UNIVERSITÉ PARIS 13, SORBONNE PARIS CITÉ

ÉCOLE DOCTORALE GALILÉE

THÈSE

Présentée par

Naïm Aber

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

SPÉCIALITÉ : INFORMATIQUE

Abstraction et Analyse de l'Espace des États des Réseaux de Petri Temporels

soutenue publiquement le 12 juillet 2013

Membres du jury :

Directeurs de thèse :		
	Kaïs KLAI	LIPN, Université Paris 13
	Laure PETRUCCI	LIPN, Université Paris 13
Président :		
	Christophe FOUQUERÉ	LIPN, Université Paris 13
Rapporteurs :		
	Béatrice BÉRARD	LIP6, Université Pierre et Marie Curie
	Franck POMMEREAU	IBISC, Université d'Évry
Examinateur :		
	Fabrice KORDON	LIP6, Université Pierre et Marie Curie

Résumé :

Les réseaux de Petri temporels sont largement utilisés pour la spécification des systèmes où le temps est exprimé explicitement. Le principal défi pour l'analyse de tels systèmes est de pouvoir construire une abstraction finie de l'espace d'états qui préserve des familles de propriétés que l'on veut vérifier. De plus, les algorithmes de construction doivent être optimaux en termes de temps de calcul et d'espace mémoire. Notre première contribution dans cette thèse est un nouveau graphe appelé Graphe des Agrégats Temporisés (TAG) qui permet l'abstraction du comportement des réseaux de Petri temporels bornés avec une sémantique de tir forte. Le principal apport de notre approche, comparée aux approches existantes, est l'encodage des informations temporelles. En effet, chaque nœud du TAG comporte des informations temporelles permettant de calculer les délais minimum et maximum écoulés sur chaque chemin du graphe. Le TAG est fini pour les réseaux de Petri temporels bornés, et permet de préserver les séquences et les marquages accessibles ainsi que la vérification des propriétés sur les événements ou sur les états. Nous proposons dans la deuxième contribution des algorithmes pour la vérification des formules TCTL et prouvons leur correction.

Mots clés : Réseaux de Petri Temporels, Abstraction de l'Espace d'États, Vérification de Propriétés Temporelles.

Abstract :

Time Petri nets (TPN models) allow the specification of real-time systems involving explicit timing constraints. The main challenge of the analysis of such systems is to construct, with as few resources (time and space), a coarse abstraction preserving timed properties. We propose a new finite graph, called Timed Aggregate Graph (TAG), abstracting the behaviour of bounded TPNs with strong time semantics. The main feature of this abstract representation compared to existing approaches is the encoding of the time information. This is done in a pure way within each node of the TAG allowing to compute the minimum and maximum time elapsed in every path of the graph. The TAG preserves runs and reachable states of the corresponding TPN and allows for verification of both event- and state-based properties. In the second contribution, we propose the appropriate algorithms for checking TCTL formulae and prove the correction of these algorithms.

Keywords : Time Petri Nets, State Space Abstraction, Model Checking of Timed Properties.

Remerciements

Mes remerciements vont à l'ensemble des personnes qui ont permis l'aboutissement de cette thèse.

En premier lieu, je souhaite remercier Béatrice BÉRARD et Franck POMMEREAU qui me font l'honneur d'être les rapporteurs de cette thèse, ainsi que Christophe FOUQUERÉ et Fabrice KORDON pour leur participation au jury de ma thèse.

Mes remerciements les plus vifs vont à ma directrice de thèse Laure PETRUCCI, qui a énormément compté dans la réussite de cette thèse. Pendant ces trois années, elle a su me guider sur le chemin adéquat et apporter des réponses à mes nombreuses interrogations.

Je ne sais pas si des remerciements seront suffisants pour Kais KLAI, directeur de ma thèse, sans qui cette thèse ne serait pas. Je tiens à le remercier pour ses compétences, son soutien et ses conseils à chaque fois que j'y avais recourt. Je lui suis gré de l'enthousiasme qu'il a manifesté pour mes recherches même lorsque le mien faillissait.

Je tiens à remercier tout particulièrement le directeur de l'école doctorale Vincent LAURENT, qui par son soutien et ses conseils, m'a aidé à mener à bout mes recherches.

Cette thèse a été réalisée au Laboratoire d'Informatique de Paris-Nord (LIPN). Je remercie donc l'ensemble des membres du laboratoire pour leurs conseils avisés au cours de ces trois années. Je remercie particulièrement Sami EVANGELISTA pour la collaboration enrichissante que nous avons eue et pour son aide scientifique. Un grand Merci à Karima MOUHOUBI, Hanane TAFAT BOUZIF, Hanane ALLAOUA, Amine CHAIBI, Hanane OCHI et Hayat CHABALLAH pour leur partage de l'amitié et des connaissances. Merci à tous les Thésards (ou ancien thésards) et à tous permanents avec qui j'ai partagé cette aventure.

Pour terminer, je remercie ma famille et mes amis pour leur soutien moral et leur accompagnement depuis toujours. Merci en particulier à mes parents qui m'ont donné un goût certain pour les études et m'ont permis d'arriver jusque là. Merci à mes frères et sœurs qui m'ont continuellement encouragé.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1.1 Contexte scientifique et problématique

L'évolution des systèmes informatiques (en termes de taille, de complexité, de modularité, d'interaction, etc) pose de nombreux défis dans leurs conceptions, réalisations et vérifications. Les systèmes temps-réel, que l'on trouve dans de multiples secteurs d'activités : contrôle des systèmes automatisés de production, aide à la conduite des véhicules (en particulier dans le domaine de l'aviation), gestion des flux d'information dans les réseaux, etc; en sont de parfaits exemples. Les systèmes temps-réel se sont progressivement établis comme une discipline à part entière et rassemblent une forte communauté issue à la fois du monde académique et de l'industrie. L'évolution de ces systèmes se caractérise par une complexité croissante, un rôle toujours plus critique et des contraintes temporelles strictes. Leur mise au point est un processus complexe. Pour cela, il est fortement recommandé d'utiliser d'une part des techniques de spécification formelle afin de décrire sans ambiguïté le comportement des applications considérées, et d'autre part, des techniques de vérification formelle afin de valider le bon fonctionnement de ces applications. Les étapes de vérification sont primordiales dans le processus de conception de tels systèmes, car elles permettent de s'assurer que le système ne rencontre pas de défaillances (des comportements non attendus) et qu'il remplit correctement ses spécifications. La vérification des systèmes temps-réel demeure toujours complexe du fait du nombre et de la complexité des paramètres pris en compte lors de leur analyse. Un défi, qui reste difficile à l'heure actuelle, est d'analyser ces systèmes en prenant en compte explicitement le temps. Ainsi, l'existence de contraintes sur le temps induit, en général, un nombre infini de configurations (surtout dans le cas où le temps est considéré comme dense). Afin d'intégrer

ces contraintes avec une garantie d'obtenir un système plus sûr, il est nécessaire d'utiliser des méthodes formelles. Pour cela, deux grandes classes de modèles temporisés ont été proposés : les automates temporisés [5] et les réseaux de Petri avec du temps [34, 43]. Dans cette thèse, nous nous intéressons aux systèmes modélisés par une extension temporelle des réseaux de Petri : les réseaux de Petri t-temporels [35]. Ce modèle est une classe de réseaux de Petri classiques où un intervalle est associé à chaque transition. Cet intervalle représente la période durant laquelle la transition reste franchissable. Notre choix pour ce modèle est motivé par le fait qu'il représente un moyen efficace pour modéliser les différents aspects des systèmes temps-réel. De plus, il a été démontré que ce modèle est plus expressif que d'autres extensions temporisées de réseaux de Petri [42].

1.2 Contributions

L'espace des états accessibles d'un réseau de Petri t-temporel est en général infini. Pour remédier à ce problème, des méthodes symboliques ont été proposées et utilisées pour l'analyse des réseaux t-temporels et pour la vérification de propriétés temporelles. La plupart des ces méthodes sont basées sur le graphe des classes d'états [8] et le graphe des zones [18]. Dans cette perspective, nous proposons une nouvelle technique de vérification de systèmes temps réel. Notre principal défi est donc de construire, avec le moins de ressources possible (en temps et espace mémoire), une abstraction finie préservant les propriétés temporelles. L'approche proposée est basée sur un graphe que nous avons appelé Graphe des Agrégats Temporisés (en anglais, Timed Aggregate Graph – TAG). Le TAG fournit une représentation exacte de l'espace des états accessibles des TPNs, et permet l'analyse de l'accessibilité et la vérification de propriétés temporisées. Le TAG fournit ainsi une technique alternative efficace aux méthodes d'analyse existantes.

1.3 Organisation du rapport

Le manuscrit est organisé comme suit : Le chapitre 2 présente les concepts liés à la vérification formelle, aux modèles temporisés (automates temporisés et réseaux de Petri avec du temps) et aux logiques utilisées dans les systèmes temporisés. Nous nous attardons sur le modèle des réseaux de Petri t-temporels, qui constitue le modèle de référence pour nos contributions. Dans le chapitre 3, nous passons en revue les méthodes existantes pour le calcul et l'analyse l'espace des états des réseaux de Petri t-temporels. Nous présentons principalement les méthodes basées sur le graphe des classes d'états et le

graphe des zones. Dans le chapitre 4, nous présentons notre approche, appelée Graphe des Agrégats Temporisés (TAG pour Timed Aggregates Graph), pour l'abstraction de l'espace des états des réseaux de Petri t-temporel. Nous donnons la définition du graphe, présentons sa méthode de construction et montrons qu'il est fini pour les réseaux de Petri t-temporels. Pour clôturer ce chapitre, nous présentons les résultats de préservation et rapportons les résultats des expérimentations. Dans le chapitre 5, nous proposons des algorithmes basés sur la TAG pour la vérification des formules de la logique TCTL et prouvons leur convergence. Enfin, dans le chapitre 6, nous concluons en anticipant sur les perspectives.

Chapitre 2

Préliminaires et notions de base

Sommaire

2.1 Introduction

La conception d'un système informatique commence par la rédaction d'un cahier des charges dans lequel doivent être écrites toutes les spécifications du système. Ces spécifications sont susceptibles d'être incomplètes, voire contradictoires. Les concepteurs du système sont chargés de détecter les erreurs

introduites au cours de chacune des étapes du cycle de vie du système. Or, la complexité croissante de ces systèmes rend la détection des erreurs de plus en plus ardue. Il est donc nécessaire d'utiliser des méthodes précises (c'est-à-dire mathématiques) permettant de faire le lien entre les spécifications informelles et le programme final. Et c'est exactement le rôle des méthodes formelles. C'est-à-dire, garantir que le système soit sûr et cela à partir du cahier des charges rédigé en langage naturel jusqu'au programme final rédigé dans un langage de programmation.

Ce chapitre a pour but d'introduire les techniques, modèles et logiques utilisés dans le contexte de la vérification formelle de systèmes. Nous n'allons pas détailler tous ces modèles, mais des présentations détaillées pourront être trouvées dans les références citées. Ainsi, sur la base de ces descriptifs, nous justifierons notre choix d'adopter le modèle de réseaux de Petri t-temporels [34, 35]. La deuxième partie de ce chapitre est entièrement réservée à ce formalisme. Nous rappelons d'abord des notions sur les réseaux de Petri classiques (Place/Transition). Ensuite, nous présentons le modèle t-temporel en donnant sa définition formelle, sa sémantique (la sémantique de tir forte) et quelques exemples. Enfin, nous introduisons la logique TCTL avec sa syntaxe et sa sémantique.

2.2 Vérification formelle

L'objectif des méthodes formelles est de modéliser formellement des systèmes informatiques et de vérifier qu'ils répondent à leurs spécifications. Ces méthodes sont classées en trois grandes catégories : le test, la preuve de théorèmes et le model-checking.

2.2.1 Tests

Le test consiste à générer, de manière automatique, des scénarios à partir des spécifications formelles. Le système n'est pas testé de manière exhaustive, car les tests sont une procédure de vérification partielle ayant pour but d'identifier le plus grand nombre possible de comportements problématiques. C'est la méthode la moins coûteuse et la plus facile à mettre en œuvre, et donc la plus répandue. Cependant, elle est limitée par l'impossibilité de tester de manière exhaustive le système.

Parmi les outils permettant l'automatisation des tests, on peut citer notamment *JUnit* [16]. JUnit est un Frame-work de test unitaire pour le langage de programmation *Java* et définit deux types de fichiers de tests : les *Test-Case*, qui servent généralement à tester le bon fonctionnement d'une classe,

et les *Test-Suite* qui permettent d'exécuter un certain nombre de Test-Case déjà définis.

2.2.2 Preuve de théorème

La preuve de théorème est une méthode formelle qui, à l'aide d'outils logiciels appelés *assistants de preuve*, tente de démontrer des théorèmes sur des formules logiques. Beaucoup de projets d'assistants de preuve ont été proposés, parmis lesquels on peut citer Coq [11, 38] et Isabelle [39] entre autres. La preuve de théorème a l'avantage d'être indépendante de la taille de l'espace des états, et peut donc s'appliquer sur des modèles avec un très grand nombre d'états (et même infini). Néanmoins, son inconvénient est qu'elle nécessite l'intervention et la créativité de l'utilisateur pour compléter la preuve, ce qui est parfois un processus assez laborieux. Un autre inconvénient est son incapacité à fournir un contre-exemple lorsque la preuve échoue.

2.2.3 Model-checking

Le model-checking consiste à analyser exhaustivement l'évolution du système lors de ses exécutions possibles. Il s'agit d'une forme de test informatique exhaustif à l'aide d'algorithmes astucieux appelés *model-checkers*. Ces algorithmes permettent d'énumérer et d'analyser tous les états que le système peut atteindre. En général, il n'est pas possible d'analyser directement le système, mais l'analyse se fait plutôt sur un modèle formel, plus ou moins abstrait par rapport au systéme informatique réel. Pour ce faire, il existe plusieurs modèles formels, les automates [25] et les réseaux de Petri [41] en sont des exemples parmi d'autres. Le model-checking a l'avantage d'être automatique et de pouvoir exhiber un exemple de comportement invalidant la propriété à vérifier (un contre-exemple) quand celle-ci est insatisfaite. En contre-partie, ce processus se heurte au problème de l'explosion combinatoire du nombre des états accessibles du système. Par ailleurs, les techniques de model-checking (classiques) sont peu appropriées à des applications directes sur des systèmes paramétrés ou infinis.

2.3 Modèles temporisés

Les systèmes temps-réel sont un domaine d'application privilégié des méthodes formelles. Ceci est dû au fait que ces systèmes présentent des exigences en termes de qualité et de sûreté, chose qui ne peut être apportée que par une approche formelle. Le choix du modèle est alors directement lié aux types de propriétés que l'on souhaite vérifier. Ainsi, avec des modèles classiques,

tels que les automates et les résaux de Petri, on peut décrire le fonctionnement logique du système, et vérifier des propriétés temporelles *qualitatives*. Pour pouvoir vérifier des propriétés temporelles *quantitatives*, il faut avoir recours à des modèles temporisés. Ces derniers ont la capacité de décrire à la fois le fonctionnement logique et les contraintes temporelles quantitatives du système. Les modèles temporisés les plus répandus dans la communauté scientifique sont les automates temporisés [5] et les deux extensions temporelles des réseaux de Petri [43, 34].

2.3.1 Automates temporisés

Les automates temporisés [5] sont une extension des automates d'états finis classiques [24, 44] où le temps est représenté à l'aide de variables explicites appelées *horloges*. Ces horloges évoluent toutes à la même vitesse, peuvent être comparées à des constantes lors des transitions et il est possible de les remettre à zéro.

Soient X un ensemble fini d'horloges prenant leur valeurs dans $\mathbb{R}_{\geq 0}$ et $x \in X$ une horloge. Alors l'ensemble des contraintes d'horloges, noté $C(X)$, est défini par : où $c \in \mathbb{N}$, φ_1 et φ_2 sont des contraintes atomiques.

Définition 2.3.1 (Automates temporisé) *Un* automate temporisé *est un tuple* $\mathcal{A} = \langle S, s_0, X, \Sigma, Inv, ResetT \rangle$ *où :*

- S *est un ensemble fini de places (*localités*) ;*
- $s_0 \subset S$ *est un ensemble de places initiales ;*
- X *est un ensemble fini variables d'horloges ;*
- Σ *est un* alphabet *fini de A ;*
- $Inv : X \to C(X)$ *est un ensemble d'*invariants *de place ;*
- $Reset \in X$ *est le sous ensemble d'horloges à remettre à zéro ;*
- $T \subseteq S \times \Sigma \times C(X) \times X \times S$ *est un ensemble fini de transitions où chaque* $e =< s, a, \varphi, r, s' > \in T$ *correspond à une transition entre la place* s *et la place* s'*, étiquetée par la lettre* a*, gardée par les contraintes de* $\varphi \in C(X)$*, et qui remet à* 0 *les variables de* $Reset \subset X$

Un état d'un automate, appelé aussi *configuration*, est une paire (s, V) où $s \in S$ est une place et $V : X \to \mathbb{R}^X_{\geq 0}$ une fonction qui associe à chaque horloge x une valeur : $v(x)$. Dans une localité d'un automate, il est possible de rester et donc de faire écouler le temps, ou bien de franchir une transition discrète. Les transitions comportent des gardes qui spécifient des contraintes sur les horloges. Le tir d'une transition n'est possible que si ses gardes sont vérifiées et peut réinitialiser certaines horloges. Des invariants peuvent être ajoutés aux localités pour restreindre les durées de séjour.

La Figure 2.1 illustre un exemple d'automate temporisé. L'automate possède deux localités : s_0 (localité initiale) et s_1, deux horloges : x et y, et deux transitions : $s_0 \to s_1$ et $s_1 \to s_0$. La localité s_0 possède un invariant : $y \leq 5$, tandis que s_1 en possède deux : $y \leq 10$ et $y \leq 8$. La transition $s_0 \to s_1$ (respectivement $s_1 \to s_0$) est étiquetée par la lettre a (respectivement b), conditionnée par la garde $y \geq 3$ (respectivement $x \geq 6 \wedge y \geq 4$) et assure l'action de remise à zéro de l'horloge x (respectivement y).

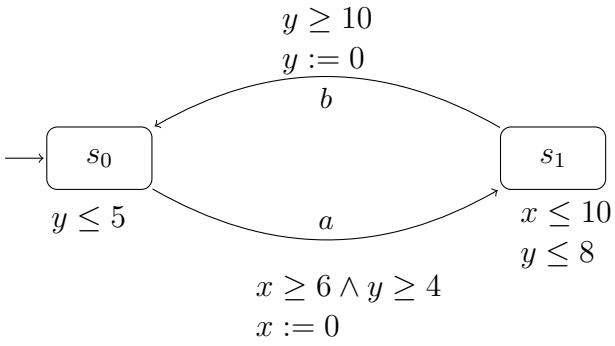


FIGURE 2.1 – Exemple d'un automate temporisé

Afin de représenter un espace fini de l'automate, les états accessibles entre le tir de deux transitions peuvent être regroupés en un seul état symbolique appelé *région* [31]. Les principaux outils manipulant les automates temporisés sont Uppaal [32] et Kronos [48].

2.3.2 Extensions temporelles des réseaux de Petri

Les deux principales extensions temporelles des réseaux de Petri sont : les *réseaux de Petri temporisés* [43] et les *réseaux de Petri temporels* [34, 35]. Les premiers considèrent des dates alors que les seconds ajoutent des intervalles de temps au modèle classique. Pour les deux extensions, le temps peut être intégré de différentes manières, en le plaçant sur les transitions, sur les places, ou sur les arcs [46].

2.3.2.1 Réseaux de Petri temporisés

Les réseaux de Petri temporisés se distinguent en trois grandes classes qui se singularisent par l'objet sur lequel la contrainte temporelle est introduite. Ainsi, les temporisations peuvent être associées : aux transitions (modèle t-temporisé), aux places (modèle p-temporisé) ou bien aux arcs (modèle a-temporisé).

Ramchandani fut le premier à introduire les réseaux de Petri temporisés via le *modèle t-temporisés* dans [43], où il associa une date à chaque transition. Ces dates représentent les durées des opérations à effectuer.

Deux interprétations sont possibles concernant le franchissement d'une transition t : (1) *Date de tir au plus tôt* : une fois que la transition t est sensibilisée, son tir n'a lieu qu'après une durée de d unités de temps. (2) *Date de tir au plus tard* : la transition t devient tout de suite franchissable après son activation et reste ainsi pendant une durée de d unités de temps.

Dans [45], Sifakis propose d'associer des dates aux places du réseau conduisant au *modèle p-temporisé*. Ces dates traduisent le temps de séjour des jetons dans les places avant de franchir des transitions. En général, la décision de tirer une transition n'est pas obligatoire.

Le modèle p-temporisé ne peut pas garantir à la fois des durées de séjour minimum et maximum. Par conséquent, les systèmes où des durées de séjour minimum et maximum sont imposées ne peuvent pas être modélisés efficacement avec les réseaux de Petri p-temporisés.

Des travaux récents apparus dans [2] considèrent des réseaux de Petri à arcs temporisés (*modèle a-temporisé*). Dans ce modèle, chaque jeton possède une horloge représentant son *âge*. Les auteurs supposent que le tir d'une transition peut être retardé même lorsque celle-ci est désensibilisée, c'est-à-dire qu'ils considèrent un comportement non-urgent du réseau.

Les Figures 2.2, 2.3 et 2.4 illustrent un exemple de réseau de Petri temporisé avec les modèles t-temporisé, p-temporisé et a-temporisé respectivement. Si on considère que pour le modèle a-temporisé, les dates sont interprétées au plus tard, alors les exemples sont équivalents.

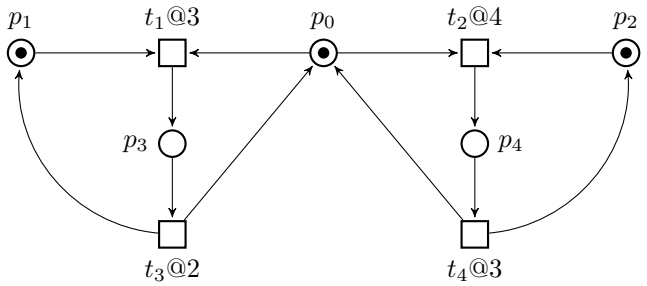


FIGURE 2.2 – Exemple d'un réseau de Petri t-temporisé

2.3.2.2 Réseaux de Petri temporels

Merlin propose dans [34] un modèle de réseaux de Petri pour l'étude des problèmes de recouvrement dans les protocoles de communication. Son

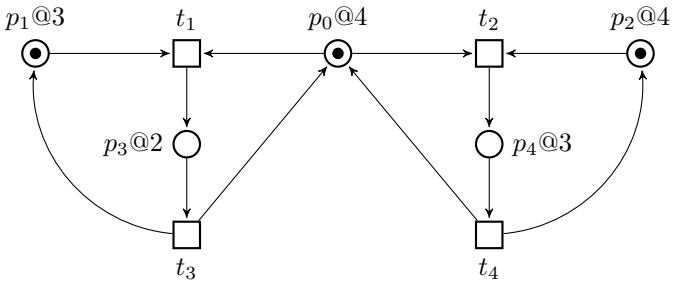


FIGURE 2.3 – Exemple d'un réseau de Petri p-temporisé

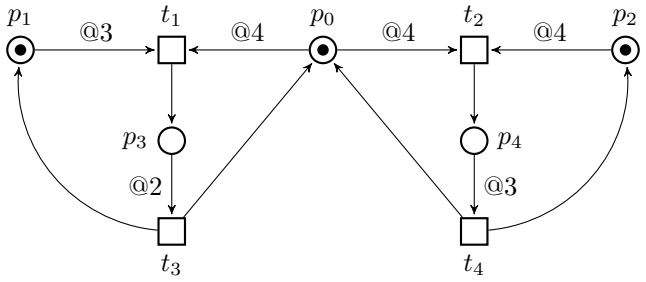


FIGURE 2.4 – Exemple d'un réseau de Petri a-temporisé

modèle, appelé *réseau de Petri t-temporel*, associe à chaque transition une contrainte temporelle de type intervalle de temps, et non pas ponctuelle comme dans le cas du modèle t-temporisé. Un intervalle $[a, b]$ associé à une transition t est relatif à la date de l'activation de t. Si la transition t est validée à l'instant τ, elle peut être tirée dans l'intervalle $a + \tau$ et $b + \tau$ sauf si elle est désensibilisée par franchissement d'une autre transition avec laquelle elle est en conflit. Nous reviendrons en détails sur ce modèle dans le chapitre 2.

Dans *le modèle p-temporel*, proposé dans [26], un intervalle statique $[a, b]$ est associé à chaque place p. Un jeton contenu dans la place p ne participe à la validation des transitions dont p est une place d'entrée que si il a séjourné pendant au moins a unités de temps, et au-delà de b unités de temps, le jeton est considéré comme *mort*, et ne participera donc plus à la validation des transitions.

Les modèles t-temporel et p-temporel de l'exemple précédent sont illustrés respectivement par les Figures 2.5 et 2.6.

En ce qui concerne l'expressivité, les auteurs de [42] ont démontré que les modèles t-temporels et p-temporels sont incomparables. De plus, il a été démontré dans [46] que les modèles t-temporisés et p-temporisés sont équivalents, et sont une sous-classe des réseaux de Petri temporels.

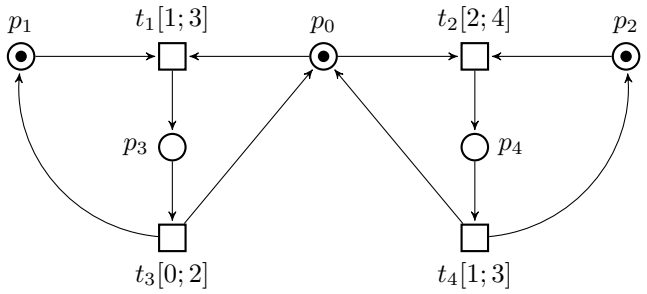


FIGURE 2.5 – Exemple d'un réseau de Petri t-temporel

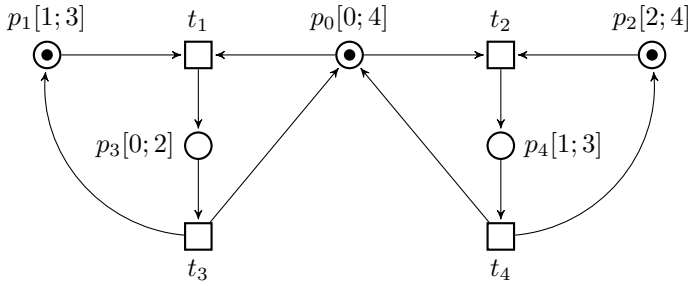


FIGURE 2.6 – Exemple d'un réseau de Petri p-temporel

2.4 Logiques temporisées

Dans [30], les auteurs classent les propriétés logiques en deux catégories principales : Les propriétés de *vivacité* assurent que sous certaines conditions, un événement (respectivement un état) finira par avoir lieu (respectivement être atteint). Les propriétés de *sûreté* expriment le fait qu'un comportement *non-désirable* ne se produira jamais. Ces propriétés sont exprimées soit dans la logique temporelle linéaire (en anglais, Linear Temporal Logic – LTL) ou dans la logique temporelle arborescente (en anglais, Computational Tree Logic – CTL). Cependant, certaines propriétés des modèles temporisés ne peuvent pas être formulées par ces logiques, car on a besoin de faire apparaître des contraintes quantitatives sur le temps. On peut par exemple vouloir énoncer la propriété suivante :
L'alarme se déclenche au plus tard « 35 millisecondes »après l'apparition d'un problème.
Dans cet exemple, la contrainte quantitative est 35 *millisecondes*. Afin de pouvoir intégrer ce genre d'informations, des extensions des logiques temporelles classiques ont été proposées. Une manière de faire est de compléter l'opérateur temporel U (modalité Until) avec une contrainte de la forme "$\sim c$" où

$\sim \in \{=, <, \leq, >, \geq\}$ et $c \in \mathbb{N}$.

Parmi les extensions temporisées de la logique linéaire, on peut citer notamment : Metric Temporal Logic (MTL) [29] qui associe des intervalles à la modalité U, et Metric Interval Temporal Logic (MITL) [6] qui est une restriction de MTL où les intervalles ne sont pas réduits à une unique valeur, c'est-à-dire que les contraintes de la forme « $= c$ »ne sont pas permises.

