les places et transitions du réseau de Petri sont associées aux pattes du microcontrôleur et représentent les actionneurs et capteurs du système

réel. Les microcontrôleurs disposent en général de quelques dizaines de pattes d'entrée et de sortie (54 pour l'Atmega 2560) : le domaine d'application de PN2A se limite donc aux petits systèmes. L'idée développée dans cet outil est de permettre une implémentation des réseaux de Petri. Cette implémentation survient avant la phase de réalisation et permet de confronter le modèle (théorique) à la réalité : son environnement, ses périphériques, ... (Figure 6.4). La finalité est de valider la performance et les aspects fonctionnels du système.

De plus, d'un point de vue de codage, l'expressivité des réseaux de Petri T-temporel offre de nouvelles possibilités dans le développement des applications embarquées. Les jetons contenus dans les places apportent un pouvoir expressif, alors que le temps offre, par exemple, des concepts de synchronisation de chien de garde ... Grâce à la synchronisation et au parallélisme, on peut imaginer mettre en œuvre des tâches concurrentes sans avoir besoin d'un système d'exploitation.

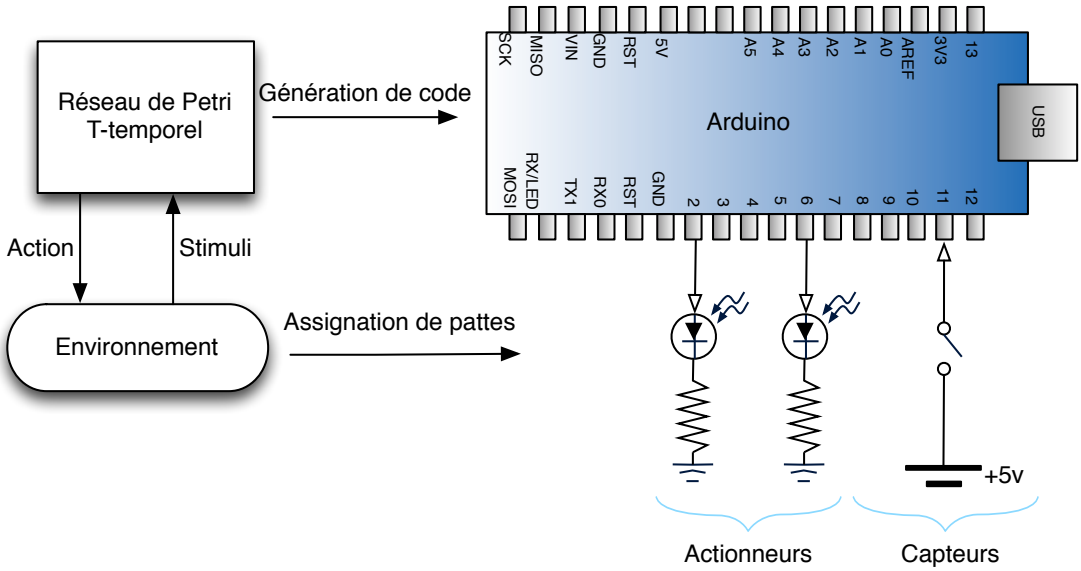


FIGURE 6.4 : Réseau de Petri T-temporel et son environnement

La connexion des nœuds du réseau avec les pattes du microcontrôleur le rend non-autonome. Le tir des transitions doit se synchroniser avec les signaux (niveau haut ou bas) observés sur les pattes du microcontrôleur. Dans la section suivante, nous présentons la façon de projeter un réseau de Petri T-temporel sur un microcontrôleur. Nous commençons par définir un réseau de Petri T-temporel à microcontrôleur synchronisé (mSTPN)

6.3.1 Réseau de Petri T-temporel à microcontrôleur synchronisé (mSTPN)

Définition 65 (mSTPn)**.** *Un réseau de Petri T-temporel à microcontrôleur synchronisé (mSTPn) est la paire* $\langle N, A \rangle$ *où :*

- *N est un réseau de Petri T-temporel et*
- $A : P \cup T \rightarrow \langle Pin, Level \rangle$ *est la fonction qui associe à chaque transition t et à chaque place p un numéro de patte du microcontrôleur (Pin) et un niveau de signal (Level). Level* $\in \{High, Low\}$ *pour niveau haut ou niveau bas.*

La sémantique opérationnelle d'un *mSTPn* est définie comme suit :

- Transitions sensibilisées par un marquage M : $Enabled(M)$ est l'ensemble des transitions sensibilisées depuis le marquage M. Sa définition est la même que celle d'un réseau de Petri T-temporel (Section 3.5.5)
- Transitions nouvellement sensibilisées : $\uparrow Enabled(M)$ désigne l'ensemble des transitions nouvellement sensibilisées (Section 3.5.5).
- Transitions $>_{sensibilisées}$: Soit t une transition telle que $M \xrightarrow{t} M'$. Une transition t' est $>_{sensibilisée}$ si et seulement si $t' \in (Enabled(M) \cap Enabled(M'))$ et $\exists p \in {}^\bullet t', M'(p) > M(p)$ (comme le montre la Figure 6.5). $Greater(M)$ désignera l'ensemble des transitions $>_{sensibilisées}$.

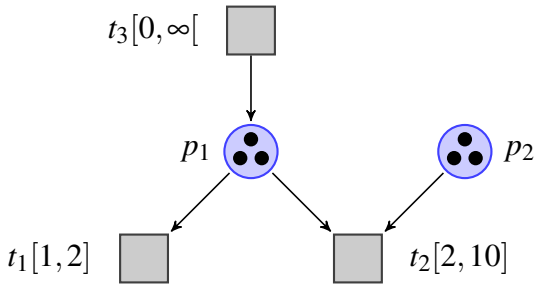


FIGURE 6.5 : Transition plus sensibilisée

- Soit v un vecteur de dimension $n = |T|$ de valeurs entières. Soit t une transition. $v(t)$ représente le temps écoulé depuis la sensibilisation de la transition t : c'est l'horloge locale de t. Initialement, v est à valeur nulle.
- Transition tirable : la transition t est tirable si et seulement si $efd(t) \leq v(t) \leq lfd(t)$. Si $v(t) > lfd(t)$, t devient non tirable (sémantique faible).

$Firable(M)$ est l'ensemble des transitions tirables à partir du marquage M.

- Activation d'une transition : Soit Pins un vecteur contenant l'état des pattes du microcontrôleur. Une transition t est activée si le niveau détecté sur la patte du microcontrôleur liée à t correspond à celui prédéfinie pour cette patte. t est activée si et seulement si $(A(t) \rightarrow pin) == pin \wedge Pins(pin) == (A(t) \rightarrow L)$. $Sensitized(M)$ est l'ensemble des transitions activées.
- Tir de t à partir du marquage M : Si $t \in (Enabled(M) \cap Sensitized(M) \cap Firable(M))$ alors $M \xrightarrow{t} M'$ est défini par

$$\forall p \in P,\ M'(p) = M(p) - Pre(p,t) + Post(t,p)$$

La Définition 66 donne la sémantique d'un *mSTPn*.

