Titre: Title:	Vers l'intégration de paramètres temporels statiques et dynamiques dans les techniques de vérification par ordre partiel
Auteur: Author:	Mohamed Karim Weslati
Date:	2014
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Weslati, M. K. (2014). Vers l'intégration de paramètres temporels statiques et dynamiques dans les techniques de vérification par ordre partiel [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1654/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/1654/
Directeurs de recherche: Advisors:	Hanifa Boucheneb
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

VERS L'INTÉGRATION DE PARAMÈTRES TEMPORELS STATIQUES ET DYNAMIQUES DANS LES TECHNIQUES DE VÉRIFICATION PAR ORDRE PARTIEL

MOHAMED KARIM WESLATI

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

DÉCEMBRE 2014

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

VERS L'INTÉGRATION DE PARAMÈTRES TEMPORELS STATIQUES ET DYNAMIQUES DANS LES TECHNIQUES DE VÉRIFICATION PAR ORDRE PARTIEL

présenté par : WESLATI Mohamed Karim

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. FOUTSE Khomh, Ph. D., président

Mme BOUCHENEB Hanifa, Doctorat, membre et directrice de recherche

M. TALHI Chemseddine, Ph. D., membre

DÉDICACE

À mes chers parents,

Et à toute ma famille.

REMERCIEMENTS

Je voudrais exprimer par ces quelques lignes de remerciements ma sincère gratitude envers tous ceux qui, directement ou indirectement, ont aidé à la finalisation de ce travail.

Je commence par remercier d'une part, Madame Hanifa Boucheneb qui m'a fait l'honneur d'être mon encadrante. Je la remercie profondément pour ses encouragements continus, ses précieuses directives, son soutien moral, sa preuve de compréhension et pour ses conseils judicieux qu'elle m'a prodigués.

Je tiens d'autre part, à remercier les membres de Jury pour le temps accordé et bien vouloir examiner mon travail malgré leurs occupations.

RÉSUMÉ

Les systèmes temps réels sont utilisés de nos jours d'une façon importante, surtout dans le domaine des applications critiques.

Pour les modéliser, nous avons opté pour l'utilisation des réseaux de Petri temporels (TPN) qui sont une extension temporelle des réseaux de Petri simples (Rdp) proposés par Carl Adam Petri.

Ce choix est justifié par le fait que les TPN sont des modèles permettant de représenter les aspects importants des systèmes temps réel comme la concurrence, la synchronisation et les contraintes temporelles d'une façon triviale.

De plus, les TPN offrent des techniques formelles de vérification qui intègrent les contraintes temporelles. Parmi ces techniques, ce mémoire s'intéresse aux approches énumératives. Ces approches se basent sur des abstractions et visent à représenter les espaces d'états infinis des TPN par des graphes finis. Plusieurs abstractions d'espaces d'états ont été proposées dans la littérature : le SCG (State Class Graph), le CSCG (Contracted State Class Graph), le ZBG (Zone Based Graph), etc. Ces graphes sont finis pour des TPN bornés (ayant un nombre fini de marquages accessibles) et préservent les marquages et les séquences de franchissement des TPN. Ce mémoire s'intéresse au CSCG car il est plus compact que les autres. Cependant, à l'instar des approches énumératives, le CSCG se heurte au problème d'explosion combinatoire. Pour atténuer ce problème, ce mémoire propose d'étendre et d'appliquer des techniques de réduction d'ordre partiel aux TPN.

La première contribution de ce mémoire est d'implémenter une technique d'ordre partiel pour les TPN. Cette technique est une extension aux TPN de l'approche de réduction Stubborn Set (proposée par Valmari pour les réseaux de Petri simples).

La deuxième contribution est le développement d'un outil qui permet de générer l'espace d'états réduit d'un TPN en appliquant notre technique. Cet outil permet aussi la construction des graphes SCG et CSCG.

Nous avons réalisé certaines optimisations dans l'implémentation de cet outil en nous basant sur des algorithmes de classes d'états de moindres complexités.

Nous avons testé l'approche d'ordre partiel proposée dans ce mémoire et comparée avec d'autres approches et variantes. Divers réseaux de Petri temporels ont été testés, incluant quelques modèles de références. Ces derniers sont utilisés pour confronter des outils de vérification, dans des compétitions organisées au sein de la conférence annuelle Petri nets.

Les résultats obtenus en utilisant ces TPN de références montrent que notre technique permet d’avoir de meilleurs résultats par rapport aux autres approches en utilisant les même TPN. En effet, le gain a varié entre 20% et 90% en termes de nombres de classes d’états, d’arcs et de temps d’exécution.

ABSTRACT

Real time systems are widely used today especially in the field of critical applications.

For modeling, we chose to use Time Petri network (TPN), which is an extension of simple Petri networks (Rdp) implemented by Carl Adam Petri.

This is justified by the fact that TPN models are used to represent important aspects of real-time systems such as concurrency and synchronization in a simple way. Besides, TPN are rich in theories that have helped us to implement our technique.

The two major issues of real-time systems are the obtaining of an infinite state space and the combinatorial explosion.

To overcome the first issue, we have chosen the method of abstraction CSCG (Contracted State Class Graph). It is a method of abstraction which is applied to the TPN to get a finite state space representation. To address the second issue, we have extended partial order reduction technique "Stubborn Set" in the context of TPN.

Note that the Stubborn Set method is a partial order reduction technique that aims to alleviate the problem of combinatorial explosion of the state space of simple Petri net i.e. without considering the time constraints. Therefore, our first contribution of our research is to participate to develop a partial order reduction technique extending the method Stubborn Set in the context of TPN.

The second contribution is the development of a tool to generate the reduced state space of TPN by applying our technique. It also allows the construction of the state space using other techniques discussed in the chapter "literature review".

We have performed some optimizations in the implementation of this tool based on algorithms with reduced complexity taken from the literature.

To test our technique with our developed tool, we chose various examples of temporel Petri nets such the benchmark TPN.

The results shows that our technique can achieve better results compared to other techniques in terms of the number of state classes, arcs and execution time.

This will allow us to use this technique in other application areas such as verification of certain properties of workflows.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ASCG	Atomic State Class Graph
CSCG	Contracted State Class Graph
DBM	Difference Bound Matrix
FMS	Flexible Manufacturing System
HC	HouseConstruction
LAAS	Laboratoire d'Analyse et d'Architecture des Systèmes
LSCG	Linear State Class Graph
MCC	Model Checking Contest
Rdp	Réseaux de Petri simple
SCG	State Class Graph
SSCG	Strong State Class Graph.
STP	Simple Time Petri Net
SwPN	Stopwatch Petri Nets
Tcl	Tool command language
TPN	Temporel Petri Net
ZBG	Zone Based Graph

CHAPITRE 1 INTRODUCTION

Les systèmes temps réel ("Système temps réel," n.d.) sont des systèmes qui sont assujettis à des contraintes temporelles souples ou strictes. À part la nécessité d'avoir des résultats exacts, ils doivent respecter ces contraintes-là. Par exemple pour calculer une métrique dans le domaine aéronautique, c'est bien d'avoir le bon résultat, mais cette information doit être livrée à temps et non pas d'une façon retardée.

De nos jours, les systèmes temps réel sont de plus en plus utilisés dans divers domaines, notamment dans :

- La télé-chirurgie.
- L'aéronautique avec le contrôle de différents capteurs.
- Les systèmes de productivités (les usines, les industries, etc.).
- Les systèmes de contrôles d'informations (vidéo de surveillance, contrôle à distance, etc.).

Les systèmes temps réel sont de plus en plus complexes surtout avec les fonctionnalités avancées qu'ils réalisent. Le non-respect des contraintes temporelles peut causer parfois des pertes importantes, et du coup la vérification de ces systèmes avant leurs utilisations s'avère indispensable.

Pour vérifier si les systèmes temps réel respectent bien les requis, on doit tout d'abord les modéliser. Pour cela, on peut utiliser plusieurs modèles comme les automates temporisés et les réseaux de Petri temporels (TPN), appelés aussi modèle de Merlin (Merlin & Farber, 1976).

Dans ce présent mémoire, nous allons nous intéresser aux TPN. Un réseau de Petri temporel est un réseau de Petri où un intervalle de temps est associé à chaque transition, spécifiant son intervalle de franchissement.

Les TPN permettent de décrire les comportements des systèmes temps réels incluant leurs contraintes temporelles d'une façon simple.

Un état représente une information caractérisant une situation dans un réseau de Petri temporel. Une partie de cette information est dédiée aux contraintes temporelles.

L’espace d’états d’un TPN regroupe tous les états possibles qui sont accessibles à partir de l’état initial. Cependant, en associant des intervalles de temps aux transitions, l'espace d’états devient infini. En effet, le temps est continu et à partir de chaque état on pourrait avoir une infinité d’états successeurs en faisant évoluer le temps.

Les abstractions permettent de remédier à ce problème car elles ont pour but de représenter cet espace infinie par des graphes finis. Cependant, même avec les abstractions, nous pouvons nous trouver face à un autre problème qui est l’explosion combinatoire due à la sémantique d'entrelacement de comportements concurrents : c’est le cas où une légère variation sur les données du TPN engendre une augmentation importante de l’espace d’états, ce qui peut rendre plus difficile la phase de vérification des propriétés sur l’espace d’états obtenu.

Comme solution, nous pouvons procéder à l’utilisation d’une technique de réduction d’ordre partiel ("Partial order reduction," n.d.). Ce sont des techniques qui permettent de réduire l’espace d’états d’un réseau de Petri temporel en éliminant les comportements équivalents par rapport aux propriétés qui nous intéressent.

En réduisant l’espace d’états, c’est-à-dire en le rendant plus compact, ça nous aide à faciliter la tâche de vérification des propriétés sur les TPN étudiés. Au lieu de parcourir l’ensemble de tous les états pour voir si une propriété est vérifiée ou non, nous pourrions parcourir seulement les états contenus dans l’espace réduit. Ceci nous permet de gagner plus de temps et pouvoir analyser les systèmes complexes.

Il y a plusieurs outils dans le domaine de recherche qui permettent la génération et la réduction de l’espace d’états d’un TPN. Toutefois, à notre connaissance, il n’existe aucun outil qui permet de réaliser ce calcul avec une technique de réduction d’ordre partiel qui intègre les paramètres temporels pour les TPN. On entend par paramètres temporels, les contraintes temporelles statiques des transitions des TPN.

Objectifs de recherche

Nos objectifs de recherche sont liés aux problèmes d'explosion combinatoire des approches énumératives des TPN. Il s'agit de :

1-Participer au développement d’une technique de réduction d’ordre partiel permettant d’atténuer le problème de l’explosion combinatoire dans le cas des réseaux de Petri temporels.