Pour les extensions temporisées de la logique arborescente, Timed Computational Tree Logic (TCTL) a été proposée dans [23] pour le temps discret et dans [3] pour le temps dense. Nous reviendrons plus en détail sur la logique TCTL dans dans la suite de ce chapitre, où nous donnons sa syntaxe et sa sémantique.

2.5 Réseaux de Petri Place/Transition

En 1962, Carl Adam Petri introduisit pour la première fois les réseaux de Petri dans sa thèse intitulée « Communication avec Automates ». Depuis, les réseaux de Petri [41] ont été largement utililisés notamment grâce à leur capacité à modéliser des systèmes dynamiques concurrents et à prendre en compte aussi bien les états (représentés par des places) que les événements (représentés par des transitions).

2.5.1 Définition

Un réseau de Petri (en anglais, Petri net) est un graphe biparti (composé de deux types de nœuds et dont aucun arc ne relie deux nœuds de même type) orienté (composé d'arcs ayant un seul sens) reliant des places et des transitions (les nœuds) et complété par un marquage initial (le nombre de jetons dans chaque place). Un réseau de Petri peut être présenté comme un système composé de deux parties : la partie statique (structurelle), représentée par un graphe et la partie dynamique (comportementale), qui consiste à faire évoluer le marquage initial en franchissant des transitions.

Définition 2.5.1 (Réseaux de Petri) *Un* réseau de Petri *est un tuple* $\mathcal{S} = \langle P, T, Pre, Post \rangle$ *où :*

- P *est un ensemble fini de places ;*
- T *est un ensemble fini de transitions avec* $P \cap T = \emptyset$ *;*
- $Pre : T \longrightarrow \mathbb{N}^P$ *est l'application d'incidence avant, correspondant aux arcs reliant les places aux transitions ;*
- $Post : T \longrightarrow \mathbb{N}^P$ *est l'application d'incidence arrière, correspondant aux arcs reliant les transitions aux places.*

$P = \{p_1, p_2, \ldots, p_n\}$ et $T = \{t_1, t_2, \ldots, t_m\}$ forment les sommets du réseau où $|P| = n$ et $|T| = m$ $(n, m \in \mathbb{N}_{>0})$ représentent respectivement le nombre de places et le nombre de transitions. La restriction $P \cap T = \emptyset$ interdit les arcs de types place-place ou transition-transition. Les applications d'incidence avant et arrière peuvent être écrites en utilisant une notation matricielle et sont ainsi définies comme suit :

$$Pre = \begin{pmatrix} w_{1,1} & \ldots & w_{1,j} & \ldots & w_{1,m} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ w_{i,1} & \ldots & w_{i,j} & \ldots & w_{i,m} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ w_{n,j} & \ldots & w_{n,j} & \ldots & w_{n,m} \end{pmatrix}$$

et

$$Post = \begin{pmatrix} w'_{1,1} & \ldots & w'_{1,j} & \ldots & w'_{1,m} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ w'_{i,1} & \ldots & w'_{i,j} & \ldots & w'_{i,m} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ w'_{n,j} & \ldots & w'_{n,j} & \ldots & w'_{n,m} \end{pmatrix}$$

où $w_{i,j} = Pre(p_i, t_j)$ et $w'_{i,j} = Post(p_i, t_j)$.

Un *marquage* d'un réseau de Petri $\mathcal{S}$ est une fonction $m : P \longrightarrow \mathbb{N}$ où $m(p)$ dénote le nombre de jetons dans la place p. Un *réseau de Petri marqué* est une paire $\mathcal{R} = \langle \mathcal{S}, m_0 \rangle$ où $\mathcal{S}$ est un réseau de Petri et m_0 son *marquage initial*.

2.5.2 Sémantique d'un réseau de Petri

Une transition t est activée (ou sensibilisée) par un marquage m si et seulement si $m \geq Pre(t)$. Une transition t est dite *franchissable (ou tirable)* à partir un marquage m si et seulement si elle est sensibilisée par m. Si t est franchissable à partir de m, alors son franchissement mène à un nouveau marquage m' tel que : $m' = m - Pre(t) + Post(t)$, on note alors : $m \xrightarrow{t} m'$. Une *exécution* $\sigma = t_1 t_2 \ldots t_k$ représente la séquence de franchissement partant du marquage initial m_0 et conduisant au marquage m_k, c'est-à-dire : $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \ldots \xrightarrow{t_k} m_k$.

L'ensemble (éventuellement infini) des marquages atteignables d'un réseau $\mathcal{R}$ à partir de son marquage initial m_0 est appelé *graphe des marquages accessibles* (ou *graphe d'accessibilité*), on le note $Reach(\mathcal{R})$. Un réseau de Petri marqué $\mathcal{R}$ est dit *vivant* si et seulement si pour tout marquage accessible m, et pour toute transition t, il existe une séquence de franchissements partant

de m et contenant t. S'il existe un marquage m où aucune transition n'est sensibilisée, m est dit *marquage mort* . Une place p est dite *bornée* s'il existe un entier naturel k, tel que pour tout marquage accessible à partir du marquage initial m_0, le nombre de jetons dans p est toujours inférieur ou égal à k. Dans ce cas, on dit que p est *k-bornée*. Un réseau de Petri $\mathcal{R}$ est borné pour un marquage initial m_0 si toutes ses places sont bornées.

2.6 Réseaux de Petri t-temporels

Les réseaux de Petri t-temporels ont été introduits par Philip M. Merlin dans [34]. C'est une extension du modèle Place/Transition qui consiste à ajouter aux transitions des contraintes quantitatives sous forme d'intervalles de temps. Dans le reste du manuscrit, on utilisera l'abréviation TPN (pour Time Petri Net) pour désigner un réseau de Petri t-temporel.

2.6.1 Définition

Un réseau de Petri t-temporel (TPN) est un réseau de Petri Place/Transition où un intervalle temporel $[t_{\min}; t_{\max}]$ est associé à chaque transition t.

Définition 2.6.1 (Définition d'un TPN) *Un* TPN *est un tuple* $\mathcal{N} = \langle P, T, Pre, Post, I\rangle$ *où :*

- $\langle P, T, Pre, Post\rangle$ *est un réseau de Petri P/T ;*
- $I : T \longrightarrow \mathbb{N}\times(\mathbb{N}\cup\{\infty\})$ *est une fonction associant un intervalle de temps* $I(t)$ *à chaque transition* t*, t.q. :* $I(t) = (t_{\min}, t_{\max})$*, avec* $t_{\min} \leq t_{\max}$*, où* $t_{\min}$ *(resp.* $t_{\max}$*) est la date de tir* au plus tôt *(resp.* au plus tard*) de la transition* t*.*

Un *marquage* d'un réseau de Petri $\mathcal{N}$ est une fonction $m : P \longrightarrow \mathbb{N}$ où $m(p)$ dénote le nombre de jetons dans la place p. Un *TPN marqué* est une paire $\mathcal{N} = \langle \mathcal{N}_1, m_0\rangle$ où $\mathcal{N}_1$ est un TPN et m_0 son *marquage initial*.

2.6.2 Sémantique d'un TPN

Une transition t est activée (ou sensibilisée) par un marquage m si et seulement si $m \geq Pre(t)$. L'ensemble des transitions activées par un marquage m, noté $Enable(m)$, est défini par $Enable(m) = \{t \in T : m \geq Pre(t)\}$.

Si une transition t_i est franchissable à partir du marquage m, alors $\uparrow(m, t_i)$ dénote l'ensemble des transitions *nouvellement sensibilisées* [7]. Formellement, $\uparrow(m, t_i) = \{t \in T \mid (m - Pre(t_i) + Post(t_i)) \geq Pre(t) \wedge (m - Pre(t_i)) < Pre(t)\}$.

Si une transition t est dans $\uparrow(m, t_i)$, t est alors dite *nouvellement sensibilisée* pour le marquage obtenu à partir de m par le tir de t_i. De même, $\downarrow(m, t_i) = \{t \in T \mid (m - Pre(t_i) + Post(t_i)) \geq Pre(t) \wedge (m - Pre(t_i)) \geq Pre(t)\}$ est l'ensemble de transitions anciennement sensibilisées. L'ensemble (éventuellement infini) des marquages accessibles du TPN $\mathcal{N}$ est noté $Reach(\mathcal{N})$. $Reach(\mathcal{N})$ est fini si le TPN $\mathcal{N}$ est borné.

Deux sémantiques différentes ont été proposées pour analyser la dynamique des TPNs : la *sémantique de tir forte (en anglais, Strong Time Semantics — STS)* et la *sémantique faible* (en anglais Weak Time Semantics — WTS) [42]. La différence entre ces deux sémantiques réside dans l'urgence de tir des transitions sensibilisées qui ont atteint leurs bornes de tir maximales. En effet, si une transition sensibilisée a atteint la valeur maximale de son intervalle, la sémantique de tir forte force son franchissement. Au contraire, avec une sémantique de tir faible, la transition n'est pas obligatoirement franchie. Dans la suite du manuscrit, nous considérons la sémantique de tir forte du fait de son l'intérêt pour exprimer l'urgence des évènements, ce qui constitue le point le plus critique pour un système temps-réel.

La sémantique d'un TPN peut être exprimée sous forme d'un *Système de Transitions Temporisé (en anglais, Timed Transitions System — TTS)* [31]. Les TTSs constituent un modèle de base pour décrire les systèmes possédant deux types de transitions : des transitions d'actions, décrivant l'évolution discrète, et des transitions de temps, décrivant l'évolution temporelle ou continue.

Définition 2.6.2 (Système de Transitions Temporisé) *Un* Système de Transitions Temporisé *est un quadruplet $\mathcal{S}_{\mathcal{N}} = \langle Q, s_0, \Sigma, \rightarrow \rangle$ où :*

- *Q est un ensemble d'états ;*
- *s_0 est un état initial ;*
- *Σ est un alphabet ;*
- *$\rightarrow$ est une relation de transition incluant les transitions discrètes et les transitions continues, où :*
  - *$\rightarrow \in Q \times \Sigma \times Q$ est une relation de transition discrète ;*
  - *$\rightarrow \in Q \times \mathbb{R}_{>0} \times Q$ est une relation de transition continue.*

Un état d'un TPN défini par un TTS est une paire $s = (m, V)$ où m est un marquage et $V : T \longrightarrow \mathbb{R}_{\geq 0} \cup \{\perp\}$ est une valuation temporelle. Dans la suite de ce manuscrit, $s.m$ et $s.V$ dénotent respectivement le marquage et la valuation temporelle de l'état s. Étant donnés un état $s = (m, V)$ et une transition t, t est franchissable à partir de s si et seulement si $t \in Enable(m) \wedge V(t) \neq \perp \wedge t_{min} \leq V(t) \leq t_{\max}$.

Définition 2.6.3 (Sémantique d'un TPN) *Soit un TPN marqué $\mathcal{N} = \langle P, T, Pre, Post, I, m_0 \rangle$. La sémantique de $\mathcal{N}$ est un TTS $\mathcal{S}_{\mathcal{N}} = \langle Q, s_0, T, \rightarrow \rangle$ où :*

1. *Q est ensemble (éventuellement infini) d'états ;*
2. *$s_0 = (m_0, V_0)$ est l'état initial tel que :* $\forall t \in T,\ V_0(t) = \begin{cases} 0 & si\ t \in Enable(m_0); \\ \perp & sinon; \end{cases}$
3. *$\rightarrow \subseteq Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$est la relation de transition tel que. :*
   (a) *(transition discrète) :* $\forall t \in T :\ (m, V) \xrightarrow{t} (m', V')$ *si seulement si :*

$$\begin{cases} t \in Enable(m) \wedge m' = m - Pre(t) + Post(t); \\ t_{min} \leq V(t) \leq t_{max} \\ \forall t' \in T : V'(t') = \begin{cases} 0 & si\ t' \in \uparrow(m, t); \\ V(t') & si\ t' \in \downarrow(m, t); \\ \perp & sinon. \end{cases} \end{cases}$$

   (b) *(transition continue) :* $\forall d \in \mathbb{R}_{\geq 0}$, $(m, V) \xrightarrow{d} (m', V')$ *si et seulement si :*

$$\begin{cases} \forall t \in Enable(m),\ V(t) + d \leq t_{max} \\ m' = m; \\ \forall t \in T,\ V'(t) = \begin{cases} V(t) + d & si\ t \in Enable(m); \\ \perp & sinon. \end{cases} \end{cases}$$

La sémantique donnée par cette définition est une sémantique le tir forte, c'est-à-dire qu'une transition activée doit être franchie si son horloge a atteint la valeur maximale. Ceci se traduit en termes de TTS par l'impossibilité de faire évoluer le temps avec une transition continue. Les changements d'états se produisent soit par le franchissement d'une transition soit par l'écoulement de temps. Le franchissement d'une transition permet d'atteindre un nouveau marquage, alors que l'écoulement de temps peut de rendre franchissables de nouvelles transitions. Si un marquage bloquant est atteint, seules les transitions de temps seront possibles et le système ne quittera jamais le marquage courant.

La figure 2.7 illustre un exemple d'un TPN marqué, où : $\{p_1, p_2, p_3\}$ est l'ensemble de ses places et $\{t_1, t_2, t_3\}$ l'ensemble des ses transitions. Les intervalles temporels $[2; 4]$, $[0; 2]$ et $[1; 3]$ sont associés respectivement aux transitions t_1, t_2 et t_3. Enfin, le marquage initial est donné par le vecteur $(1, 0, 1)$.

2.6.3 Séquence de franchissement

Soit un TPN $\mathcal{N}$ et soit $\mathcal{S}_\mathcal{N}$ le TTS correspondant. Une exécution $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots$, où $\alpha_i \in (T \cup \mathbb{R}_{\geq 0})$, est une séquence de $\mathcal{S}_\mathcal{N}$ si et seulement si $(s_i, \alpha_i, s_{i+1}) \in \rightarrow$ pour $i = 0, 1, \ldots$. La longueur d'une séquence π est notée

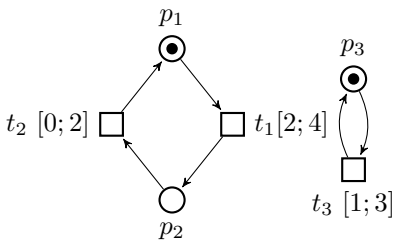


FIGURE 2.7 – Exemple de réseau de Petri t-temporels

$| \pi |$. Par hypothèse, quand deux transitions continues λ_i et λ_{i+1} se succèdent, elles sont agglomérées en une seule transition continue valant $(\lambda_i + \lambda_{i+1})$. L'ensemble (possiblement infini) des séquences de $\mathcal{N}$ est noté $[\mathcal{S}_\mathcal{N}]$.

Soient deux systèmes de transitions temporisées $\mathcal{S}_1$ et $\mathcal{S}_2$. Nous pouvons définir une équivalence de comportement entre $\mathcal{S}_1$ et $\mathcal{S}_2$ par l'intermédiaire d'une relation de bisimulation temporelle. Celle-ci assure que toute action de l'un des deux systèmes peut être simulée par l'autre (et inversement), c'est-à-dire qu'elle a un équivalent dans cet autre système. La relation de bisimulation temporelle est une bisimulation à la fois pour la relation de transition temporelle et pour la relation de transition discrète.

2.7 Exemple d'application

Dans cette section, nous présentons une application des TPNs pour la modélisation d'un systèmes temps-réel. L'application considérée est un système de contrôle d'un procédé de fabrication d'un produit appelé $ProdF$. Ce produit est réalisé par l'assemblage de deux parties $PartA$ et $PartB$ fabriquées respectivement dans les ateliers $AtelierA$ et $AtelierB$. L'assemblage des deux parties est ensuite réalisé dans l'atelier $AtelierC$. Le modèle TPN de ce procédé de fabrication est donné dans la Figure 2.8.

Les modèles t-temporel et p-temporel de l'exemple précédent sont illustrés respectivement par les Figures 2.5 et 2.6.

La partie $PartA$ consiste en deux composants : un premier composant fabriqué par la machine $MacA1$ à partir des matières premières $MatA1$ et $MatA2$; et un deuxième composant fabriqué par la machine $MacA2$ à partir des matières premières $MatA1$ et $MatA3$.

De même, la partie $PartB$ consiste en deux composants : un premier composant fabriqué par la machine $MacB1$ à partir de la matière première $MatB1$; et un deuxième composant fabriqué par la machine $MacB2$ à partir de la matière première $MatB2$.

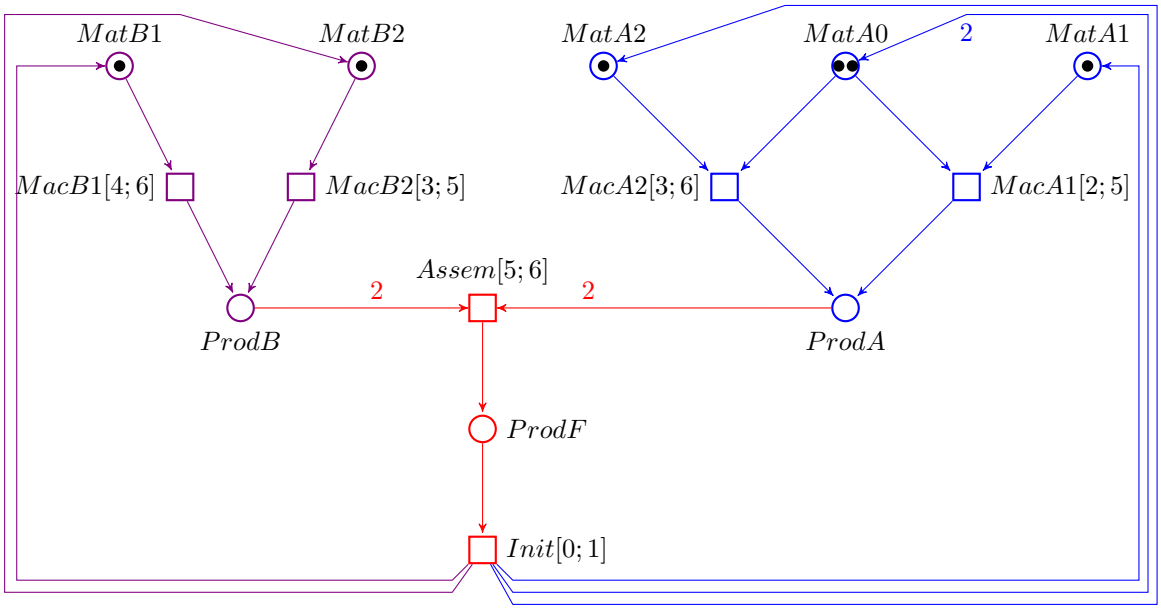


FIGURE 2.8 – Processus de fabrication du produit F

Une fois produites par les ateliers $AtelierA$ et $AtelierB$, les deux parties $PartA$ et $PartB$ sont assemblées par la machine $Assemblage$ de l'$AtelierC$ pour obtenir le produit $ProdF$.

Toutes les matières premières sont préparées à l'avance, c'est-à-dire, avant le début du processus de fabrication. Cependant, une fois que sa préparation est achevée, chaque matière première n'est utilisable qu'à partir d'un certain délai et sa durée de vie est limitée. Pour la partie $PartA$, la matière $MatA1$ est utilisable entre les instants 2 et 5 (unités de temps), $MatA2$ est utilisable entre les instants 3 et 6 et $MatA3$ entre les instants 2 et 6. De même, pour la partie B, les matières $MatB1$ et $MatB2$ sont utilisables respectivement entre 4 et 6 et entre 3 et 5. Enfin, la contrainte temporelle sur la machine $Assemblage$ indique que le processus de l'assemblage du produit $ProdF$ dure entre 5 et 6 unités de temps. Une fois la production terminée, les matières premières utilisées par les ateliers $AtelierA$ et $AtelierB$ sont livrées dans 1 unité de temps au plus tard.

L'utilisation d'un TPN pour modéliser ce systèmes temps-réel a permis de garantir l'exactitude du comportement et le respect des contraintes temporelles. Le modèle TPN ne écrit pas simplement le procédés de fabrication du produit $ProdF$, il décrit aussi le fait que ce produit doit être fabriqué dans des délais imposés. Cet exemple d'application nous permet aussi d'illustrer le fait qu'il est plus facile d'exprimer l'urgence des événements avec une sémantique

de tir forte.

2.8 Logique TCTL

La logique TCTL est une extension de CTL qui intègre des contraintes temporelles quantitatives. Nous considérons une logique TCTL [23] dans laquelle un intervalle temporel est ajouté sur les opérateurs temporels afin d'indiquer des contraintes sur les temps d'exécution. Pour les TPNs, une version basée sur les contraintes d'exclusion mutuelle généralisées (en anglais Generalized Mutual Exclusion Constraints — GMEC) a été proposée dans [12].

Soit l'ensemble des places ($P = \{p_1, p_2, \ldots, p_n\}$) d'un TPN N. Les GMECs sont définies par la grammaire suivante :

$$(\sum_{i=1}^{n} b_i * m(p_i)) \sim c \quad | \quad g_1 \vee g_1 \quad | \quad g_2 \wedge g_2 \quad | \quad g_1 \Rightarrow g_2$$

où $b_i \in \mathbb{Z}$, $c \in \mathbb{N}$, $\sim \in \{<, \leq, =, >, \geq\}$, g_1, g_2 sont des contraintes GMEC et $\vee, \wedge, \Rightarrow$ sont les opérateurs usuels.

Pour un marquage m, $m(p_i) \sim c$ signifie que le marquage courant de la place p_i est en relation"$\sim$" avec c. Les termes $m(p_1) > 0$, $2m(p_1) + m(p_2) = 2$, $(m(p_1) = 1) \Rightarrow (m(p_2) = 0)$, ... sont des exemples de contraintes GMEC.

2.8.1 Syntaxe de TCTL

Soit $N = \langle P, T, Pre, Post, I, m_0 \rangle$ un TPN marqué sur lequel les formules TCTL sont définies.

Définition 2.8.1 (Syntaxe des formules TCTL) *Les formules TCTL sont définies par la grammaire suivante :*

$$\textbf{\textit{false}} \quad | \quad g \quad | \quad \neg\varphi \quad | \quad \varphi \vee \psi \quad | \quad \exists(\varphi U_{[d,D]} \psi) \quad | \quad \forall(\varphi U_{[d,D]} \psi)$$

où g est une contrainte GMEC, $d \in \mathbb{N}$, $D \in \mathbb{N} \cup \{\infty\}$ avec $d \leq D$, φ et ψ sont des formules TCTL.

On remarquera que l'opérateur X, appelée également $Next$, n'est pas représenté dans la définition 2.8.1. En effet, la notion de successeur immédiat n'est pas définie lorsque l'on considère le temps dense.

Pour un marquage m, on note $m \models \varphi$ si la propriété φ est **vraie** et $m \not\models \varphi$ si la propriété ψ est **fausse**.

2.8.2 Sémantique de TCTL

La sémantique de la logique TCTL est donnée par la relation de satisfaction $\models$ sur le système de transitions temporisées $S_N = \langle Q, s_0, \rightarrow \rangle$.

Définition 2.8.2 (Sémantique des formules TCTL) *La* sémantique des formules TCTL *est définie comme suit :*

$(S_N, s_0) \models \varphi$ *si et seulement si* $m \not\models \varphi$

$(S_N, s_0) \models \textbf{\textit{false}}$ *si et seulement si* $(S_N, s_0) \not\models \varphi$

$(S_N, s_0) \models \neg\varphi$ *si et seulement si* $(S_N, s_0) \not\models \varphi$

$(S_N, s_0) \models \varphi \vee \psi$ *si et seulement si* $(S_N, s_0) \models \varphi \vee (S_N, s_0) \models \psi$

$(S_N, s_0) \models \exists(\varphi U_{[d,D]} \psi)$ *si et seulement s'il existe une exécution* $\pi = s_0 \xrightarrow{(d_0,t_0)} s_1 \xrightarrow{(d_1,t_1)} s_2 \ldots \xrightarrow{(d_{n-1},t_{n-1})} s_n$ *dans* S_N, *tel que* $d \leq \sum_{i=o}^{n-1} d_i \leq D$, $s_0 \models \varphi$, $s_1 \models \varphi \ \ldots \ s_{n-1} \models \varphi$ *et* $s_n \models \psi$

$(S_N, s_0) \models \exists(\varphi U_{[d,D]} \psi)$ *si et seulement si pour toute exécution* $\pi = s_0 \xrightarrow{(d_0,t_0)} s_1 \xrightarrow{(d_1,t_1)} s_2 \ldots \xrightarrow{(d_{n-1},t_{n-1})} s_n$ *dans* S_N, *tel que* $d \leq \sum_{i=o}^{n-1} d_i \leq D$, $s_0 \models \varphi$, $s_1 \models \varphi \ \ldots \ s_{n-1} \models \varphi$ *et* $s_n \models \psi$

La sémantique de TCTL est définie sur des arbres de branchement temporisés. La formule $\exists(\varphi U_{[d,D]} \psi)$ spécifie qu'il est possible d'atteindre un état satisfaisant la propriété ψ entre d et D unités de temps et que tous les états intermédiaires satisfont φ (Figure 2.9). La formule $\forall(\varphi U_{[d,D]} \psi)$ spécifie que tous les chemins mènent à un état satisfaisant la propriété ψ entre d et D unités de temps et que tous les états précédants cet état vérifient φ (Figure 2.10).

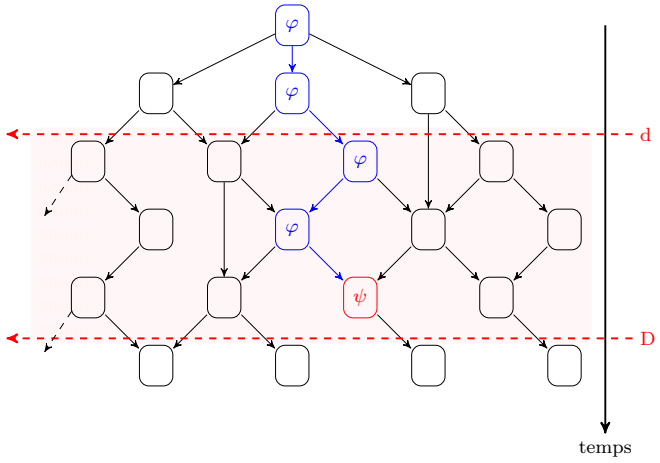


FIGURE 2.9 – Sémantique de la formule $\exists(\varphi U_{[d,D]} \psi)$

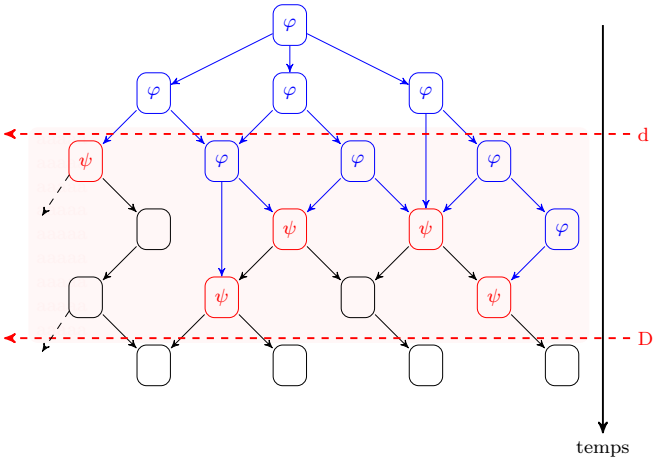


FIGURE 2.10 – Sémantique de la formule $\forall(\varphi U_{[d,D]}\psi)$

2.8.3 Propriétés d'acessibilité temporisée en TCTL

Les propriétés de vivacité, sûreté et d'équité sont exprimables en TCTL de la même manière on qu'en CTL. Les contraintes sur les horloges permettent en plus de spécifier des propriétés d'acessibilité temporisée avec un temps continu. Ci-dessous, nous formulons dans la logique TCTL les proporiétés d'accessibilité usuelles.

$\exists\Diamond_{[d;D]}\varphi$: Il existe un état situé entre d et D unités de temps qui vérifie la propriété φ.

$\forall\Box_{[d;D]}\varphi$: La propriété φ est vérifiée dans tous les états situés entre les instants d et D.

$\forall\Diamond_{[d;D]}\varphi$: Le long de chaque chemin, il est toujours possible d'atteindre un état qui vérifie la propriété φ entre les instants d et D.

$\exists\Box_{[d;D]}\varphi$: Il existe un chemin situé entre d et D unités de temps où tous les états vérifient φ.

Considérons les propriétés Φ_1 et Φ_2 pour le TPN du procédés de fabrication illustré dans la Figure 2.8.