Définition 66 (Sémantique d'un *mSTPn*)**.** *La sémantique d'un mSTPn est définie par un système de transitions $\langle Q, \{q_0\}, \Sigma, \rightarrow \rangle$ tel que :*

- $Q = \mathbb{N}^{|P|} \times \mathbb{N}^{|T|}$
- $q_0 = \langle M_0, v_0 \rangle \in Q$
- $\Sigma = T$
- $\rightarrow \subseteq Q \times (T \cup \mathbb{N}) \times Q$
  - *Evolution temporelle :*

$$\langle M, v \rangle \xrightarrow{\theta} \langle M, v' \rangle \Rightarrow \begin{cases} \forall t \in T, M \geq {}^\bullet t \Rightarrow v'(t) \leq lfd(t) \\ v' := v + \theta \end{cases} \tag{6.1}$$

  - *Tir d'une transition :*

$$\langle M, v \rangle \xrightarrow{t} \langle M', v' \rangle \Rightarrow \begin{cases} M \geq {}^\bullet t \\ M' := M - {}^\bullet t + t^\bullet \\ efd(t) \leq v(t) \leq lfd(t) \\ Pins(A(t) \rightarrow pin) == A(t) \rightarrow level \\ \forall t' \in T, v'(t') := \begin{cases} 0 \ si\ t' \in \uparrow Enabled(M) \cup Greater(M) \\ v(t') \ sinon \end{cases} \end{cases} \tag{6.2}$$

6.3.2 Algorithme d'activation d'un mSTPN

Dans la pratique, l'activation d'un mSTPN est donnée par l'Algorithme 9

Algorithme 9 (Algorithme d'activation d'un mSTPN)**.**

1. *Tant que Vrai faire :*
   (a) *Lire les pattes d'entrée du microcontrôleur ;*
   (b) *Mettre à jour le vecteur Sensitized ;*
   (c) *Pour tout* $t \in T$ *faire :*
      i. *Si* $t \in Enabled(M) \cap Sensitized(M) \cap Firable(M)$ *alors*
         - *M := M - Pre(.,t) + Post(t,.)*
      ii. $Firable[t] := False$
      iii. $Sensitized[t] := False$
      iv. $v[t] := 0$
   (d) *Pour tout* $\uparrow Enabled(M) \cup Greater(M)$ *alors*
      - *v[t] := 0 ;*
   (e) *Les pattes de sortie du microcontrôleur correspondent aux places qui sont activées ;*
   (f) *Délai complet d'un cycle*

6.3.3 Génération de croquis Arduino

Parsing

Les réseaux de Petri T-temporel utilisés par PN2A sont réalisés avec Roméo ou TINA. Cest deux outils produisent des fichiers correspondant à la description du TPN. Le *parsing* consiste à produire à partir des fichiers produits par Roméo ou TINA des déclarations en langage C de la structure du TPN. Pour la Figure 6.5, nous avons la déclaration suivante :

```
int M0[2] = {3,3}; /* Initial marking */
int M1[2] = {3,3};
int M2[2] = {3,3};
int Pre[2][3] = {{1,1,0},{0,1,0}};
int Post[2][3] = {{0,0,1},{0,0,0}};
int C[2][3];
...
unsigned long Horloge[3]={0,0,0};
```

```
unsigned long alpha[3]={1,2,0};
unsigned long beta[3]={2,10,INT32_MAX};
...
```

$M0$ est le marquage initial. Lors du tir d'une transition t passant du marquage M à M' ($M \xrightarrow{t} M'$), dans la déclaration précédente, $M1$ correspond à M et $M2$ à M'. On retrouve dans les déclarations les matrices *Pre* et *Post*. Les intervalles statiques de tir des transitions sont stockés respectivement dans les variables $alpha$ et $beta$. Sur l'exemple, t_3 a pour intervalle statique $[0, \infty[$. L'infini est représenté par $INT32\_MAX$ (2^{32}). Les intervalles dynamiques sont enregistrés dans la variable $Horloge$.

Fonction d'interruption

La fonction d'interruption est appelée à chaque débordement du *timer* T_2. TCNT2 fixe cette période à 1ms. $deltaT$ définit le vieillissement des horloges (deltaT=50). Ce timer T_2 [6] est commun aux *ATmega328 et ATmega2560* qui sont compatibles à plusieurs cartes Arduino cartes : *Uno, Mega, Fio, Nano (v> 3.0), Lilypad, Pro, Pro mini.*

```
ISR(TIMER2_OVF_vect) {
int t;
count++;
if(count > deltaT){
  count = 0;
  for(t=0;t<NB_TRANSITIONS;t++){
    if((ENABLED[t])&&Horloge[t]<beta[t]) {
      Horloge[t]=Horloge[t]+deltaT;
    }
  }
}
  TCNT2 = 130;
  TIFR2 = 0x00;
};
```

Génération de code

Un projet ou croquis Arduino est un répertoire contenant des fichiers. Le répertoire contient au moins un fichier portant le même nom que le croquis avec l'extension *.ino* : c'est le fichier principal du projet. Le fichier principal comporte au moins deux fonctions :

- la fonction d'initialisation setup() ;

- la fonction de boucle loop().

La fonction setup()

Arduino exécute une seule fois au début du programme, les instructions contenue dans la fonction setup(). La fonction setup() initialise les registres comme le Timer T_2 à 1ms, calcule la matrice d'incidence C, initialise les pattes d'entrées et de sorties ainsi que le bus de communication.

La fonction loop()

Une fois la phase d'initialisation terminée, Arduino exécute toutes les instructions contenues dans la fonction loop(). Les instructions sont répétées indéfiniment tant que la carte est sous tension.

A chaque cycle, la fonction $loop()$ lit l'état des capteurs d'entrée, puis calcule l'ensemble des transitions sensibilisées. Ensuite, elle exécute l'Algorithme 9 d'activation d'un mSTPN. Les horloges de chaque transition sont mises à jour et les sorties affichées : il s'agit dans nos tests, d'allumer ou d'éteindre une led.

6.3.4 Exemple

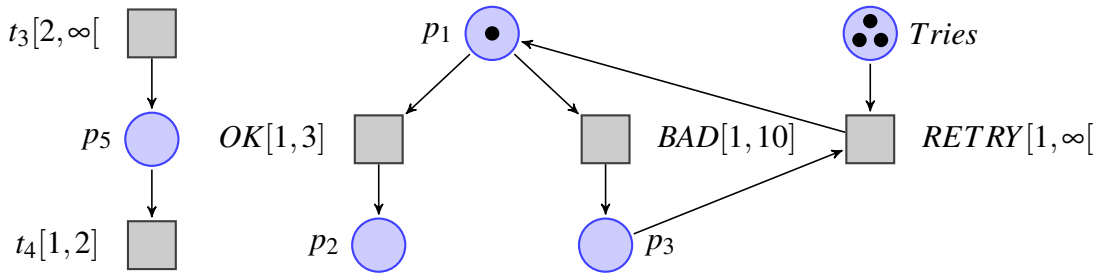


FIGURE 6.6 : Système d'identification

L'exemple de la Figure 6.6 modélise un système d'identification. L'utilisateur dispose de trois essais pour s'identifier (identification réussie s'il y a un jeton dans la place p_2) avec des contraintes temporelles. En parallèle, les utilisateurs peuvent agir sur les transitions t_3 ou t_4. Des boutons ont été affectés aux transitions OK, BAD, $RETRY$, t_3 et t_4, alors que des leds ont été affectés aux places p_1, p_2, p_3, p_5 et $Tries$.

En exécutant le modèle de la Figure 6.6 sur une carte Arduino, on a observé l'effet de rebond [5] au niveau des boutons. L'attribution de nouveaux intervalles de temps aux transitions et une nouvelle durée de cycle a permis d'éliminer l'effet de rebond. Ces tests ont permis d'adapter le modèle théorique avec une implémentation réelle sur une carte arduino en prenant en compte des contraintes matérielles.

Dans Roméo le temps évolue par tir de transitions alors que PN2A fait écouler le temps par une horloge temps réel. Cette mise en œuvre confronte vraiment le modèle à son environnement.

6.3.5 Complexité

La plus grande complexité de l'algorithme est le calcul des transitions sensibilisées par un marquage $Enabled(M)$ et le tir d'une transition. La complexité algorithmique est de l'ordre de $|P| \times |T|$. Pour un ATmega2560 avec un oscillateur à 16 MHz, le tableau suivant présente les temps d'exécution du code généré par PN2A en fonction de la complexité du réseau de Petri T-temporel. Les tests de complexité ont été réalisés à partir des exemples de réseaux de taille croissante par duplication d'un modèle de base. L'occupation mémoire augmente avec la taille des vecteurs *Pre* et *Post* qui permettent de caractériser le réseau. Les temps d'exécution quant à eux, augmentent de façon plus significative que l'occupation mémoire. Pour le dernier test, (25 places et 25 transitions) le temps d'exécution est de 4 ms. Cette taille correspond presque au nombre maximal d'entrées et de sorties de l'Atmega 2560 (54 pattes).