2-Développer un outil permettant la génération de l'espace d'états en se basant sur notre technique de réduction d'ordre partiel pour pouvoir la comparer avec d'autres techniques existantes dans la littérature.

Plan du mémoire

Le présent mémoire se divise en cinq chapitres. Le deuxième chapitre constitue une introduction aux notions de base des réseaux de Petri simples (Rdp) et temporels (TPN). Nous y abordons aussi d'autres concepts utilisés dans la suite du mémoire.

Le troisième chapitre est consacré à la revue de littérature en relation avec les problèmes abordés dans ce mémoire.

Nous présentons les travaux de recherche portant sur les abstractions d'espaces d'états des TPN, les techniques d'ordre partiel et les outils développés pour la construction et la réduction de l'espace d'états.

Le quatrième chapitre décrit notre technique de réduction d'ordre partiel permettant d'atténuer le problème d'explosion combinatoire et du coup de réduire l'espace d'états.

Dans le cinquième chapitre, nous présentons notre outil développé qui permet de générer l'espace d'états en appliquant soit notre technique soit d'autres techniques vues dans le chapitre « revue de littérature ».

Le dernier chapitre permet de présenter un résumé sur nos travaux de recherche, leurs limites et quelques pistes de recherche futures.

CHAPITRE 2 PRÉLIMINAIRES

Comme nous l'avons mentionné dans le chapitre « introduction », nous utilisons les réseaux de Petri pour modéliser les systèmes temps réel.

C'est un modèle riche permettant l'analyse des systèmes simples ou complexes incluant les concepts de synchronisation et de concurrence.

Dans ce présent chapitre, nous présentons dans un premier temps les notions de base et la dynamique des réseaux de Petri simples (Rdp).

Ensuite, nous présentons succinctement quelques extensions temporelles des réseaux de Petri, et définissons formellement la sémantique des réseaux de Petri temporels. Nous terminons ce chapitre par des définitions d'autres concepts utilisés dans ce mémoire.

2.1 Notions de base

2.1.1 Présentation des réseaux de Petri

C'est un modèle mathématique crée par Carl Adam Petri ("Réseau de Petri ", n.d.) en 1962 lors de ses travaux de thèse. Les réseaux de Petri permettent de modéliser différents systèmes dynamiques à variables ou événements discrets.

Un réseau de Petri (Rdp) (S.Vialle, 2007) (V.Augusto, 2012) est formé de places, de transitions et d'arcs :

- Place : peut correspondre à une précondition, une post condition, une donnée en entrée et/ou en sortie ou encore une ressource. Elle est représentée par un cercle. L'ensemble des places est désigné par P. Une place contient 0 ou n jetons.

Figure 2-1 : Une place contenant 0 jeton.

Figure 2-2 : Une place contenant 2 jetons.

- Transition : peut correspondre à un traitement ou à une activité. Généralement, elle est représentée par un rectangle. L'ensemble des transitions est désigné par T.

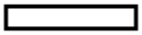

Figure 2-3 : Modélisation d'une transition.

- Arc : permet de lier soit une place à une transition soit une transition à une place. C'est donc un élément de (P X T) U (T X P).

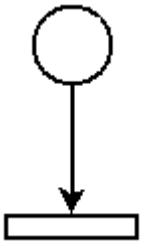

Figure 2-4 : Un arc connectant une place à une transition.

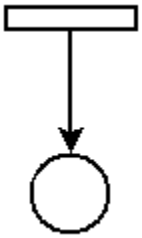

Figure 2-5 : Un arc connectant une transition à une place.

2.1.2 Marquage

Chaque place du réseau peut contenir 0 ou n jetons. Le marquage dénoté M correspond au nombre de jetons dans chaque place du réseau. C'est une fonction de P dans N, N étant l'ensemble des entiers naturels. Pour chaque p, M(p) est le nombre de jetons disponibles dans p.
Considérons le réseau de Petri ci-dessous, composé de 4 places et de 3 transitions :

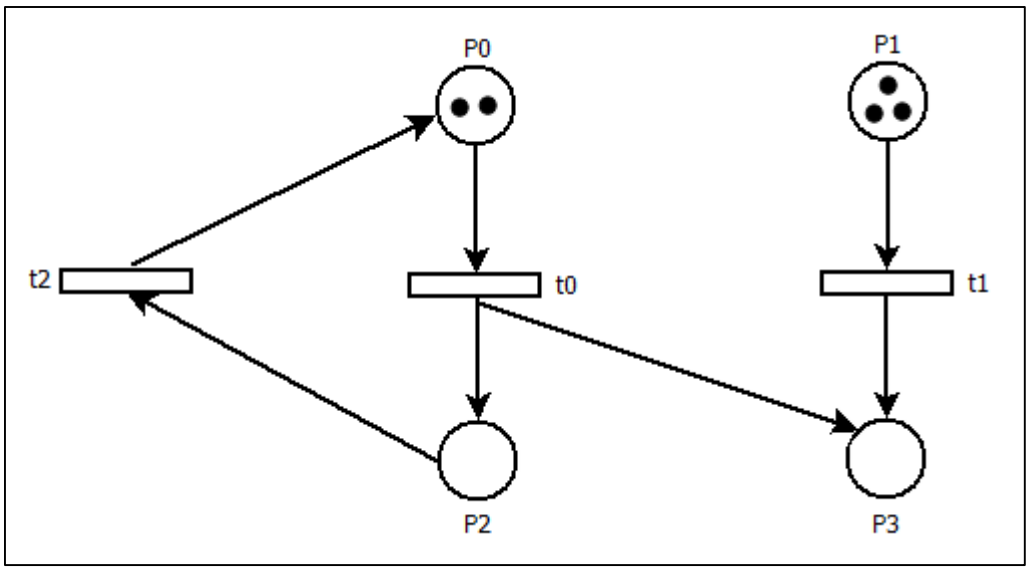

Figure 2-6 : Un exemple de réseau de Petri.

Le marquage M est : M(P0)=2, M(P1)=3, M(P2)=M(P3)=0. En effet, on a 2 jetons dans la place P0, 3 jetons dans la place P1 et 0 jeton dans les deux places P2 et P3.

À noter que le marquage M peut aussi être vu comme un vecteur où chaque élément de ce vecteur correspond au nombre de jetons contenu dans une place du réseau de Petri étudié. Dans le cas du Rdp de la figure 2-6, le marquage M est le vecteur [2 3 0 0].

2.1.3 Matrice Pre et Post

- **Pre :** c'est une matrice d'ordre |P| x |T| permettant de spécifier les poids des arcs en entrée des transitions. Pour une place p et une transition t, Pre(p, t) est le nombre de jetons de p nécessaires au franchissement de t. Pre(p, t) peut aussi être vu comme une fonction. Ci-dessous la matrice Pre du réseau de Petri de la figure 2-6.

$$Pre = \begin{array}{c} \\ p0 \\ p1 \\ p2 \\ p3 \end{array} \begin{array}{c} \begin{array}{ccc} t0 & t1 & t2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

À noter que Pre(ti) est le vecteur colonne de la matrice Pre d'indice i. Par exemple, Pre(t1)= [0,1,0,0]. Il représente le nombre de jetons par place, nécessaire au franchissement ou à la sensibilisation de la transition ti.

- **Post :** c'est une matrice d'ordre |P| x |T| permettant de spécifier les poids des arcs en sortie des transitions. Pour une place p et une transition t, Post(p, t) est le nombre de jetons produits par t dans p. Post(p, t) peut aussi être vu comme une fonction.
  Ci-dessous la matrice Post du réseau de Petri de la figure 2-6.

$$Post = \begin{array}{c} \\ p0 \\ p1 \\ p2 \\ p3 \end{array} \begin{array}{c} \begin{array}{ccc} t0 & t1 & t2 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

  À noter que Post(ti) est le vecteur colonne de la matrice Post d'indice i. Par exemple, Post(t1)=[0,0,0,1]. Il représente le nombre de jetons par place produit suite au franchissement de la transition t1. Dans l'exemple, on a seulement un jeton à produire dans la place p3.

2.1.4 Définition d'un Rdp

On définit un Rdp par le n-uplet (P, T, Pre, Post, M0) où :

-P est l'ensemble de toutes les places du réseau.

-T est l'ensemble de toutes les transitions du réseau.

-Pre est la matrice d'incidence avant, c'est la matrice qui contient les poids sur les arcs reliant les places aux transitions.

-Post est la matrice d'incidence arrière, c'est la matrice qui contient les poids sur les arcs reliant les transitions aux places.

-M0 est le marquage initial.

Soient t une transition et p une place. On désigne par (Boucheneb & Barkaoui, 2014) :

°t : est l'ensemble de places en entrées de la transition t : $\{p \in P \mid Pre(p,t) > 0\}$.

t° : est l'ensemble de places en sorties de la transition t : $\{p \in P \mid Post(p,t) > 0\}$.

°p : est l'ensemble de transitions en entrées de la place p : $\{t \in T \mid Post(p,t) > 0\}$.

p° : est l'ensemble de transitions en sorties de la place p : $\{t \in T \mid Pre(p,t) > 0\}$.

° (°t) : est l'ensemble de transitions en entrée de la transition t, on pourrait l'écrire ainsi

$$\bigcup_{p \in {}^\circ t} {}^\circ p.$$

(t°)° : est l'ensemble de transitions en sortie de la transition t, on pourrait l'écrire ainsi

$$\bigcup_{p \in t^\circ} p^\circ.$$

2.2 La dynamique des Rdps

2.2.1 Sensibilisation d'une transition

Une transition t est sensibilisée si chaque place en entrée de t contient au moins le nombre suffisant de jetons qui est indiqué sur l'arc entre la place et la transition t.

C'est-à-dire, $\forall p \in P, on\ a\ M(p) \geq Pre(p,t)$.

Dans le marquage courant du Rdp de la figure 2-7, nous avons 2 jetons dans P0 et 1 jeton dans P1. La transition t est sensibilisée, car il faut au moins 2 jetons dans P0 (valeur indiquée sur l'arc) et au moins un jeton dans P1 (si on ne mentionne pas le nombre de jetons nécessaires pour franchir la transition sur l'arc, c'est que c'est égal à 1).

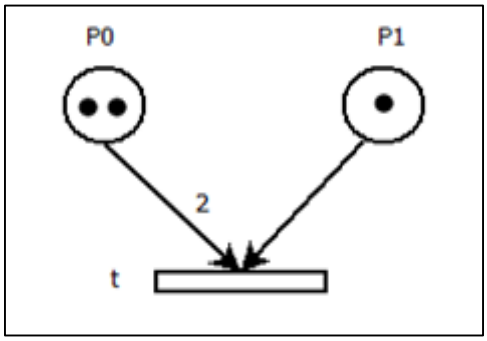

Figure 2-7 : Cas d'une transition sensibilisée.