Φ_1 : « Le produit $ProdF$ est toujours délivré dans les délais. »

Φ_2 : « Il est possible que le produit $ProdF$ soit délivré hors des délais »

Les propriétés Φ_1 et Φ_2 peuvent être spécifiées par les formules TCTL suivantes :

Φ_1 : $\forall\Box_{[9;11]}m(ProdF) = 1$.
« Pour toute exécution du TPN, la place $ProdF$ est toujours marquée entre 9 et 11 unités de temps. »

Φ_2 : $\exists\Diamond_{[0;9[} m(ProdF) = 1) \vee (\exists\Diamond_{]11;\infty]} m(ProdF) = 1$.
« Il existe une exécution dans TPN contenant un état où la place $ProdF$ est marquée avant 9 ou après 11 unités de temps. »

Le but du TPN modélisant ce procédés de fabrication est de satisfaire la proporiété Φ_1 mais pas la proporiété Φ_2.

Dans le Chapitre 5, nous présentons des algorithmes de vérification d'accessibilité temporisée et de Model Checking TCTL en se servant du graphe temporisé proposé dans le Chapitre 4. En effet, montrer que l'évaluation d'une formule TCTL sur le TPN est équivalent à sa vérification sur un arbre temporisé constitue l'un des résultats les plus importants de notre thèse. Enfin, grâce à ces algorithmes, nous confirmeront la satisfaction de Φ_1 et la non satisfaction de Φ_2.

2.9 Conclusion

Depuis leur introduction, les réseaux de Petri sont largement utilisés pour la modélisation et l'analyse des systèmes à événements discrets. Rapidement, la nécessité de la prise en compte explicite du paramètre temps a conduit à l'émergence des différents modèles temporisés. C'est ainsi que parmi les modèles temporisés existants, les réseaux de Petri t-temporels ont été introduits pour la modélisation et l'analyse des systèmes temps-réel. En effet, les durées associées aux transitions permettent d'exprimer parfaitement les points critiques de ces systèmes, comme par exemple l'urgence des événements.

Ainsi, dans ce chapitre nous avons d'abord exposé brièvement les principaux modèles temporisés avant de présenter les réseaux de Petri t-temporels comme étant notre modèle de référence. Nous avons alors rappelé la définition formelle d'un réseau de Petri t-temporel ainsi que sa sémantique. Nous avons ensuite exposé quelques notions de base liés à ce modèle.

L'ensemble des états accessibles d'un TPN est généralement infini même lorsque celui-ci est borné. Pour faire face à ce problème, des méthodes symboliques de représentation de l'espace d'états ont été proposées. À cet effet, dans le prochain chapitre nous présentons les différentes méthodes pour la représentation et l'analyse de l'espace d'états des réseaux de Petri t-temporels.

Chapitre 3

Méthodes d'analyse des réseaux de Petri t-temporels

Sommaire

3.1 Introduction

Dans les systèmes temps-réel, le temps est considéré comme dense, par exemple s'il prend ses valeurs dans le domaine $\mathbb{R}_{\geq 0}$. Par conséquent, l'ensemble des états accessibles d'un TPN est généralement infini même lorsque celui-ci

est borné. Ceci est dû au fait qu'un état peut avoir une infinité de successeurs temporels. Des méthodes symboliques de représentation de l'espace d'états sont donc utilisées. Ces méthodes ont pour but de fournir un graphe fini sur lequel on peut vérifier des familles de propriétés. Si $G_{\mathcal{N}}$ désigne l'espace d'états d'un TPN $\mathcal{N}$, Υ une abstraction de $G_{\mathcal{N}}$, et Φ une formule logique de la famille que l'on souhaite préserver, alors, l'objectif est d'avoir $G_{\mathcal{N}} \models \Phi$ si et seulement si $\Upsilon \models \Phi$.

Parmi les travaux existants, se dégagent principalement les approches basées sur les *classes d'états* [8] et celles basées sur le *graphe des régions* [5]. Il existe d'autres méthodes, on peut citer notamment [7, 47, 9, 18, 10, 33, 12, 21], qui sont des raffinements, des améliorations ou encore des dérivées de ces deux principales techniques. Les abstractions existantes se distinguent principalement par le critère d'agglomération des états, le type de propriétés qu'elles préservent et la taille des graphes produits. Dans ce chapitre, nous passons en revue les techniques proposées dans la littérature permettant l'abstraction et l'analyse de l'espace d'états des TPNs. Nous présenterons les principes de fonctionnement des deux méthodes citées ci-dessus. Nous parlerons ensuite des différentes améliorations qui ont été apportées ainsi que les outils qui leur sont dédiés.

3.2 Méthodes basées sur les classes d'états

Le *Graphe des Classes d'États (en anglais, States Class Graph — SCG)* [8] fut la première solution apportée pour l'abstraction de l'espace d'états des TPNs. Les contraintes temporelles sont représentées sous forme d'un système d'inéquations. Le système d'inéquations, appelé *domaine*, porte sur les variables associées aux intervalles de tir des transitions sensibilisées. Une classe d'états est ainsi une paire « marquage-domaine ». Les nœuds du SCG sont des classes et les arcs sont étiquetés par des transitions.

3.2.1 Classe d'états

Soit un ensemble variables $X = \{x_{t_1}, \ldots, x_{t_i}, \ldots, x_{t_n}\}$ associées à un ensemble de transitions $T = \{t_1, \ldots, t_i, \ldots, t_n\}$, où x_{t_i} dénote la variable relative à la transition t_i. Une *classe d'état* est une paire $C = (m, D)$ où m est un marquage et D un *domaine* représenté par un système d'inéquations portant sur les variables associées aux transitions sensibilisées par m.

3.2.2 Classe initiale

La classe *initiale* C_0 est représentée par le marquage initial m_0 et le domaine D_0 qui associe à chaque transition sensibilisée dans m_0 un intervalle temporel. Formellement, $C_0 = (m_0, D_0)$ où :

- m_0 est le marquage initial ;
- D_0 est défini par le système d'inéquations associées aux variables des transitions sensibilisées par m_0, où pour chaque transition $t \in Enable(m_0)$, on ajoute la contrainte $t_{min} \leq x_t \leq t_{max}$.

3.2.3 Classe Successeur

Le calcul de la classe $C' = (m', D')$, successeur de la classe $C = (m, D)$ par le franchissement de la transition t, est effectué comme suit :

- Une transition t est franchissable depuis une classe $C = (m, D)$ si et seulement si :
  - **(i)** La transition t est sensibilisée par le marquage m ;
  - **(ii)** Dans le système D augmenté des contraintes $\forall t' \neq t, \quad x_t \leq x_{t'}$, la transition t est tirée dans son intervalle de tir et peut l'être avant les autres transitions sensibilisées par le marquage m.
- Si la transition t est franchissable depuis la classe $C = (m, D)$, alors la classe $C' = (m', D')$, successeur de C par le tir de t, est obtenue comme suit :
  - $m' = m - Pre(t) + Post(t)$
  - D' est déterminé par les étapes suivantes :
    1. Les conditions (ii) de franchissement de t sont ajoutées au système D ;
    2. Les variables associées aux transitions en conflit avec t sont éliminées ;
    3. Chaque variable $x_{t'}, t' \neq t$, est remplacée par $x_t + x_{t'}$; x_t est ensuite éliminée ;
    4. Pour chaque transition t' nouvellement sensibilisée, on ajoute les contraintes $t'_{min} \leq x_{t'} \leq t'_{max}$.

L'ensemble des solutions du système déterminé à l'étape (3) peut être vu comme le domaine de tir des transitions distinctes de t qui sont restées sensibilisées pendant le tir de t, leur nouvelle origine du temps est la date à laquelle la transition t a été tirée. Les éliminations effectuées dans les étapes (2) et (3) sont obtenues en utilisant la méthode d'élimination de Fourier-Motzkin [13]. Deux classes sont égales si et seulement si leurs marquages et domaines respectifs sont égaux. La comparaison de domaines et la relation d'accessibilité entre

classes d'états permettent de construire le graphe de classes d'états (SCG). Le SCG contient la classe initiale, il y a un arc étiqueté t d'origine C et d'extrémité C' si t est franchissable depuis la classe C et son tir depuis C conduit vers C'. Le nombre de classes d'états est fini si le réseau est borné. Ainsi, il peut être utilisé comme une abstraction des états accessibles pour vérifier des propriétés exprimées dans des logiques temporelles comme CTL ou LTL. Il permet aussi une prise en compte de contraintes temporelles quantitatives lors de la vérification de propriétés. Pour cela, une sous-classe de la logique TCTL a été utilisée dans [22]. Toutefois, dans le graphe des classes, la durée de la séquence de franchissement permettant de passer d'un marquage à un autre ne peut être caractérisée de manière précise. En effet, à chaque franchissement de transition une imprécision est introduite. L'approche basée sur le graphe des classes d'états a été mise en œuvre dans un outil dédié appelé *tina* [1, 10]. Le SCG permet pour les TPNs bornés la vérification des propriétés d'accessibilité ainsi que les propriétés exprimées en CTL* [21] et TCTL [22].

3.2.4 Exemple

Considérons le TPN de la figure 2.7. Son SCG est obtenu comme suit :

- La classe initiale est $C_0 = (m_0, D_0)$, avec :

  $m_0 = (1, 0, 1)$

  D_0 est l'ensemble des solutions en x_{t_1} et x_{t_3} du système

  $2 \leq x_{t_1} \leq 4$

  $1 \leq x_{t_3} \leq 3$

- Le tir de t_1 depuis C_0 à une date $\theta_1 \in [2, 4]$, mène à $C_1 = (m_1, D_1)$ avec :

  $m_1 = (0, 1, 1)$

  D_1 : est calculé en 4 étapes, selon l'algorithme donné ci-dessous :

  Étape (1) : ajout des conditions de franchissabilité de t_1 à D_0 :

  $2 \leq x_{t_1} \leq 4 \qquad x_{t_1} \leq x_{t_3}$

  $1 \leq x_{t_3} \leq 3$

  Étape (2) : Pas de transition en conflit avec t_1.

  Étape (3) : Le changement d'origine produit le système suivant :

  $2 \leq x_{t_1} \leq 4 \qquad 1 \leq x_{t_1} + x_{t_3} \leq 3 \qquad x_{t_1} \leq x_{t_1} + x_{t_3}$

  Par élimination de x_{t_1}, on obtient le système suivant :

  $0 \leq x_{t_3} \leq 1$

  Étape (4) : t_2 est une transition nouvellement sensibilisée.

  $0 \leq x_{t_2} \leq 2$

$$0 \leq x_{t_3} \leq 1$$

– Le tir de t_3 depuis C_0 à une date $\theta_3 \in [1, 3]$, mène à $C_2 = (m_2, D_2)$ avec :

$m_2 = m_0 = (1, 0, 1)$

D_2 : est calculé par les 4 étapes suivantes :

Étape (1) : ajout des conditions de franchissabilité de t_3 à D_0 :

$$2 \leq x_{t_1} \leq 4 \qquad x_{t_3} \leq x_{t_1}$$

$$1 \leq x_{t_3} \leq 3$$

Étape (2) : Pas de transition en conflit avec t_3.

Étape (3) : Le changement d'origine produit le système suivant :

$$1 \leq x_{t_3} \leq 3 \qquad 2 \leq x_{t_3} + x_{t_1} \leq 4 \qquad x_{t_3} \leq x_{t_3} + x_{t_1}$$

Par élimination de x_{t_3}, on obtient le système suivant :

$$0 \leq x_{t_1} \leq 3$$

Étape (4) : t_3 est à nouveau sensibilisée.

$$0 \leq x_{t_1} \leq 3$$

$$1 \leq x_{t_3} \leq 3$$

Des contraintes diagonales associées aux transitions sensiblisées peuvent être calculées pour chaque classe. Pour chaque couple de transitions t et t' sensiblisées dans une classe $C = (m, D)$, avec $t_{1_{min}} \leq x_{t_1} \leq t_{1_{max}}$ et $t_{2_{min}} \leq x_{t_1} \leq t_{2_{max}} \in D$, on associe les deux contraintes $x_{t_1} - x_{t_2} \leq t_{1_{max}} - t_{2_{min}}$ et $x_{t_2} - x_{t_1} \leq t_{2_{max}} - t_{1_{min}}$.

3.2.5 Améliorations et raffinements du graphe des classes d'états

Dans sa première version, le SCG permet la vérification de certaines propriétés sur les TPN bornés comme l'accessibilité des états. Toutefois, il préserve uniquement les propriétés de logique linéaire. Pour remédier à cette limitation, un raffinement de la méthode a été proposée dans [47], sous forme d'un graphe appelé *Graphe des Classes d'États Atomiques (en anglais, Atomic States Class Graph — ASCG)*. Les auteurs utilisent un découpage des classes de l'état en ajoutant des contraintes linéaires de sorte que chaque état d'une classe atomique possède un successeur dans toutes les classes suivantes. Grâce à cette amélioration, le ASCG permet la vérification des propriétés CTL* sur les TPNs, mais avec une limitation sur les intervalles de temps qui doivent être bornés. Une nouvelle approche pour les classes atomiques a été proposée dans [9], qui permet la vérification des propriétés CTL* sans restriction sur les intervalles de temps. Lime et Roux utilisent l'approche des classes

d'états pour construire un automate temporisé qui préserve le comportement du TPN [33]. Ainsi, le nombre de variables est réduit d'une manière considérable. Le modèle résultant est alors vérifié avec les outils dédiés aux automates temporisés comme UPPAAL [32]. Cependant, UPPAAL ne peut vérifier qu'un sous-ensemble de la logique TCTL. Les améliorations apportées à l'approche des classes d'états ont été intégrées dans l'outil logiciel Tina [10].

3.3 Méthodes basées sur le graphe des zones

Le *Graphe Basé sur les Zones (en anglais, Zones Based Graph — ZBG)* [18, 17] est une une autre méthode pour la représentation de l'espace d'états d'un TPN. C'est une adaptation du *graphe des régions* [3, 15, 4, 5], proposé initialement pour les automates temporisés, dans laquelle les valeurs des horloges sont regroupées dans un ensemble de classes d'équivalence : les *régions*. En pratique, le nombre de régions est trop important pour effectuer une analyse efficace. Les régions sont donc regroupées dans un ensemble de *zones*. Une zone est une union convexe de régions et peut être représentée par une *matrice de différences bornées* (en anglais Difference Bound Matrice — DBM) [36, 37]. Ainsi, contrairement au graphe des classes, ce sont directement les horloges associées aux transitions qui sont encodées dans les zones.

3.3.1 Régions

Une région est obtenue à partir d'une relation d'équivalence, notée "$\approx$", définie pour un ensemble d'horloges X prenant leurs valeurs dans $\mathbb{R}_{\geq 0}$ et munies d'une constante maximale K. Le quotient obtenu est noté $\mathbb{R}_{\geq 0}^{X,K}$. Soit $x \in X$ une horloge, v et v' deux de ses valuations, alors, $v \approx v'$ si :

1. Pour toute horloge x, $ent(v(x)) = v'(x)$ ou $(ent(v(x)) \geq K$ et $v'(x) \geq K)$;
2. Pour toute horloge x telle que $v(x) \leq K$, $frac(v(x)) = 0$ si et seulement si $frac(v'(x)) = 0$;
3. Pour tout couple (x, y) d'horloges tel que $v(x) \leq K$ et $v'(x) \leq K$, $frac(v(x)) \leq frac(v(y))$ si et seulement si $frac(v'(x)) \leq frac(v'(y))$.

où, pour $d \in \mathbb{R}_+$, $ent(d)$ désigne la partie entière de d et $frac(d)$ sa partie fractionnelle.

Dans la partition obtenue, les régions sont : les points à coordonnées entières, les segments ouverts entre deux tels points, les intérieurs des triangles délimités par les segments, les demi-droites ouvertes et les régions ouvertes délimitées par ces demi-droites. La figure 3.1 représente l'ensemble des régions

pour les contraintes de la forme $x \sim K$ avec $X = \{x_1, x_2\}$, $\sim \in \{<, \leq, =, >, \geq\}$, $K = 0, 1, 2$ pour x_1 et $K = 0, 1$ pour x_2. Il y a 28 régions, certaines sont ré-

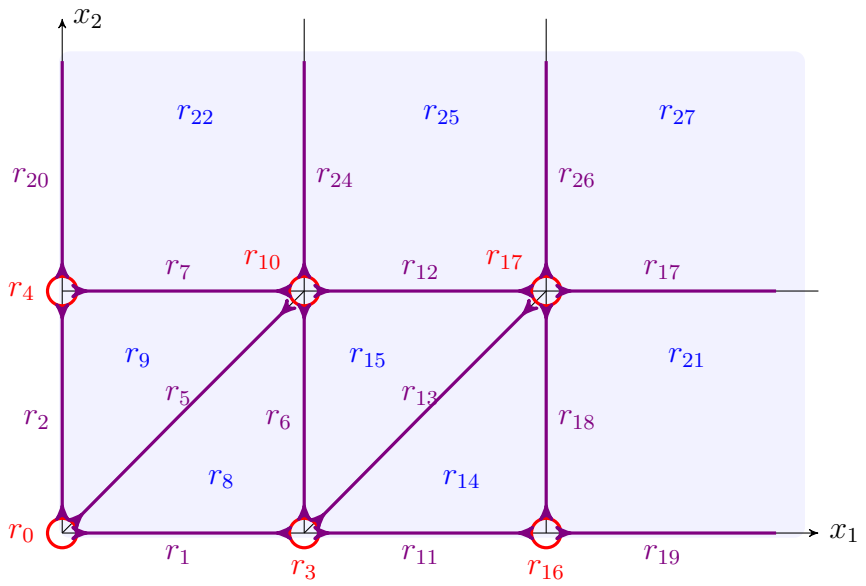


FIGURE 3.1 – Régions pour les contraintes $x_1, x_2 \sim K$ avec $K = 0, 1, 2$

duites à une unique configuration (comme la région r_0 définie par $x_1 = x_2 = 0$), d'autres sont des parties ouvertes du plan (comme la région r_{22} décrite par la contrainte $(0 < x_1 < 1) \wedge (x_2 > 1)$, les autres sont des segments (comme la région r_5 décrite par la contrainte $0 < x_1 = x_2 < 1$). Le système démarre dans la région initiale r_0. Lorsque le temps passe, on accède à la région r_5, puis à la région r_{10}, etc; jusqu'à la région r_{27}. Si, au lieu de laisser passer le temps, on effectue une transition discrète, la remise à zéro de certaines horloges nous conduit aux régions situées sur les axes. Par exemple, la remise à zéro de x_1 dans la région r_5 nous conduit à la région r_1 d'où l'on ira en r_8, etc.

3.3.2 Régions pour les TPNs

Pour les TPNs, une horloge x_t est associée à chaque transition t. Une constante maximale K_{x_t} est définie pour chaque horloge x_t comme suit :

$$\forall t \in \mathbb{T}, K_{x_t} = \begin{cases} t_{min} \text{ si } t_{max} = \infty \\ t_{max} \text{ sinon} \end{cases}$$

Une région est une classe d'équivalence des valuations d'horloges induite par la relation $\approx$ où $v \approx v'$ si et seulement si :

1. Pour tout $x_t \in X$, $ent(v(x_t)) = ent(v'(x_t))$ ou $(v(x_t) > K_{x_t}$ et $v'(x_t) > K_{x_t})$;
2. Pour tout $x_t, x_{t'} \in X$ avec $v(x_t) \leq K_{x_t}$ et $v(x_{t'}) \leq K_{x_{t'}}$, $frac(v(x_t)) \leq frac(v(x_{t'}))$ si et seulement si $frac(v'(x_t)) \leq frac(v'(x_{t'}))$;
3. Pour tout $x \in X$ avec $v(x_t) \leq K_x$, $frac(v(x_t)) = 0$ si et seulement si $frac(v'(x_t)) = 0$.

3.3.3 Zones

Une zone est un ensemble convexe de valuations d'horloges représentant un ensemble de contraintes de la forme $x_{t_i} - x_{t_j} \leq c_{ij}$, $x_{t_i} \leq c_{i0}$, $x_{t_i} \geq c_{0j}$ avec $c_{ij}, c_{i0}, c_{0j} \in \mathbb{Z}$. Une zone peut être encodée par une matrice de différences bornées en ajoutant une horloge additionnelle x_{t_0}. L'horloge x_{t_0} est associée à une transition t_0 toujours sensibilisée mais jamais tirée, et donc sa valeur vaut toujours zéro. En introduisant l'horloge x_{t_0}, une zone Z peut alors être écrite sous forme des contraintes atomiques suivantes : $\forall x_{t_i}, x_{t_j}, (x_{t_i} - x_{t_j} \leq c_{ij})$. Cela permet d'encoder la zone Z par une matrice de différences bornées (en anglais, Difference Bound Matrices — DBM) [14, 36]. Enfin, la forme canonique de la matrice est obtenue par un calcul basé sur l'algorithme Floyd-Warshall [28].

3.3.4 Encodage des zones avec les matrices de différences bornées

Les matrices de différences bornées [14, 36] ont pour but la représentation des zones d'horloges. Soit un ensemble de transitions $T = \{t_1, \ldots, t_n\}$ à qui on associe un ensemble d'horloges $X = \{x_1, \ldots, x_n\}$. Une matrice de différences bornées est une matrice indexée par $X \cup \{x_0\}$ où x_0 est une variable factice dont la valeur est toujours 0. Chaque élément de la matrice est de la forme $c_{i,j}$ et représente la contrainte atomique $(x_i - x_j \leq c_{ij})$, avec $x_i, x_j \in X$ et $c_{i,j} \in \mathbb{Q} \cup \{\infty\}$. Si $c_{i,j} = \infty$, ceci représente l'absence de borne supérieure pour $c_i - x_j$. Les éléments $c_{i,0}$ et $c_{0,j}$ représentent respectivement $(x_i \leq c_{i,0})$ et $(-x_j \leq c_{0,j})$.

Considérons l'exemple suivant : $x_1 - x_2 \leq 2 \wedge 0 \leq x_2 \leq 2 \wedge 1 \leq x_1$. Une représentation possible de ces contraintes par une DBM peut être donnée comme suit :

	x_0	x_1	x_2
x_0	0	-1	0
x_1	∞	0	2
x_2	2	∞	0

Une zone d'horloges ne se représente pas par une unique matrice. Ainsi, deux matrices distinctes peuvent correspondre à la même zone d'horloges.

Par exemple, $x_1 - x_2 \leq 2$ et $x_2 - x_0 \leq 2$ implique $x_1 - x_0 \leq 4$, donc $c_{1,0}$ pourrait être 4. D'où la nécessité de la mise sous forme canonique des DBMs. Le principe est comme suit : pour chaque x_i, x_j, x_k dans X, si $x_i - x_j \leq c_{i,j}$ et $x_j - x_k \leq c_{j,k}$, remplacer la borne de $x_i - x_k$ par $c_{i,j} + c_{j,k}$. L'algorithme permettant d'obtenir la forme canonique d'une DBM est appelé algorithme de Floyd-Warshal [28]. La forme canonique de l'exemple précédent est donnée par :

	x_0	x_1	x_2
x_0	0	-1	0
x_1	4	0	2
x_2	2	1	0

3.3.5 État symbolique

Un état symbolique est un couple (m, Z) où m est un marquage et Z est une zone sur l'ensemble des horloges associées aux transitions sensibilisées par le marquage m. L'état (m, Z) représente l'ensemble de valuations pour lesquelles un marquage est accessible.

3.3.6 Successeurs discrets

Le successeurs discret d'un état (m, Z) par le tir d'une transition t, noté $Post_t((m, Z))$, est donné par l'expression suivante :

$$Post_t((m, Z)) = (m', Z') = \begin{cases} m' = m - Pre(t) + Post(t); \\ Z' = (Z \cap \{x_t \geq t_{min}\})[x_{t_i} := 0, \quad \forall t_i \in \uparrow (m, t)] \end{cases}$$

où $\uparrow (m', t)$ est l'ensemble de transitions nouvellement sensibilisées après le tir de t. Cela permet de remettre à zéro les horloges des transitions nouvellement sensibilisées. Ainsi, Z' est la zone pour laquelle le nouveau marquage est accessible.

3.3.7 Successeurs temporels

Les successeurs temporels de (m, Z), notés $\overrightarrow{Post}((m, Z))$, sont les états accessibles à partir de (m, Z) en laissant le temps s'écouler jusqu' à ce qu'une transition atteigne sa borne de tir maximal. Les états $\overrightarrow{Post}((m, Z))$ sont obtenus comme suit :

$$\overrightarrow{Post}((m, Z)) = (m', Z') = \begin{cases} m' = m; \\ Z' = \overrightarrow{Z} \cap (\{x_t \leq t_{max}\}, \forall x_t \in Enable(m)) \end{cases}$$

Cette opération permet de calculer la zone maximale pour laquelle le marquage est m, autrement dit, Z' est la couverture maximale de Z permettant de rester dans le marquage m.

3.3.8 Graphe des zones

Le graphe des zones est obtenu en appliquant successivement un ensemble d'itérations à partir de l'état initial (m_0, Z_0) où m_0 représente la marquage initial et $Z_0 = \{x_{t_i} = 0 | t_i \in T \wedge t_i \in Enable(m_0)\}$. Ainsi, à chaque fois qu'un état (m, Z) est atteint, les trois opérations suivantes sont appliquées :

1. Calculer les successeurs temporels (par écoulement de temps) de l'état (m, Z) : $(m', Z') = \overrightarrow{Post}(m, Z)$.
2. Déterminer les transitions franchissables à partir de (m', Z') : t_i est franchissable si $Z' \cap \{x_{t_i} > t_{i_{min}}\}$ n'est pas une zone vide.
3. Pour chaque transition t franchissable dans (m', Z'), calculer le successeur de (m', Z') par le tir de t : $(m'', Z'') = Post_t(m', Z')$.

Pour assurer la terminaison du processus de construction des successeurs temporels, à chaque étape, la zone courante est comparée à celles précédemment obtenues. Si celle-ci est incluse ou égale à l'une d'entre elles, l'état courant est considéré comme déjà atteint et par conséquent il ne sera pas ajouté à l'ensemble des états du graphe et son exploration est évitée.

3.3.9 Exemple

Considérons le TPN de la figure 2.7.

L'état initial est (m_0, Z_0), avec $m_0 = (1, 0, 1)$ et $Z_0 = \{x_{t_1} = x_{t_3} = 0\}$. En appliquant les trois opérations ci-dessous sur (m_0, Z_0), on aura :

1. Calcul des successeurs temporels (m'_0, Z'_0), i.e. le temps maximal durant lequel on peut rester dans m_0 :
   $(m'_0, Z'_0) = \overrightarrow{Post}(m_0, Z_0) =$

$$\begin{cases} m' = m = (1, 0, 1); \\ Z' = \overrightarrow{Z_0} \cap (\{x_t \leq t_{max}\}, \forall t \in Enable(m_0)) \\ \quad = \{x_{t_1} = x_{t_3} \in [0; \infty[\} \cap \{x_{t_1} \leq 4 \wedge x_{t_3} \leq 3\} \\ \quad = \{(x_{t_1} = x_{t_3}) \wedge (x_{t_1} \leq 4) \wedge (x_{t_3} \leq 3)\} \end{cases}$$

2. À partir de (m'_0, Z'_0), deux transitions sont franchissables : t_1 et t_3.
3. Calcul des successeurs de (m'_0, Z'_0) par le tir de t_1 et t_3 :
   Le franchissement de t_1 :

$Post_{t_1}((m'_0, Z'_0)) = (m_1, Z_1) =$

$$\begin{cases} m_1 = m - Pre(t_1) + Post(t_1) = (0,1,1); \\ Z_1 = (Z'_0 \cap \{x_{t_1} \geq t_{1_{min}}\})[x_{t_2} := 0] \\ \quad = (\{(x_{t_1} = x_{t_3}) \wedge (x_{t_1} \leq 4) \wedge (x_{t_3} \leq 3)\} \cap \{x_{t_1} \geq 2\})[x_{t_2} := 0] \\ \quad = (\{(x_{t_1} = x_{t_3}) \wedge (2 \leq x_{t_1} \leq 4) \wedge (2 \leq x_{t_3} \leq 3)\})[x_{t_2} := 0] \\ \quad = (\{2 \leq x_{t_3} \leq 3)\})[x_{t_2} := 0] \\ \quad = \{(2 \leq x_{t_3} \leq 3) \wedge (x_{t_2} := 0)\} \end{cases}$$

Le franchissement de t_3 :
$Post_{t_3}((m'_0, Z'_0)) = (m_2, Z_2) =$

$$\begin{cases} m_2 = m - Pre(t_3) + Post(t_3) = (1,0,1); \\ Z_2 = (Z'_0 \cap \{x_{t_3} \geq t_{3_{min}}\})[x_{t_3} := 0] \\ \quad = (\{(x_{t_1} = x_{t_3}) \wedge (x_{t_1} \leq 4) \wedge (x_{t_3} \leq 3)\} \cup \{x_{t_3} \geq 1\})[x_{t_3} := 0] \\ \quad = (\{(x_{t_1} = x_{t_3}) \wedge (1 \leq x_{t_1} \leq 4) \wedge (1 \leq x_{t_3} \leq 3)\})[x_{t_3} := 0] \\ \quad = (\{(1 \leq x_{t_1} \leq 4)[x_{t_3} := 0] \\ \quad = \{(1 \leq x_{t_1} \leq 4) \wedge (1 \leq x_{t_3} \leq 3)\} \end{cases}$$

3.3.10 Approximations et améliorations

Comme pour les automates temporisés, un inconvénient de la méthode est le recours nécessaire à des méthodes d'approximation (k-approximation ou k_x-approximation [17]) dans le cas où l'infini est utilisé dans les bornes des intervalles temporels, et dans ce cas, la finitude du graphe n'est pas garantie. Pour résoudre ce problème, une étape supplémentaire est alors introduite. Celle ci consiste à appliquer des approximations sur les zones admettant des bornes infinies. Ci-dessous, nous présentons deux des approximations qui ont été proposées. Soit Z une zone sur un ensemble d'horloges X et $k \in \mathbb{Z}$.