Complexité	*Temps d'exécution*	*Taille du programme*	*Mémoire allouée*
$\|P\| = 3\ \|T\| = 2$	150 μsec	5796 bytes	444 bytes (5%)
$\|P\| = 4\ \|T\| = 9$	180 μsec	6024 bytes	594 bytes (7%)
$\|P\| = 10\ \|T\| = 10$	840 μsec	6296 bytes	1242 bytes (15%)
$\|P\| = 20\ \|T\| = 20$	2500 μsec	7394 bytes	3302 bytes (40%)
$\|P\| = 25\ \|T\| = 25$	4000 μsec	8666 bytes	4872 bytes (59%)

TABLE 6.1 : Temps d'exécution

6.3.6 Exécutable et utilisation de PN2A

PN2A est écrit en LISP (DrRacket). Les exécutables sont produits pour les plateformes Windows, Mac OS et Linux et sont librement distribués sur *rts-*

[5] En électronique, un rebond est une suite d'impulsions parasites délivrées par les boutons poussoirs et les interrupteurs quand on les actionne.

software [6]. Le fichier d'entrée décrivant le TPN peut être réalisé soit avec l'outil Roméo ou soit avec TINA. L'utilisateur a la possibilité de choisir le type de microcontrôleur, d'assigner des pattes du microcontrôleur aux places et aux transitions. Le délai d'un cycle peut être ajusté. Le bouton Generate Arduino sketch génère le croquis Arduino. Les Figures C.1, C.2, C.3 et C.4 de l'Annexe C montrent l'interface de PN2A.

6.4 Conclusion

Dans ce chapitre, nous avons présenté deux outils développés dans le cadre de cette thèse : Penelope et PN2A.

Penelope offre un cadre logiciel pour l'analyse et le calcul des processus maximaux issus du dépliage des réseaux de Petri. Les axiomes, théorèmes, règles de dérivation, règles de réduction et distributivités des opérateurs de l'algèbre de branchement ont été codés en LISP avec le paquet PLT/Redex.

Quant à PN2A, c'est un outil qui permet d'embarquer un réseau de Petri T-temporel sur une carte Arduino. Ceci a nécessité de définir une nouvelle sémantique basée sur la sémantique faible (Section 3.5.5) : lorsque l'horloge d'une transition t atteint sa date de tir au plus tard $Lfd(t)$, la transition devient non tirable. On peut étendre PN2A à d'autres sémantiques et à d'autres type de microcontrôleur. Actuellement, les places du réseau sont associés aux pattes du microcontrôleur. Il pourrait être possible d'associer des blocs de code non bloquant aux places pour offrir des possibilités plus larges dans le développement de petites applications embarquées.

[6] http ://pn2a.rts-software.org

7

Conclusion générale

Sommaire

7.1 Bilan

Les travaux présentés dans ce manuscrit portent sur la contribution au dépliage des réseaux de Petri et à l'analyse des processus qui y sont issus. Le dépliage des réseaux de Petri est une méthode d'ordre partiel qui constitue une meilleure alternative (moins coûteuse) que la construction de l'espace d'états. Il contient toutes les informations nécessaires pour accéder à tous les états accessibles et permet alors, au même titre que le graphe des états accessibles ou graphe des marquages, de faire des vérifications.

Il existe de plus en plus d'algorithmes de calcul du dépliage des réseaux de Petri différents les des autres, en fonction de la performance recherchée et des vérifications à faire. Nos travaux se sont basés sur l'algorithme de Esparza, Römer et Vogler [31] qui permet de capturer l'espace d'états. En nous basant sur cet algorithme, nous avons proposé une contribution aux dépliages des réseaux de Petri avec des arcs de reset.

Pour y arriver, nous avons démontré l'inexistence d'une traduction structurelle d'un réseau de Petri avec des arcs de reset vers un réseau de Petri ordinaire qui préserve à la fois la bisimulation et la sémantique d'ordre partiel. Un nouvel algorithme a été proposé pour calculer le dépliage d'un réseau de Petri avec des arcs de reset et, plus particulièrement, un préfixe fini et complet du dépliage pour un réseau de Petri borné avec des arcs de reset.

Le dépliage est l'union de tous les processus (exécutions possibles du système). Les processus issus du dépliage, souvent appelés processus de branchement, possèdent quelques particularités qui les distinguent des processus classiques de Milner (CCS [63]), de Hoare (CSP [41]) pour ne citer que ceux-là. L'algèbre des processus de branchement présentée dans ce manuscrit, se base notamment sur *CCS* pour proposer une analyse algébrique qui prend en compte les spécificités des processus de branchement.

L'algèbre de processus proposée offre un cadre de travail pour l'analyse algébrique des processus de branchement. Elle définit un ensemble d'opérateurs, d'axiomes, de théorèmes, de distributivitiés, de règles de dérivation et un processus de réduction conduisant à la forme canonique d'un dépliage. La forme canonique d'un dépliage est unique et est caractéristique d'un dépliage. On peut alors comparer des dépliages sur la base de leurs formes canoniques. Cette comparaison établit une équivalence entre deux ou plusieurs réseaux de Petri qui ne préserve que les conflits : l'équivalence de conflits. L'équivalence de conflits est une équivalence faible (en la comparant à l'équivalence observationnelle par exemple). Ceci est dû à l'introduction de l'opérateur de processus $\oplus$ qui agrège les opérateurs de causalité ($\prec$) et de concurrence ($\wr$).

Deux outils ont été présentés dans ce manuscrit. Il s'agit de Penelope qui automatise le calcul de la forme canonique d'un dépliage et PN2A qui permet d'embarquer sur un microcontrôleur un réseau de Petri T-temporel.

7.2 Perspectives

A partir des travaux présentés dans cette thèse, quelques perspectives intéressantes méritent d'être explorées.

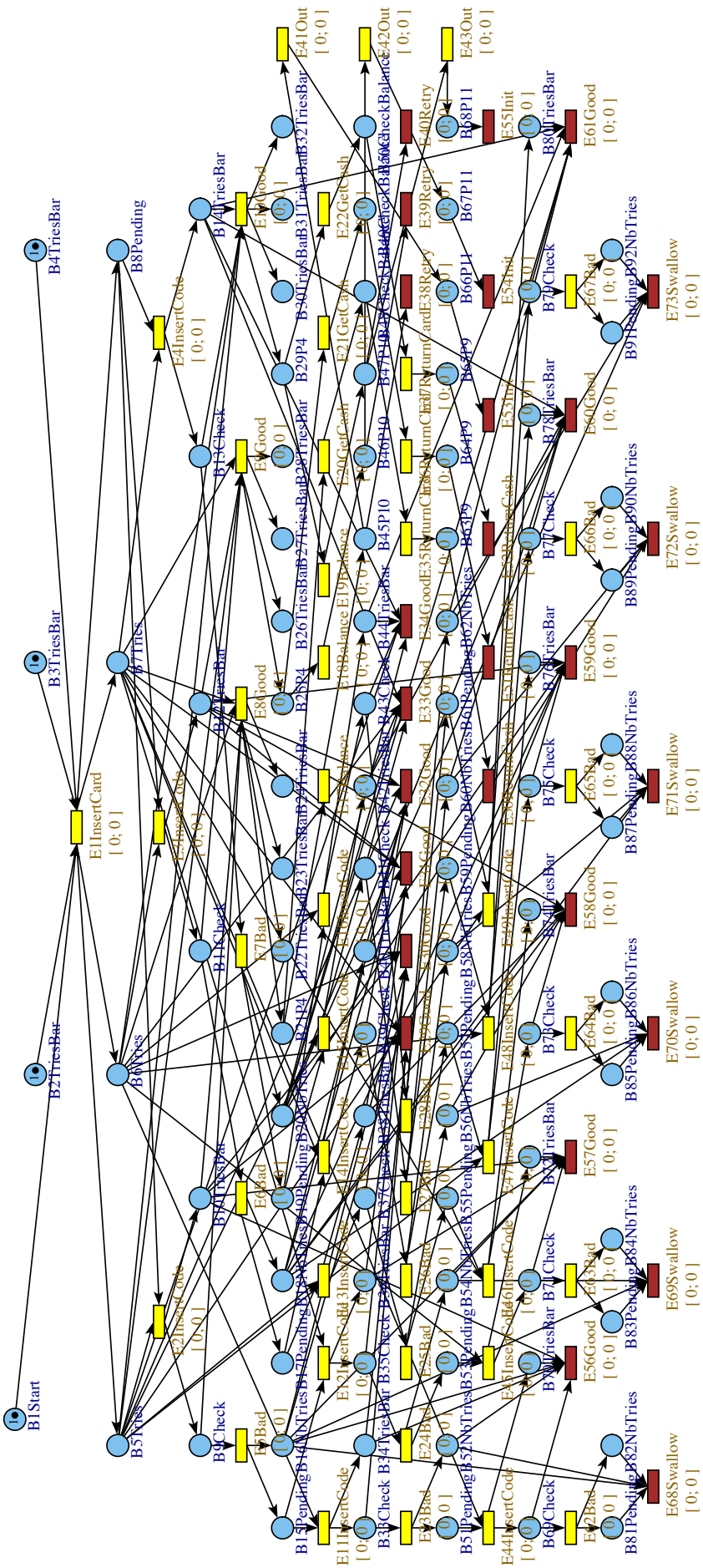
La construction du préfixe fini et complet du dépliage d'un réseau de Petri avec des arcs de reset bornés se fait en trois étapes. Premièrement, nous construisons sa transformation structurelle. Deuxièmement, on construit le préfixe fini et complet de cette transformation structurelle. Et pour finir, nous fusionnons les transitions et les places créées dans la transformation structurelle puis nous supprimons les causalités engendrées en remplaçant les arcs qui les matérialisent par des arcs de reset. Bien entendu, les arcs de reset sont rajoutés tels que décrit dans la Définition 58. Tout ceci à cause de l'incapacité d'utiliser les évènements cut-off de Esparza, Römer et Vogler [31]. Leurs évènements cut-off ne donnent pas la bonne profondeur pour avoir tous les états accessibles. Il serait intéressant de définir les bons évènements cut-off sur le réseau de Petri sous-jacent. Ceci permettra de construire directement le préfixe fini du réseau sous-jacent à la bonne profondeur. En ajoutant à ce préfixe fini les arcs de reset, nous obtiendrons le préfixe fini et *complet* du dépliage des réseaux de Petri avec des arcs de reset.