Dans la figure 2-8, la transition t n'est pas sensibilisée vu que P0 contient un seul jeton alors que t nécessite au moins 2 jetons dans P0.

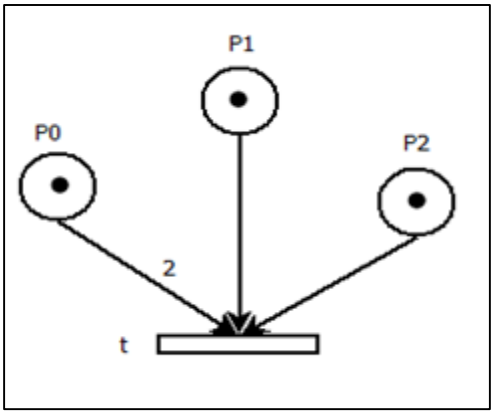

Figure 2-8 : Cas d'une transition non sensibilisée.

À noter que l'ensemble de transitions sensibilisées pour le marquage M est défini par :
$En(M) = \{t \in T \mid \forall p \in P, M(p) > Pre(p,t)\}$.

2.2.2 Tir d'une transition

Une transition sensibilisée peut être franchie. L'opération de tir (franchissement) d'une transition consiste à consommer les jetons des places en entrée et à produire les jetons aux places de sorties. L'ajout et l'enlèvement des jetons sont suivant les poids des arcs qui lient la transition aux places d'entrées et de sorties.

Le nouveau marquage M' en franchissant la transition t à partir d'un marquage M est calculé comme suit :

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

Dans la figure 2-9, nous avons un Rdp avec un marquage initial M0= [2 3 0].

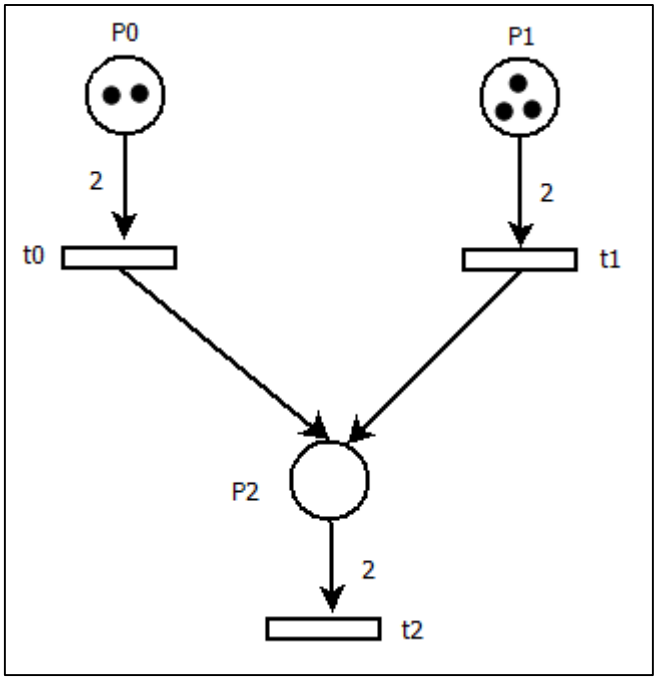

Figure 2-9 : Rdp avant le franchissement de la transition t1.

Dans la figure 2-10, nous franchissons la transition t1 ce qui nous mène à un nouveau marquage M' qui est [2 1 1]. Nous pouvons écrire ça d'une façon formelle : $M0\ [t1 >\ M'$.

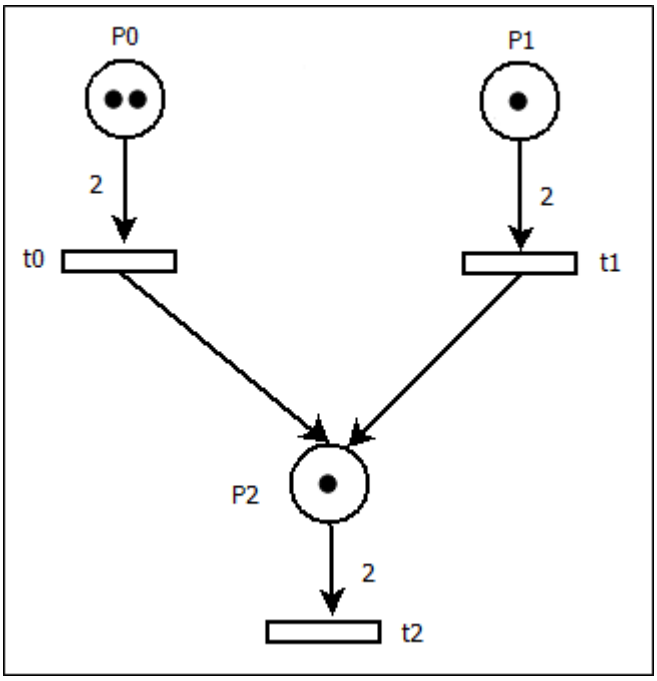

Figure 2-10 : Rdp après le franchissement de la transition t1.

2.2.3 Séquence de franchissement

Une séquence de franchissement w est une suite de transition (t1..., tk... tn) qui permet d'atteindre le marquage M_{n+1} à partir du marquage M1 en tirant successivement les transitions qui composent w.

On a : $M1\ [t1\ > M2\ \ldots . Mk\ [tn >\ M_{n+1}$.

2.2.4 Graphe de marquages

C'est un graphe qui contient tous les marquages accessibles à partir du marquage initial M0 et les séquences de transitions franchissables.

Il est défini par un triplé, $GM\ =\ (Marq, M0, [>)$ où $Marq = \{Mi, \exists\ w\ \in\ T^{+} \mid M0\ [w\ >\ Mi\}$.

Le graphe de marquages du Rdp de la figure 2-9 est présenté dans la figure ci-dessous.

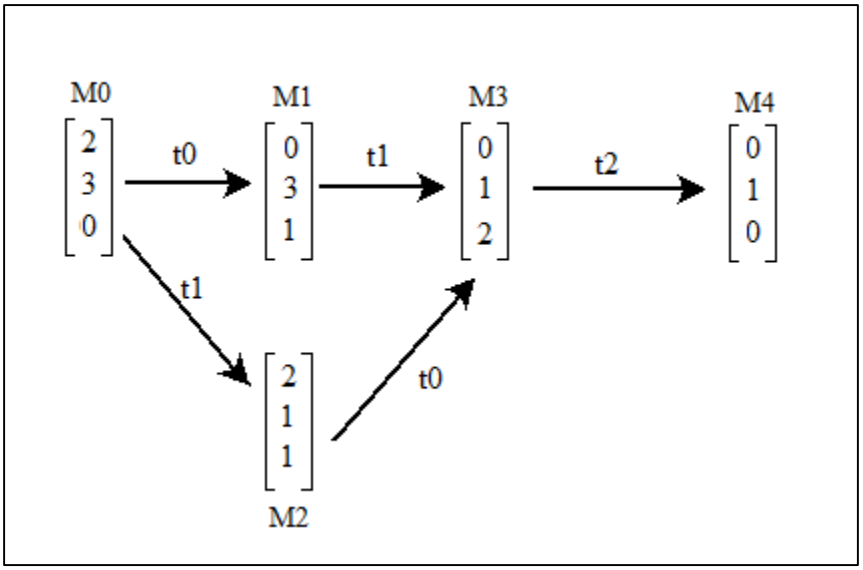

Figure 2-11 : Graphe de marquages du Rdp de la figure 2-9.

Dans le graphe de marquages (figure 2-11), un exemple de séquence qui peut être franchie à partir du marquage initial M0= [2 3 0] est w= t0 t1 t2, en effet nous avons $M0\ [t0\ t1\ t2 >\ M4$ (avec M4= [0 1 0]).

Par contre la séquence w = t2 t0 t1 ne peut pas être franchie à partir du marquage initial M0.

2.2.5 Conflit et parallélisme

2.2.5.1 Conflit

- **Conflit structurel**

Deux transitions t et t' sont en conflit structurel ssi elles ont au moins une place commune en entrée (Morère, 2012). C'est à dire $\exists\, p \in {}^\circ t \cap {}^\circ t'$.

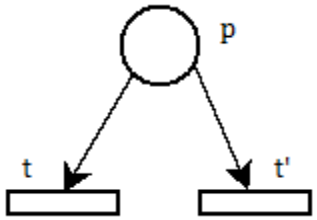

Figure 2-12 : Cas de conflit structurel.

À noter que (°t)° est l'ensemble de transitions qui sont en conflit structurel avec la transition t, nous pouvons l'écrire ainsi : $\bigcup_{p \in {}^\circ t} p^\circ$.

- **Conflit effectif**

Soit M un marquage, t et t' deux transitions sensibilisées dans M. On dit que t et t' sont en conflit effectif ssi elles sont déjà en conflit structurel et qu'il existe une place p commune en entrée à ces deux transitions telle que : $M(p) < Pre(p,t) + Pre(p,t')$.

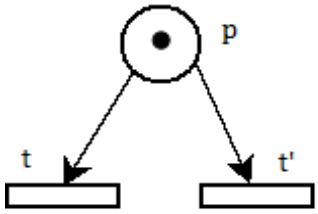

Figure 2-13 : Cas de conflit effectif.

On n'aurait pas de conflit effectif si on avait 2 jetons dans la place p.

2.2.5.2 Parallélisme

- **Parallélisme structurel**

Deux transitions t et t' sont parallèles structurellement (J-C.Geffroy, 2002) s'il n'existe aucune place commune en entrée de ces deux transitions $^\circ t \cap {}^\circ t' = \emptyset$.

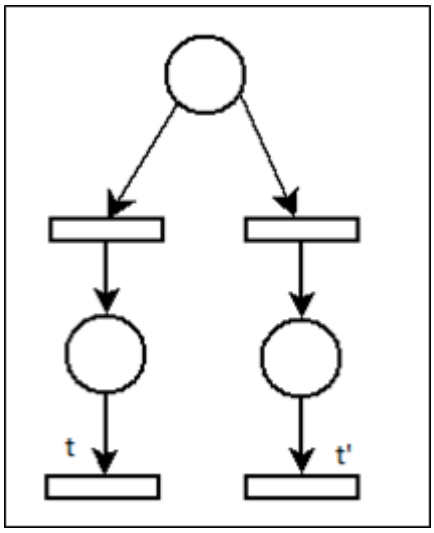

Figure 2-14 : Cas de deux transitions qui sont parallèles structurellement.