1. k-approximation : L'idée est que, si une horloge va au-delà de la valeur k, sa valeur précise n'a pas d'importance et les comportements résultant de la zone obtenue sont les mêmes. Formellement, la k-approximation est obtenue comme suit :

$$\forall t_i, t_j \in Enable(m) \cup \{t_0\}, c_{ij} = \begin{cases} \infty \text{ si } c_{ij} > k \\ -k \text{ si } c_{ij} < -k \\ c_{ij} \text{ sinon} \end{cases}$$

2. k_x-approximation : La k_x-approximation améliore l'approximation précédente en prenant des valeurs de k différentes selon la transition non bornée considérée. Pour une transition $t_i \in T$, $k_{x_i} = \begin{cases} t_{i_{min}} \text{ si } t_{i_{max}} = \infty \\ t_{i_{max}} \text{ sinon} \end{cases}$

La k_x-approximation est alors donnée comme suit :

$$\forall t_i, t_j \in Enable(m) \cup \{t_0\}, c_{ij} = \begin{cases} \infty \text{ si } c_{ij} > k_{x_i} \\ -k_{x_j} \text{ si } c_{ij} < -k_{x_j} \\ c_{ij} \text{ sinon} \end{cases}$$

Cela permet d'obtenir une abstraction plus compacte en restant exacte vis-à-vis de l'accessibilité de marquage et de la vérification des propriétés TCTL.

Dans sa première version, le graphe des zones permet la vérification des propriétés temporelles quantitatives mais pas des propriétés CTL*. L'ajout d'une nouvelle transition pour mesurer le temps écoulé a été proposé dans [12]. Ce raffinement conduit au graphe appelé *Graphe Basé sur les Zones atomiques (en anglais, Atomic Zones Based Graph)*. Grâce à cet ajout de transition, les formules TCTL sont converties en formules CTL ce qui permet de les vérifier sur des TPNs bornés.

3.4 Conclusion

L'ajout des contraintes temporelles aux réseaux de Petri, via le modèle t-temporels, rend leurs analyse encore plus complexe du fait que le nombres des états accessibles devient, en général, infini. Dans ce chapitre, nous avons présenté les méthodes proposées dans la littérature ayant pour but de contrer le problème de l'explosion combinatoire de l'espaces des états d'un TPN. en construisant une représentation finie de l'espace d'états. Ces méthodes tentent de construire une représentation finie de l'espace d'états. Elles sont basées basées soit sur le graphe classes d'états soit sur le graphe des zones.

Une classe d'états est représentée par un marquage et un système système d'inéquations linéaires appelé domaine. Les nœuds du graphe sont donc des classes et les arcs sont étiquetés par les transitions du TPN. Le graphe des classes ainsi que ces extensions sont implémentés dans l'outils Tina. Des algorithmes de vérification des propriétés LTL, CTL et CTL* basés sur ce graphe ont été proposés et sont également implémentés dans l'outil.

Le graphe des zones a été proposé initialement pour les automates temporisés sous la forme de graphe des régions. Au contraire du graphe des classes, les temporisations sont représentées par des horloges et non pas par des variables. Une zone est une union convexe d'un ensemble de classes d'équivalence d'horloges (régions). Les algorithmes de construction des graphes basés sur les zones ainsi que ceux de vérification des propriétés CTL et TCTL sont implémentés dans l'outil logiciel Romeo.

Les graphes résultants de ces approches sont principalement caractérisés par leurs tailles (nombre de nœuds/arcs), les conditions de finitude, et le type

de propriétés préservées. Le calcul d'une classe d'états nécessite la résolution d'un système d'inéquations linéaires. Le nombre d'inéquations dans un système est proportionnelle au nombre de transitions activées. Le système d'inéquations nécessite en général un certain nombre d'éliminations et de changements de variables. Une zone est encodée par une matrice de différences bornées dont la taille est également proportionnelle au nombre de transitions activées. Des calculs supplémentaires sont nécessaires pour les approximations des bornes infinies dans les intervalles temporels. Ces calculs, qui ne sont pas toujours triviaux, peuvent être évités en encodant le temps de séjour dans chaque nœud du graphe. La prise en compte de ces paramètres fut le défi que nous avons relevé dans la nouvelle approche que nous proposons dans le prochain chapitre.

Chapitre 4

Graphe des agrégats temporisés

Sommaire

4.1 Introduction

Ce chapitre est consacré à notre principale contribution : le *Graphe des Agrégats Temporisés* (en anglais, *Timed Aggregate Graph —TAG*). Le TAG est une approche alternative pour l'abstraction et de l'analyse de l'espace des états des TPNs et fournit une représentation finie tout en préservant les propriétés avec des contraintes quantitatives sur le temps.

Le chapitre est organisé comme suit : la première partie est consacrée à la présentation du TAG, nous donnons sa définition formelle, illustrons sa construction (en donnant des exemples) et prouvons sa finitude pour les TPNs bornés. Dans la deuxième partie, nous établissons les principaux résultats issus de l'approche que nous proposons. D'un point de vue théorique, nous montrons que le TAG est une représentation exacte pour le comportement du TPN associé. D'un point de vue expérimental, les résultats obtenus montrent d'une manière générale un gain important en termes de taille des graphes (nœuds/arcs) par rapport aux approches basées sur les SCG et les ZBGs.

4.2 Graphe des agrégats temporisés

Le principale particularité du TAG par rapport aux deux approches décrites dans le chapitre précédent est le fait d'encoder les durées de séjour dans les nœuds du graphe. Le TAG peut être décrit comme un graphe où les nœuds, appelés *agrégats*, regroupent un ensemble d'états du TTS. Les temporisations relatives à un agrégat sont de deux types : (1) L'intervalle du temps de séjour dans l'agrégat courant : permet d'agréger les transitions de délai. Ainsi, le TAG ne comporte pas de transitions de délai et ses arcs sont étiquetés uniquement avec les transitions du TPN. (2) Les intervalles de tir dynamiques : permettent, de mettre à jour dynamiquement les dates de tir au plus tôt et au plus tard pour chaque transition sensibilisée dans l'agrégat courant. Ils permettent également de maintenir l'ordre de franchissement entre transitions (les contraintes diagonales).

4.2.1 Agrégats temporisés

Avant de définir formellement le TAG, commençons d'abord par donner la définition formelle d'un agrégat.

Définition 4.2.1 (Agrégat Temporisé) *Un* agrégat temporisé *associé à un* TPN $\mathcal{N} = \langle P, T, Pre, Post, I\rangle$ *est un 4-tuple* $a = (m, h, H, E)$*, où :*

- m *est un marquage ;*
- $E = \{\langle t, \alpha_t, \beta_t\rangle \mid t \in Enable(m),\ \alpha_t \in (\mathbb{Z} \cup \{-\infty\}) \wedge \beta_t \in \mathbb{N} \cup \{\infty\}\}$ *est l'ensemble des transitions sensibilisées associées à leurs délais de franchissement dynamiques au plus tôt* (α) *et au plus tard* (β)*.*
- $h = \min_{\langle t,\alpha_t,\beta_t\rangle \in E}(max(0, \alpha_t))$ *: est le temps de séjour minimum du système dans l'agrégat* a *;*
- $H = \min_{\langle t,\alpha_t,\beta_t\rangle \in E}(\beta_t)$ *: est le temps de séjour maximum du système dans l'agrégat* a*.*

Un agrégat a est caractérisé par un marquage m, un ensemble de transitions sensibilisées E où chaque transition est associée à son intervalle de tir *dynamique* $[\alpha_t; \beta_t]$, et enfin h et H représentant respectivement le temps de séjour minimum et maximum du système dans l'agrégat courant. Comme pour les états d'un TTS, les attributs d'un agrégats a sont notés $a.m$, $a.E$, $a.h$ et $a.H$.

4.2.2 Graphe des agrégats temporisés

Nous pouvons maintenant définir formellement le TAG associé à un TPN marqué. C'est un système de transitions étiqueté avec un ensemble d'agrégats, un agrégat initial, un ensemble d'arcs étiquetés par des transitions et une

relation de transition. L'agrégat initial est calculé à partir des informations statiques du TPN comme suit : (1) L'attribut marquage est le marquage initial du TPN. (2) Chaque transition sensibilisée t est franchissable entre $t_{\min}$ et $t_{\max}$. (3) Le temps minimum (respectivement maximum) de séjour du système dans l'agrégat initial est le minimum des instants statiques de tir au plus tôt (respectivement au plus tard) des transitions sensibilisées.

Définition 4.2.2 (Graphe des Agrégats Temporisés) *Un* TAG *associé à un TPN* $\mathcal{N} = \langle P, T, Pre, Post, I, m_0\rangle$ *est un tuple* $G = \langle \mathcal{A}, T, a_0, \delta\rangle$ *où :*

1. $\mathcal{A}$ *est l'ensemble d'agrégats temporisés ;*
2. $a_0 = \langle m_0, h_0, H_0, E_0\rangle$ *est l'agrégat initial tel que :*
   (a) m_0 *est le marquage initial de* $\mathcal{N}$ *;*
   (b) $h_0 = \min_{t\in Enable(m_0)}(t_{\min})$ *;*
   (c) $H_0 = \min_{t\in Enable(m_0)}(t_{\max})$ *;*
   (d) $E_0 = \{\langle t, t_{\min}, t_{\max}\rangle \mid t \in Enable(m_0)\}$.
3. $\delta \subseteq \mathcal{A} \times T \times \mathcal{A}$ *est la relation de transition telle que :*
   pour un agrégat $a = \langle m, h, H, E\rangle$*, une transition* t *avec* $\langle t, \alpha_t, \beta_t\rangle \in E$ *et un agrégat* $a' = \langle m', h', H', E'\rangle$, $(a, t, a') \in \delta$ *si et seulement si :*
   (a) $m' = m - Pre(t) + Post(t)$ *;*
   (b) $\alpha_t \leq a.H$ *;*
   (c) $\forall\langle t', \alpha_{t'}, \beta_{t'}\rangle \in a.E$ *;*
   $t_{min} > t'_{max} \Rightarrow (t_{min} - \alpha_t) - (t'_{min} - \alpha_{t'}) \geq (t_{min} - t'_{max})$
   (d) $E' = E'_1 \cup E'_2$*, où :*
   - $E'_1 = \bigcup_{t'\in\uparrow(a,t)}\{\langle t', t'_{\min}, t'_{\max}\rangle\}$ *;*
   - $E'_2 = \bigcup_{t'\in\downarrow(a,t)}\{\langle t', \alpha_{t'} - a.H, \beta_{t'} - max(0, \alpha_t)\rangle\}$ *;*

   (e) $h' = \min_{\langle t', \alpha_{t'}, \beta_{t'}\rangle\in E'}(max(0, \alpha_{t'}))$ *;*
   (f) $H' = \min_{\langle t', \alpha_{t'}, \beta_{t'}\rangle\in E'}(\beta_{t'})$.

Étant donnés un agrégat $a = \langle m, h, H, E\rangle$ et une transition $t \in T$, t est sensibilisée dans a, noté $a \xrightarrow{t}$, si et seulement si : (1) $\exists(\alpha_t, \beta_t) \in (\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{N} \cup \{\infty\})$ tel que $(t, \alpha_t, \beta_t) \in E$ et $\alpha_t \leq H$.
(2) Il n'existe pas d'autre transition t', sensibilisée par $a.m$, qui doit être franchie avant t.
En effet, la première condition est suffisante si l'intervalle statique de t chevauche tous les intervalles des transitions sensibilisées. Mais ceci n'est pas le cas quand t_{min} est supérieur à un certain t'_{max} d'une autre transition sensibilisée t', et dans ce cas, la condition (2) est nécessaire. Si le franchissement d'une transition t à partir d'un agrégat a mène à un nouvel agrégat $a' = \langle m', h', H', E'\rangle$, alors les attributs temporels de a' sont calculés grâce aux ensembles $\uparrow(a, t)$ et $\downarrow(a, t)$. Ceci constitue l'idée clé sous-jacente au TAG :

– Les éléments de E' sont calculés en fonction de leur appartenance aux ensembles $\uparrow(a,t)$ et $\downarrow(a,t)$. Pour chaque transition $t' \in Enable(a'.m)$, si t' est nouvellement sensibilisée, ses délais de tir dynamiques au plus tôt et au plus tard sont définis statiquement par t'_{min} et t'_{max} respectivement. Sinon ($t' \in \downarrow(a,t)$), si $\langle t', \alpha_{t'}, \beta_{t'} \rangle \in a'.E$, alors le temps maximum de séjour du système dans $a.m$ (c'est-à-dire $a.H$) est soustrait de $\alpha_{t'}$ et la date de tir dynamique au plus tôt de la transition t (c'est-à-dire α_t) est soustraite de $\beta_{t'}$. En effet, plus le système séjourne dans $a.m$ moins il le fera en $a'.m$ et vice-versa. Ainsi, le nouveau délai de tir au plus tôt de la transition t à partir de a' est $max(0, \alpha_{t'} - a.H)$ et son nouveau délai de tir au plus tard est $\beta_{t'} - \alpha_t$.
– Le calcul de $a'.h$ (respectivement $a'.H$) est donné par le minimum des délais de tir dynamiques au plus tôt (respectivement au plus tard) des transitions sensibilisées.

4.2.3 Relation d'equivalence entre les agrégats

D'après la définition du TAG (Définition 4.2.2), le délai de tir dynamique au plus tôt d'une transition peut diminuer infiniment. Ceci implique que le TAG peut être infini. C'est pour cela que nous avons défini une relation permettant d'identifier les agrégats équivalents. Cette relation d'équivalence est utilisée dans la construction du TAG de sorte que chaque agrégat nouvellement construit ne soit pas exploré si un agrégat équivalent a été déjà construit.

Définition 4.2.3 (Relation d'equivalence entre les agrégats) *Soient a et a' deux agrégats. a est équivalent à a', noté $a \equiv a'$, si et seulement si :*

1. $a.h = a'.h$, $a.H = a'.H$ *et* $a.m = a'.m$;
2. $\forall(\langle t, \alpha_t, \beta_t \rangle, \langle t, \alpha'_t, \beta'_t \rangle) \in (a.E \times a'.E)$, $\beta_t = \beta'_t$:

   (a) $\alpha_t = \alpha'_t$ *ou,*
   (b) $max(0, \alpha_t) = max(0, \alpha'_t)$ *et*
      i. $\nexists t' \in T : ((t'_{max} < t_{min}) \vee (t_{max} < t'_{min})$ *ou*
      ii. $\exists t' \in (T \setminus Enable(a.m))\ (t_{max} < t'_{min})$ *ou*
      iii. $\exists t' \in (T \setminus Enable(a.m))\ t'_{max} < t_{min} \wedge (\alpha_t \leq t'_{max}) \wedge (\alpha'_t \leq t'_{max})$ *ou*
      iv. $\exists t' \in Enable(a.m)\ t'_{max} < t_{min} \wedge ((\alpha_t - \alpha_{t'}) \leq (t'_{max} - t'_{min}) \wedge (\alpha'_t - \alpha'_{t'}) \leq (t'_{max} - t'_{min})) \vee ((\alpha_{t'} - \alpha_t) = (\alpha'_{t'} - \alpha'_t))$.

Informellement, la définition précédente garantit que le comportement du système à partir de deux agrégats équivalents est le même. Le point 1 garantie

que a et a' ont le même marquage et les mêmes temps de séjour minimum et maximum. Le point 2 assure que les délais de tir au plus tard de chaque transition sensibilisée t dans a et a' sont les mêmes. Si le délai de tir au plus tôt de t dans a est différent de son délai au plus tôt dans a', alors cela ne doit pas influencer les comportements futurs de a et a'. Premièrement, s'il n'y a pas une transition t' telle que $t'_{max} < t_{min}$, alors cette différence est sans conséquence tant que les deux délais de tir au plus tôt sont négatifs ou nuls. Deuxièmement, s'il existe une transition t' telle que $t'_{max} < t_{min}$ et que t' n'est pas activée dans a, alors α_t et α'_t doivent être tous les deux inférieurs ou égales à t'_{max}. Enfin, dans le cas où il existe une transition t' activée dans a, alors : soit t est franchissable dans les deux agrégats a et a' soit la différence $(t_{min} - \alpha_t) - (t'_{min} - \alpha_{t'})$ est égale à $(t_{min} - \alpha'_t) - (t'_{min} - \alpha'_{t'})$.

4.2.4 Exemples

Considérons les exemples de TPNs suivants. Bien que ces deux modèles soient simples, ils sont assez représentatifs pour expliquer la construction d'un TAG. En effet, dans le premier modèle, les intervalles associés aux transitions se chevauchent, tandis que le deuxième modèle illustre le cas où le délai maximum de tir d'une transition est infini. Des exemples plus significatifs sont donnés dans la expérimentations de ce chapitre.

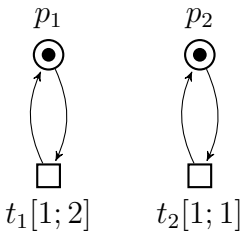


FIGURE 4.1 – TPN (a)

Les Figures 4.3 et 4.4 illustrent les TAGs associés aux TPNs des Figures 4.1 et 4.2 respectivement. Les graphes des classes d'états obtenus à partir de ces TPNs (avec l'outil Tina) sont illustrés dans les Figures 4.5 et 4.6 respectivement. Enfin, les graphes des zones correspondant sont représentés dans les Figures 4.7 et 4.8 respectivement. Dans les graphes associés au premier TPN, les marquages n'apparaissent pas dans les nœuds car ils sont tous identiques au marque initial. Les TAGs et SCGs sont identiques pour les TPNs des Figures 4.1 et 4.2 (6 noeuds/10 arcs et 5 noeuds/7 arcs respectivement). Les ZBG de ces deux TPNs contiennent 7 nœuds/12 arcs pour le premier et 4 nœuds/6 arcs pour le deuxième.

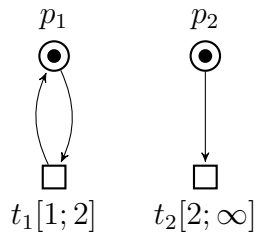
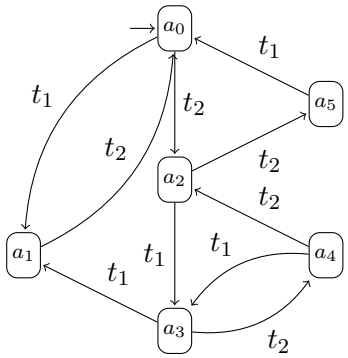
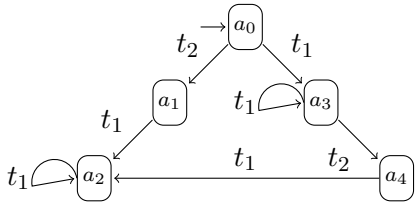


FIGURE 4.2 – TPN (c)



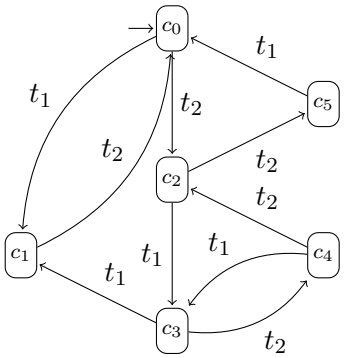
agrégat	(h,H)	E
a_0	(1,1)	$\{\langle t_1,1,2\rangle,\langle t_2,1,1\rangle\}$
a_1	(0,0)	$\{\langle t_1,1,2\rangle,\langle t_2,0,0\rangle\}$
a_2	(0,1)	$\{\langle t_1,0,1\rangle,\langle t_2,1,1\rangle\}$
a_3	(0,0)	$\{\langle t_1,1,2\rangle,\langle t_2,0,1\rangle\}$
a_4	(0,1)	$\{\langle t_1,0,2\rangle,\langle t_2,1,1\rangle\}$
a_5	(0,0)	$\{\langle t_1,-1,0\rangle,\langle t_2,1,1\rangle\}$

FIGURE 4.3 – Le TAG du TPN de le Figure 4.1



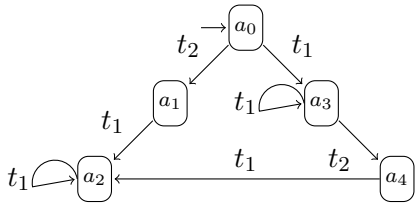
agrégat	marquage	(h,H)	E
a_0	(1,1)	(1,2)	$\{\langle t_1,1,2\rangle,\langle t_2,2,\infty\rangle\}$
a_1	(1,0)	(0,0)	$\{\langle t_1,-1,0\rangle\}$
a_2	(1,0)	(1,2)	$\{\langle t_1,1,2\rangle\}$
a_3	(1,1)	(0,2)	$\{\langle t_1,1,2\rangle,\langle t_2,0,\infty\rangle\}$
a_4	(1,0)	(0,2)	$\{\langle t_1,-1,2\rangle\}$

FIGURE 4.4 – Le TAG du TPN de la Figure 4.2



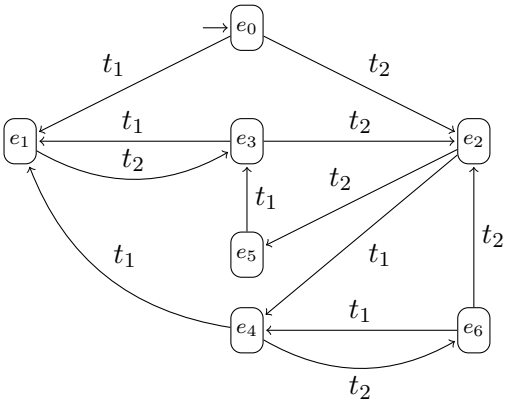
classe	D
c_0	$\{t_1[1,2], t_2[1,1]\}$, $\{t_1 - t_2 \leq 1, t_2 - t_1 \leq 0\}$
c_1	$\{t_1[1,2], t_2[0,0]\}$, $\{t_1 - t_2 \leq 2, t_2 - t_1 \leq -1\}$
c_2	$\{t_1[0,1], t_2[1,1]\}$, $\{t_1 - t_2 \leq 0, t_2 - t_1 \leq 0\}$
c_3	$\{t_1[1,2], t_2[0,1]\}$, $\{t_1 - t_2 \leq 2, t_2 - t_1 \leq 0\}$
c_4	$\{t_1[0,0], t_2[1,1]\}$, $\{t_1 - t_2 \leq -1, t_2 - t_1 \leq 1\}$
c_5	$\{t_1[0,2], t_2[1,1]\}$, $\{t_1 - t_2 \leq 1, t_2 - t_1 \leq 1\}$

FIGURE 4.5 – Le SCG du TPN de la Figure 4.1



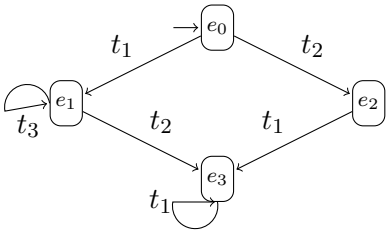
classe	m	D
c_0	(1 , 1)	$\{t_1[1,2], t_2[2,\infty]\}$, $\{t_1 - t_2 \leq 0\}$
c_1	(1 , 1)	$\{t_1[1,2], t_2[0,\infty]\}$, $\{t_1 - t_2 \leq 2\}$
c_2	(1 , 0)	$\{t_1[0,0]\}$
c_3	(1 , 0)	$\{t_1[0,2]\}$
c_4	(1 , 0)	$\{t_1[1,2]\}$

FIGURE 4.6 – Le SCG du TPN de la Figure 4.2



état	Z
e_0	$\{t_1[0,1], t_2[0,1]\}$, $\{t_1 - t_2 \leq 0, t_2 - t_1 \leq 0\}$
e_1	$\{t_1[0,0], t_2[1,\infty]\}$, $\{t_1 - t_2 \leq -1\}$
e_2	$\{t_1[1,\infty], t_2[0,\infty]\}$, $\{t_2 - t_1 \leq -1\}$
e_3	$\{t_1[0,\infty], t_2[0,\infty]\}$, $\{t_1 - t_2 \leq 0, t_2 - t_1 \leq 0\}$
e_4	$\{t_1[0,\infty], t_2[0,\infty]\}$, $\{t_1 - t_2 \leq 0\}$
e_5	$\{t_1[2,\infty], t_2[0,0]\}$, $\{t_2 - t_1 \leq -2\}$
e_6	$\{t_1[0,\infty], t_2[0,\infty]\}$, $\{t_2 - t_1 \leq 0\}$

FIGURE 4.7 – Le ZBG du TPN de la Figure 4.1



état	m	Z
e_0	(1 , 1)	$\{t_1[0,2], t_2[0,2]\}$, $\{t_1 - t_2 \leq 0, t_2 - t_1 \leq 0\}$
e_1	(1 , 1)	$\{t_1[0,\infty], t_2[0,\infty]\}$
e_2	(1 , 0)	$\{t_1[0,\infty], t_2[0,0]\}$, $\{t_2 - t_1 \leq -2\}$
e_3	(1 , 0)	$\{t_1[0,\infty], t_2[0,\infty]\}$, $\{t_2 - t_1 \leq 0\}$

FIGURE 4.8 – Le ZBG du TPN de la Figure 4.2

4.3 Preuve de finitude

Dans cette section, nous démontrons que le TAG associé à un TPN borné est fini. Pour cela, nous montrons que le nombre de ses agrégats est borné et nous calculons cette borne. Les deux lemmes suivants établissent deux résultats préliminaires avant de prouver la finitude du TAG.

Lemme 4.3.1 *Soient un TPN $\mathcal{N}$ et le TAG $G = \langle \mathcal{A}, T, a_0, \delta \rangle$. Soit un agrégat a de G et soit une transition $t \in Enable(a.m)$. Alors, le nombre d'intervalles dynamiques possibles $[\alpha_t; \beta_t]$ associés à la transition t dans tous les agrégats non équivalents est au plus égal à $\mathbb{B}_t$ où*

$$\mathbb{B}_t = \begin{cases} b_t & si \ \nexists t' \in T : \ t'_{max} < t_{min} \\ b_t * c_t & sinon \end{cases}$$

où $b_t = (t_{min} + 1) * (t_{max} + 1) - (t_{min} + (t_{min} * (t_{min} + 1)/2))$ et $c_t = Max_{t' \in T:\, t'_{max} < t_{min}}(t_{min} - t'_{max}) + 1$

Preuve Les intervalles possibles sont obtenus par toutes les combinaisons possibles pour les délais de tir au plus tôt et au plus tard α_t et β_t, excepté ceux qui sont invalides (c'est-à-dire ceux où $\alpha_t > \beta_t$ et ceux où $\alpha_t \neq t_{min} \wedge \alpha_t = \beta_t$). Dans le premier cas, nous considérons qu'il n'existe pas de transition t' telle que $t'_{max} < t_{min}$. Puisque les valeurs négatives de α_t sont équivalentes à 0, les valeurs possibles et équivalentes de α_t sont alors $\{t_{min}, t_{min} - 1, \ldots, 0\}$ (c'est-à-dire $t_{min} + 1$ valeurs possibles). De plus, $\beta_t \in \{t_{max}, t_{max} - 1, \ldots, 0\}$ (c'est-à-dire $t_{max} + 1$ valeurs possibles) et il y a $t_{min} + (t_{min} * (t_{min} + 1)/2)$ intervalles invalides. Les t_{min} premiers intervalles sont invalides par construction, puisque β_t ne peut jamais atteindre α_t par décrémentation ($\alpha \leq H$). De plus, il existe $(t_{min} * (t_{min} + 1)/2)$ intervalles où $\alpha_t > \beta_t$, ceux-là sont également invalides. Dans le deuxième cas, le raisonnement précédent est toujours valable, à condition de considérer la différence entre t et t'. Selon la définition de la relation d'équivalence entre agrégats, cette différence est ignorée si elle permet le franchissement de t. Ainsi, il y a $t_{min} - t'_{max} + 1$ valeurs à distinguer (pour t' ayant la plus petite valeur t'_{max} telle que $t'_{max} < t_{min}$) et qui est donc à combiner avec le nombre de possibilités obtenues dans le premier cas.