L'algèbre des processus de branchement présentée dans le Chapitre 5 ne concerne que les réseaux de Petri "ordinaires". Elle a été étendue aux réseaux de Petri avec des arcs de reset. On pourrait envisager de l'étendre aux autres extensions des réseaux de Petri notamment aux réseaux de Petri T-temporels. De plus, d'autres équivalences moins faibles que l'équivalence de conflits peuvent être établies.

ANNEXES

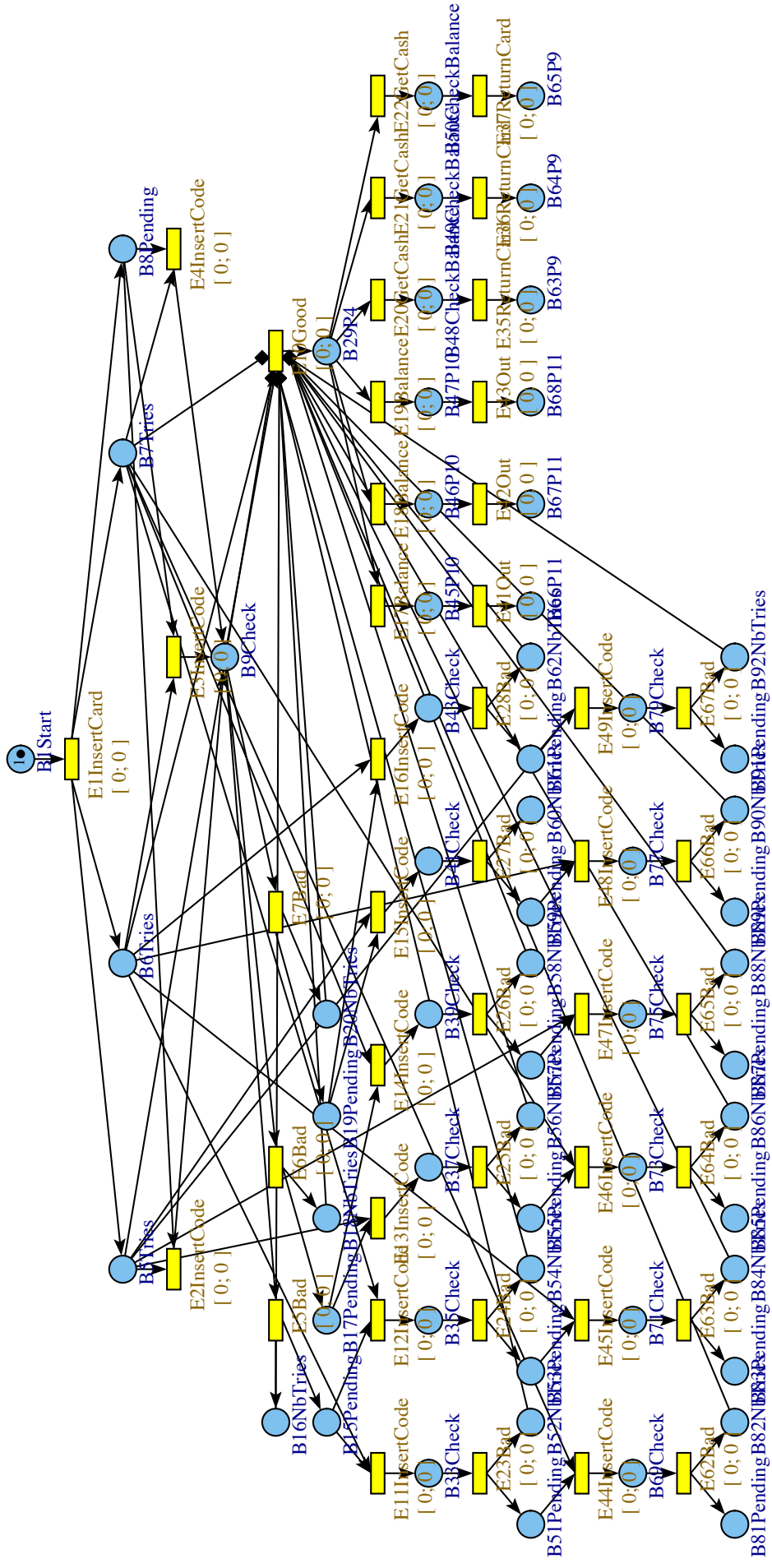
A

Préfixe fini et complet du dépliage de la transformation structurelle du DAB de la Figure 4.18


B1Start
B2TriesBar
B3TriesBar
B4TriesBar
B5Tries
B6Tries
B7Tries
B8Pending
E1InsertCard [0; 0]
E2InsertCode [0; 0]
E3InsertCode [0; 0]
E4InsertCode [0; 0]
B9Check
B10Check
B11Check
B12Check
B13Check
E5Bad [0; 0]
E6Good [0; 0]
E7Bad [0; 0]
E8Good [0; 0]
E9Good [0; 0]
E10Good [0; 0]
E11InsertCode [0; 0]
E12InsertCode [0; 0]
E13InsertCode [0; 0]
E14InsertCode [0; 0]
E15InsertCode [0; 0]
E16InsertCode [0; 0]
E17InsertCode [0; 0]
E18Balance [0; 0]
E19Balance [0; 0]
E20GetCash [0; 0]
E21GetCash [0; 0]
E22GetCash [0; 0]
E23Bad [0; 0]
E24Bad [0; 0]
E25Bad [0; 0]
E26Bad [0; 0]
E27Bad [0; 0]
E28Bad [0; 0]
E29Bad [0; 0]
E30Bad [0; 0]
E31Bad [0; 0]
E32Bad [0; 0]
E33Good [0; 0]
E34Good [0; 0]
E35ReturnCard [0; 0]
E36ReturnCard [0; 0]
E37ReturnCard [0; 0]
E38ReturnCard [0; 0]
E39Retry [0; 0]
E40Retry [0; 0]
E41Out [0; 0]
E42Out [0; 0]
E43Out [0; 0]
E44InsertCode [0; 0]
E45InsertCode [0; 0]
E46InsertCode [0; 0]
E47InsertCode [0; 0]
E48InsertCode [0; 0]
E49InsertCode [0; 0]
E50InsertCode [0; 0]
E51InsertCode [0; 0]
E52InsertCode [0; 0]
E53Init [0; 0]
E54Init [0; 0]
E55Init [0; 0]
E56Good [0; 0]
E57Good [0; 0]
E58Good [0; 0]
E59Good [0; 0]
E60Good [0; 0]
E61Good [0; 0]
E62Bad [0; 0]
E63Bad [0; 0]
E64Bad [0; 0]
E65Bad [0; 0]
E66Bad [0; 0]
E67Bad [0; 0]
E68Swallow [0; 0]
E69Swallow [0; 0]
E70Swallow [0; 0]
E71Swallow [0; 0]
E72Swallow [0; 0]
E73Swallow [0; 0]
B81Pending B82NbTries
B83Pending B84NbTries
B85Pending B86NbTries
B87Pending B88NbTries
B89Pending B90NbTries
B91Pending B92NbTries
B2P4
B3P4
B4P4
B5P4
B6P4
B7P4
B8P4
B9P4
B10P4
B11P4
B12P4
B13P4
B14P4
B15P4
B16P4
B17P4
B18P4
B19P4
B20P4
B21P4
B22P4
B23P4
B24P4
B25P4
B26P4
B27P4
B28P4
B29P4
B30P4
B31P4
B32P4
B33P4
B34P4
B35P4
B36P4
B37P4
B38P4
B39P4
B40P4
B41P4
B42P4
B43P4
B44P4
B45P4
B46P4
B47P4
B48P4
B49P4
B50P4
B51P4
B52P4
B53P4
B54P4
B55P4
B56P4
B57P4
B58P4
B59P4
B60P4
B61P4
B62P4
B63P4
B64P4
B65P4
B66P11
B67P11
B68P11
B69P11
B70P11
B71P11
B72P11
B73P11
B74P11
B75P11
B76P11
B77P11
B78P11
B79P11
B80P11