- **Parallélisme effectif**

  Deux transitions t et t' sont en parallélisme effectif si elles sont déjà en parallélisme structurel et de plus on a : $\forall p \in P, M(p) \geq Pre(p,t) + Pre(p,t')$.

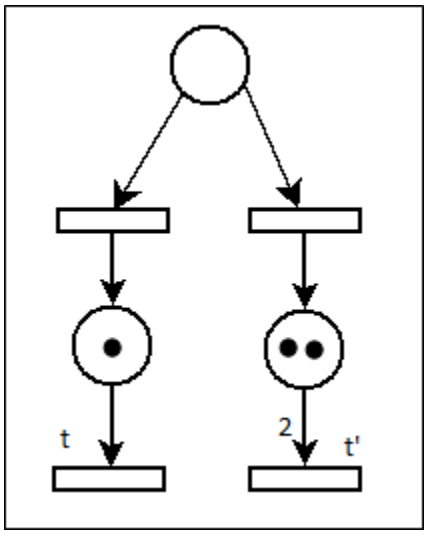

Figure 2-15 : Cas de deux transitions qui sont en parallélisme effectif.

2.2.6 Quelques propriétés

Dans ce qui suit, nous allons définir quelques propriétés utiles :

- Sureté (Safety) : cette propriété assure qu'un mauvais comportement ne pourra jamais arriver dans un système.
- Vivacité (Liveness) : un comportement valide finira par se réaliser.
- Adjacent : $pour\ t\ \in T, Adj(t) = \{t' \in T \mid (Pre(t) + Post(t))\ X\ (Pre(t') + Post(t'))\ \neq\ 0\ \}$ désigne l'ensemble de transitions qui sont connectées à au moins une place p qui est en entrée ou en sortie de t.
- Visibilité : une transition t est dite visible par rapport à une propriété φ si l'une de ses places en entrée ou en sortie est contenue dans cette propriété.
- Inter-blocage (Deadlock) : C'est atteindre un marquage où on pourrait plus franchir des transitions. C'est-à-dire qu'il existe un marquage Md tel que l'ensemble de transitions sensibilisées ou franchissables dans ce marquage est vide : $En(Md) = \emptyset$.

2.3 Réseaux de Petri incluant le temps

Il existe plusieurs types de réseaux de Petri qui incluent la notion du temps en associant des contraintes sur la durée ou le délai de franchissement de transitions ou encore sur la disponibilité des jetons dans les places.

Parmi ces réseaux, nous citons :

-Les réseaux de Petri temporels où on associe des intervalles de temps soit aux places et dans ce cas on les appelle réseaux de Petri P-temporels, soit aux transitions et on les appelle réseaux de Petri T-temporels.

- Les réseaux de Petri temporisés où on associe une durée de séjour minimale à chaque jeton soit aux places et dans ce cas on les appelle réseaux de Petri P-temporisés, soit aux transitions et on les appelle réseaux de Petri T-temporisés.

Dans notre étude, nous allons nous intéresser aux réseaux de Petri temporels (TPN) avec des intervalles de temps associés aux transitions. En effet, les TPN représentent un bon compromis entre la modélisation des systèmes temps réels (la concurrence, la synchronisation et la

spécification des contraintes temporelles directement sur les transitions) et les techniques de vérification. Les autres approches comme la modélisation à l'aide des automates sont beaucoup plus compliquées que les TPN. En effet, elles demandent plus d'horloges pour spécifier les contraintes temporelles, ce qui rend la phase d'analyse plus complexe. (Boucheneb & Barkaoui, 2014) (Boucheneb & Rakkay, 2008).

2.3.1 Définition d'un TPN

Un réseau de Petri temporel (TPN) est un n-uplet N= (P, T, Pre, Post, M0, Is) où (P, T, Pre, Post, M0) est un réseau de Petri simple et :

-Is : est une fonction qui indique l'intervalle statique de tir d'une transition t. ↓Is(t) représente la borne inférieure de cet intervalle tandis que ↑Is(t) représente la borne supérieure.

À partir du marquage initial M0, le réseau de Petri temporel évolue soit en faisant progresser le temps soit en franchissant des transitions. Lorsqu'une transition est sensibilisée, son intervalle de temps de tir est initialisé à son intervalle statique. Les bornes de cet intervalle diminuent progressivement avec le temps jusqu'à ce que la transition soit franchie ou désensibilisée par une autre transition en conflit.

Elle ne pourrait être franchie si la borne inférieure de son intervalle de temps de tir n'a pas encore atteint 0. Elle doit être immédiatement franchie sans délai supplémentaire si la borne supérieure de son intervalle de tir atteint 0.

Soit M un marquage et t une transition sensibilisée dans M (Boucheneb & Barkaoui, 2013):

- **Nw(M, t)** est l'ensemble de transitions nouvellement sensibilisées. Considérant le marquage M' qui est le marquage successeur de M en franchissant la transition t, Nw(M, t) contient t si $t \in En(M')$ et chaque transition de En(M') qui n'est pas sensibilisé dans le marquage intermédiaire $M - Pre(t)$.
  $Nw(M,t) = \{t' \in En(M') \mid t' = t \vee M - Pre(t) < Pre(t')\}$
- **CF(M, t)** est l'ensemble de transitions sensibilisées dans le marquage M et qui sont en conflit effectif avec t. $CF(M,t) = \{ t' \in En(M) | t' = t \vee M < Pre(t) + Pre(t') \}$.

2.3.2 Sémantique

On définit l'état d'un réseau de Petri temporel par un couple s= (M, I) où M est le marquage et I est la fonction qui indique l'intervalle de tir de chaque transition sensibilisée dans M.

Soient s= (M, I) et s'= (M', I') deux états, dh un réel strictement positif, t une transition (t ∈ T) et → la relation de transition définie par (Boucheneb & Barkaoui, 2013) :

$s \xrightarrow{dh} s'$ Si s' est accessible à partir de l'état s en faisant progresser le temps de dh unités, formellement on écrit :

$\forall t \in En(M), dh \leq \uparrow I(t), M' = M \; et \; \forall t' \in En(M'), I'(t') = [Max(0, \downarrow I(t') - dh),$
$\uparrow I(t') - dh]$.

En faisant progresser le temps, on n'aura pas un changement sur le marquage, seul le franchissement d'une transition permet de le faire. La modification va être au niveau des intervalles de tirs des transitions sensibilisées dans M' où on va retrancher dh unités de temps de leurs bornes inférieures et supérieures. Soit t' une transition sensibilisée dans M', si $dh \geq \downarrow I(t')$ alors la borne inférieure de tir de la transition t' va être égale à 0.

$s \xrightarrow{t} s'$ Si s′ est accessible à partir de l'état s en franchissant la transition t, formellement on écrit :

$t \in En(M), \downarrow I(t) = 0, \forall p \in P, \; M'(p) = M(p) - Pre(p,t) + Post(p,t)$ Et

$\forall t' \in En(M'), I'(t') = \quad Is(t') \; si \; t' \in Nw(M,t) \; , I(t') \; sinon.$

En franchissant une transition, on obtient un nouveau marquage M' calculé comme pour un réseau de Petri simple. Quant aux intervalles de tirs des transitions sensibilisées dans M', ils vont être égaux à leurs intervalles de tirs statiques si ces transitions sont nouvellement sensibilisées sinon ils restent inchangés.

La sémantique d'un TPN est définie par un système de transitions (S, →, s0) avec S est l'ensemble de tous les états accessibles à partir de l'état initial s0 par → * (la fermeture transitive et réflexive de la relation de transition → définie ci-dessus).

Un exemple de séquence : $Seq = s1 \xrightarrow{dh1} s1 + dh1 \xrightarrow{t1} s2 \xrightarrow{dh2} \ldots . sk \xrightarrow{tk} sn.$

Les séquences dh1 t1… dhk tk sont appelées les traces temporisées parce qu'elles incluent le temps (dh1…, dhk) et les séquences composées seulement de transitions sont appelées les traces non temporisées.

Dans la section suivante, nous allons introduire d'autres concepts qui sont importants pour le reste du mémoire. Parmi ces notions, on trouve les matrices de bornes (DBM) qui permettent de représenter les contraintes temporelles des TPN, et aussi la forme canonique qui facilite les calculs sur les DBM.

2.4 Autres notions

2.4.1 Matrice de bornes

C'est une structure de donnée qui a été définie par David Dill (Dill, 1990). Elle permet de représenter les contraintes entre deux variables vi et vj où chaque élément mij de cette matrice M représente la borne supérieure de la différence de ces deux variables.

Soit n le nombre de variables, la matrice DBM aura pour dimension (n+1) x (n+1). En effet, on ajoute la ligne et la colonne 0 pour représenter les contraintes contenant une seule variable comme par exemple le cas de $vi \leq c$.

Voici comment on calcule les éléments mij de la matrice de borne à partir des contraintes (Ismail Berrada, 2005). Si on a une contrainte de la forme :

- $vi - vj \leq c \; alors \; mij = c$.
- $vi - vj \geq c \; alors \; mij = -c$.
- $vi \leq c \; alors \; mi0 = c$.
- $vi \geq c \; alors \; m0i = -c$.
- Si on n'a pas de contraintes entre deux variables alors $mij = \infty$.

À noter que la DBM peut être représentée par un graphe appelé graphe de contraintes où chaque variable vi (y compris 0) est représentée par un sommet et chaque élément mij est représenté par un arc qui a pour poids la valeur c : $vj \xrightarrow{c} vi \; (si \; i \neq j)$.

Considérons l'ensemble de contraintes suivant : $v1 \leq 6 \wedge \; v2 \geq 3 \; \wedge \; v1 - v2 \leq 2$.

La DBM sera comme suit : MB=$\begin{pmatrix} 0 & \infty & -3 \\ 6 & 0 & 2 \\ \infty & \infty & 0 \end{pmatrix}$.

Soit la matrice de bornes MD=$\begin{pmatrix} 0 & \infty & \infty & 2 \\ \infty & 0 & \infty & 1 \\ 2 & \infty & 0 & \infty \\ 1 & 1 & \infty & 0 \end{pmatrix}$, le graphe associé est :

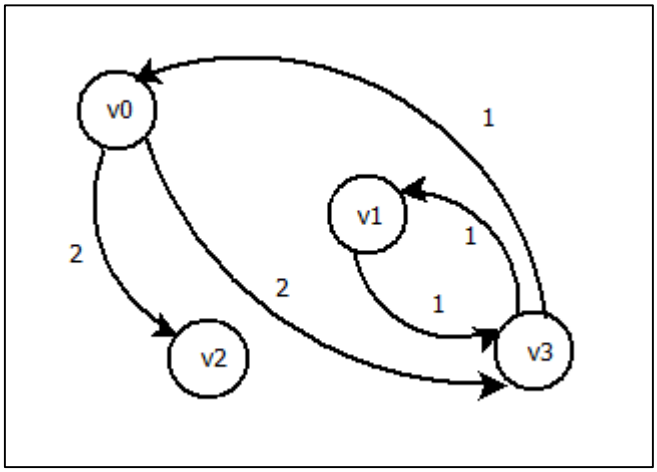

Figure 2-16 : Graphe associé à la matrice de bornes MD.