Lemme 4.3.2 *Soient un TPN $\mathcal{N}$ et le TAG $G = \langle \mathcal{A}, T, a_0, \delta \rangle$ correspondant et soit un marquage accessible $m \in Reach(\mathcal{N})$. Le nombre d'agrégats dans G ayant m comme marquage est au plus égal à $\sum_{s \in 2^{Enable(m)}} (\prod_{t \in s} \mathbb{B}_t)$.*

Preuve Si m est un marquage accessible, alors les agrégats accessibles ayant m comme marquage se distinguent par la valeur de leur attribut E. Il y a autant de possibilités de partitionnement des transitions anciennement et

nouvellement sensibilisées que le nombre de sous-ensembles de $Enable(m)$. D'après le Lemme 4.3.1, pour un sous-ensemble s de $Enable(m)$, il existe au plus $\prod_{t\in s} \mathbb{B}_t$ intervalles possibles pour ses transitions.

Les deux lemmes précédents permettent de déduire le résultat suivant.

Théorème 4.3.3 *Étant donné un TPN $\mathcal{N}$, si $\mathcal{N}$ est borné, alors le TAG associé est fini et le nombre de ses agrégats est au plus égal à $\sum_{m\in Reach(\mathcal{N})} \sum_{s\in 2^{Enable(m)}} \prod_{t\in Enable(m)} \mathbb{B}_t$.*

4.4 Résultats de préservation

Dans cette section, nous présentons les principaux résultats théoriques de notre approche et nous les appuyons par des preuves. Nous montrons principalement les deux résultats suivants : (1) Le TAG préserve les marquages accessibles du TPN associé. (2) À chaque exécution du TAG correspond une séquence de franchissement dans le TPN associé, et vice-versa. De plus, la duré de toute séquence de franchissement peut être borné en utilisant les informations encodées dans les agrégats du TAG.

Théorème 4.4.1 *Soit $\mathcal{N}$ un TPN et soit $G = \langle \mathcal{A}, T, a_0, \delta\rangle$ le TAG associé à $\mathcal{N}$. Alors, pour toute exécution $\pi = a_0 \xrightarrow{t_1} a_1 \rightarrow \ldots \xrightarrow{t_n} a_n$ dans le TAG, $\forall d \in \mathbb{R}_{\geq 0}$ tel que $d \leq a_n.H$, Il existe une exécution $\overline{\pi} = (m_0, V_0) \xrightarrow{(d_1,t_1)} (m_1, V_1 \rightarrow \ldots \xrightarrow{(d_n,t_n)} (m_n, V_n \xrightarrow{d}$ dans $\mathcal{N}$, tel que $m_i = a_i.m$ et $a_{i-1}.h \leq d_i \leq a_{i-1}.H$, pour $i = 1 \ldots n$.*

Preuve Soit $\pi = a_0 \xrightarrow{t_1} a_1 \rightarrow \ldots \xrightarrow{t_n} a_n$ un chemin dans G et soit $d \in \mathbb{R}_{\geq 0}$ avec $d \leq a_n.H$. On note α_{i_t} (respectivement β_{i_t}) le maximum entre 0 et le délai de tir dynamique au plus tôt (respectivement le délai de tir dynamique au plus tard) d'une transition t dans l'agrégat a_i.

Prouvons que le chemin $(m_0, V_0) \xrightarrow{(\alpha_{0_{t_1}},t_1)} (m_1, V_1) \rightarrow \ldots \xrightarrow{(\alpha_{n-1_{t_n}},t_n)} (m_n, V_n) \xrightarrow{d}$ vérifie le résultat requis. La particularité de cette exécution est le fait que le franchissement de chaque transition t est effectué au plus tôt (même si t_i aurait pu être tirée plus tôt dans un état antérieur a_j, $j < i$). Procédons par récurrence sur la longueur de π.
Notons tout d'abord que, $a_{i-1}.h \leq \alpha_{i-1_{t_i}}$ (par construction) et $\alpha_{i-1_{t_i}} \leq a_{i-1}.H$, pour $i = 1 \ldots n$.

- $\mid \pi \mid = 0$: puisque, $m_0 = a_0.m$, $a_0.H$ est la valeur minimale des délais de tir au plus tard pour toutes les transitions sensibilisées dans m_0 et $a_0.h \leq d \leq a_0.H$. Donc, $(m_0, V_0) \xrightarrow{d}$.

– Supposons maintenant que pour toute exécution $a_0 \xrightarrow{t_1} a_1 \rightarrow \ldots \xrightarrow{t_n} a_n$ de longueur n et pour tout $d \in \mathbb{R}_{\geq 0}$ vérifiant $d \leq a_n.H$, l'exécution $(m_0, V_0) \xrightarrow{(\alpha_{0_{t_1}}, t_1)} (m_0, V_0) \rightarrow \ldots \xrightarrow{(\alpha_{n-1_{t_n}}, t_n)} (m_n, V_n) \xrightarrow{d}$ est une exécution de $\mathcal{N}$. Soit $\pi = a_0 \xrightarrow{t_1} a_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} a_n \xrightarrow{t_{n+1}} a_{n+1}$ une exécution de longueur $n+1$. Puisque $\alpha_{n_{t_{n+1}}} < 0 \vee a_n.h \leq \alpha_{n_{t_{n+1}}} \leq a_n.H$. Nous pouvons alors utiliser l'hypothèse de récurrence pour exhiber une exécution $(m_0, V_0) \xrightarrow{(\alpha_{0_{t_1}}, t_1)} (m_1, V_1) \rightarrow \ldots \xrightarrow{(\alpha_{n-1_{t_n}}, t_n)} (m_n, V_n) \xrightarrow{\alpha_{n_{t_{n+1}}}}$. Montrons maintenant qu'il est possible de compléter cette exécution en franchissant t_{n+1} à partir de m_n (après un délai $\alpha_{n_{t_{n+1}}}$). Notons tout d'abord que t_{n+1} est sensibilisée par m_n (car $m_n = a_n.m$).
Soit l (avec $l < n$) le plus grand nombre entier, tel que $t_{n+1} \in \uparrow (a_{l-1}, t_l)$, alors $\langle t_{n+1}, t_{n+1_{min}}, t_{n+1_{max}} \rangle \in a_l.E$. De plus, $\forall i = l+1 \ldots n$, $\langle t_{n+1}, \alpha_{i_{t_{n+1}}}, \beta_{i_{t_{n+1}}} \rangle \in a_i.E$ avec $\alpha_{i_{t_{n+1}}} = \alpha_{i-1_{t_{n+1}}} - a_{i-1}.H$ et $\beta_{i_{t_{n+1}}} = \beta_{i-1_{t_{n+1}}} - \max(0, \alpha_{i-1_{t_i}})$.
Par définition, nous savons que $\alpha_{i-1_{t_i}} \leq a_{i-1}.H$ (condition pour le franchissement de t_i à partir de a_{i-1}), pour tout $i = l+1 \ldots n$. En utilisant le fait que $\alpha_{l_{t_{n+1}}} \leq \beta_{l_{t_{n+1}}}$ et le fait que $\alpha_{l_{t_{l+1}}} \leq a_l.h \leq \beta_{l_{t_{n+1}}}$, nous pouvons déduire que $\alpha_{l_{t_{n+1}}} - a_l.H \leq \beta_{l+1_{t_{n+1}}} = (\beta_{l_{t_{n+1}}} - \alpha_{l_{t_{l+1}}})$ et que $\beta_{l+1_{t_{n+1}}} \geq 0$, donc $\alpha_{l+1_{t_{n+1}}} \leq \beta_{l+1_{t_{n+1}}}$. En itérant ce résultat nous déduisons que $\alpha_{n_{t_{n+1}}} \leq \beta_{n_{t_{n+1}}}$ et que $\beta_{n_{t_{n+1}}} \geq 0$. Par ailleurs, il est clair que $\beta_{i-1_{t_{n+1}}} \geq \beta_{i_{t_{n+1}}}$ (pour tout $i = l+1 \ldots n$). En particulier, nous avons $\alpha_{n_{t_{n+1}}} \leq \beta_{n_{t_{n+1}}} \leq \beta_{n-1_{t_{n+1}}} \leq \cdots \leq \beta_{l_{t_{n+1}}} = t_{n+1_{max}}$. Ainsi, le système peut rester dans m_n pour une durée $\alpha_{n_{t_{n+1}}}$. En plus, le fait que t_{n+1} est franchissable dans a_n implique qu'elle est également franchissable à partir m_n. L'écoulement de temps conduit à a_{n+1} dont le marquage est le même que ce lui de a_n. Montrons maintenant qu'il est possible de rester dans m_{n+1} pour une durée $a_{n+1}.H$ (ou toute durée $d \leq a_{n+1}.H$) unités de temps. Nous pouvons utiliser le même raisonnement pour montrer que pour toute transition t sensibilisée par m_{n+1}, alors $\alpha_{n+1_t} \leq t_{max}$. Puisque $a_{n+1}.H = \min_{t \in Enable(m_{n+1})} \beta_t$, nous pouvons déduire que le système peut rester dans m_{n+1} au plus un délai de $a_{n+1}.H$ unités de temps.

Théorème 4.4.2 *Soit $\mathcal{N}$ un TPN et soit $G = \langle \mathcal{A}, T, a_0, \delta \rangle$ le TAG associé à $\mathcal{N}$. Alors $\forall \overline{\pi} = (m_0, V_0) \xrightarrow{(d_1,t_1)} (m_1, V_1) \xrightarrow{(d_2,t_2)} \ldots \xrightarrow{(d_n,t_n)} (m_n, V_n) \xrightarrow{d_{n+1}}$, avec $d_i \in \mathbb{R}_{\geq 0}$, pour $i = 1 \ldots n+1$, $\exists \pi = a_0 \xrightarrow{t_1} a_1 \rightarrow \ldots \xrightarrow{t_n} a_n$ tel que : $a_{i-1}.h \leq d_i \leq a_{i-1}.H$, $m_i = a_i.m$, pour $i = 0 \ldots n$ et $d_{n+1} \leq a_n.H$.*

Preuve Soit $\overline{\pi} = (m_0, V_0) \xrightarrow{(d_1,t_1)} (m_1, V_1) \xrightarrow{(d_2,t_2)} \ldots \xrightarrow{(d_n,t_n)} (m_n, V_n) \xrightarrow{d_{n+1}}$ une exécution de $\mathcal{N}$, avec $d_i \in \mathbb{R}_{\geq 0}$, pour $i = 1 \ldots n+1$. Montrons par

récurrence sur la longueur de $\overline{\pi}$ l'existence d'une exécution π dans le TAG vérifiant le Théorème 4.4.2.

- $\mid \overline{\pi} \mid = 0$: Évident puisque $m_0 = a_0.m$ (par construction) et d_1 est inférieure ou égale à $\min_{t \in Enable(m_0)} t_{max}$ ce qui est exactement la valeur de $a_0.H$.
- Supposons que le Théorème 4.4.2 soit vrai pour une exécution $\overline{\pi}$ telle que $\mid \overline{\pi} \mid \leq n$.
  Soit $\overline{\pi} = (m_0, V_0) \xrightarrow{(d_1,t_1)} (m_1, V_1) \xrightarrow{(d_2,t_2)} \ldots \xrightarrow{(d_n,t_n)} (m_n, V_n) \xrightarrow{(d_{n+1},t_{n+1})} (m_{n+1}, V_{n+1}) \xrightarrow{d_{n+2}}$ une exécution de longueur $n+1$. Soit $\pi = a_0 \xrightarrow{t_1} a_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} a_n$ l'exécution dans le TAG associé au préfixe de longueur n de $\overline{\pi}$ (par l'hypothèse de récurrence). On note α_{i_t} (respectivement β_{i_t}) le maximum entre 0 et le délai de tir dynamique au plus tôt (respectivement le délai de tir dynamique au plus tard) de la transition t dans l'agrégat a_i.
  Puisque $a_n.m = m_n$ et $d_{n+1} \leq a_n.H$, t_{n+1} est également activée par a_n (par construction) et peut être franchie conduisant à a_{n+1} dont le marquage est le même que m_{n+1}. Montrons maintenant que $d_{n+2} \leq a_{n+1}.H$. Si $Enable(m_{n+1}) = \emptyset$ alors m_{n+1} est un état bloquant comme l'est a_{n+1} et $d_{n+1} < a_{n+1}.H = \infty$. Sinon, soit $t \in Enable(m_{n+1})$ et soit $1 \leq k \leq n+1$ le plus grand entier, vérifiant $t \in\uparrow (m_k, t_{n+1})$. Le fait que $m_{n+1} \xrightarrow{d_{n+2}}$ implique que $\sum_{i=k}^{n+1} d_i \leq t_{max}$, ainsi $d_{n+1} \leq t_{max} - \sum_{i=k}^{n} d_i$. En prouvant que $d_i \geq \alpha_{i_{t_{i+1}}}$ pour $i = k \ldots n$, nous déduisons que $d_{n+1} \leq t_{max} - \sum_{i=k}^{n} \alpha_{i_{t_{i+1}}} = \beta_{n+1_t}$. Ce résultat est valable pour toute transition t sensibilisée, ainsi $d_{n+1} \leq \min_{t \in Enable(m_{n+1})}(\beta_{n+1_t}) = a_{n+1}.H$. Enfin, montrons que $d_i \geq \alpha_{i_{t_{i+1}}}$ pour tout $i = k \ldots n$. Soit $k \leq i \leq n$, si $\alpha_{i_{t_{i+1}}} \leq 0$ alors $d_i \geq \alpha_{i_{t_{i+1}}}$. Sinon, soit $1 \leq l \leq i$ la valeur entière la plus élevée, s'il existe une transition t_{i+1} telle que $t_{i+1} \in\uparrow (m_l, t_l)$ (si une telle valeur n'existe pas, alors $l = 0$). Alors $\sum_{j=l}^{i} d_j \geq t_{i+1_{min}}$ (puisque t_{i+1} est franchissable à partir de m_i), ce qui veut dire que $d_i \geq t_{i+1_{min}} - \sum_{j=l}^{i-1} d_j$. En utilisant l'hypothèse de récurrence, on obtient $d_j > a_{j-1}.H$ pour tout $j = l \ldots i-1$, ce qui implique $d_i \geq (t_{i+1_{min}} - \sum_{j=l}^{i-1} a_{j-1}.H) = \alpha_{i_{t_{i+1}}}$.

4.4.1 Bisimulation TAG-TTS

Dans le but d'associer une transition $a \xrightarrow{t} a'$ d'un TAG $G = \langle \mathcal{A}, T, a_0, \delta \rangle$ à une transition $(m, V) \xrightarrow{d} (m, V') \xrightarrow{t} (m', V'')$ de $\mathcal{S}_{\mathcal{N}}$, et inversement, on définit la relation de bisimulation suivante :

Définition 4.4.3 (Bisimulation TAG-TTS) *Soit $\mathcal{N}$ un TPN et soit $G = \langle \mathcal{A}, T, a_0, \delta \rangle$ le TAG associé à $\mathcal{N}$. Alors,*

- *Pour toute transition* $a_i \xrightarrow{t} a_{i+1}$ *dans le TAG,* $\forall d \in \mathbb{R}_{\geq 0}$ *tel que* $d \leq a_i.H$*, alors existe une transition continue* $(m, V) \xrightarrow{d} (m, V')$ *suivie d'une transition discrète* $(m, V') \xrightarrow{t} (m', V'')$ *dans* $\mathcal{N}$*, tel que* $m = a_i.m$ *et* $d \in [a_{i-1}.h; a_{i-1}.H]$.
- *Pour toute transition continue* $(m, V) \xrightarrow{d} (m, V')$ *suivie d'une transition discrète* $(m, V') \xrightarrow{t} (m', V'')$ *dans* $\mathcal{N}$*, alors il existe une transition* $a_i \xrightarrow{t} a_{i+1}$ *dans le TAG telle que :* $a_i.m = m$ *et* $d \in [a_{i-1}.h; a_{i-1}.H]$, $\forall d \in \mathbb{R}_{\geq 0}$ *tel que* $d \leq a_i.H$.

4.5 Algorithme de construction du TAG

Dans cette section, nous présentons les algorithmes utilisés pour construire un TAG à partir d'un TPN. Avant de présenter l'algorithme de construction du TAG, nous présentons tout d'abord les fonctions utilisées par ce dernier.

- `Agrégat-Initial(`$\mathcal{N}, m_0$`)` : permet de calculer l'agrégat initial, elle est donnée par L'Algorithme 1. Le calcul des attributs de l'agrégat initial a_0

Algorithme 1 Calcul de l'agrégat initial

1: `Entrées: TPN` $\mathcal{N}$ `et le marquage initial` m_0
2: $a_0.m \leftarrow m_0$, $a_0.h \leftarrow \infty$, $a_0.H \leftarrow 0$, $a_0.E \leftarrow \emptyset$
3: **pour tout** $t \in Enable(m_0)$ **faire**
4: $\quad a_0.E \leftarrow a_0.E \cup \{\langle t, t_{\min}, t_{\max}\rangle\}$
5: $\quad$ **si** $t_{\min} < a_0.h$ **alors**
6: $\quad\quad a_0.h \leftarrow t_{\min}$
7: $\quad$ **si** $t_{\max} > a_0.H$ **alors**
8: $\quad\quad a_0.H \leftarrow t_{\max}$
9: `Retourner` $\langle a_0.m, a_0.h, a_0.H, a_0.E\rangle$

est fait à partir des donnés statiques du TPN comme suit : $a_0.m$ est le marquage initial du TPN, l'attribut $a_0.E$ est l'ensemble des transitions sensibilisées par $a_0.m$ avec leurs intervalles de tir statiques (c'est-à-dire le t_{min} et le t_{max} pour chaque transition sensiblisé t), et enfin le temps de séjour minimum (respectivement maximum) $a_0.h$ (respectivement $a_0.H$) est le minimum de des t_{min} (respectivement t_{max}) pour les transitions appartenant à $a_0.E$

- `Franchissable(`$\mathcal{N}, a, t_i$`)` : consiste à tester si une transition sensibilisée dans un agrégat a est franchissable. L'Algorithme 2 illustre le test de la franchissabilé d'une transitions t_i dans un agrégat a donné. La transition t est franchissable à partir d'un agrégat a SSi : t est activée dans a, son délai de tir dynamique au plus tôt α_{t_i} est inférieur au temps de séjour

Algorithme 2 Test de franchissabilité de la transition t_i dans un agrégat a

```
Entrées: un TPN 𝒩, un agrégat a et la transition t_i
si α_{t_i} ≤ a.H alors
    pour tout ⟨t, α_t, β_t⟩ ∈ a.E faire
        si t_{i_min} ≤ t_max ∨ (t_{i_min} − α_{t_i}) − (t_min − α_t) ≥ (t_{i_min} − t_max) alors
            retourner vrai
retourner faux
```

minimum dans a et il n'y aucune autre transition qui doit être franchie avant t (condition sur l'ordre de franchissement entres transitions).

– `Existe(`a_i`, Reach)` : consiste à chercher dans l'ensembles des agrégats déjà atteints `Reach` un agrégat a équivalent à a_i selon la relation d'équivalence définie dans la Définition 4.2.3.

Algorithme 3 Test d'existence de l'agrégat a_i dans `Reach`

Précondition : `Un TPN` $\mathcal{N}$`, un agrégat` a `et la transition` t_i

```
pour tout a ∈ Reach faire
    si a ≡ a_i alors
        retourner vrai
    sinon
        retourner faux
```

– `Successeur(` $\mathcal{N}, a, t_i$ `)` : permet de calculer les successeur de l'agrégat a par le franchissement de la transition t_i, elle est illustrée par L'Algorithme 4.

La construction du successeur de a par le tir de t_i, noté a_i, consiste à calculer le marquage $a_i.m$, les temps de séjour minimum et maximum $a_i.h$ et $a_i.H$ et l'attribut $a_i.E$ qui représente l'ensemble des transitions sensibilisées dans a_i avec leurs intervalles de tir dynamiques.

La construction du TAG est illustrée par l'Algorithme 5. L'appel à la fonction `Agrégat-Initial(`$\mathcal{N}, m_0$`)` permet de calculer l'agrégat initial a_0. L'ensemble des agrégats accessibles, noté `Reach`, et l'ensemble des arcs du TAG, noté `Arcs`, sont initialisés à l'ensemble vide et l'ensemble des agrégats à analyser, noté `Wait`, contient initialement l'agrégat initial a_0. Chaque agrégat dans `Wait` est analysé dans le but de construire ses éventuels successeurs. L'analyse d'un agrégat a est faite par l'appel des deux fonctions `Franchissable(`$\mathcal{N}, a, t_i$`)` et `Successeur(`$\mathcal{N}, a, t_i$`)`. À chaque fois qu'un nouvel agrégat a_i est atteint, l'algorithme teste s'il existe déjà dans `Reach` en utilisant à la fonction `Existe(`a_i`, Reach)`. Si un agrégat équivalent à a_i existe dans `Reach`, alors a_i ne sera pas exploré. Sinon, l'agrégat a_i est ajouté à `Wait` pour être analysé ultérieurement. Enfin, à chaque fois qu'un agrégat est analysé, il

Algorithme 4 Calcul du successeur de a par le tir de t_i

1: `Entrées: un TPN` $\mathcal{N}$`, un agrégat` a `et la transition` t_i
2: $a_i.m \leftarrow a.m - Pre(t_i) + Post(t_i)$
3: $a_i.h \leftarrow \infty, \quad a_i.H \leftarrow 0, \quad a_i.E \leftarrow \emptyset$
4: **pour tout** $t' \in Enable(a_i.m)$ **faire**
5: **si** $\langle t', \alpha_{t'}, \beta_{t'} \rangle \in a.E$ **alors**
6: $\alpha_{t'} \leftarrow \alpha_{t'} - a.H$
7: $\beta_{t'} \leftarrow \beta_{t'} - max(0, \alpha_{t_i})$
8: **sinon**
9: $\alpha_{t'} \leftarrow t'_{\min}$
10: $\beta_{t'} \leftarrow t'_{\max}$
11: **si** $\alpha_{t'} < a_i.h$ **alors**
12: $a_i.h \leftarrow \alpha_{t'}$
13: **si** $\beta_{t'} > a_i.H$ **alors**
14: $a_i.H \leftarrow \beta_{t'}$
15: $a_i.E \leftarrow a_i.E \cup \{\langle t', \alpha_{t'}, \beta_{t'} \rangle\}$
16: `retourner` $\langle a_i.m, a_i.h, a_i.H, a_i.E \rangle$

est ajouté à `Reach`.

4.6 Expérimentations

L'algorithme de construction du TAG a été implémenté dans un prototype d'outil que nous avons utilisé pour effectuer des expérimentations afin de comparer la taille des graphes générés par l'approche. Le prototype implémenté nous a permis d'avoir une première comparaison avec les approches existantes en terme de taille des graphes obtenus. Il est important de noter que le prototype n'est pas encore optimisé en terme de temps de calcul, et c'est pour cette raison que le paramêtre "temps de construction" n'est pris en compte dans les résultats expérimentaux donnés dans ce chapitre.

4.6.1 Modèles TPN considérés

Nous avons testé notre approche sur plusieurs modèles de TPN, et nous rapportons ici les résultats obtenus pour 4 modèles . Les modèles considérés sont représentatifs des caractéristiques que peut avoir un TPN, tels que : la concurrence, la synchronisation, les intervalles de tir disjoints et les bornes de tir infinies. Les deux premiers modèles (Figure 4.9 et Figure 4.10) sont deux modèles paramétrés où le nombre de processus peut être augmentés. Dans la Figure 4.9, c'est le nombre de boucles $p_n \rightarrow t_n \rightarrow p_n$ ($n \geq 4$) qui est incrémenté

Algorithme 5 Construction du TAG

```
Entrées: un TPN 𝒩 et le marquage initial m0
a0 ← Agrégat-Initial(𝒩, m0)
Reach ← ∅,   Arcs ← ∅,   Wait ← {a0}
tant que Wait ≠ ∅ faire
  a ← dépiler (Wait)
  Reach ← Reach ∪ {a}
  pour tout ⟨ti, αti, βti⟩ ∈ a.E faire
    si Franchissable(𝒩, a, ti) alors
      ai ← Successeur(𝒩, a, ti)
      si Existe(ai, Reach) alors
        Wait ← Wait ∪ {ai}
        Retourner ⟨ Reach, Arcs ⟩
  Arcs ← Arcs ∪ {(a, ti, ai)}
```

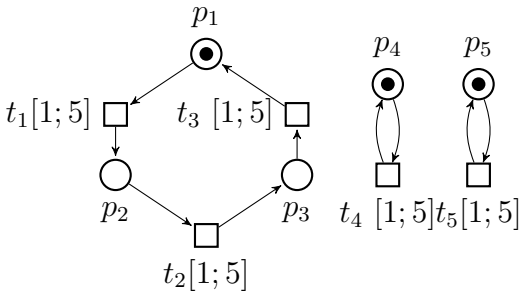


FIGURE 4.9 – Exemple d'un TPN avec des concurrences

alors que dans la Figure 4.10, c'est le nombre de processus qui est augmenté. Dans cet exemple, un processus a un comportement local, avec la transition t_i, ou global par synchronisation avec tous les autres processus, avec la transition t_0. En plus de ces deux exemples, nous avons testé deux autres modèles de TPN paramétrés qui sont connus dans la littérature. Le premier (Figure 4.11), le modèle est disponible dans [22]) représente une composition de modèles de producteurs/consommateurs par la fusion d'un tampon mémoire unique (de taille 5). Enfin, nous avons considéré un modèle TPN du protocole de Fischer pour l'exclusion mutuelle (Figure 4.12). Ce protocole met en place n processus qui s'exécutent de facon concurrente, et une variable globale à tous les processus. Cette variable peut prendre $n+1$ valeurs, une pour chacun des processus plus une valeur neutre. Le modèle TPN de la Figure 4.12 est une version adaptée de celui présenté dans [40] auquel nous avons apporté quelques corrections en retirant les transitions mortes et corrigeant les situations d'interblocage.

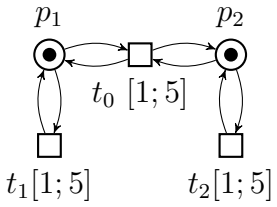


FIGURE 4.10 – Exemple d'un TPN avec des synchronisations

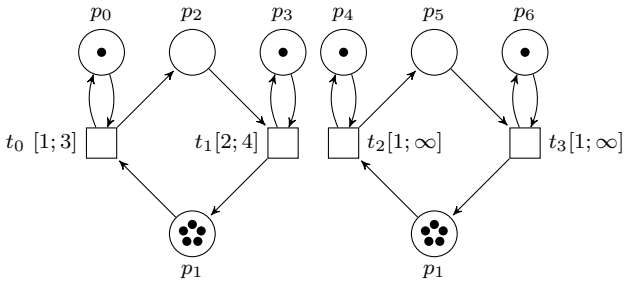


FIGURE 4.11 – Modèles TPN de producteurs/consommateurs

4.6.2 Résultats obtenus

Les Tableaux 4.1, 4.2, 4.3 et 4.4 rapportent les résultats obtenus avec les approches SCG, le ZBG et TAG. Les résultats obtenus pour le modèle producteurs/consommateurs (Tableau 4.1) montrent que le TAG donne une meilleure abstraction (ordre linéaire) en terme de taille des graphes par rapport aux graphes SCG et ZBG.

À chaque fois qu'un nouveau module de producteur/consommateur est introduit, la taille des graphes augmente pour les trois approches. Toutefois, le SAG réalise une meilleure performance par rapport aux deux autres approches.