B

Préfixe fini et complet du dépliage du DAB de la Figure 4.17


B1Start
E1InsertCard
[0; 0]
B2Tries
E2InsertCode
[0; 0]
B3Check
E3BadCode
[0; 0]
B4Tries
E4InsertCode
[0; 0]
B5Check
E5Bad
[0; 0]
B6Bad
[0; 0]
B7Tries
E6Bad
[0; 0]
B8Pending
E7Bad
[0; 0]
B9Check
E8GoodCode
[0; 0]
B10Pending
B11Tries
B12Pending
B13Tries
B14Pending
B15Tries
B16NbTries
B17Pending
B18Tries
B19Pending
B20NbTries
B21Pending
B22Tries
B23Pending
B24NbTries
B25Pending
B26Tries
B27Pending
B28NbTries
B29R4
E9Good
[0; 0]
E10Out
[0; 0]
B30Check
E11InsertCode
[0; 0]
E12InsertCode
[0; 0]
E13InsertCode
[0; 0]
E14InsertCode
[0; 0]
E15InsertCode
[0; 0]
E16InsertCode
[0; 0]
E17Balance
[0; 0]
E18Balance
[0; 0]
E19Balance
[0; 0]
E20GetCash
[0; 0]
E21GetCash
[0; 0]
E22GetCash
[0; 0]
B31Check
B32Check
B33Check
B34Check
B35Check
B36Check
B37Check
B38Check
B39Check
B40Check
B41Check
B42Check
B43Check
B44Check
B45P10
B46P10
B47P10
B48Check
B49Check
B50Check
B51Pending
B52NbTries
B53Pending
B54NbTries
B55Pending
B56NbTries
B57Pending
B58NbTries
B59Pending
B60NbTries
B61Pending
B62NbTries
B63P9
B64P9
B65P9
B66P11
B67P11
B68P11
E23Bad
[0; 0]
E24Bad
[0; 0]
E25Bad
[0; 0]
E26Bad
[0; 0]
E27Bad
[0; 0]
E28Bad
[0; 0]
E29Out
[0; 0]
E30Out
[0; 0]
E31Out
[0; 0]
E32ReturnCard
[0; 0]
E33ReturnCard
[0; 0]
E34ReturnCard
[0; 0]
E35ReturnCard
[0; 0]
E36ReturnCard
[0; 0]
E37ReturnCard
[0; 0]
E41InsertCode
[0; 0]
E42InsertCode
[0; 0]
E43InsertCode
[0; 0]
E44InsertCode
[0; 0]
E45InsertCode
[0; 0]
E46InsertCode
[0; 0]
E47InsertCode
[0; 0]
E48InsertCode
[0; 0]
E49InsertCode
[0; 0]
B69Check
B70Check
B71Check
B72Check
B73Check
B74Check
B75Check
B76Check
B77Check
E61Bad
[0; 0]
E62Bad
[0; 0]
E63Bad
[0; 0]
E64Bad
[0; 0]
E65Bad
[0; 0]
E66Bad
[0; 0]
E67Bad
[0; 0]
B78Pending
B79NbTries
B80Pending
B81NbTries
B82Pending
B83NbTries
B84Pending
B85NbTries
B86Pending
B87NbTries
B88Pending
B89NbTries
B90Pending
B91NbTries
B92NbTries