2.4.2 Forme canonique

Pour comparer deux DBM, il suffit de comparer leurs formes canoniques. La forme canonique est une forme réduite de la matrice DBM en propageant l'effet de chaque variable sur les autres. Pour cela, on utilise l'algorithme Floyd-Warshall pour retourner les chemins les plus courts entre les variables composant la DBM. Voici l'algorithme de Floyd-Warshall :

$FC = MD$;

$Pour\ k\ de\ 0$ à $n+1$

$Pour\ i\ de\ 0$ à $n+1$

$Pour\ j\ de\ 0$ à $n+1$

$FC_{ij} = min\left(FC_{ij}, FC_{ik} + FC_{kj}\right)$;

En appliquant cet algorithme sur la DBM MD, nous obtenons la matrice :

$$FC=\begin{pmatrix} 0 & 3 & \infty & 2 \\ 2 & 0 & \infty & 1 \\ 2 & 5 & 0 & 4 \\ 1 & 1 & \infty & 0 \end{pmatrix}.$$

2.4.3 Modèle cheking

Le modèle cheking (Clarke, Grumberg, & Peled, 1999) est une technique qui permet de vérifier d'une façon automatique divers types de systèmes.

Le modèle cheking prend en entrée le modèle du système étudié et la propriété à prouver. Il vérifie alors si une propriété est satisfaite pour le système étudié.
Si la propriété n'est pas satisfaite, le modèle cheking fournit un contre-exemple. La figure 2-17 résume le fonctionnement du modèle cheking.

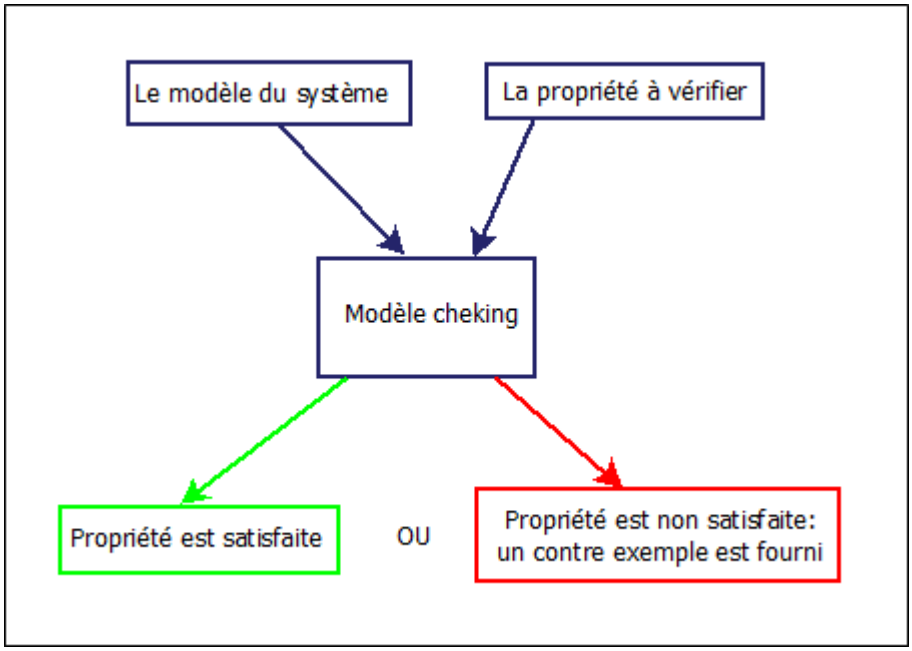

Figure 2-17 : Modèle cheking.

Ce chapitre a pour but de se familiariser avec les concepts de base des réseaux de Petri simples (Rdp) et des réseaux de Petri temporels (TPN).
Nous avons détaillé leurs fonctionnements et leurs sémantiques, ceci permettra de bien comprendre les notions à voir dans les chapitres suivants.
Dans le prochain chapitre, nous allons nous concentrer sur les réseaux de Petri temporels.
Nous allons voir quelles sont les problématiques liées aux TPN ainsi que les méthodes et les techniques utilisées dans la littérature pour pallier ces problèmes.

CHAPITRE 3 REVUE DE LITTÉRATURE

Dans ce chapitre, nous allons nous concentrer sur les techniques de vérification par énumération du modèle de Merlin (TPN). Le graphe d'états dans les TPN contient tous les états s= (M, I) qui sont calculés en franchissant les séquences de transitions possibles dans ce TPN à partir de l'état initial.

En associant des intervalles de temps aux transitions, cet espace d'états devient infini. En effet, le temps est continu et à partir de chaque état on pourrait avoir une infinité d'états successeurs en faisant évoluer le temps.

Du coup, la phase d'analyse et de vérification des systèmes modélisés avec les TPN s'avère très compliquée sur le graphe d'états.

Pour surmonter ce problème, plusieurs méthodes dites d'abstraction ont été proposées dans la littérature. Le but est d'abstraire le temps et de représenter dans un graphe fini les séquences de franchissement réalisables.

De plus, et dans le but de réduire l'espace d'états pour avoir de meilleures performances lors de la vérification des propriétés, on a conçu les techniques de réduction d'ordre partiel. Le principe de ces techniques est de minimiser le nombre de transitions à franchir à partir de chaque état, ceci permet de réduire d'une façon considérable l'espace d'états des systèmes temps réel.

Nous allons parcourir les méthodes d'abstractions ainsi que les techniques de réduction d'ordre partiel qui sont les plus répandues dans la littérature, et aussi présenter certains outils développés pour la construction de l'espace d'états.

3.1 Les méthodes d'abstractions

Pour pallier au problème de l'infinité d'états successeurs causé par la continuité du temps, on pourrait procéder à réaliser des abstractions. Ces dernières permettent de réunir dans une même classe, les états résultant du franchissement, à des dates différentes, de la même séquence de transitions. Tous ces états partagent le même marquage, mais ayant des informations temporelles différentes.

On distingue plusieurs abstractions d'états pour un TPN, elles diffèrent dans la façon de représenter, de calculer et de regrouper les états :

-Abstractions basées sur les intervalles de temps de tirs des transitions sensibilisées (Berthomieu, Bernard & Vernadat, 2003).

-Abstractions basées sur les horloges (Boucheneb & Hadjidj, 2006).

Elles se distinguent aussi par les propriétés préservées. Les propriétés linéaires d'un réseau de Petri temporel sont les propriétés qui portent sur les séquences de tir et de marquages du TPN. Une abstraction de l'espace d'états d'un TPN préserve les propriétés linéaires si et seulement si toutes les séquences de tir du TPN sont exactement les séquences de tir de l'abstraction.

Dans notre mémoire, nous allons nous intéresser aux propriétés linéaires et aux abstractions basées sur les intervalles car elles sont plus compactes (c'est-à-dire qu'elles permettent d'avoir un espace d'états plus réduit) que celles qui sont basées sur des horloges (Boucheneb & Rakkay, 2008).

3.1.1 SCG (State Class Graph)

C'est une abstraction introduite par Berthomieu et Menasche en 1982 ainsi que par Berthomieu et Diaz en 1991 (Berthomieu, B. & Diaz, 1991). On caractérise une classe d'états par le couple $\alpha = (M, F)$ où M est le marquage et F la formule qui caractérise l'union de domaines de tirs des états formant α.

La figure 3-1 montre qu'à partir de la classe d'états initiale $\alpha 0$ et en franchissant la même transition t, on obtient plusieurs états : s1, s2... sn qui partagent le même marquage M1, mais qui sont franchis à des dates différentes. Par conséquent, on pourrait les réunir tous dans une seule classe d'états α.

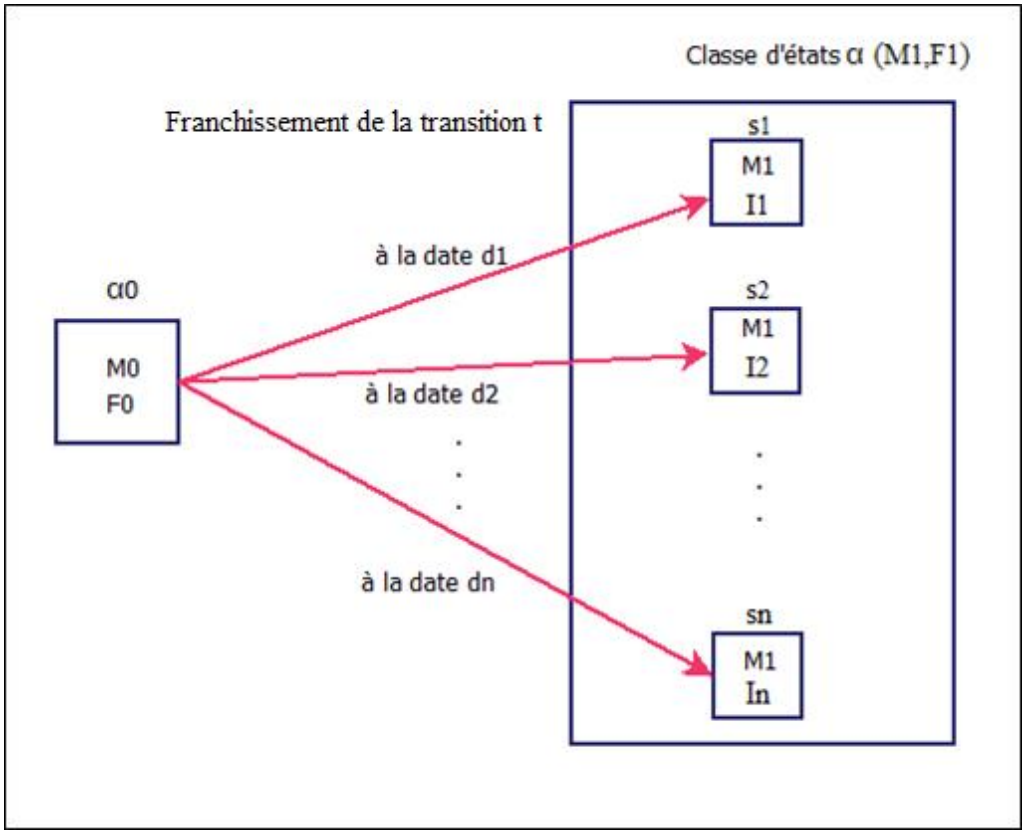

Figure 3-1 : Regroupement des états dans une seule classe.