Pour le TPN de la Figure 4.9, les résultats obtenus montrent que la taille du TAG augmente de façon exponentielle lorsque le parallélisme se produit dans la structure du TPN (Tableau 4.2).

Il en est de même pour le ZBG et le SCG, et nous pouvons voir que notre méthode se comporte mieux quand on incrémente le nombre de boucles dans le modèle. Les exécutions du ZBG et du SCG sont arrêtées à 5 boucles en raison d'un manque d'espace mémoire. Le nombre d'arcs dans les graphes obtenus suit la même proportion.

Avec l'exemple de synchronisation (Figure 4.3), le TAG se comporte aussi bien. En effet, avec 1, 2 et 3 processus, la taille des graphes obtenus sont presque similaires avec les trois approches. Mais, à partir de 4 processus syn-

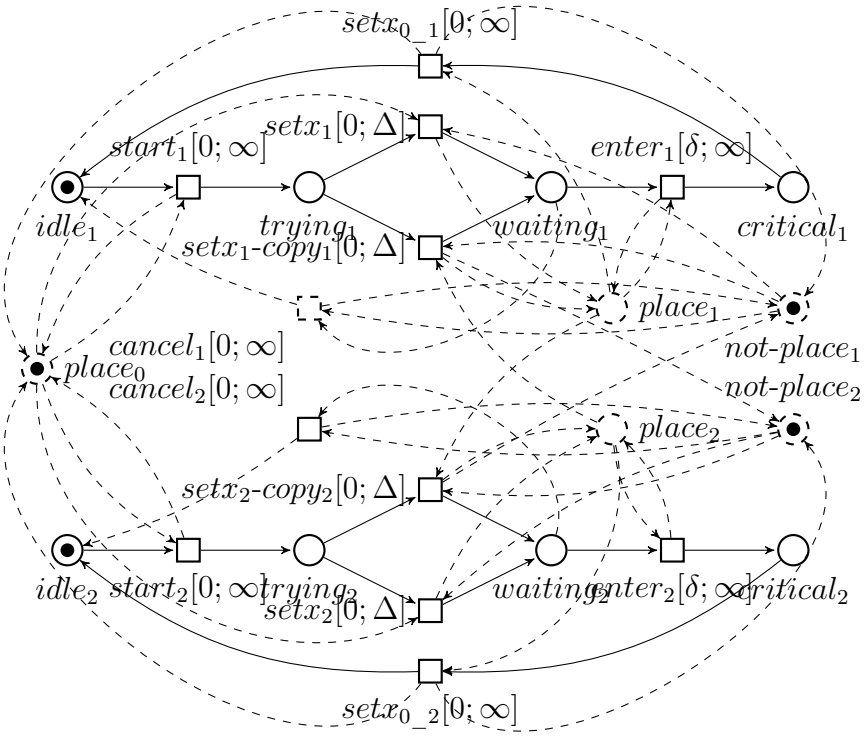


FIGURE 4.12 – Modèle TPN du protocole de Fischer pour l'exclusion mutuelle

Nombre de prod/cons	SCG (avec Tina) (noeuds / arcs)	ZBG (avec Romeo) (noeuds / arcs)	TAG-TPN (noeuds / arcs)
1	34 / 56	34 / 56	34 / 56
2	748 / 2460	593 / 1 922	407 / 1 255
3	4 604 / 21891	3 240 / 15 200	1 618 / 6 892
4	14 086 / 83 375	9 504 / 56 038	3 972 / 20 500
5	31 657 / 217 423	20 877 / 145 037	8 175 / 48 351
6	61 162 / 471 254	39 306 / 311 304	15 157 / 99 539
7	107 236 / 907 708	67 224 / 594 795	26 113 / 186 363
8	175 075 / 1 604 319	107 156 / 1 044 066	42 503 / 324 600
9	270 632 / 2 655 794	161 874 / 1 718 104	66 103 / 534 055
10	400 648 / 4 175 413	234 398 / 2 687 147	99 036 / 839 011

TABLE 4.1 – Résultats expérimentaux pour le modèle producteur/consommateur (Figure 4.11)

Nombre de self-loops	SCG (avec Tina) (noeuds / arcs)	ZBG (avec Romeo) (noeuds / arcs)	TAG-TPN (noeuds / arcs)
1	1 / 2	2 / 4	1 / 2
2	13 / 35	14 / 38	13 / 35
3	157 / 553	158 / 557	118 / 409
4	2 245 / 10 043	2 246 / 10 048	915 / 3 909
5	3 9781 / 21 7681	39 782 / 217 687	6 496 / 33 071
6	848 893 / 5 495 603	848 894 / 5 495 610	43 637 / 258 051
7	? / ?	? / ?	282 514 / 1.90282e+06

TABLE 4.2 – Résultats expérimentaux pour le TPN avec des concurrences (Figure 4.9)

chronisés, la taille des SCGs et ZBGs augmentent de façon exponentielle, ce qui conduit à une explosion de l'espace d'états avec 7 processus, tandis que les TAGs ont été calculés avec succès avec 7 processus (et même plus).

Le modèle TPN du protocole Fischer est le seul modèle où notre approche conduit relativement à de mauvais résultats (bien que la différence avec les deux autres approches soit linéaire) (Tableau 4.4).

Une première explication est que, dans le cas où les intervalles de tir sont disjoints : le TAG se comporte moins bien pour certains cas. En effet, quand une transition t est activée par un agrégat a et qu'il existe une transition t', non

Nombre de processus	SCG (avec Tina) (noeuds / arcs)	ZBG (avec Romeo) (noeuds / arcs)	TAG-TPN (noeuds / arcs)
1	39 / 72	40 / 74	39 / 72
2	471 / 1 296	472 / 1 299	354 / 963
3	6 735 / 25 056	6 736 / 25 060	2 745 / 9 888
4	119 343 / 563 040	119 344 / 563 045	19 488 / 87 375
5	2 546 679 / 14 564 016	? / ?	130 911 / 701 748

TABLE 4.3 – Résultats expérimentaux pour le TPN avec des synchronisations (Figure 4.10)

Nombre de processus	SCG (avec Tina) (noeuds / arcs)	ZBG (avec Romeo) (noeuds / arcs)	TAG-TPN (noeuds / arcs)
1	4 / 4	4 / 4	4 / 4
2	18 / 29	19 / 32	20 / 32
3	65 / 146	66 / 153	80 / 171
4	220 / 623	221 / 652	308 / 808
5	727 / 2 536	728 / 2 615	1 162 / 3 645
6	2 378 / 9 154	2 379 / 10 098	4 274 / 15 828
7	7 737 / 24 744	7 738 / 37 961	15 304 / 66 031
8	25 080 / 102 242	25 081 / 139 768	53 480 / 265 040

TABLE 4.4 – Résultats expérimentaux pour le protocole de Fischer (Figure 4.12)

activée par a, tel que $t_{min} > t'_{max}$, a est considéré comme non équivalent (bien qu'il puisse l'être) à tous les agrégats où le temps de tir au plus tôt de t n'est pas le même. Cependant, les transitions t et t' peuvent ne jamais être activées simultanément (la différence ne joue aucun rôle dans ce cas). Cette question sera étudiée et améliorée de sorte que certaines propriétés structurelles du modèle soient prises en compte pour affiner la relation d'équivalence entre agrégats.

4.7 Conclusion

Les résultats expérimentaux montrent (en général) un gain important en performances en termes de taille du graphe (noeuds/arcs) par rapport aux approches SCG et ZBG pour les exemples testés. Nous sommes conscients

que notre approche doit être confronté à des applications plus importantes, mais les résultats préliminaires obtenus sont prometteurs.

En outre, dans les approches existantes vues dans le chapitre 4, les informations sur l'écoulement de temps n'apparaissent pas explicitement dans les nœuds. Ceci conduit à des calculs supplémentaires et complexes tels que : la manipulation des DBM pour encoder les zones (pour les approches basées sur les zones) et les classes (pour les approches basées sur les classes d'états), conversion des graphes en automates temporisés (avec UPPAAL) pour la vérification de propriétés, etc. Avec notre approche, les informations temporelles sont encodées dans les agrégats permettant ainsi de vérifier les propriétés temporisées par un simple parcours du graphe, ce qui a un impact significatif sur la complexité à la fois de la construction et de la vérification.

L'encodage de l'information temporelle dans les agrégats est fait de telle manière que les temps minimum et maximum écoulés sur chaque chemin du TAG puissent être calculés à tout moment. Un autre avantage du TAG est la préservation de séquences du TPN correspondant ainsi que les états accessibles, ce qui permet la préservation des logiques basées sur les événements et sur les états (et ce pour les deux logiques, linéaire et arborescente). Grâce aux résultats de préservation, le TAG peut aisément être utilisé, par exemple, pour vérifier si un marquage (respectivement une transition) donné est accessible (respectivement est franchissable) avant (ou après) un certain délai. La vérification de propriétés en exploitant la structure du TAG est le sujet du prochain chapitre.

Chapitre 5

Vérification basée sur le TAG

5.1 Introduction

Dans le chapitre précédent, nous avons présenté un des principaux résultats de notre approche : à chaque exécution du TAG correspond une séquence de franchissement dans le TPN associé, et vice-versa. Autrement dit, le TAG est une représentation exacte du comportement du TPN associé. Dans ce chapitre, nous étendons, dans un premier temps, ce résultat à l'accessibilité temporisée des marquages et des séquences. Nous montrons par la suite que le TAG préserve les formules de la logique TCTL. Enfin, nous proposons des algorithmes de vérification pour quelques formules TCTL, nous prouvons leur correction et leur convergence.

5.2 Temps d'accessibilité et temps de franchissement

Dans cette section, nous présentons et appuyons par des preuves les deux résultats suivants : (1) le TAG préserve l'accessiblité temporisée des marquages du TPN associé. (2) La durée d'une séquence de franchissement d'un TPN peut être borné en utilisant les informations encodées dans les agrégats du TAG associé.

La Définition 5.2.1 introduit le temps écoulé pour atteindre un marquage (respectivement pour franchir une transition) dans une exécution $overline\pi$.

Définition 5.2.1 *Soient un TPN $\mathcal{N}$ et soit $\overline{\pi} = (m_0, V_0) \xrightarrow{(d_1,t_1)} (m_1, V_1) \rightarrow \ldots \xrightarrow{(d_n,t_n)} (m_1, V_1) \xrightarrow{d}$ une exécution de $\mathcal{N}$. Le temps d'accessibilité d'un marquage m dans $\overline{\pi}$, noté $AT_{\overline{\pi}}(m)$ (pour Access Time) et le temps de franchissement d'une transition t dans $\overline{\pi}$, noté $FT_{\overline{\pi}}(t)$ (pour Firing Time), sont définis comme suit :*

- $AT_{\overline{\pi}}(m_0) = 0$
- $\forall 1 \leq i \leq n,\ AT_{\overline{\pi}}(m_i) = FT_{\overline{\pi}}(t_i) = \sum_{k=1}^{i} d_k$.

De même, la Définition 5.2.2 introduit le temps écoulé pour atteindre un agrégat (respectivement pour franchir une transition) dans un chemin π du TAG.

Définition 5.2.2 *Soit $\mathcal{N}$ un TPN et soit $G = \langle \mathcal{A}, T, a_0, \delta \rangle$ le TAG associé à $\mathcal{N}$. Soit $\pi = m_0 \xrightarrow{(d_1,t_1)} m_1 \xrightarrow{(d_2,t_2)} \dots \xrightarrow{(d_n,t_n)} m_n$ un chemin dans G. Alors, $\forall i = 0 \dots n$:*

- *Les temps minimum et maximum d'accessibilité d'un agréget a_i, notés $MinAT_\pi(a_i)$ et $MaxAT_\pi(a_i)$ respectivement, sont définis comme suit :*
  - $MinAT(a_0) = 0$ *et* $MaxAT(a_0) = a_0.H$
  - $MinAT_\pi(a_i) = MinAT_\pi(a_{i-1}) + \max(0, \beta_{i-1_{t_i}} - (t_{i_{max}} - t_{i_{min}}))$
  - $MaxAT_\pi(a_i) = MaxAT(a_{i-1}) +$
    $\min(\min_{t \in \uparrow(a_{i-1},t_i)}(t_{\min}), \min_{t \in \downarrow(a_{i-1},t_i)}(\beta_{i-1_t} - a_{i-1}.H))$
- *Les temps minimum et maximum de franchissement d'une transition t_i, notés $MinFT_\pi(t_i)$ et $MaxFT_\pi(t_i)$ respectivement, sont définis comme suit :*
  - $MinFT_\pi(t_i) = MinAT_\pi(a_i)$
  - $MaxFT_\pi(t_i) = MaxAT_\pi(a_{i-1})$

5.2.1 Marquages et séquences

Les résultats présentés dans la Proposition 5.2.3 permettent de borner le temps écoulé avant d'atteindre un marquage (respectivement le franchissement d'une transition). Ces bornes sont données en fonction des attributs de l'agrégat contenant le marquage correspondant (respectivement la transition correspondante).

Proposition 5.2.3 *Soit $\mathcal{N}$ un TPN et soit $G = \langle \mathcal{A}, T, a_0, \delta \rangle$ le TAG associé à $\mathcal{N}$. Soit l'exécution $\pi = a_0 \xrightarrow{t_1} a_1 \rightarrow \dots \xrightarrow{t_n} a_n$ dans le TAG G et soit $\overline{\pi} = (m_0, V_0) \xrightarrow{(d_1,t_1)} (m_1, V_1) \rightarrow \dots \xrightarrow{(d_n,t_n)} (m_n, V_n) \xrightarrow{d}$ une séquence correspondante dans le TPN $\mathcal{N}$. Alors, $\forall i = 1 \dots n$:*

- *Le temps d'accessibilité d'un marquage m_i est compris entre les temps minimum et maximum d'accessibilité de l'agréget correspondant. Formelement, $minAT_\pi(a_i) \leq AT_{\overline{\pi}}(m_i) \leq maxAT_\pi(a_i)$, avec $a_i.m = m_i$;*
- *Le temps de franchissement d'une transition t_i dans N est compris entre les temps minimum et maximum de son franchissement dans G. Formelement, $minFT_\pi(t_i) \leq FT_{\overline{\pi}}(t_i) \leq maxFT_\pi(t_i)$.*

Ces résultats nous conduisent à énoncer le Corollaire 5.2.4 qui établit une équivalence d'accessibilité temporisée entre un TPN et le TAG associé.

Corollaire 5.2.4 *Soit $\mathcal{N}$ un TPN et soit $G = \langle \mathcal{A}, T, a_0, \delta \rangle$ le TAG associé à $\mathcal{N}$. Soient un marquage m, une transition t et un délai $d \in \mathbb{R}_{\geq 0}$. Alors :*

- *Le marquage m est accessible* ***avant*** *(respectivement* ***après****) d unités de temps si et seulement s'il existe une exécution $\pi = a_0 \xrightarrow{t_1} a_1 \rightarrow \ldots \xrightarrow{t_n} a_n$ dans G telle que $a_n.m = m$ et $d \leq maxAT_\pi(a_n)$ (respectivement $d \leq minAT_\pi(a_n)$) ;*
- *La transition t est franchissable* ***avant*** *(respectivement* ***après****) d unités de temps si et seulement s'il existe une exécution $\pi = a_0 \xrightarrow{t_1} a_1 \rightarrow \ldots \xrightarrow{t_n} a_n$ dans G telle que $t_n = t$ et $d \leq maxFT_\pi(t_n)$ (respectivement $d \leq minFT_\pi(t_n)$).*

5.2.2 Préservation des formules TCTL

L'équivalence d'accessibilité entre un TAG et le TPN associé est valable pour des propriétés basées sur les états comme sur les événements. On peut par exemple vérifier si un marquage (respectivement une transition) est accessible (respectivement franchissable) avant (ou après) un certain délai. La Proposition 5.2.5 permet d'établir, d'une manière plus générale, que : si une propriété d'accessibilité temporisée est vérifiée dans le TTS d'un TPN, alors elle l'est aussi dans le TAG associé et vice-versa.

Proposition 5.2.5 *Soit $\mathcal{N}$ un TPN et soit $G = \langle \mathcal{A}, T, a_0, \delta \rangle$ le TAG associé à $\mathcal{N}$, et soit une formule TCTL ϕ. Alors, ϕ est vérifiée sur $\mathcal{N}$ si et seulement si ϕ est vérifiée sur G.*

Le Théorème 4.4.2 établit que le TAG préserve les marquages et les séquences. De plus, nous avons vu dans le Corollaire 5.2.4 que le TAG préserve aussi les traces du temps écoulé. La Proposition 5.2.3 permet de borner le temps écoulé pour une exécution d'un TPN. Ces bornes sont calculées en fonction des informations encodées dans les agrégats du TAG. Il s'ensuit que le TAG n'affecte ni les marquages ni les durées et peut donc être utilisé pour vérifier si un marquage est accessible dans une fenêtre de temps ou si une transition est franchissable dans un intervalle temporel.

5.3 Analyse de l'accessibilité temporisée des marquages

Dans cette section, nous décrivons la démarche pour la vérification des propriétés d'accessibilité temporisée. L'idée générale est d'utiliser l'algorithme de construction du TAG pour vérifier à la volée l'accessibilité d'un marquage M entre d et D unités de temps. Notons que M n'est pas nécessairement un marquage, mais peut être une contrainte sur un marquage, par exemple, une

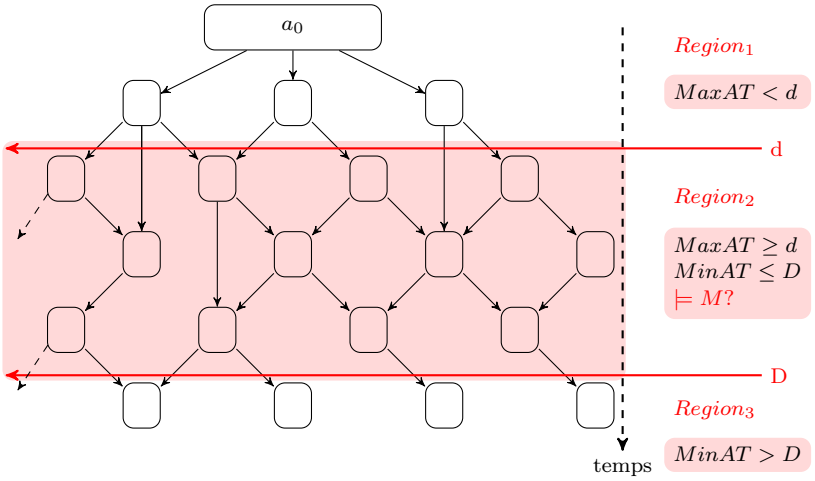


FIGURE 5.1 – Analyse de l'accessibilité temporisée basée sur la TAG

contrainte de la forme $m(p_i) < c$, pour un marquage m, une place p_i et une constante $c \in \mathbb{N}$.

Afin d'intégrer la vérification d'accessibilité de M durant la construction du TAG, nous associons à chaque agrégat a_i les temps d'accessibilité minimum et maximum $minAT(a_i)$ et $maxAT(a_i)$ respectivement. Ainsi, chaque nœud du graphe est représenté par le triplet $\langle a_i, minAT(a_i), minAT(a_i)\rangle$. Puisque les nœuds du graphe sont des agrégats et leurs temps d'accessibilité, le graphe prend la forme d'un arbre dont les feuilles sont situées par rapport à un axe de temps (comme c'est illustré dans la Figure 5.1).

La première région ($Region_1$) contient les agrégats accessibles strictement avant d unités de temps. La seconde région ($Region_2$) contient les agrégats accessibles entre d et D unités de temps. Enfin, la troisième région ($Region_3$) contient les agrégats accessibles strictement après D unités de temps. Il est important de noter que si $D = \infty$, la troisième région est vide. Les nœuds de l'arbre sont traités en fonction de la région à laquelle ils appartiennent. Ainsi, seuls les agrégats appartenant à $Region_2$ sont analysés en fonction du marquage M. La région $Region_1$ est explorée afin de calculer les temps d'accessibilité maximum et minimum des agrégats parcourus sans que le marquage M ne soit testé. La région $Region_3$ n'est jamais explorée puisque dès qu'un agrégat appartenant à cette région est atteint, l'exploration du chemin courant est arrêté. Ci-dessous, nous décrivons la demarche pour la vérifictaion des propriétés d'accessibilité usuelles que nous avons formulé dans la logique TCTL.

Vérification de $\exists\Diamond_{[d;D]}M$: consiste à vérifier si le marquage M est accessible entre les instants entre d et D unités de temps (Algorithme 9).

Si l'agrégat courant a_i appartient à $Region_2$, on vérifie alors si son marquage est égal à M. Si c'est est le cas, la construction est arrêtée et

Algorithme 6 Check $\exists\Diamond_{[d;D]}M$

Précondition : un TPN N **et le marquage** M

```
1: calculer l'agrégat initial a_0
2: ajouter a_0 à wait
3: tant que wait ≠ ∅ faire
4:    soit π = a_0 --t_0--> a1 ... --t_{i-1}--> a_i ... le chemin courant
5:    si a_i ∈ Region_1 alors
6:       ajouter a_i au chemin courant
7:    sinon
8:       si a_i ∈ Region_2 alors
9:          si a_i = M alors
10:            retourner vrai
11:         sinon
12:            ajouter a_i au chemin courant
13:      sinon
14:         changer de chemin à partir de a_{i-1}
15: retourner faux
```

l'algorithme retourne **vrai** indiquant que le marquage M est accessible entre les instants d et D. Sinon, la construction continue en calculant un nouveau triplet $\langle a_{i+1}, minAT(a_{i+1}), minAT(a_{i+1})\rangle$ sur le chemin courant.

Si l'agrégat courant a_i appartient à $Region_3$, la construction du chemin courant est arrêtée et un nouveau chemin partant de l'agrégat précedent est exploré. Lorsque tous les nœuds de la $Region_2$ sont construit, l'algorithme retourne **faux** signifiant que le marquage M n'est pas accessible entre les instants d et D.

Vérification de $\forall\Box_{[d;D]}M$: la vérification de cette propriété revient à vérifie si le marquage M est accessible dans tous les états situés entre les instants d et D unités de temps (Algorithme 7).

Si l'agrégat courant a_i appartient à $Region_2$, on vérifie alors si son marquage est égal à M. Si ce n'est pas le cas, la construction est arrêtée et l'algorithme retourne **faux**. indiquant que a_i est situé entre les instants d et D mais ne satisfait pas le marquage M. Sinon, un nouveau triplet $\langle a_{i+1}, minAT(a_{i+1}), minAT(a_{i+1})\rangle$ successeur de $\langle a_i, minAT(a_i), minAT(a_i)\rangle$ est calculé sur le chemin courant.

Si maintenant a_i appartient à $Region_3$, la construction du chemin courant est arrêtée et un nouveau chemin partant de l'agrégat précedent

Algorithme 7 Check $\forall\Box_{[d;D]}M$

Précondition : un TPN N **et le marquage** M

```
1: calculer l'agrégat initial a_0
2: ajouter a_0 à wait
3: tant que wait ≠ ∅ faire
4:    soit π = a_0 --t_0--> a1 ... --t_{i-1}--> a_i ... le chemin courant
5:    si a_i ∈ Region_1 alors
6:       ajouter a_i au chemin courant
7:    sinon
8:       si a_i ∈ Region_2 alors
9:          si a_i = M alors
10:            ajouter a_i au chemin courant
11:         sinon
12:            retourner faux
13:      sinon
14:         changer de chemin à partir de a_{i-1}
15: retourner vrai
```

est exploré. Lorsque tous les nœuds de la $Region_2$ sont construit, l'algorithme retourne **vrai** signifiant que le marquage M est toujours accessible entre les instants d et D.

vérification de $\forall\Diamond_{[d;D]}M$: consiste à vérifier si le long de chaque chemin, il est toujours possible d'atteindre le marquage M entre d et D unités de temps (Algorithme **??**).

Si l'agrégat a_i appartient à $Region_2$, on vérifie alors si son marquage est égal à M. Si c'est le cas, la construction du chemin courant est arrêtée et un nouveau chemin partant de l'agrégat précedent est exploré. Si ce n'est pas le cas, la construction continue en calculant un nouveau triplet $\langle a_{i+1}, minAT(a_{i+1}), minAT(a_{i+1})\rangle$ sur le chemin courant.

Si l'agrégat courant a_i appartient à $Region_3$, la construction est arrêtée et l'algorithme retourne **faux**. Lorsque tous les nœuds de la $Region_2$ auront été construits, l'algorithme retourne **vrai**.

Vérification de $\exists\Box_{[d;D]}M$: la vérification de cette propriété revient à vérifie l'existence d'un chemin situé entre d et D unités de temps où le marquage M est toujours accessible (Algorithme **??**).

Si l'agrégat courant a_i appartient à $Region_2$, on vérifie alors si son marquage est égal à M. Si c'est le cas, un nouveau triplet $\langle a_{i+1}, minAT(a_{i+1}), minAT(a_{i+1})\rangle$ successeur de $\langle a_i, minAT(a_i), minAT(a_i)\rangle$ est calculé sur le chemin courant. Sinon, la construction du chemin courant est arrêtée et un nouveau

Algorithme 8 Check $\forall\Diamond_{[d;D]}M$

Précondition : un TPN N **et le marquage** M

```
1: calculer l'agrégat initial a_0
2: ajouter a_0 à wait
3: tant que wait ≠ ∅ faire
4:    soit π = a_0 --t_0--> a1 ... --t_{i-1}--> a_i ... le chemin courant
5:    si a_i ∈ Region_1 alors
6:       ajouter a_i au chemin courant
7:    sinon
8:       si a_i ∈ Region_2 alors
9:          si a_i = M alors
10:            changer de chemin à partir de a_{i-1}
11:         sinon
12:            ajouter a_i au chemin courant
13:      sinon
14:         retourner faux
15: retourner vrai
```

Algorithme 9 Check $\exists\Box_{[d;D]}M$

Précondition : un TPN N **et le marquage** M

```
1: calculer l'agrégat initial a_0
2: ajouter a_0 à wait
3: tant que wait ≠ ∅ faire
4:    soit π = a_0 --t_0--> a1 ... --t_{i-1}--> a_i ... le chemin courant
5:    si a_i ∈ Region_1 alors
6:       ajouter a_i au chemin courant
7:    sinon
8:       si a_i ∈ Region_2 alors
9:          si a_i = M alors
10:            ajouter a_i au chemin courant
11:         sinon
12:            changer de chemin à partir de a_{i-1}
13:      sinon
14:         retourner vrai
15: retourner faut
```

chemin partant de l'agrégat précedent est exploré.
Si a_i appartient à $Region_3$, l'algorithme retourne **vrai**. Lorsque tous les nœuds de la $Region_2$ sont construit, l'algorithme retourne **faux** signifiant qu'il n'existe pas de chemin entre les instants d et D où M

est toujours accessible

5.4 Vérification de formules TCTL

Dans cette section, nous proposons des algorithmes pouvant être utilisé pour vérifier des propriétés de la logique temporisée sur des systèmes temps-réel. Pour cela, nous supposons que : (1) les comportements des systèmes sont décrits par des réseaux de Petri t-temporels et (2) la propriété que l'on souhaite vérifier est formulée dans la logique TCTL. Ainsi, les algorithmes de vérification doivent retourner **vrai** si la formule TCTL est satisfaite par le modèle TPN, sinon ils retournent **faux**. Les algorithmes de vérification à la volée, qui consistent à tester la propriété durant la construction de l'espace d'états, sont très adaptés pour les formules TCTL. Cette approche permet d'explorer uniquement les parties de l'espace d'états qui sont concernées par la propriété en cours de vérification. Cela justifie notre choix d'adopter une approche à la volée dans les algorithmes présentés ci-dessous.

5.4.1 Algorithmes de vérification

Soit une formule φ que l'on souhaite vérifier dans un intervalle de temps $[d; D]$. Comme pour l'accessibilité, la vérification s'effectue sur un TAG représenté sous forme d'un arbre partitionné en trois régions (Figure 5.2). Seuls les agrégats appartenant à $Region_2$ sont analysés en fonction de la formule en cours de vérification φ. La région $Region_1$ est explorée afin de calculer les temps d'accessibilité maximum et minimum des agrégats parcourus sans que la formule φ ne soit testée. La région $Region_3$ n'est jamais exploré puisque dès qu'un agrégat appartenant à cette région est atteint, l'exploration du chemin courant est arrêtée.