C

Interfaces de PN2A

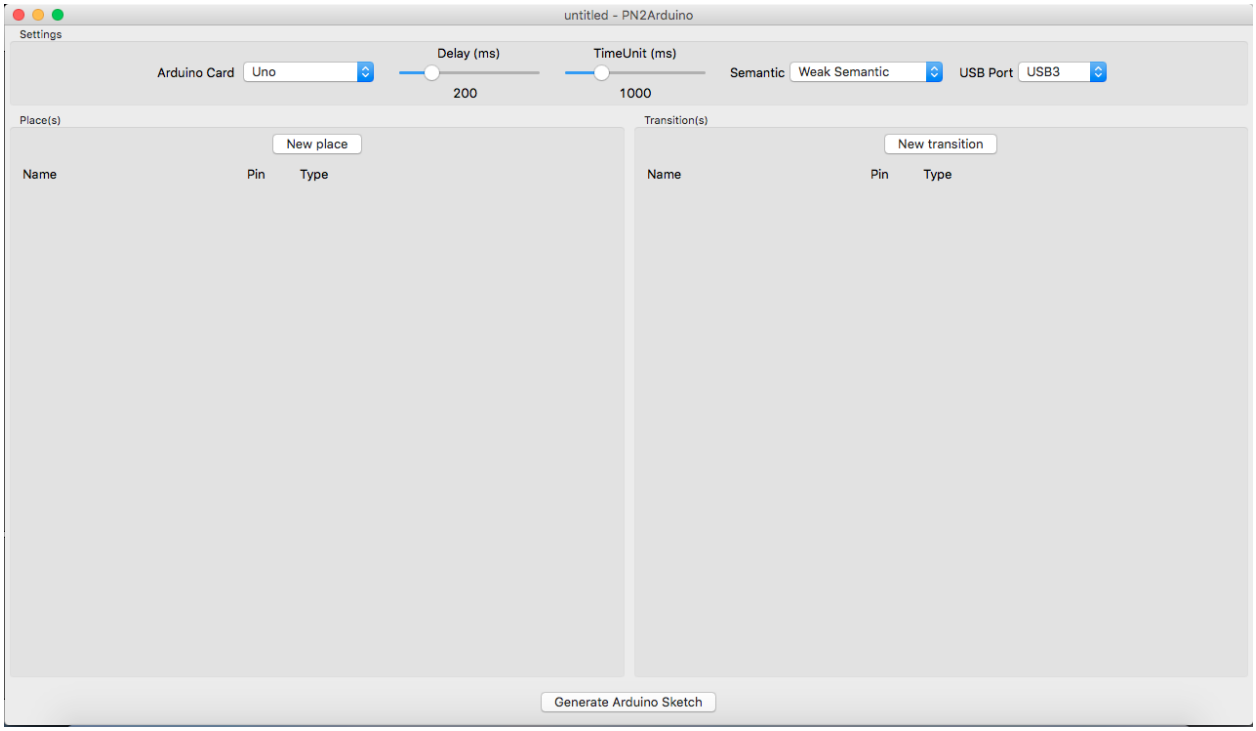


FIGURE C.1 : Accueil de PN2A

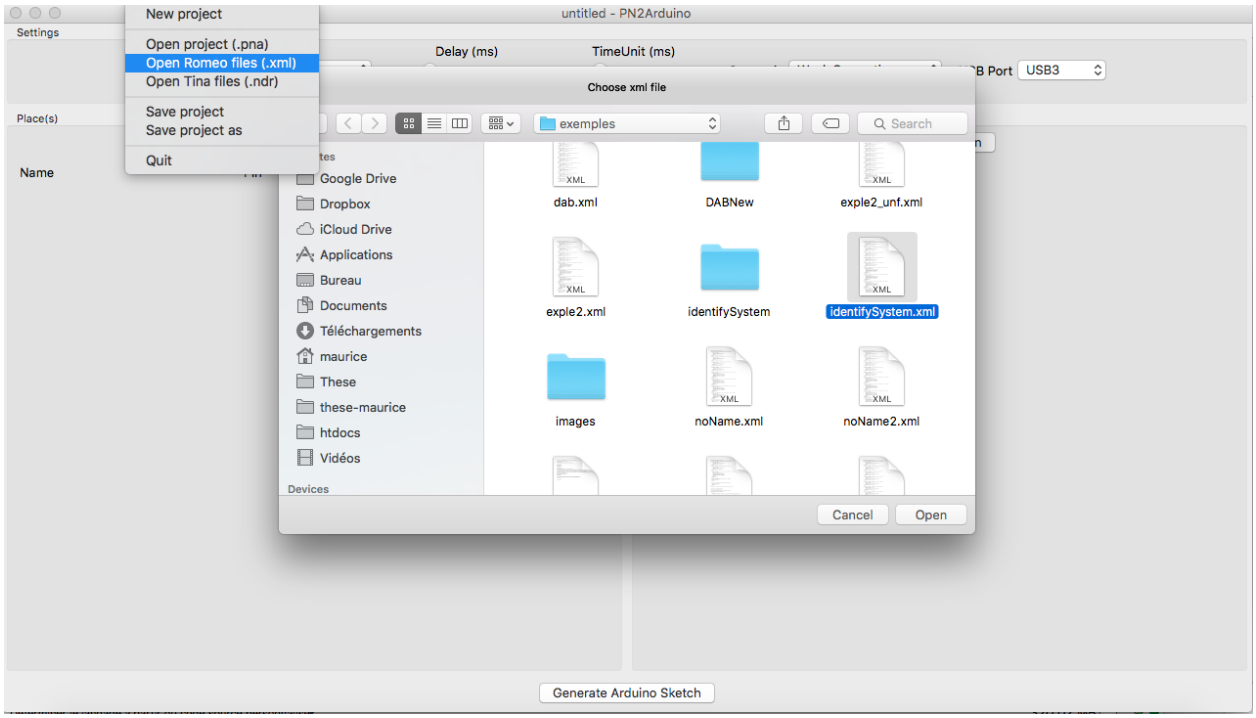


FIGURE C.2 : Ouverture d'un fichier

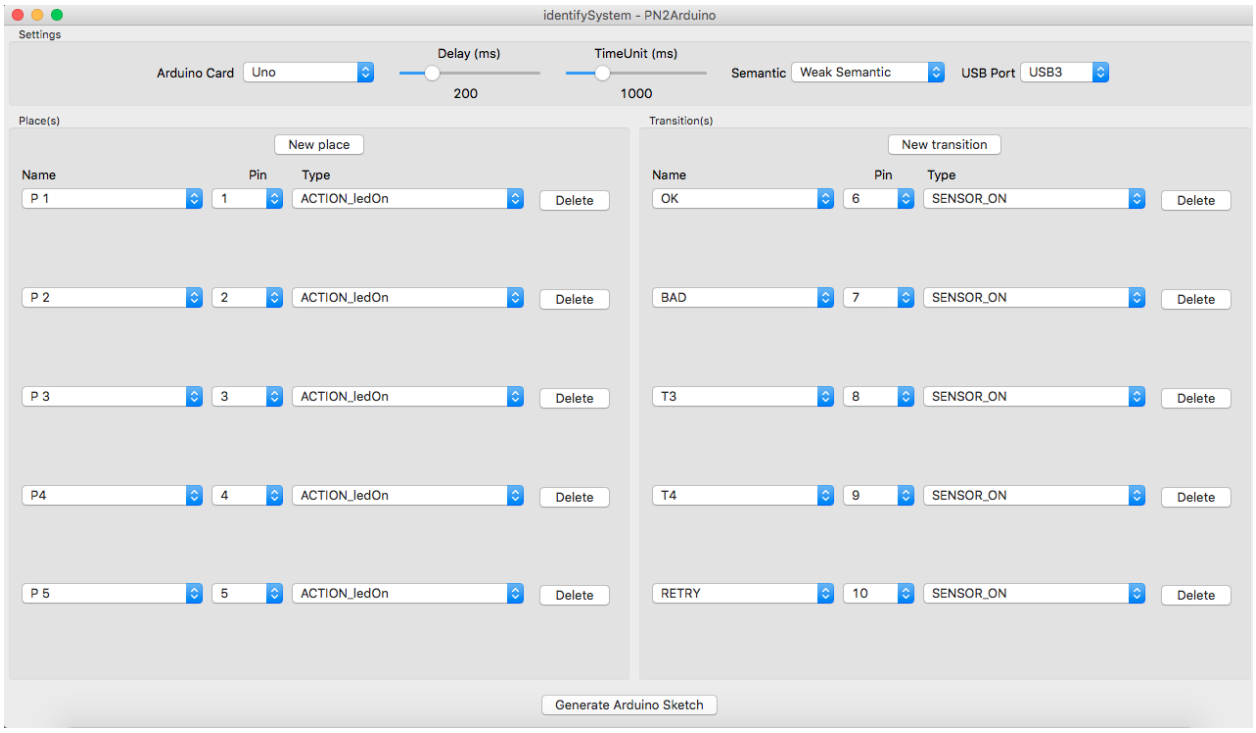


FIGURE C.3 : Assignation des pattes

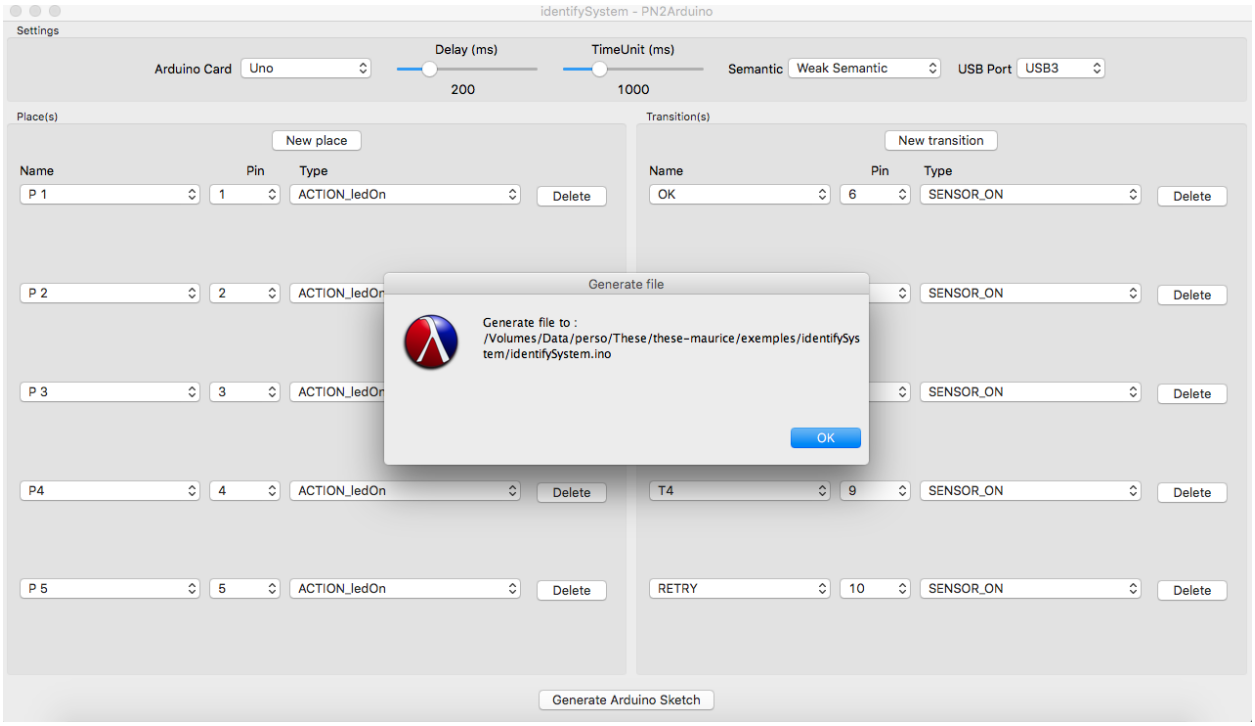


FIGURE C.4 : Génération du croquis Arduino

Bibliographie

[1] Parosh Aziz Abdulla, S. Purushothaman Iyer, and Aletta Nylén. *Computer Aided Verification : 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings*, chapter Unfoldings of Unbounded Petri Nets, pages 495–507. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[2] Choquet-Geniet Annie. *Les réseaux de Petri - Un outil de modélisation*. Dunod, 2006.