La formule F est définie par une conjonction de contraintes atomiques. Une contrainte atomique a la forme suivante : $x - y \prec c, x \prec c$ *ou encore* $-x \prec c$, où x et y sont des variables réelles, $\prec \in \{<, =, \leq, \geq, >\}$ et c un nombre rationnel.

Soit la classe d'états initiale définie par α0 = (M0, F0) avec M0 le marquage initial et F0 la formule qui contient les contraintes temporelles définie par : $F0 = \bigwedge_{t \in En(M0)} \downarrow I0(t) \leq t \leq \uparrow I0(t)$, où ↑I0(t) désigne la borne supérieure de l'intervalle de tir statique de la transition t et par↓I0(t) sa borne inférieure.

En franchissant la transition tf à partir de α= (M, F), on obtient une nouvelle classe d'états $\alpha' = (M', F')$, où M' est calculé comme pour un Rdp classique c'est-à-dire $\forall p \in P, M'(p) = M(p) - Pre(p, tf) + Post(p, tf)$ et F' est calculée comme suit :

1) Initialiser F' à $F \wedge \bigwedge_{t \in En(M)} tf - t \leq 0 \wedge \bigwedge_{t\prime \in Nw(M,tf)} \downarrow Is(t') \leq t'^{n} - tf \leq \uparrow Is(t')$.

2) Mettre F' à la forme canonique et éliminer toutes les transitions qui sont en conflit avec tf (c'est-à-dire éliminer les transitions qui appartiennent à CF(M, tf)).

3) Renommer chaque t'^{n} par t'.

La notation t'^{n} traite la situation où t' est sensibilisée avant le franchissement de tf et nouvellement sensibilisée par tf. La nouvelle instance de t' est temporairement représenté par t'^{n} dans l'étape 1.

Considérons l'exemple de Rdp de la figure 3-2, la classe initiale est $\alpha 0 = (p1 + p2, -1 \leq t1 - t2 \leq 1)$. Il y a deux transitions t1 et t2 qui sont sensibilisées et qui sont aussi franchissables à partir de α0.

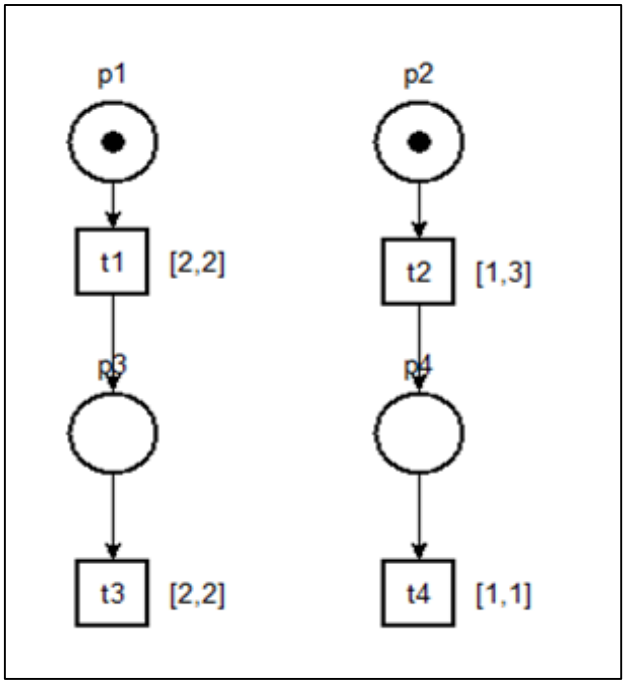

Figure 3-2 : Exemple de TPN pour le calcul de classe d'états successeur.

Nous allons calculer la classe α1 successeur de α0 en franchissant la transition t1.

1) Mettre la formule F' à la condition de tir de t1 à partir de α0 et ajouter la contrainte de la transition t3 qui est nouvellement sensibilisée par t1 : $-1 \leq t1 - t2 \leq 1 \wedge t1 \leq t2 \wedge t'^{3} - t1 = 2;$

2) Mettre la formule F' sous la forme canonique et éliminer t1 en réalisant des substitutions (on a l'équation $t'^{3} - t1 = 2$, d'où $t1 = t'^{3} + 2$. On substitue alors chaque t1 par $t'^{3} + 2$). Finalement, on obtient : $-2 \leq t2 - t'^{3} \leq -1;$

3) Renommer t'^{3} par t3 : $-2 \leq t2 - t3 \leq -1.$

La classe obtenue est $\alpha 1 = (p2 + p3,\ -2 \leq t2 - t3 \leq -1)$.

De point de vue pratique, F est représentée à l'aide d'une matrice de bornes DBM (Difference Bound Matrices) qui permet de représenter les bornes des différences entre les variables et de calculer la propagation de l'effet de chaque entrée sur les autres de la DBM.

Considérons le Rdp de la figure 3-2. Dans cet exemple, nous allons représenter les contraintes temporelles F0 de la classe initiale α0 à l'aide d'une matrice de bornes, notée D0.

Soit M un marquage, ti et tj deux transitions tel que $ti, tj \in En\ (M)$. Pour le calcul des éléments de D0, on procède comme suit :

$D0(i,i) = 0, D0\ (0,ti) = -\downarrow Is(ti), D0\ (ti,0) = \uparrow Is(ti)$.

$D0(ti,tj) = \uparrow Is(ti) - \downarrow Is(tj)$.

Voici la matrice de bornes D0 du réseau de Petri de la figure 3-2.

$$D0 = \begin{array}{c} \\ \mathbf{0} \\ \mathbf{t1} \\ \mathbf{t2} \end{array} \begin{array}{c} \begin{array}{ccc} \mathbf{0} & \mathbf{t1} & \mathbf{t2} \end{array} \\ \begin{pmatrix} 0 & -2 & -1 \\ 2 & 0 & 1 \\ 3 & 1 & 0 \end{pmatrix} \end{array}$$

Dans le chapitre 5 nous allons voir l'algorithme de calcul de D' qui représente la matrice de bornes de la classe d'états α' successeur de la classe d'états α.

On utilise la représentation canonique de la DBM, ceci rend les opérations sur les formules beaucoup plus faciles. Une classe d'états est désormais représentée par α= (M, D).

Le graphe résultant de cette technique contient toutes les classes d'états calculées en éliminant celles qui sont équivalentes. Deux classes d'états sont égales s'ils ont le même marquage M et la même matrice D.

Soit StatesClassG l'ensemble de toutes les classes d'états contenues dans le graphe de classes d'états.

Dans la figure suivante, on représente l'algorithme qui permet de générer le graphe de classes d'états.

```
State_Class_Graph ( Graphe G) {

Tant que ( vrai ) {

pour chaque classe d'états α dans StatesClassG

    VectFR ← TR_Franchissable ( α )

        pour chaque tf dans VectFR faire {

        α' ← Successeur ( α , tf)

        si ( α' ∉ StatesClassG )
        Ajouter α' à G


        }


si ( Il n'y a plus de nouvelle classe d'états )
break

}

}
```

Figure 3-3 : Algorithme de construction du graphe SCG.

Au début, le graphe de classes d'états contient la classe initiale α0. À partir de chaque classe d'états, on calcule l'ensemble de transitions franchissables. Ensuite, on calcule les successeurs de cette classe par ces transitions-là. Enfin, on s'assure pour chaque successeur qu'il n'existe pas dans le graphe avant de l'ajouter. Une fois qu'on n'a plus de successeur, soit parce qu'il n'y a plus de transitions franchissables ou soit parce que tous les successeurs sont déjà existants, la construction du graphe est terminée.

3.1.2 CSCG (Contracted State Class Graph)

C'est une version contractée de la méthode SCG (Boucheneb & Rakkay, 2008). En effet, dans la matrice D qui représente la forme canonique de la matrice F de la classe d'états α, on a une ligne 0 et une colonne 0 : elles représentent les délais minimaux et maximaux pour que chaque transition sensibilisée soit franchie. Dans le cas de CSCG, on ne tient pas compte de cette ligne et de cette colonne dans la comparaison de deux classes d'états.

Dans (Boucheneb & Rakkay, 2008) les auteurs ont montré que la colonne et la ligne 0 ne sont pas nécessaires pour calculer le successeur de la classe d'états α= (M, D). De plus, les classes qui sont équivalentes sans tenir compte de la ligne et de la colonne 0 ont les mêmes séquences de tirs.

Soit la classe initiale définie par α0 = (M0, F0) avec M0 le marquage initial et $F0 = \bigwedge_{t,t' \in En(M0)} t - t' \leq \uparrow Is(t) - \downarrow Is(t')$, en franchissant une transition tf on obtient une nouvelle classe d'états α'= (M', F') telle que M' est calculé de la même façon qu'un réseau de Petri simple et F' est calculée en appliquant le même algorithme que pour la méthode SCG.

Pour la construction du graphe de classes d'états, il s'agit du même calcul que pour la méthode SCG, sauf que pour comparer deux états afin de voir s'ils s'agissent du même état ou non, on compare les deux marquages et les deux matrices D sans considérer la ligne et la colonne 0. Le résultat de la comparaison permet de retenir une classe calculée et l'ajouter dans le graphe de classes d'états.

La méthode d'abstraction CSCG permet alors d'avoir un espace d'états plus compacte que celui de SCG et par conséquent CSCG est plus performante.

3.2 Les techniques de réduction d'ordre partiel :

Les techniques de réduction d'ordre partiel sont des techniques permettant la réduction de l'espace d'états des réseaux de Petri (Valmari, 1992), (Dams, Gerth, Knaack, & Kuiper, 1998), (Penczek & Półrola, 2001). En effet, elles permettent de représenter une partie bien définie du comportement du système global, d'où vient le mot partiel. Cette représentation 'partielle' doit contenir les informations nécessaires pour permettre de bien analyser le système temps réel en question et surtout de pouvoir vérifier les propriétés.

Du coup, la réduction du graphe ne doit pas éliminer les informations pertinentes, il faut garder l'information utile afin de vérifier les propriétés d'intérêt.

Deux transitions sont dites indépendantes, si aucune d'entre elles ne peut sensibiliser ou désensibiliser l'autre et le franchissement de ces deux transitions dans différents ordres (c'est-à-dire tirer la première puis la deuxième et vice-versa) mène vers le même état.