Nous illustrons notre approche pour la vérification des formules TCTL à travers les Algorithmes 10 et 11 qui vérifient la satisfaction des formule TCTL $\exists(\varphi U_{[d,D]}\psi)$ et $\forall(\varphi U_{[d,D]}\psi)$ respectivement. Les algorithmes pour la vérification des autres formules TCTL peuvent être déduits à partir de ces deux algorithmes.

Check $\exists(\varphi U_{[d,D]}\psi)$: vérifier cette formule revient à vérifier l'existence d'un chemin partant de l'état initial où tous les états satisfont φ et conduisant à un état qui satisfait ψ entre d et D unités de temps ($Region_2$). Soit $\pi = a_0 \xrightarrow{t_0} a1 \ldots \xrightarrow{t_{i-1}} a_i \ldots$ un chemin du TAG. Le long de π, l'algorithme analyse chaque agrégat a_i en fonction de la région à laquelle il appartient.

1. Si le nœud a_i appartient à la région $Region_1$, nous nous intéressons à la satisfaction de φ par a_i. Si a_i satisfait φ, alors il est ajouté au chemin

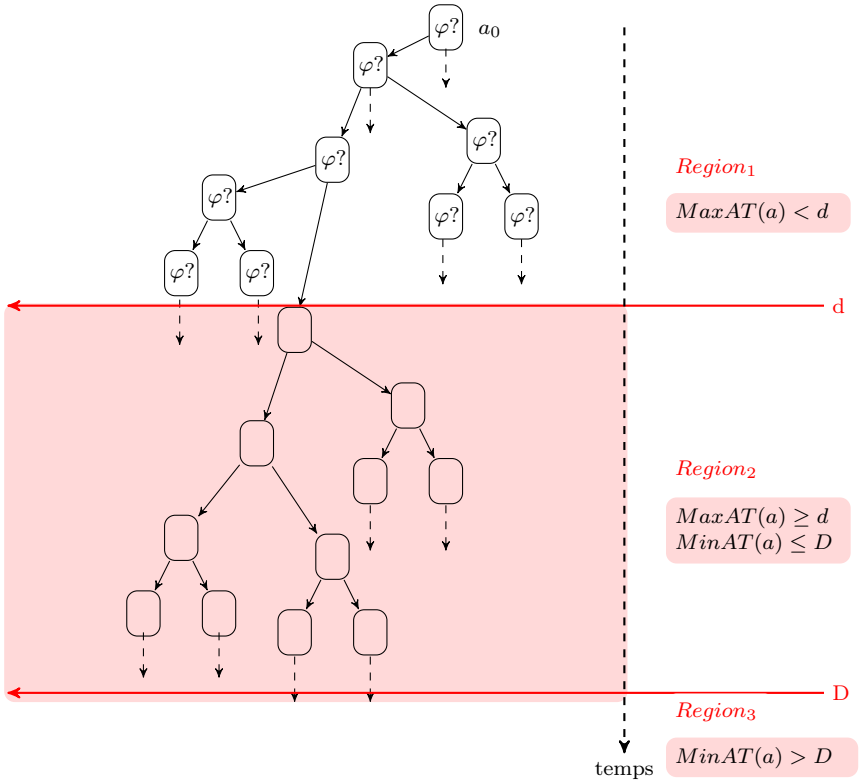


FIGURE 5.2 – Vérification basée sur le TAG

courant et l'exploration continue. Sinon, un nouveau chemin partant du prédécesseur de a est exploré.

2. Si le nœud a_i appartient à la région $Region_2$, nous nous intéressons à la satisfaction de ψ par a_i. Si a_i satisfait ψ, alors l'algorithme retourne **vrai** signifiant ainsi que le formule est satisfaite. Si a_i ne satisfait pas ψ, alors on vérifie si a_i satisfait φ, si c'est le cas, alors a_i est ajouté au chemin courant, sinon, un nouveau chemin partant de a_{i-1} est exploré.
3. Si le nœud a_i appartient à la région $Region_3$, un nouveau chemin partant de a_{i-1} est exploré. Une fois que tous les nœuds de la région $Region_2$ sont explorés, l'algorithme retourne **faux** indiquant que la propriété n'est pas satisfaite par le modèle.

Check $\forall(\varphi U_{[d,D]}\psi)$: l'Algorithme 11 vérifie si tous les chemins mènent à un état qui vérifie ψ entre d et D unités de temps ($Region_2$) et tous les états précédant cet état de chaque chemin vérifient φ. L'algorithme analyse l'agrégat a_i le long de chaque chemin $\pi = a_0 \xrightarrow{t_0} a1 \ldots \xrightarrow{t_{i-1}} a_i \ldots$ selon les étapes suivantes :

1. Si a_i appartient à la $Region_1$, alors nous nous intéressons à la satisfac-

Algorithme 10 Check $\exists(\varphi U_{[d,D]}\psi)$

Précondition : un TPN N **et les propriétés** φ **et** ψ

```
1: calculer l'agrégat initial a_0
2: ajouter a_0 à wait
3: tant que wait ≠ ∅ faire
4:    soit π = a_0 --t_0--> a1 ... --t_{i-1}--> a_i ... le chemin courant (dans wait)
5:    si a_i ∈ Region_1 alors
6:       si a_i ⊨ φ alors
7:          ajouter a_i au chemin courant
8:    sinon
9:       si a_i ∈ Region_2 alors
10:         si a_i ⊨ ψ alors
11:            retourner vrai
12:         sinon
13:            si a_i ⊨ φ alors
14:               ajouter a_i au chemin courant
15:            sinon
16:               changer de chemin à partir de a_{i-1}
17:      sinon
18:         changer de chemin à partir de a_{i-1}
19: retourner faux
```

tion de φ par a_i. Si a_i satisfait φ, il est ajouté au chemin courant et l'exploration du TAG continue. Sinon, l'algorithme retourne **Faux** et dans ce cas, le modèle ne vérifie pas la propriété.

2. Si a_i appartient à la $Region_2$, dans ce cas on s'intéresse à la satisfaction de ψ par a_i. Si a_i satisfait ψ, un nouveau chemin partant de a_{i-1} est exploré. Si a_i ne satisfait pas ψ, alors il est ajouté au chemin courant si il satisfait φ, sinon l'exploration est arrêté et l'algorithme retourne **faux**.

3. L'exploration est arrêté dans la cas où le nœud a_i appartient à la $Region_3$. Une fois que tous les nœuds de la $Region_2$ sont explorés, l'algorithme retourne **vrai**, ce qui signifie que la propriété est satisfaite par le modèle.

Dans le cas où la propriété à vérifier est imbriquée, les Algorithmes 10 et 11 sont appelés récurssivement à chaque fois qu'un nouvel agrégat est atteint. C'est-à-dire que lorsqu'on s'intéresse à vérifier une propriété ϕ sur un aggrégat a_i, les algorithmes sont relancés en considérant a_i comme étant un 'état initial.

Algorithme 11 Check $\forall(\varphi U_{[d,D]}\psi)$

Précondition : un TPN N **et les propriétés** φ **et** ψ

calculer l'agrégat initial a_0
ajouter a_0 à *wait*
tant que *wait* $\neq \emptyset$ **faire**
  soit $\pi = a_0 \xrightarrow{t_0} a1 \ldots \xrightarrow{t_{i-1}} a_i \ldots$ le chemin courant (dans *wait*)
  si $a_i \in Region_1$ **alors**
    si $a_i \models \varphi$ **alors**
      ajouter a_i au chemin courant
  sinon
    si $a_i \in Region_2$ **alors**
      si $a_i \models \psi$ **alors**
        changer de chemin à partir de a_{i-1}
      sinon
        si $a_i \models \varphi$ **alors**
          ajouter a_i au chemin courant
        sinon
          retourner **faux**
    sinon
      retourner **faux**
retourner **vrai**

5.4.2 Preuves de terminaison

Théorème 5.4.1 *Les Algorithmes 10 et 11 terminent.*

La terminaison des algorithmes 10 et 11 est garantie par des critères d'arrêt qui assurent leur terminaison.

– L'Algorithme 10 : **Check** $\exists(\varphi U_{[d,D]}\psi)$. Le but de cet algorithme est de chercher un chemin $(s_0, v_0) \to \cdots \to (s_n, v_n)$ où tous les états (s_i, v_i) vérifient φ et l'état (s_n, v_n) vérifie ψ dans un intervalle de temps $[d; D]$. L'algorithme procède par la construction du TAG et l'analyse des agrégats atteints le long de chaque chemin. L'analyse est effectuée en fonction de trois critères, qui sont : (1) l'intervalle de temps, (2) la satisfaction de ψ et (3) la satisfaction de φ. Soit a l'agrégat à analyser. Si la propriété ψ est vraie pour une valuation appartenant à l'intervalle $[d; D]$, l'algorithme retourne **vrai**, ceci constitue le premier critère d'arrêt. Si la propriété φ n'est pas vérifiée, l'algorithme n'explore pas les successeurs de cet agrégat sur le chemin courant, c'est le deuxième critère d'arrêt. Le troisième critère est le fait qu'il ne reste plus de temps pour que la propriété ψ soit satisfaite. Cela intervient lorsque tous les nœuds appar-

tenant à $Region_2$ sont atteints. Dans ce cas la construction est arrêtée et l'algorithme retourne **faux**.

– L'Algorithme 11 : **Check** $\forall(\varphi U_{[d,D]}\psi)$. L'algorithme consiste à vérifier que le long de chaque chemin $(s_0, v_0) \rightarrow \cdots \rightarrow (s_n, v_n)$, la propriété φ est satisfaite par les agrégats (s_i, v_i) et conduisant à un agrégat (s_n, v_n) qui vérifie ψ dans l'intervalle de temps $[d; D]$. Ceci consiste à explorer chaque agrégat du TAG et l'analyser par les étapes suivantes : (1) tant que l'intervalle $[d; D]$ n'est pas encore atteint, si un agrégat ne satisfait pas φ, alors l'algorithme retourne **faux**, ce qui constitue un premier critère d'arrêt. (2) Pendant l'intervalle $[d; D]$ tous les agrégats doivent satisfaire au moins une des deux propriétés φ et ψ afin de continuer l'exploration. Si un agrégat ne vérifie aucune des deux propriétés, l'algorithme retourne **faux**, ce qui en fait le deuxième critère d'arrêt. Le troisième critère d'arrêt est le fait que l'exploration s'arrête et l'algorithme retourne **vrai** lorsque tous les agrégats situés entre d et D ont été analysés.

Ces derniers résultats nous permettent d'affirmer que si un TPN $\mathcal{N}$ satisfait une formule TCTL ϕ, l'algorithme correspondant à la formule retournera **vrai**.

Théorème 5.4.2 *Soit un TPN $\mathcal{N}$ et soit une formule TCTL ϕ. Alors,*
*$\exists(\varphi U_{[d,D]}\psi)$ est vérifiée par $\mathcal{N}$ si et seulement si l'Algorithme 10 retourne **vrai**.*
*$\forall(\varphi U_{[d,D]}\psi)$ est vérifiée par $\mathcal{N}$ si et seulement si l'Algorithme 11 retourne **vrai**.*

5.4.3 Exemples d'application pour la vérification des formules TCTL

Nous avons construit le TAG associé au TPN du procédé de fabrication illustré dans la Figure 2.8. Nous rapportons aussi les graphe des classes d'états et des zones obtenus pour ce même modèle. Le TAG, le SCG et le ZBG de ce modèle sont illustrés dans les Annexes 7.1, 7.2 et 7.3 respectivement.

Les graphes obtenus sont tous les trois différents et contiennent 28 noeuds/50 arcs, 17 noeuds/34 arcs et 18 noeuds/38 arcs pour le TAG, le SCG et le ZBG respectivement. Ces résultats confirment notre hypothèse sur le fait que, dans le cas où les intervalles de tir sont disjoints, le TAG se comporte moins bien. Cependant, ces résultats confirment aussi que la structure des états dans le SCG et le ZBG est plus complexe que dans le TAG.

La satisfaction (ou la non satisfaction) des propriétés Φ_1 et Φ_2 par le TPN du procédé de fabrication (Figure 2.8), peut être vérifiée en appliquant les Algorithmes 7 et 9 respectivement.

Propriété Φ_1 : L' algorithme **Check** $(\forall\Box_{[9;11]} m(ProdF) = 1)$ retourne **vrai**. En effet, nous remarquons bien que dans le TAG du TPN mo-

délisant le processus de fabrication, le temps d'accessibilté de tous les agrégats où la place $ProdF$ est marquée est toujours compris entre 9 et 11 unités de temps. Ceci confirme la satisfaction de la propriété Φ_1.

Propriété Φ_2 : L' algorithme **Check** $(\exists\Diamond_{[0;9[}m(ProdF) = 1) \vee (\exists\Diamond_{]11;\infty]}m(ProdF) = 1)$ retourne **faux**. Ceci peut être vérifié sur le TAG du TPN du processus de fabrication, où nous remarquons qu'il n'existe aucun agrégat contenant un état où la place $ProdF$ est marquée et cela ni avant 9 ni après 11 unités de temps. Cela permet de confirmer la non satisfaction de la formule Φ_2.

5.5 Conclusion

Dans ce chapitre, nous avons démontré que le TAG préserve les contraintes temporelles, les marquages accessibles, les séquences et les propriétés TCTL. Nous avons montré que les temps de séjour minimum et maximum sauvegardés dans les nœuds permettent de borner les durées des séquence d'un TPN, et par conséquent, de vérifier des propriétés temporisées, et cela avec un simple parcours du TAG. Ces résultats nous ont conduit à proposer des algorithmes pour l'analyse de l'accessibilité temporisée des marquages et la vérifications des propriétés exprimées dans la logique TCTL. Nous avons enfin prouvé que ces algorithmes terminent. Notre principal objectif étant d'appliquer ces algorithmes pour des applications réelles, leurs implémentation est la première perspective qui sera étudiée à l'avenir.

Chapitre 6

Conclusion générale

Les systèmes temps-réel sont aujourd'hui présents dans de nombreux secteurs d'activités : dans l'automobile et l'aéronautique (systèmes de pilotage embarqués dans les voitures, avions et satellites), dans le domaine ferroviaire (systèmes de signalisation), dans l'industrie de production (systèmes de contrôle de procédés), dans la finance (traitement des données boursières en temps réel), ... etc. Les systèmes temps-réel se distinguent par la prise en compte de contraintes temporelles dont le respect est aussi important que l'exactitude de la modélisation. Autrement dit, le système ne doit pas simplement délivrer des résultats exacts, il doit les délivrer dans des délais imposés.

L'objet du premier chapitre a donc été de présenter les différentes techniques et modèles utilisés pour la modélisation et l'analyse des systèmes temps-réel. Nous avons fait remarquer que, pour garantir le respect des contraintes temporelles (c'est-à-dire le fait que chaque action soit effectuée dans les limites temporelles imposées), il est nécessaire d'utiliser des modèles formels temporisés. Parmi les modèles temporisés reconnus dans la communauté scientifique, nous en avons présenté trois : les automates temporisés, les réseaux de Petri temporels et les réseaux de Petri temporisés.

Dans les automates temporisés, le temps est représenté par des variable qu'on appelle horloges et qui sont ajoutées au modèle des automates d'état finis. Ces horloges évoluent toutes à la même vitesse et peuvent être comparées à des constantes lors du franchissement d'une transition. Il est également possible de remettre les horloges à zéro. Les états sont appelés localités et peuvent contenir des contraintes temporelles supplémentaires appelées invariants.

Dans les réseaux de Petri temporisés, les contraintes temporelles sont exprimées sous forme de dates qui sont ajoutées soit aux transitions (modèle t-temporisé), aux places (modèle p-temporisé) ou aux arcs (modèle a-temporisé). Une date d est interprétée de deux manières : (1) au plus tard : dans ce cas, une transition t est franchissable entre sa date d'activation et l'instant d. (2) au plus tôt : dans ce cas, une transition t n'est franchissable qu' à partir de la date d.

Dans les réseaux de Petri temporels, le temps est introduit sous forme d'un intervalle et non pas une date. Cet intervalle peut être associé soit aux transitions (modèle t-temporel), aux places (modèle p-temporel) ou aux arcs (modèle a-temporel). Dans les trois modèles, chaque transition possède une

durée d'activation et peut alors être franchie si et seulement si l'instant de tir appartient à cet intervalle.

Dans la deuxième partie du Chapitre 2, nous nous somme intéressés plus particulièrement au modèle des réseaux de Petri t-temporels. La particularité de ce modèle est sa capacité à modéliser d'une manière efficace et simple l'urgence des événements qui constitue une contrainte critique pour les systèmes temps-réel. En effet, la garantie de cette contrainte permet de renforcer la vivacité de ces systèmes, c'est-à-dire, éviter que le système puisse rester indéfiniment dans un état sans franchir une transition. Dans un réseau de Petri t-temporel, l'urgence des événements est introduite à travers la sémantique de tir forte. Cette sémantique force le franchissement d'une transition activée si son délai de tir maximum est atteint, et de ce fait, l'écoulement du temps ne peut pas désactiver une telle transition.

L'intégration du facteur temps dans les réseaux de Petri t-temporel ne fait qu'augmenter la complexité de leurs analyse. La raison principale de cette complexité, est le fait que l'espaces des états d'un réseau de Petri t-temporel est généralement infini. Afin de remédier à ce problème, plusieurs méthodes ont été proposées dans le but de fournir une représentation finie. Elles ont pour vocation de fournir une abstraction de l'espace des états plus au moins exacte par rapport aux types de propriétés que l'on souhaite vérifier. Ces différentes méthodes ont été présentées dans le deuxième chapitre.

Les techniques proposées dans la littérature peuvent être classées en deux grandes familles : les méthodes basées sur le concept de classe d'états et celles basées sur les zones.

- *Les méthodes basées sur le concept de classe d'états* : une classe d'états se caractérise par un marquage et un domaine représenté par un système d'inéquations linéaires. Les nœuds du graphe des classes d'états sont des classes et ses arcs sont étiquetés uniquement par les transitions du réseau de Petri temporel. Le calcul d'une classe d'états nécessite la résolution d'un système d'inéquations linéaires. Le nombre d'inéquations dans un système est proportionnel au nombre de transitions activées. Le système d'inéquations nécessite en général un certain nombre d'éliminations et de changements de variables qui sont parfois non triviaux. Le graphe des classes d'états a été revu et amélioré dans le but de réduire le nombre de nœuds et d'étendre le type des propriétés préservées. Ces améliorations ont conduit à des algorithmes permettent la vérification des propriétés LTL, CTL et CTL*. Les algorithmes de construction et de vérification basés sur les classes d'états sont implémentés dans un outils logiciel appelé Tina.
- *Les méthodes basées sur les zones* : une méthode adaptée à partir du

graphe des régions qui a été proposé initialement pour les automates temporisés. Les temporisations sont représentées par des horloges et non pas par des variables. Les valeurs des horloges sont regroupées dans un ensemble de classes d'équivalence appelées régions. En pratique, le nombre de régions étant trop important, les régions sont regroupées dans un ensemble de zones. Une zone est donc une union convexe de régions et peut être encodée par une matrice de différences bornées. Des améliorations on été apportées dans le but de résoudre le problème des intervalles à borne maximale infinie. Des algorithmes utilisant cette méthode pour la vérification des propriétés CTL et TCTL sur les réseaux de Petri t-temporels on été proposés et implémentés dans l'outil logiciel Romeo.

Les graphes résultants de ces approches sont principalement caractérisées par leur tailles (nombre de nœuds/arcs), les conditions de finitude, et le type de propriétés préservées. La génération de l'abstraction de l'espace des états est donc une étape clé dans le processus de vérification d'un système temps-réel. Plusieurs paramètres sont pris en compte durant cette étape comme la tailles (nombre de nœuds/arcs) et le temps de calcul des graphes. Pour toutes les approches mentionnées ci-dessus, nous avons remarqué que le temps de séjour dans un état n'apparaît pas explicitement dans les nœuds des graphes, ce qui engendre des calculs supplémentaires pour obtenir les durées des chemins lors de la vérification. L'encodage des temps de séjour dans les nœuds est le point clé qui nous a motivé à proposer une nouvelle approche pour l'abstraction et l'analyse de l'espace des états des réseaux de Petri t-temporels. Notre approche, proposée et présentée dans le Chapitre 4, est basée sur un graphe appelé Graphes des Agrégats Temporisés. Dans ce graphe, un seul agrégat regroupe les différents états situés entre deux délais, et cela en exploitant le fait que le marquage et l'ensemble des transitions activées ne changent pas entre ces deux instants. Les deux délais associés à chaque arégat, appelés temps de séjour minimum et maximum, constituent le principal avantage de notre approche. Comme cela a été démontré, l'approche proposée permet de construire une représentation finie et exacte pour les réseau de Petri t-temporel bornés. L'algorithme de construction du Graphes des Agrégats Temporisés a été implémenté et testé sur plusieurs exemples de réseaux de Petri temporels. Les résultats obtenus avec notre approche ont été comparés à ceux obtenus avec les graphes des classes et des zones. La comparaison des résultats expérimentaux nous a permis d'établir que notre approche se comporte bien pour la plus part des modèles testés, même si les résultats sont moins bons que ceux des deux autres approches pour certain exemples.

Les résultats de préservation du temps écoulé, des marquages, des sé-

quences et des propriété TCTL ont été exposées et prouvées dans le Chapitre 5. En outre, l'encodage des temps de séjour minimum et maximum dans les nœuds du graphe permet de vérifier des propriétés temporisées avec un simple parcours du graphe. C'est en exploitant ces résultats, que nous avons proposé des algorithmes pour l'analyse de l'accessibilité temporisée des marquages et la vérifications des propriétés TCTL. La convergence de ces algorithmes a été également prouvée à la fin de ce chapitre.

Ce travail a montré la nécessité d'avoir des modèles concis pour l'analyse des systèmes temps-réel. Cette concision doit être obtenue dès la phase de modélisation par l'utilisation de modèles adaptés aux contraintes exigées par ce genre de systèmes. Ceci peut être apporté par les réseaux de Petri t-temporels. Pour les phases d'analyse et de vérification, la génération de l'abstraction de l'espace des états doit être du même degré de concision. C'est ce que nous avons essayé d'apporter dans l'approche que nous avons proposé. Mais, si les résultats théoriques et expérimentaux sont prometteurs, nous sommes conscients que notre approche doit être confronté à des applications plus importantes. Ainsi, les perspectives à court terme de ce travail vont vers des expérimentations supplémentaires (sur des cas plus significatifs) afin de mieux comprendre les limites de notre approche et éventuellement trouver des solutions pour les corriger. À moyen terme, plusieurs perspectives seront étudiées : Tout d'abord, nous prévoyons l'implémentation des algorithmes de vérification de propriétés exprimées dans la logique *TCTL* que nous avons proposé dans le Chapitre 5. Cela va nous permettre de comparer notre approche par rapport au temps exigé pour achever la vérification d'une propriété. Deuxièmement, nous pensons que la taille du graphe peut encore être réduite tout en préservant les propriétés temporelles et sans nécessairement préserver toutes les exécutions sous-jacentes du TPN. Enfin, nous pensons que des techniques de réduction peuvent aisément être adaptées aux graphes des agrégats temporisés. Nous prévoyons notamment d'utiliser les techniques basées sur les diagrammes de décision (comme cela a été fait dans [20, 27]) et les techniques d'ordres partiels [19].

Chapitre 7

Annexes

7.1 Annexe A : Le graphe des agrégats temporisés associé au TPN du procédé de fabrication

```
------------------------------------------------------------------
aggregate a_0
------------------------------------------------------------------
 h : 2---   H : 5
Enabled transitions :
{maca1, [2 , 5]}
{maca2, [3 , 6]}
{macb1, [4 , 6]}
{macb2, [3 , 5]}
marking :
<mata0:2, mata1:1, mata2:1, matb1:1, matb2:1>
------------------------------------------------------------------
 --- macb2 --->
------------------------------------------------------------------
aggregate a_1
------------------------------------------------------------------
 h : 0---   H : 2
Enabled transitions :
{maca1, [-3 , 2]}
{maca2, [-2 , 3]}
{macb1, [-1 , 3]}
marking :
<mata0:2, mata1:1, mata2:1, matb1:1, prodb:1>
------------------------------------------------------------------
 --- macb1 --->
------------------------------------------------------------------
aggregate a_2
------------------------------------------------------------------
```

```
 h : 0---   H : 2
Enabled transitions :
{maca1, [-5 , 2]}
{maca2, [-4 , 3]}
marking :
<mata0:2, mata1:1, mata2:1, prodb:2>
-----------------------------------------------------------------
 --- maca2 --->
-----------------------------------------------------------------
aggregate a_3
-----------------------------------------------------------------
 h : 0---   H : 2
Enabled transitions :
{maca1, [-7 , 2]}
marking :
<mata0:1, mata1:1, proda:1, prodb:2>
-----------------------------------------------------------------
--- maca1 --->
-----------------------------------------------------------------
aggregate a_4
-----------------------------------------------------------------
 h : 5---   H : 6
Enabled transitions :
{assem, [5 , 6]}
marking :
<proda:2, prodb:2>
-----------------------------------------------------------------
--- assem --->
-----------------------------------------------------------------
aggregate a_5
-----------------------------------------------------------------
 h : 0---   H : 1
Enabled transitions :
{init, [0 , 1]}
Marking :
<prodf:1>
-----------------------------------------------------------------
--- init --->
-----------------------------------------------------------------
a_0
-----------------------------------------------------------------
```

```
--- maca1 --->
-----------------------------------------------------------------
aggregate a_6
-----------------------------------------------------------------
 h : 0---   H : 3
Enabled transitions :
{maca2, [-6 , 3]}
marking :
<mata0:1, mata2:1, proda:1, prodb:2>
-----------------------------------------------------------------
--- maca2 --->
-----------------------------------------------------------------
a_4
-----------------------------------------------------------------
--- maca2 --->
-----------------------------------------------------------------
aggregate a_7
-----------------------------------------------------------------
 h : 0---   H : 2
Enabled transitions :
{maca1, [-5 , 2]}
{macb1, [-3 , 3]}
marking :
<mata0:1, mata1:1, proda:1, matb1:1, prodb:1>
-----------------------------------------------------------------
--- macb1 --->
-----------------------------------------------------------------
a_3
-----------------------------------------------------------------
--- maca1 --->
-----------------------------------------------------------------
aggregate a_8
-----------------------------------------------------------------
 h : 0---   H : 3
Enabled transitions :
{macb1, [-5 , 3]}
marking :
<proda:2, matb1:1, prodb:1>
-----------------------------------------------------------------
--- macb1 --->
-----------------------------------------------------------------
```

```
a_4
------------------------------------------------------------------
--- maca1 --->
------------------------------------------------------------------
aggregate a_9
------------------------------------------------------------------
 h : 0---   H : 3
Enabled transitions :
{maca2, [-4 , 3]}
{macb1, [-3 , 3]}
marking :
<mata0:1, mata2:1, proda:1, matb1:1, prodb:1>
------------------------------------------------------------------
--- macb1 --->
------------------------------------------------------------------
a_6
------------------------------------------------------------------
--- maca2 --->
------------------------------------------------------------------
a_8
------------------------------------------------------------------
--- macb1 --->
------------------------------------------------------------------
aggregate a_10
------------------------------------------------------------------
 h : 0---   H : 1
Enabled transitions :
{maca1, [-3 , 1]}
{maca2, [-2 , 2]}
{macb2, [-2 , 1]}
marking :
<mata0:2, mata1:1, mata2:1, matb2:1, prodb:1>
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
aggregate a_11
------------------------------------------------------------------
 h : 0---   H : 1
Enabled transitions :
{maca1, [-4 , 1]}
{maca2, [-3 , 2]}
```

```
marking :
<mata0:2, mata1:1, mata2:1, prodb:2>
-----------------------------------------------------------------
--- maca2 --->
-----------------------------------------------------------------
aggregate a_12
-----------------------------------------------------------------
 h : 0---   H : 1
Enabled transitions :
{maca1, [-5 , 1]}
marking :
<mata0:1, mata1:1, proda:1, prodb:2>
-----------------------------------------------------------------
--- maca1 --->
-----------------------------------------------------------------
a_4
-----------------------------------------------------------------
--- maca1 --->
-----------------------------------------------------------------
aggregate a_13
-----------------------------------------------------------------
 h : 0---   H : 2
Enabled transitions :
{maca2, [-4 , 2]}
marking :
<mata0:1, mata2:1, proda:1, prodb:2>
-----------------------------------------------------------------
--- maca2 --->
-----------------------------------------------------------------
a_4
-----------------------------------------------------------------
--- maca2 --->
-----------------------------------------------------------------
aggregate a_14
-----------------------------------------------------------------
 h : 0---   H : 1
Enabled transitions :
{maca1, [-4 , 1]}
{macb2, [-3 , 1]}
marking :
<mata0:1, mata1:1, proda:1, matb2:1, prodb:1>
```

```
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
a_12
------------------------------------------------------------------
--- maca1 --->
------------------------------------------------------------------
aggregate a_15
------------------------------------------------------------------
 h : 0---   H : 1
Enabled transitions :
{macb2, [-4 , 1]}
marking :
<proda:2, matb2:1, prodb:1>
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
a_4
------------------------------------------------------------------
--- maca1 --->
------------------------------------------------------------------
aggregate a_16
------------------------------------------------------------------
 h : 0---   H : 1
Enabled transitions :
{maca2, [-3 , 2]}
{macb2, [-3 , 1]}
marking :
<mata0:1, mata2:1, proda:1, matb2:1, prodb:1>
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
a_13
------------------------------------------------------------------
--- maca2 --->
------------------------------------------------------------------
a_15
------------------------------------------------------------------
--- maca2 --->
------------------------------------------------------------------
aggregate a_17
```

```
------------------------------------------------------------------
 h : 0---   H : 2
Enabled transitions :
{maca1, [-3 , 2]}
{macb1, [-1 , 3]}
{macb2, [-2 , 2]}
marking :
<mata0:1, mata1:1, proda:1, matb1:1, matb2:1>
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
a_7
------------------------------------------------------------------
--- macb1 --->
------------------------------------------------------------------
aggregate a_18
------------------------------------------------------------------
 h : 0---   H : 2
Enabled transitions :
{maca1, [-5 , 2]}
{macb2, [-4 , 2]}
marking :
<mata0:1, mata1:1, proda:1, matb2:1, prodb:1>
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
a_3
------------------------------------------------------------------
--- maca1 --->
------------------------------------------------------------------
aggregate a_19
------------------------------------------------------------------
 h : 0---   H : 2
Enabled transitions :
{macb2, [-6 , 2]}
marking :
<proda:2, matb2:1, prodb:1>
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
a_4
```

```
------------------------------------------------------------------
--- maca1 --->
------------------------------------------------------------------
aggregate a_20
------------------------------------------------------------------
 h : 0---   H : 2
Enabled transitions :
{macb1, [-3 , 3]}
{macb2, [-4 , 2]}
marking :
<proda:2, matb1:1, matb2:1>
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
a_8
------------------------------------------------------------------
--- macb1 --->
------------------------------------------------------------------
a_19
------------------------------------------------------------------
--- maca1 --->
------------------------------------------------------------------
aggregate a_21
------------------------------------------------------------------
 h : 0---   H : 3
Enabled transitions :
{maca2, [-2 , 4]}
{macb1, [-1 , 4]}
{macb2, [-2 , 3]}
marking :
<mata0:1, mata2:1, proda:1, matb1:1, matb2:1>
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
aggregate a_22
------------------------------------------------------------------
 h : 0---   H : 4
Enabled transitions :
{maca2, [-5 , 4]}
{macb1, [-4 , 4]}
marking :
```

```
<mata0:1, mata2:1, proda:1, matb1:1, prodb:1>
-----------------------------------------------------------------
--- macb1 --->
-----------------------------------------------------------------
aggregate a_23
-----------------------------------------------------------------
 h : 0---   H : 4
Enabled transitions :
{maca2, [-9 , 4]}
marking :
<mata0:1, mata2:1, proda:1, prodb:2>
-----------------------------------------------------------------
--- maca2 --->
-----------------------------------------------------------------
a_4
-----------------------------------------------------------------
--- maca2 --->
-----------------------------------------------------------------
aggregate a_24
-----------------------------------------------------------------
 h : 0---   H : 4
Enabled transitions :
{macb1, [-8 , 4]}
marking :
<proda:2, matb1:1, prodb:1>
-----------------------------------------------------------------
--- macb1 --->
-----------------------------------------------------------------
a_4
-----------------------------------------------------------------
--- macb1 --->
-----------------------------------------------------------------
aggregate a_25
-----------------------------------------------------------------
 h : 0---   H : 3
Enabled transitions :
{maca2, [-5 , 4]}
{macb2, [-5 , 3]}
marking :
<mata0:1, mata2:1, proda:1, matb2:1, prodb:1>
-----------------------------------------------------------------
```

```
--- macb2 --->
------------------------------------------------------------------
a_23
------------------------------------------------------------------
--- maca2 --->
------------------------------------------------------------------
aggregate a_26
------------------------------------------------------------------
 h : 0---   H : 3
Enabled transitions :
{macb2, [-8 , 3]}
marking :
<proda:2, matb2:1, prodb:1>
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
a_4
------------------------------------------------------------------
--- maca2 --->
------------------------------------------------------------------
aggregate a_27
------------------------------------------------------------------
 h : 0---   H : 3
Enabled transitions :
{macb1, [-4 , 4]}
{macb2, [-5 , 3]}
marking :
<proda:2, matb1:1, matb2:1>
------------------------------------------------------------------
--- macb2 --->
------------------------------------------------------------------
a_24
------------------------------------------------------------------
--- macb1 --->
------------------------------------------------------------------
a_26
------------------------------------------------------------------
```