[3] Toshiro Araki and Tadao Kasami. Some decision problems related to the reachability problem for petri nets. *Theoretical Computer Science*, 3(1) :85 – 104, 1976.

[4] A Arnold and M Latteux. Recursivite et cones rationnels fermes par intersection. *Calcolo*, 15(4) :381–394, 1978.

[5] André Arnold, Aymeric Vincent, and Igor Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical computer science*, 303(1) :7–34, 2003.

[6] Atmel. Atmel atmega640/v-1280/v-1281/v-2560/v-2561/v, 8-bit atmel microcontroller with 16/32/64kb in-system programmable flash, 2014.

[7] Eric Badouel and Javier Oliver. Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes within Workflow Systems. Research Report RR-3339, INRIA, 1998.

[8] Paolo Baldan, Nadia Busi, Andrea Corradini, and G Michele Pinna. Functional concurrent semantics for petri nets with read and inhibitor arcs. In *CONCUR 2000—Concurrency Theory*, pages 442–457. Springer, 2000.

[9] Paolo Baldan, Andrea Corradini, and Ugo Montanari. Contextual petri nets, asymmetric event structures, and processes. *Information and Computation*, 171(1) :1 – 49, 2001.

[10] Albert Benveniste, Eric Fabre, Stefan Haar, and Claude Jard. Diagnosis of asynchronous discrete-event systems : a net unfolding approach. *Automatic Control, IEEE Transactions on*, 48(5) :714–727, 2003.

[11] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE transactions on software engineering*, 17(3) :259, 1991.

[12] Bernard Berthomieu and François Vernadat. State class constructions for branching analysis of time petri nets. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 442–457. Springer, 2003.

[13] Eike Best, Raymond Devillers, Astrid Kiehn, and Lucia Pomello. Concurrent bisimulations in petri nets. *Acta Informatica*, 28(3) :231–264, 1991.

[14] Tommaso Bolognesi, Ferdinando Lucidi, and Sebastiano Trigila. From timed petri nets to timed lotos. In *Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol Specification, Testing and Verification X*, pages 395–408, Amsterdam, The Netherlands, The Netherlands, 1990. North-Holland Publishing Co.

[15] Christos G Cassandras and Stephane Lafortune. *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[16] Thomas Chatain and Claude Jard. Time supervision of concurrent systems using symbolic unfoldings of time Petri nets. In Paul Pettersson and Wang Yi, editors, *Proceedings of the 3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 196–210, Uppsala, Sweden, November 2005. Springer.

[17] Thomas Chatain and Claude Jard. Complete finite prefixes of symbolic unfoldings of safe time petri nets. In Susanna Donatelli and P.S. Thiagarajan, editors, *Petri Nets and Other Models of Concurrency - ICATPN 2006*, volume 4024 of *Lecture Notes in Computer Science*, pages 125–145. Springer Berlin Heidelberg, 2006.

[18] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[19] Jean-Michel Couvreur. *Contribution à l'algorithmique de la vérification*. PhD thesis, Université Bordeaux I, 2004.

[20] David Delfieu, Maurice Comlan, and Medesu Sogbohossou. Algebraic analysis of branching processes. In *Sixth International Conference on Advances in System Testing and Validation Lifecycle*, pages 21–27, 2014. Best paper award.

[21] David Delfieu, Maurice Comlan, and Medesu Sogbohossou. Conflict Equivalence of Branching Processes. *International Journal On Advances in Systems and Measurements*, 8(1&2) :85–203, 2015.

[22] Jörg Desel, Gabriel Juhás, and Christian Neumair. *Applications and Theory of Petri Nets 2004 : 25th International Conference, ICATPN 2004, Bologna, Italy, June 21–25, 2004. Proceedings*, chapter Finite Unfoldings of Unbounded Petri Nets, pages 157–176. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[23] Michel Diaz and Patrick Sénac. Time stream petri nets a model for timed multimedia information. In *International Conference on Application and Theory of Petri Nets*, pages 219–238. Springer, 1994.

[24] Catherine Dufourd, Petr Jančar, and Philippe Schnoebelen. Boundedness of reset p/t nets. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Proceedings of the 26th ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310, Prague, Czech Republic, July 1999. Springer.

[25] Best Eike and Ferandez C. Cesar. *Nonsequential Processes*. A Petri Net View, 1988.

[26] Joost Engelfriet. Branching processes of petri nets. *Acta Informatica*, 28(6) :575–591, 1991.

[27] David de Frutos Escrig, Valentín Valero Ruiz, and Olga Marroquín Alonso. Decidability of properties of timed-arc petri nets. In *21st International Conference, ICATPN, Denmark*, pages 187–206, 2000.

[28] Javier Esparza. Model checking using net unfoldings. *Science of Compututer Programming*, 23(2-3) :151–195, 1994.

[29] Javier Esparza and Keijo Heljanko. Unfoldings - a partial-order approach to model checking. *EATCS Monographs in Theoretical Computer Science*, 2008.

[30] Javier Esparza, Stefan Römer, and Walter Vogler. *An Improvement of McMillan's Unfolding Algorithm*. Mit Press, 1996.

[31] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of mcmillan's unfolding algorithm. *Formal Methods in System Design*, 20(3) :285–310, 2002.

[32] Jean Fanchon and Rémi Morin. Regular sets of pomsets with autoconcurrency. In *Proceedings of the 13th International Conference on Concurrency Theory*, CONCUR '02, pages 402–417, London, UK, UK, 2002. Springer-Verlag.

[33] M. Felleisen, R.B. Findler, and M. Flatt. *Semantics Engineering With PLT Redex*. Mit Press, 2009.

[34] Alain Finkel. *Advances in Petri Nets 1993*, chapter The minimal coverability graph for Petri nets, pages 210–243. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[35] Alain Finkel and Philippe Schnoebelen. Fundamental structures in well-structured infinite transition systems. In *Latin American Symposium on Theoretical Informatics*, pages 102–118. Springer, 1998.

[36] Matthew Flatt and Robert Bruce Findler. Guide : Racket. 2010.

[37] Hubert Garavel. Défense et illustration des algèbres de processus. *Actes de l'Ecole d'été Temps Réel ETR*, 2003.

[38] Michael R Gary and David S Johnson. Computers and intractability : A guide to the theory of np-completeness, 1979.

[39] Jean Yves Girard. Linear logic. *Theorical Computer Science*, 50, 1987.

[40] S. Hickmott, J. Rintanen, S. Thiebaux, and L. White. Planning via Petri net unfolding. In *Proc. of 20th Int. Joint Conference on Artificial Intelligence*, pages 1904–1911, 2007.

[41] Charles Antony Richard Hoare. *Communicating sequential processes*, volume 178. Prentice-hall Englewood Cliffs, 1985.

[42] ISO/IEC. LOTOS - A formal description technique based on the temporal ordering of observational behaviour. *International Standard n° 8807, International Organisation for Standardization - Information Processing Systems - Open Systems Interconnection, Geneva*, 1989.

[43] Claude Jard. Synthesis of distributed testers from true-concurrency models of reactive systems. *Information and Software Technology*, 45(12) :805–814, 2003.

[44] Billington Jonathan. *Extensions to coloured Petri nets and their applications to protocol*. PhD thesis, PhD thesis, University of Cambridge, UK, May 1991.