Le choix de l'ensemble de transitions à tirer à partir de chaque état est basé sur cette notion d'indépendance. En effet, l'idée derrière les techniques de réduction d'ordre partiel est d'identifier

les séquences équivalentes. Ces séquences sont obtenues par la permutation de certaines transitions qui sont indépendantes.

Au lieu d'explorer les classes d'états accessibles par toutes ces séquences équivalentes, on choisit seulement une séquence qui sera la représentante d'elles. Cette dernière est suffisante pour vérifier les propriétés d'intérêt.

Si les méthodes d'abstractions permettent de regrouper des états dans une même classe, les techniques de réduction d'ordre partiel permettent à partir d'une classe d'états de ne tirer que certaines transitions de l'ensemble de transitions qu'on pourrait tirer. C'est un sous ensemble de transitions représentatif qui permet d'englober les classes d'états accessibles, en franchissant toutes les transitions à tirer à partir d'une classe d'états α, dans une seule classe d'états.

La figure 3-4 est un exemple qui montre le gain possible en appliquant les techniques de réduction d'ordre partiel. Sur la gauche, nous avons un exemple de graphe de classes d'états initial, c'est-à-dire sans appliquer une technique de réduction d'ordre partiel.

À partir de la classe d'états α0, on pourrait tirer quatre transitions qui nous mèneront vers les quatre classes d'états : α1, α2, α3 et α4. Dans le graphe de droite, il s'agit du graphe de classes d'états possible en appliquant une technique de réduction d'ordre partiel. Au lieu de tirer quatre transitions, on tire t1 et t4 seulement.

En effet, la classe d'états α0 préserve la propriété φ1. En franchissant t1 et t3, nous aurons les classes d'états α1 et α3. Ces deux classes d'états préservent la même propriété qui est φ2 et du coup nous pouvons retenir seulement la classe d'états α1. De même, à partir de α0 nous pouvons tirer t2 et t4 qui nous mènent vers deux classes d'états α2 et α4 qui préservent directement ou indirectement la propriété φ3, nous pouvons alors garder la classe d'états α4.

Nous remarquons alors le gain obtenu en termes de classes d'états en utilisant une technique de réduction d'ordre partiel.

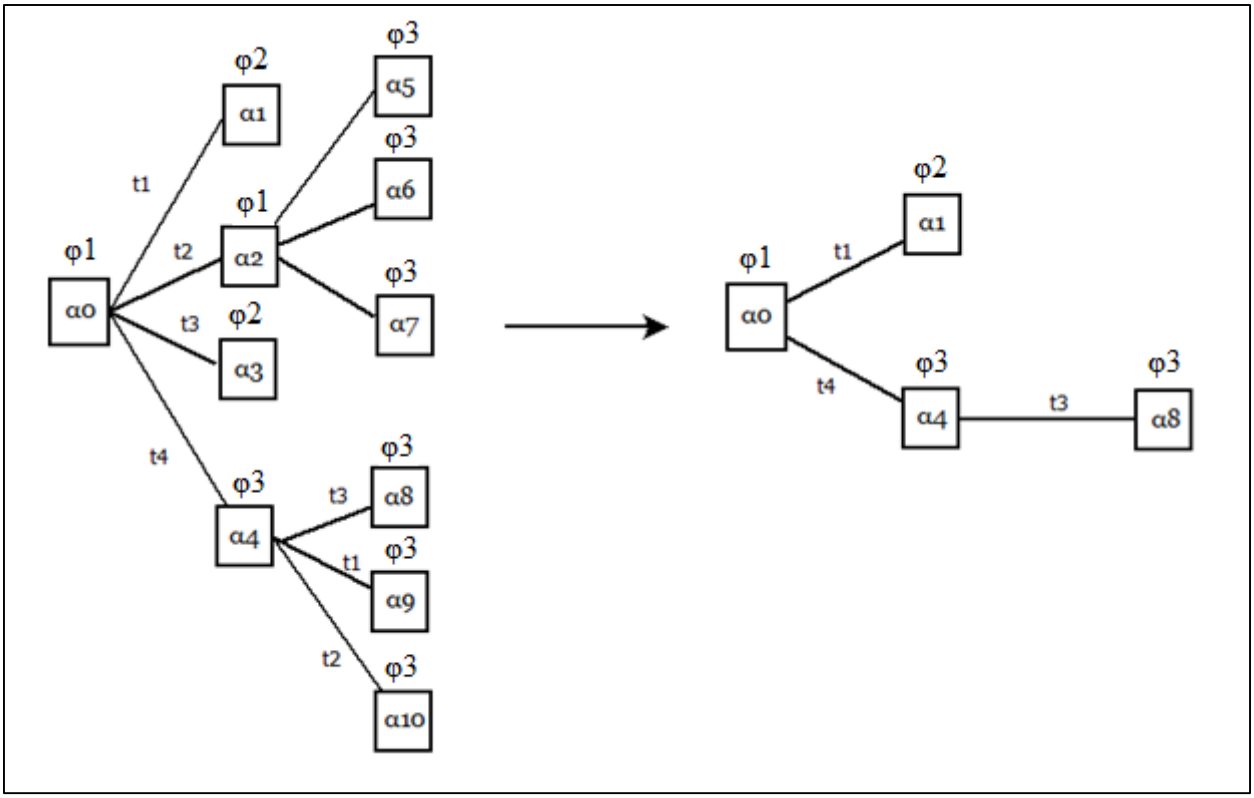

Figure 3-4 : Exemple de résultat en appliquant une technique de réduction d'ordre partiel.

L'explosion combinatoire dans notre contexte, est le cas où une petite variation du nombre d'entrées c'est-à-dire : les transitions, les places ou les arcs d'un réseau de Petri engendrera un espace d'états très grand qui tend vers l'infini. C'est-à-dire plus la complexité du système croît, plus le nombre de classes d'états croît d'une façon exponentielle.

Les techniques de réduction d'ordre partiel permettent d'atténuer le problème d'explosion combinatoire en enlevant la redondance et les différentes similitudes dans la représentation exhaustive du comportement du système à étudier.

Considérons le réseau de Petri de la figure 3-5, parcourir le chemin t1t2 ou bien t2t1 nous mènera vers le même marquage.

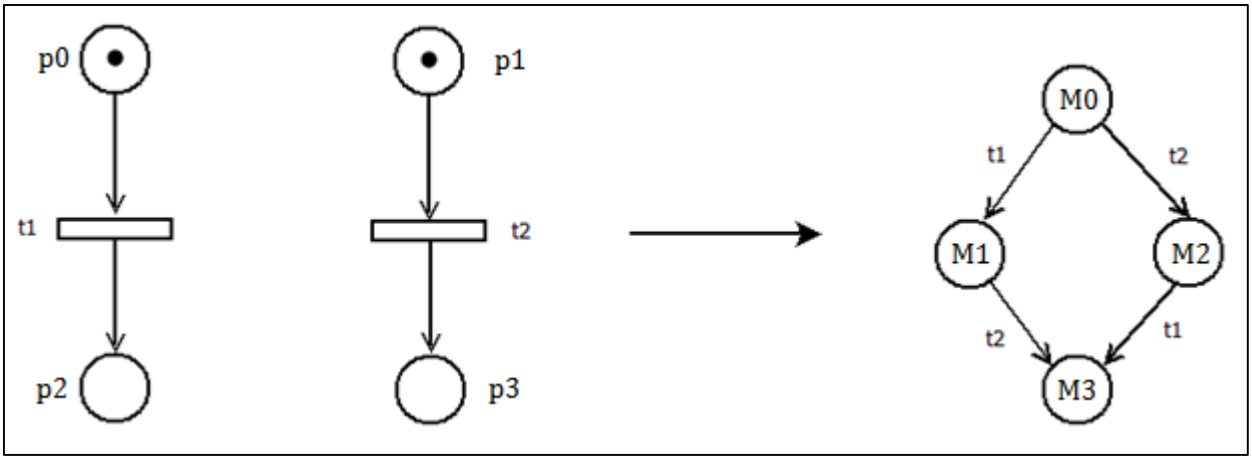

Figure 3-5 : Un exemple de réseau de Petri avec son graphe de marquages.

D'où les techniques de réduction d'ordre partiel tentent à remédier la cause du problème d'explosion combinatoire qui est la représentation du parallélisme avec l'entrelacement des transitions. Puisque les deux chemins t1t2 et t2t1 mènent vers le même marquage, ça sera plus optimal de n'explorer qu'un seul chemin parmi les chemins possibles si les marquages intermédiaires n'ont pas d'impacts sur la valeur de vérité des propriétés à vérifier.

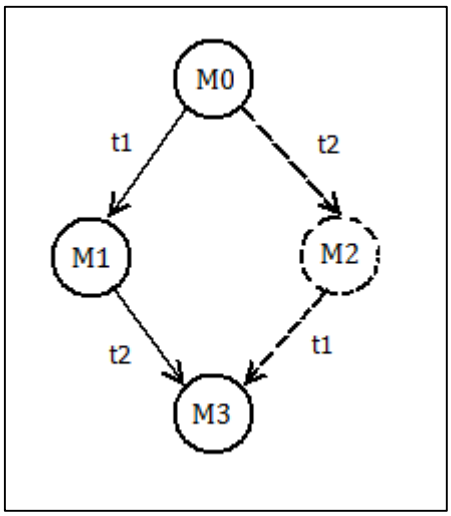

Figure 3-6 : Exploration d'un seul chemin.

Les techniques de réduction d'ordre partiel permettent alors de choisir une séquence représentative de l'ensemble de séquences de transitions équivalentes.

3.2.1 Stubborn Set

C'est une technique de réduction d'ordre partiel développée par Valmari (Valmari, 1991). Cette méthode préserve les interblocages (deadlocks) et les propriétés linéaires comme le safety et le liveness.

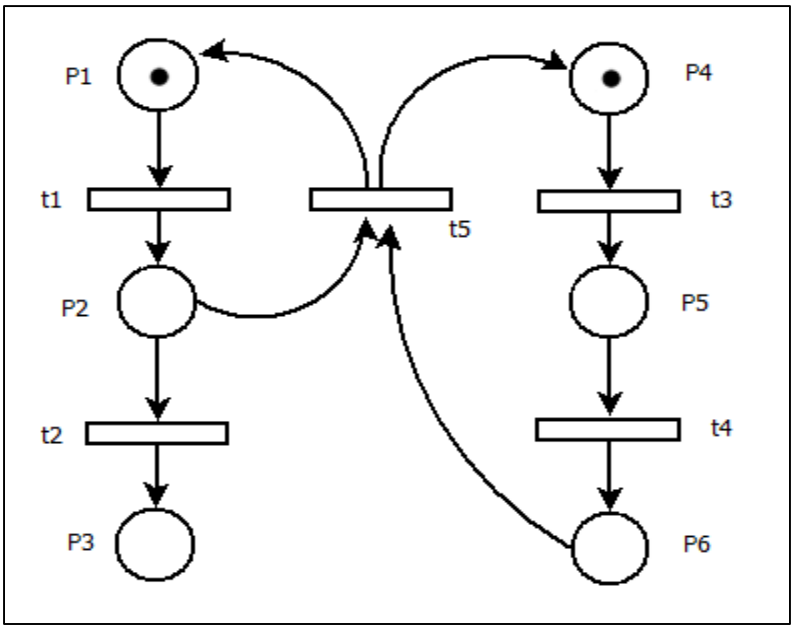

Figure 3-7 : Réseau de Petri Stubs.