7.2 Annexe B : Le graphe des classes d'états associé au TPN du procédé de fabrication

```
-----------------------------------------
C0
-----------------------------------------
Marking
2   MatA0
1   MatA1
1   MatA2
1   MatB1
1   MatB2
0   ProdA
0   ProdB
0   ProdF

Firing Domain
MacA1 [2,5]
MacA2 [3,6]
MacB1 [4,6]
MacB2 [3,5]
MacA1 - MacA2 <= 2
MacA1 - MacB1 <= 1
MacA1 - MacB2 <= 2
MacA2 - MacA1 <= 4
MacA2 - MacB1 <= 2
MacA2 - MacB2 <= 3
MacB1 - MacA1 <= 4
MacB1 - MacA2 <= 3
MacB1 - MacB2 <= 3
MacB2 - MacA1 <= 3
MacB2 - MacA2 <= 2
MacB2 - MacB1 <= 1
-----------------------------------------
-----------------------------------------
C1
-----------------------------------------
Marking
1   MatA0
0   MatA1
```

```
1   MatA2
1   MatB1
1   MatB2
1   ProdA
0   ProdB
0   ProdF

Firing Domain
MacA2 [0,4]
MacB1 [0,4]
MacB2 [0,3]
MacA2 - MacB1 <= 2
MacA2 - MacB2 <= 3
MacB1 - MacA2 <= 3
MacB1 - MacB2 <= 3
MacB2 - MacA2 <= 2
MacB2 - MacB1 <= 1
-----------------------------------------
-----------------------------------------
C2
-----------------------------------------
Marking
1   MatA0
1   MatA1
0   MatA2
1   MatB1
1   MatB2
1   ProdA
0   ProdB
0   ProdF

Firing Domain
MacA1 [0,2]
MacB1 [0,3]
MacB2 [0,2]
MacA1 - MacB1 <= 1
MacA1 - MacB2 <= 2
MacB1 - MacA1 <= 3
MacB1 - MacB2 <= 3
MacB2 - MacA1 <= 2
MacB2 - MacB1 <= 1
```

```
-----------------------------------------
-----------------------------------------
C3
-----------------------------------------
Marking
2   MatA0
1   MatA1
1   MatA2
0   MatB1
1   MatB2
0   ProdA
1   ProdB
0   ProdF

Firing Domain
MacA1 [0,1]
MacA2 [0,2]
MacB2 [0,1]
MacA1 - MacA2 <= 1
MacA1 - MacB2 <= 1
MacA2 - MacA1 <= 2
MacA2 - MacB2 <= 2
MacB2 - MacA1 <= 1
MacB2 - MacA2 <= 1
-----------------------------------------
-----------------------------------------
C4
-----------------------------------------
Marking
2   MatA0
1   MatA1
1   MatA2
1   MatB1
0   MatB2
0   ProdA
1   ProdB
0   ProdF

Firing Domain
MacA1 [0,2]
MacA2 [0,3]
```

```
MacB1 [0,3]
MacA1 - MacA2 <= 2
MacA1 - MacB1 <= 1
MacA2 - MacA1 <= 3
MacA2 - MacB1 <= 2
MacB1 - MacA1 <= 3
MacB1 - MacA2 <= 3
-----------------------------------------
-----------------------------------------
C5
-----------------------------------------
Marking
0   MatA0
0   MatA1
0   MatA2
1   MatB1
1   MatB2
2   ProdA
0   ProdB
0   ProdF

Firing Domain
MacB1 [0,3]
MacB2 [0,2]
MacB1 - MacB2 <= 3
MacB2 - MacB1 <= 1
-----------------------------------------
-----------------------------------------
C6
-----------------------------------------
Marking
1   MatA0
0   MatA1
1   MatA2
0   MatB1
1   MatB2
1   ProdA
1   ProdB
0   ProdF

Firing Domain
```

```
MacA2 [0,2]
MacB2 [0,1]
MacA2 - MacB2 <= 2
MacB2 - MacA2 <= 1
-----------------------------------------
-----------------------------------------
C7
-----------------------------------------
Marking
1   MatA0
0   MatA1
1   MatA2
1   MatB1
0   MatB2
1   ProdA
1   ProdB
0   ProdF

Firing Domain
MacA2 [0,3]
MacB1 [0,3]
MacA2 - MacB1 <= 2
MacB1 - MacA2 <= 3
-----------------------------------------
-----------------------------------------
C8
-----------------------------------------
Marking
1   MatA0
1   MatA1
0   MatA2
0   MatB1
1   MatB2
1   ProdA
1   ProdB
0   ProdF

Firing Domain
MacA1 [0,1]
MacB2 [0,1]
MacA1 - MacB2 <= 1
```

```
MacB2 - MacA1 <= 1
-----------------------------------------
-----------------------------------------
C9
-----------------------------------------
Marking
1   MatA0
1   MatA1
0   MatA2
1   MatB1
0   MatB2
1   ProdA
1   ProdB
0   ProdF

Firing Domain
MacA1 [0,2]
MacB1 [0,3]
MacA1 - MacB1 <= 1
MacB1 - MacA1 <= 3
-----------------------------------------
-----------------------------------------
C10
-----------------------------------------
Marking
2   MatA0
1   MatA1
1   MatA2
0   MatB1
0   MatB2
0   ProdA
2   ProdB
0   ProdF

Firing Domain
MacA1 [0,1]
MacA2 [0,2]
MacA1 - MacA2 <= 1
MacA2 - MacA1 <= 2
-----------------------------------------
-----------------------------------------
```

```
C11
-----------------------------------------
Marking
0   MatA0
0   MatA1
0   MatA2
0   MatB1
1   MatB2
2   ProdA
1   ProdB
0   ProdF

Firing Domain
MacB2 [0,1]
-----------------------------------------
-----------------------------------------
C12
-----------------------------------------
Marking
0   MatA0
0   MatA1
0   MatA2
1   MatB1
0   MatB2
2   ProdA
1   ProdB
0   ProdF

Firing Domain
MacB1 [0,3]
-----------------------------------------
-----------------------------------------
C13
-----------------------------------------
Marking
1   MatA0
0   MatA1
1   MatA2
0   MatB1
0   MatB2
1   ProdA
```

```
2   ProdB
0   ProdF

Firing Domain
MacA2 [0,2]
-----------------------------------------
-----------------------------------------
C14
-----------------------------------------
Marking
1   MatA0
1   MatA1
0   MatA2
0   MatB1
0   MatB2
1   ProdA
2   ProdB
0   ProdF

Firing Domain
MacA1 [0,1]
-----------------------------------------
-----------------------------------------
C15
-----------------------------------------
Marking
0   MatA0
0   MatA1
0   MatA2
0   MatB1
0   MatB2
2   ProdA
2   ProdB
0   ProdF

Firing Domain
Assem [5,6]
-----------------------------------------
-----------------------------------------
C16
-----------------------------------------
```

```
Marking
0   MatA0
0   MatA1
0   MatA2
0   MatB1
0   MatB2
0   ProdA
0   ProdB
1   ProdF

Firing Domain
Init [0,1]
-----------------------------------------
```

```
-----------------------------------------
C0 -- MacA1 --> C1
C0 -- MacA2 --> C2
C0 -- MacB1 --> C3
C0 -- MacB2 --> C4
-----------------------------------------
C1 -- MacA2 --> C5
C1 -- MacB1 --> C6
C1 -- MacB2 --> C7
-----------------------------------------
C2 -- MacA1 --> C5
C2 -- MacB1 --> C8
C2 -- MacB2 --> C9
-----------------------------------------
C3 -- MacA1 --> C6
C3 -- MacA2 --> C8
C3 -- MacB2 --> C10
-----------------------------------------
C4 -- MacA1 --> C7
C4 -- MacA2 --> C9
C4 -- MacB1 --> C10
-----------------------------------------
C5 -- MacB1 --> C11
C5 -- MacB2 --> C12
```

```
-----------------------------------------
C6 -- MacA2 --> C11
C6 -- MacB2 --> C13
-----------------------------------------
C7 -- MacA2 --> C12
C7 -- MacB1 --> C13
-----------------------------------------
C8 -- MacA1 --> C11
C8 -- MacB2 --> C14
-----------------------------------------
C9 -- MacA1 --> C12
C9 -- MacB1 --> C14
-----------------------------------------
C10 -- MacA1 --> C13
C10 -- MacA2 --> C14
-----------------------------------------
C11 -- MacB2 --> C15
-----------------------------------------
C12 -- MacB1 --> C15
-----------------------------------------
C13 -- MacA2 --> C15
-----------------------------------------
C14 -- MacA1 --> C15
-----------------------------------------
C15 -- Assem --> C16
-----------------------------------------
C16 -- Init --> C0
-----------------------------------------
```

7.3 Annexe C : Le graphe des zones associé au TPN du procédé de fabrication

```
-----------------------------------------
C0
-----------------------------------------
Marking
2   MatA0
1   MatA1
1   MatA2
1   MatB1
1   MatB2
0   ProdA
0   ProdB
0   ProdF

Firing Domain
MacA1 [0,5]
MacA2 [0,5]
MacB1 [0,5]
MacB2 [0,5]
MacA1 - MacA2 <= 0
MacA1 - MacB1 <= 0
MacA1 - MacB2 <= 0
MacA2 - MacA1 <= 0
MacA2 - MacB1 <= 0
MacA2 - MacB2 <= 0
MacB1 - MacA1 <= 0
MacB1 - MacA2 <= 0
MacB1 - MacB2 <= 0
MacB2 - MacA1 <= 0
MacB2 - MacA2 <= 0
MacB2 - MacB1 <= 0
-----------------------------------------
-----------------------------------------
C1
-----------------------------------------
Marking
1   MatA0
0   MatA1
```

```
1   MatA2
1   MatB1
1   MatB2
1   ProdA
0   ProdB
0   ProdF

Firing Domain
MacA2 [2,inf[
MacB1 [2,inf[
MacB2 [2,inf[
MacA2 - MacB1 <= 0
MacA2 - MacB2 <= 0
MacB1 - MacA2 <= 0
MacB1 - MacB2 <= 0
MacB2 - MacA2 <= 0
MacB2 - MacB1 <= 0
-----------------------------------------
-----------------------------------------
C2
-----------------------------------------
Marking
1   MatA0
1   MatA1
0   MatA2
1   MatB1
1   MatB2
1   ProdA
0   ProdB
0   ProdF

Firing Domain
MacA1 [3,inf[
MacB1 [3,inf[
MacB2 [3,inf[
MacB1 - MacA1 <= 0
MacB1 - MacB2 <= 0
-----------------------------------------
-----------------------------------------
C3
-----------------------------------------
```

```
Marking
2   MatA0
1   MatA1
1   MatA2
0   MatB1
1   MatB2
0   ProdA
1   ProdB
0   ProdF

Firing Domain
MacA1 [4,inf[
MacA2 [4,inf[
MacB2 [4,inf[
-----------------------------------------
-----------------------------------------
C4
-----------------------------------------
Marking
2   MatA0
1   MatA1
1   MatA2
1   MatB1
0   MatB2
0   ProdA
1   ProdB
0   ProdF

Firing Domain
MacA1 [3,inf[
MacA2 [3,inf[
MacB1 [3,inf[
MacB1 - MacA1 <= 0
MacB1 - MacA2 <= 0
-----------------------------------------
-----------------------------------------
C5
-----------------------------------------
Marking
0   MatA0
0   MatA1
```

```
0   MatA2
1   MatB1
1   MatB2
2   ProdA
0   ProdB
0   ProdF

Firing Domain
MacB1 [3,inf[
MacB2 [3,inf[
MacB1 - MacB2 <= 0
-----------------------------------------
-----------------------------------------
C6
-----------------------------------------
Marking
1   MatA0
0   MatA1
1   MatA2
0   MatB1
1   MatB2
1   ProdA
1   ProdB
0   ProdF

Firing Domain
MacA2 [4,inf[
MacB2 [4,inf[
-----------------------------------------
-----------------------------------------
C7
-----------------------------------------
Marking
1   MatA0
0   MatA1
1   MatA2
1   MatB1
0   MatB2
1   ProdA
1   ProdB
0   ProdF
```

```
Firing Domain
MacA2 [3,inf[
MacB1 [3,inf[
MacB1 - MacA2 <= 0
-----------------------------------------
-----------------------------------------
C8
-----------------------------------------
Marking
1   MatA0
1   MatA1
0   MatA2
0   MatB1
1   MatB2
1   ProdA
1   ProdB
0   ProdF

Firing Domain
MacA1 [4,inf[
MacB2 [4,inf[
-----------------------------------------
-----------------------------------------
C9
-----------------------------------------
Marking
1   MatA0
1   MatA1
0   MatA2
1   MatB1
0   MatB2
1   ProdA
1   ProdB
0   ProdF

Firing Domain
MacA1 [3,inf[
MacB1 [3,inf[
MacB1 - MacA1 <= 0
-----------------------------------------
```

```
-----------------------------------------
C10
-----------------------------------------
Marking
2   MatA0
1   MatA1
1   MatA2
0   MatB1
0   MatB2
0   ProdA
2   ProdB
0   ProdF

Firing Domain
MacA1 [4,inf[
MacA2 [4,inf[
-----------------------------------------
-----------------------------------------
C11
-----------------------------------------
Marking
0   MatA0
0   MatA1
0   MatA2
0   MatB1
1   MatB2
2   ProdA
1   ProdB
0   ProdF

Firing Domain
MacB2 [4,inf[
-----------------------------------------
-----------------------------------------
C12
-----------------------------------------
Marking
0   MatA0
0   MatA1
0   MatA2
1   MatB1
```

```
0   MatB2
2   ProdA
1   ProdB
0   ProdF

Firing Domain
MacB1 [3,inf[
-----------------------------------------
-----------------------------------------
C13
-----------------------------------------
Marking
1   MatA0
0   MatA1
1   MatA2
0   MatB1
0   MatB2
1   ProdA
2   ProdB
0   ProdF

Firing Domain
MacA2 [4,inf[
-----------------------------------------
-----------------------------------------
C14
-----------------------------------------
Marking
1   MatA0
1   MatA1
0   MatA2
0   MatB1
0   MatB2
1   ProdA
2   ProdB
0   ProdF

Firing Domain
MacA1 [4,inf[
-----------------------------------------
-----------------------------------------
```

```
C15
-----------------------------------------
Marking
0   MatA0
0   MatA1
0   MatA2
0   MatB1
0   MatB2
2   ProdA
2   ProdB
0   ProdF

Firing Domain
Assem [0,inf[
-----------------------------------------
-----------------------------------------
C16
-----------------------------------------
Marking
0   MatA0
0   MatA1
0   MatA2
0   MatB1
0   MatB2
0   ProdA
0   ProdB
1   ProdF

Firing Domain
Init [0,inf[
-----------------------------------------
-----------------------------------------
C17
-----------------------------------------
Marking
2   MatA0
1   MatA1
1   MatA2
1   MatB1
1   MatB2
0   ProdA
```

```
0   ProdB
0   ProdF

Firing Domain
MacA1 [0,inf[
MacA2 [0,inf[
MacB1 [0,inf[
MacB2 [0,inf[
MacA1 - MacA2 <= 0
MacA1 - MacB1 <= 0
MacA1 - MacB2 <= 0
MacA2 - MacA1 <= 0
MacA2 - MacB1 <= 0
MacA2 - MacB2 <= 0
MacB1 - MacA1 <= 0
MacB1 - MacA2 <= 0
MacB1 - MacB2 <= 0
MacB2 - MacA1 <= 0
MacB2 - MacA2 <= 0
MacB2 - MacB1 <= 0
-----------------------------------------




-----------------------------------------
C0 -- MacA1 --> C1
C0 -- MacA2 --> C2
C0 -- MacB1 --> C3
C0 -- MacB2 --> C4
-----------------------------------------
C1 -- MacA2 --> C5
C1 -- MacB1 --> C6
C1 -- MacB2 --> C7
-----------------------------------------
C2 -- MacA1 --> C5
C2 -- MacB1 --> C8
C2 -- MacB2 --> C9
-----------------------------------------
C3 -- MacA1 --> C6
C3 -- MacA2 --> C8
```

```
C3 -- MacB2 --> C10
-----------------------------------------
C4 -- MacA1 --> C7
C4 -- MacA2 --> C9
C4 -- MacB1 --> C10
-----------------------------------------
C5 -- MacB1 --> C11
C5 -- MacB2 --> C12
-----------------------------------------
C6 -- MacA2 --> C11
C6 -- MacB2 --> C13
-----------------------------------------
C7 -- MacA2 --> C12
C7 -- MacB1 --> C13
-----------------------------------------
C8 -- MacA1 --> C11
C8 -- MacB2 --> C14
-----------------------------------------
C9 -- MacA1 --> C12
C9 -- MacB1 --> C14
-----------------------------------------
C10 -- MacA1 --> C13
C10 -- MacA2 --> C14
-----------------------------------------
C11 -- MacB2 --> C15
-----------------------------------------
C12 -- MacB1 --> C15
-----------------------------------------
C13 -- MacA2 --> C15
-----------------------------------------
C14 -- MacA1 --> C15
-----------------------------------------
C15 -- Assem --> C16
-----------------------------------------
C16 -- Init --> C17
-----------------------------------------
C17 -- MacA1 --> C1
C17 -- MacA2 --> C2
C17 -- MacB1 --> C3
C17 -- MacB2 --> C4
-----------------------------------------
```

Bibliographie

[1] *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA*. IEEE Computer Society, 2006.

[2] Parosh Aziz Abdulla and Aletta Nylén. Timed petri nets and b.q.os. In *ICATPN*, pages 53–70, 2001.

[3] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425, 1990.

[4] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1) :2–34, 1993.

[5] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2) :183–235, 1994.

[6] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1) :116–146, 1996.

[7] Bernard Berthomieu and Michel Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. Software Eng.*, 17(3) :259–273, 1991.

[8] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time petri nets. In *IFIP Congress*, pages 41–46, 1983.

[9] Bernard Berthomieu and François Vernadat. State Class Constructions for Branching Analysis of Time Petri Nets. In Hubert Garavel and John Hatcliff, editors, *TACAS 2003*, volume 2619 of *Lecture Notes in Computer Science*, pages 442–457. Springer, 2003.

[10] Bernard Berthomieu and François Vernadat. Time Petri Nets Analysis with TINA. In *QEST*, pages 123–124, 2006.

[11] Yves Bertot. A short presentation of coq. *TPHOLs*, pages 12–16, 2008.

[12] Hanifa Boucheneb, Guillaume Gardey, and Olivier H. Roux. TCTL Model Checking of Time Petri Nets. *J. Log. Comput.*, 19(6) :1509–1540, 2009.

[13] George B. Dantzig and B. Curtis Eaves. Fourier-motzkin elimination and its dual. *J. Comb. Theory, Ser. A*, 14(3) :288–297, 1973.

[14] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the international workshop on Automatic verification methods for finite state systems*, pages 197–212, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[15] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1990.

[16] Erich Gamma and Kent Beck. JUnit : A Regression Testing Framework. http ://www.junit.org, 2000.

[17] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier H. Roux. Romeo : A tool for analyzing time petri nets. In *CAV*, pages 418–423. Springer, 2005.

[18] Guillaume Gardey, Olivier H. Roux, and Olivier F. Roux. Using Zone Graph Method for Computing the State Space of a Time Petri Net. In *FORMATS 2003*, volume 2791 of *Lecture Notes in Computer Science*, pages 246–259. Springer, 2003.

[19] Patrice Godefroid. Using partial orders to improve automatic verification methods. In *CAV*, volume 531 of *Lecture Notes in Computer Science*, pages 176–185. Springer, 1990.

[20] Serge Haddad, Jean-Michel Ilié, and Kais Klai. Design and evaluation of a symbolic and abstraction-based model checker. In *ATVA*, pages 196–210, 2004.

[21] Rachid Hadjidj and Hanifa Boucheneb. Improving state class constructions for CTL* model checking of time Petri nets. *STTT*, 10(2) :167–184, 2008.

[22] Rachid Hadjidj and Hanifa Boucheneb. On-the-fly TCTL model checking for time Petri nets. *Theor. Comput. Sci.*, 410(42) :4241–4261, 2009.

[23] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. In *LICS*, pages 394–406, 1992.

[24] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Introduction to automata theory, languages, and computation, 2nd edition. *SIGACT News*, 32(1) :60–65, 2001.

[25] John E. Hopcroft and Jeffrey D. Ullman. Introduction to automata theory, languages and computation. *Addison-Wesley*, 1979.

[26] W. Khansa. Réseaux de petri p-temporels : Contribution à l'etude des systèmes à evénements discrets. *Thèse de Doctorat de l'Université de Savoie, France*, 1997.

[27] Kais Klai and Denis Poitrenaud. MC-SOG : An LTL model checker based on symbolic observation graphs. In *Petri Nets*, pages 288–306, 2008.

[28] Howard Kleiman. The floyd-warshall algorithm, the ap and the tsp. *CoRR*, math.CO/0111309, 2001.

[29] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4) :255–299, 1990.

[30] Leslie Lamport. What good is temporal logic ? In *IFIP Congress*, pages 657–668, 1983.

[31] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Model-checking for real-time systems. In Horst Reichel, editor, *FCT '95*, volume 965 of *Lecture Notes in Computer Science*, pages 62–88. Springer, 1995.

[32] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL : Status and Developments. In *CAV*, pages 456–459, 1997.

[33] Didier Lime and Olivier H. Roux. Model Checking of Time Petri Nets Using the State Class Timed Automaton. *Discrete Event Dynamic Systems*, 16(2) :179–205, 2006.

[34] Philip M. Merlin. A study of the recoverability of communication protocols. *Thesis, Computer Science Dep., University of California*, 1974.

[35] Philip M. Merlin and David J. Farber. Recoverability of modular systems. *Operating Systems Review*, 9(3) :51–56, 1975.

[36] Antoine Miné. A new numerical abstract domain based on difference-bound matrices. In *PADO*, pages 155–172, 2001.

[37] Antoine Miné. A new numerical abstract domain based on difference-bound matrices. *CoRR*, abs/cs/0703073, 2007.

[38] Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors. *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *Lecture Notes in Computer Science*. Springer, 2008.

[39] Lawrence C. Paulson. *Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[40] Wojciech Penczek, Agata Pólrola, and Andrzej Zbrzezny. SAT-Based (Parametric) Reachability for a Class of Distributed Time Petri Nets. *T. Petri Nets and Other Models of Concurrency*, 4 :72–97, 2010.

[41] C. A. Petri. Concepts of net theory. In *MFCS'73*, pages 137–146. Mathematical Institute of the Slovak Academy of Sciences, 1973.

[42] Mauro Pezzè and M. Young. Time Petri Nets : A Primer Introduction. In *Tutorial presented at the Multi-Workshop on Formal Methods in Performance Evaluation and Applications*, 1999.

[43] C. Ramchandani. Analysis of asynchronous concurrent systems by timed Petri nets. Technical report, Cambridge, MA, USA, 1974.

[44] Jacques Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009.

[45] Joseph Sifakis. Use of petri nets for performance evaluation. In *Performance*, pages 75–93, 1977.

[46] Joseph Sifakis. Use of Petri nets for performance evaluation. *Acta Cybern.*, 4 :185–202, 1980.

[47] T. Yoneda and H. Ryuba. CTL model checking of time Petri nets using geometric regions. 1998.

[48] Sergio Yovine. KRONOS : A Verification Tool for Real-Time Systems. *STTT*, 1(1-2) :123–133, 1997.