[45] Neil D Jones, Lawrence H Landweber, and Y Edmund Lien. Complexity of some problems in petri nets. *Theoretical Computer Science*, 4(3) :277–299, 1977.

[46] Kari Kähkönen, Olli Saarikivi, and Keijo Heljanko. Unfolding based automated testing of multithreaded programs. *Automated Software Engineering*, 22(4) :475–515, 2015.

[47] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2) :147 – 195, 1969.

[48] Wael Khansa. *Reseaux de Petri p-temporels contribution a l'étude des systèmes à événements discrets*. PhD thesis, Universit de Chambry, FRANCE, 1997.

[49] Wael Khansa, Jean-Paul Denat, and Simon Collart-Dutilleul. P-time petri nets for manufacturing systems. In *International Workshop on Discrete Event Systems, WODES*, volume 96, pages 94–102, 1996.

[50] Victor Khomenko, Maciej Koutny, and Walter Vogler. Canonical prefixes of petri net unfoldings. *Acta Informatica*, 40(2) :95–118, 2003.

[51] Jean Krivine. *Reversible Process Algebra*. PhD thesis, Université Pierre et Marie Curie - Paris VI, November 2006.

[52] Charles Lakos and Søren Christensen. *Application and Theory of Petri Nets 1994 : 15th International Conference Zaragoza, Spain, June 20–24, 1994 Proceedings*, chapter A general systematic approach to arc extensions for coloured Petri Nets, pages 338–357. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[53] Jérôme Leroux. The general vector addition system reachability problem by Presburger inductive invariants. *Logical Methods in Computer Science*, 6(3), 2010.

[54] Ben Li, Manel Khlif-Bouassida, and Armand Toguyéni. Diagnosticabilité de Réseaux de Petri Labellisés basée sur les explications minimales et les T-semiflots. In Stephan Merz and Jean-François Pétin, editors, *Modélisation des Systèmes Réactifs (MSR 2015)*, Nancy, France, 2015.

[55] Traonouez Louis-Marie. *Vérification et dépliages de réseaux de Petri temporels paramétrés*. PhD thesis, Université de Nantes, [S.l.], 2009.

[56] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3) :441–460, 1984.

[57] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4) :184–195, 1960.

[58] Kenneth L McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Computer Aided Verification*, pages 164–177. Springer, 1993.

[59] Kenneth L McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Computer Aided Verification*, pages 164–177. Springer, 1993.

[60] K.L. McMillan and D.K. Probst. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1) :45–65, 1995.

[61] Philip Meir Merlin. A study of the recoverability of computing systems. *University of California, Irvine*, 1974.

[62] Diaz Michel. *Vérification et mise en oeuvre des réseaux de Petri*. Traité IC2. Hermès Science, Paris, 2003.

[63] Robin Milner. *A calculus of communicating systems*. Berlin ; New York : Springer-Verlag, 1980.

[64] Robin Milner. *Communication and concurrency*, volume 84. Prentice hall New York etc., 1989.

[65] Sogbohossou Médésu. *Contributions à l'analyse ordre partiel quantitative et à l'abstraction modulaire des réseaux de Petri temporels*. PhD thesis, Université de Nantes, [S.l.], 2009.

[66] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part i. *Theoretical Computer Science*, 13(1) :85 – 108, 1981. Special Issue Semantics of Concurrent Computation.

[67] David Park. *Theoretical Computer Science : 5th GI-Conference Karlsruhe, March 23–25, 1981*, chapter Concurrency and automata on infinite sequences, pages 167–183. Springer Berlin Heidelberg, Berlin, Heidelberg, 1981.

[68] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[69] Carl Adam Petri. *Communication with automata*. PhD thesis, PhD thesis, Institut fuer Instrumentelle Mathematik, 1962.

[70] Vaughan Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1) :33–71, 1986.

[71] Chander Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. *Massachusetts Institute of Technology*, 1974.

[72] Klaus Reinhardt. Reachability in petri nets with inhibitor arcs. *Rapport technique WSI-96-30, Wilhelm-Schickard Institut für Informatik, Universität Tübingen*, 1996.

[73] Christophe Reutenauer. *Aspects mathématiques des réseaux de Pétri [i.e. Petri]*. Études et recherches en informatique. Masson, Paris, Milan, Barcelone, 1989.

[74] Médésu Sogbohossou. *Contributions à l'analyse ordre partiel quantitative et à l'abstraction modulaire des réseaux de Petri temporels*. PhD thesis, Nantes, 2009.

[75] J. A. Stankovic. Misconceptions about real-time computing : a serious problem for next-generation systems. *Computer*, 21(10) :10–19, Oct 1988.

[76] Robert Valette. Les réseaux de petri. *LAASCNRS Toulouse, Septembre*, 2000.

[77] Antti Valmari. A stubborn attack on state explosion. *Form. Methods Syst. Des.*, 1(4) :297–322, December 1992.

[78] Rob van Glabbeek and Frits Vaandrager. Petri net models for algebraic theories of concurrency. In *PARLE Parallel Architectures and Languages Europe*, pages 224–242. Springer, 1987.

[79] Walter Vogler, Alex Semenov, and Alex Yakovlev. *CONCUR'98 Concurrency Theory : 9th International Conference Nice, France, September 8–11, 1998 Proceedings*, chapter Unfolding and finite prefix for nets with read arcs, pages 501–516. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[80] Bernd Walter. Timed petri-nets for modelling and analyzing protocols with real-time characteristics. In *Protocol Specification, Testing, and Verification*, pages 149–159, 1983.

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Maurice COMLAN

Contribution au dépliage des réseaux de Petri et à l'analyse des processus de branchement.

Contribution to the unfolding of Petri nets and to the analysis of branching process.

Résumé

Les réseaux de Petri et leurs extensions constituent un formalisme très connu pour la modélisation de la spécification des systèmes à évènements discrets temps réel. Pour ces systèmes, les exigences de vérification et de validation sont indispensables pour garantir leur bon fonctionnement car la moindre erreur peut entrainer des conséquences catastrophiques. Pour analyser le comportement du système, il est courant de construire le graphe d'états (ou espace d'états) qui énumère de façon exhaustive les différents états accessibles. Ce graphe permet de vérifier des propriétés génériques comme le caractère borné, l'accessibilité, la terminaison, la vivacité, l'absence de blocage, etc. Mais la construction de cet espace d'états est confrontée au problème d'explosion combinatoire du nombre d'états lié à la complexité et à la concurrence du système. L'une des alternatives pour contenir cette combinatoire est de conserver uniquement les ordres partiels entre les évènements. Le dépliage est une technique d'ordre partiel adaptée à la vérification, au diagnostic et à la planification. Cette thèse contribue à l'étude du dépliage des réseaux de Petri et, plus particulièrement, apporte une contribution au dépliage des réseaux de Petri avec des arcs de reset. De plus, le dépliage explicite les exécutions possibles du système qui sont, par définition, des processus. Il s'agit plus précisément des processus de branchement qui diffèrent des processus classiques. Ainsi, nous proposons une algèbre adaptée aux processus de branchement issus du dépliage des réseaux de Petri.

Mots clés

Réseaux de Petri, arcs de reset, dépliage, processus de branchement, algèbre de processus.

Abstract

Petri nets and their extensions are a well-known formalism for modeling the specification of discrete events systems real-time. For these systems, the demands of verification and validation are essential to ensure their smooth functioning because the least error can lead to catastrophic consequences. To analyze the behavior of the system, it is common to construct the state graph (or state space) which lists exhaustively the different accessible states. This graph allows to check the generic properties such as boundedness, accessibility, termination, vivacity, the absence of blocking, etc. But building this state space is confronted with the problem of combinatorial explosion of the number of states related to the complexity and concurrence of the system. One of the alternatives to contain this combination is to keep only partial orders between events. Unfolding is a partial order adapted to technical verification, diagnosis and planning. This thesis contributes to the study of the unfolding of Petri nets and, in particular, his contributes to the unfolding of Petri nets with reset arcs. Moreover, the unfolding explains the possible executions of the system which are, by definition, the processes. This is specifically the branching processes that differ from conventional processes. Thus, we propose an algebra adapted to the branching process from unfolding of Petri nets.

Key Words

Petri net, reset arcs, unfolding, branching process, algebraic of branching process.