Considérons le réseau de Petri de la figure 3-7 (Valmari, 1994). Initialement, les deux transitions t1 et t3 sont sensibilisées. Vu qu'elles sont indépendantes, ça ne sera pas nécessaire d'explorer les deux transitions. En effet, comme le montre la figure 3-8 (une portion du graphe de marquages du Rdp de la figure 3-7), à partir du marquage 100100 ça revient au même de tirer t1t3 ou t3t1 : le résultat est atteindre le marquage 010010.

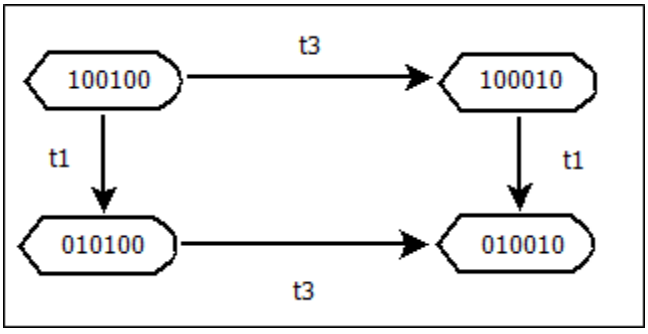

Figure 3-8 : Une partie du graphe de marquages du Rdp Stubs.

Au marquage 010100, les deux transitions t2 et t3 sont indépendantes. Choisir d'étudier le franchissement de t2 seulement n'est pas suffisant, car t2 et t5 ne sont pas indépendantes. Même si t5 n'est pas sensibilisée dans ce marquage, elle pourrait l'être en franchissant t3 et t4. Si on franchit seulement t2 on va perdre d'autres marquages qui sont utiles et du coup la méthode Stubborn Set n'est pas seulement basée sur l'indépendance des transitions, elle prend en considération les dépendances indirectes.

Un ensemble de transitions TS est dit Stubborn Set (Evangelista & Pradat-Peyre, 2006) (Sami Evangelista, 2006) dans le marquage M si :

(1) Pour chaque transition t non sensibilisée dans TS, pour une place vide p en entrée de t, toutes les transitions °p sont incluses dans TS.

Par cette condition on voudrait s'assurer que toute transition qui soit capable de sensibiliser t est dans TS, c'est-à-dire aucune transition à l'extérieur de TS ne pourrait avoir un impact sur une transition à l'intérieur de TS.

À noter que tj peut sensibiliser t, s'il existe au moins un état où le franchissement de tj conduit à la sensibilisation de t.

(2) Aucune transition ti sensibilisée dans TS ne possède une place en entrée en commun avec une transition tj à l'extérieur de TS même si tj n'est pas sensibilisée.

On voudrait s'assurer qu'aucune transition en conflit avec ti (qui pourrait désensibiliser ti) n'est à l'extérieur de TS.

(3) TS contient au moins une transition sensibilisée.

En résumé, les transitions se trouvant à l'intérieur de TS doivent être toutes indépendantes à toutes les transitions qui sont à l'extérieur. Aucun effet des transitions à l'extérieur sur l'ensemble TS.

Pour un marquage M, l'ensemble TS est construit comme suit :

-On choisit une transition sensibilisée t et on initialise TS à {t}.

-On applique les 2 règles suivantes jusqu'à l'obtention d'un point fixe (c'est-à-dire qu'il n'y a plus de changement sur l'ensemble de transitions TS) :

1/ Si t est sensibilisée dans M alors on inclut (°t)° dans TS.

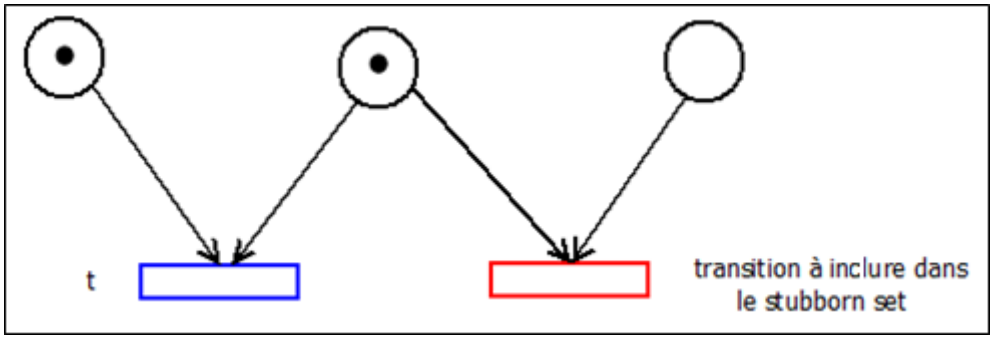

Figure 3-9 : Inclure (°t)° dans l'ensemble TS.

2/ Si t n'est pas sensibilisée pour M, on choisit p ∈ °t et on inclut °p dans TS

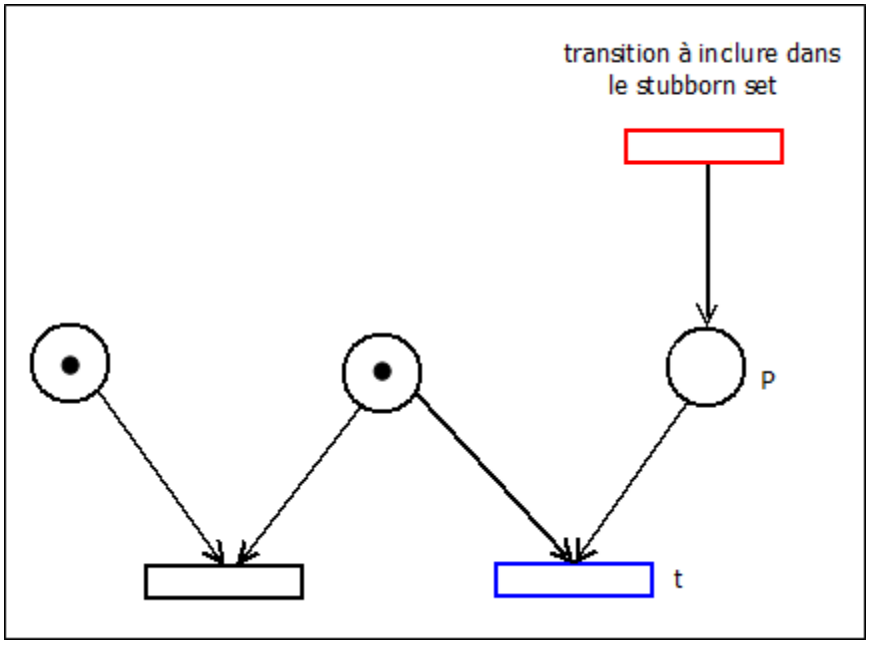

Figure 3-10 : Inclure °p dans l'ensemble TS.

3.2.1.1 Extension de la technique dans le contexte temporel

Une extension de la méthode de Stubborn Set dans le contexte temporel a été proposée par R.H. SLOAN et U.BUY (Sloan & Buy, 1997) à l'université d'Illinois de Chicago.

Dans cette méthode, les auteurs définissent le Simple Time Petri Net (STP). Un STP est défini par n-uplet (P, T, F, M0, DI) où P, T, F et M0 sont identiques pour un réseau de Petri simple, c'est-à-dire P représente l'ensemble des places, T l'ensemble de transition, F l'ensemble des arcs : on distingue les arcs qui relient les places aux transitions et les arcs qui relient les transitions aux places, M0 le marquage initial et DI contient les contraintes de temps sur les transitions.

Le STP est un cas particulier du modèle de Merlin (c'est-à-dire les TPN) où tous les intervalles de franchissement sont des singletons.

En effet DI représente le délai pour qu'une transition sensibilisée puisse être franchie, c'est-à-dire que t doit être franchie après DI unités de temps à moins qu'elle soit désensibilisée par une autre transition.

Ils définissent le délai dynamique par di(t) qui est initialisée à DI(t), ça représente le temps restant pour tirer la transition t. À tout moment, on a : $0 \leq di(t) \leq DI(t)$.

Un état est défini comme suit : e = (M, FI) où M est le marquage et FI est un vecteur contenant les délais dynamiques de l'ensemble des transitions sensibilisées.

Une transition t est franchissable dans un état e ssi elle est sensibilisée et qu'elle possède le plus petit délai dynamique entre toutes les transitions sensibilisées dans e, c'est à dire $di(t) <= di(t')$ avec t' sensibilisée dans e.

Une transition est dite courante si elle est franchissable ou bien son délai statique DI est égal à zéro, sinon on l'appelle retardée.

3.2.1.1.1 Définition d'un semi-Stubborn Set

Soit un état défini par e = (M, FI). Un ensemble Ens de transitions d'un réseau STP est dit semi-Stubborn Set dans e si pour chaque transition t ∈ Ens : t est courante dans e et de plus on a :

1/ Si t est franchissable dans e, alors $\forall\, t' \in T - Ens\ soit\ {}^\circ t \cap {}^\circ t' = \emptyset$ ou bien t' est retardée.

2/ Si t n'est pas sensibilisée dans M, alors pour certaine place vide p ∈ °t, chaque transition dans °p est soit dans Ens ou bien retardée.

3.2.1.1.2 Définition d'un Stubborn Set

Un ensemble Ens de transitions d'un réseau STP est dit Stubborn Set s'il est semi-Stubborn Set et qu'il contient au moins une transition franchissable.

3.2.1.2 Autre extension basée sur le calcul de l'ensemble G(α)

La technique basée sur le calcul de l'ensemble G(α) est une technique de réduction d'ordre partiel. C'est une extension de la méthode Stubborn Set dans le cas des réseaux de Petri temporels.

Soit $A = (Ac, succ, \alpha 0)$ un espace d'états d'un TPN.

Les auteurs (Boucheneb & Barkaoui, 2014) ont défini l'espace d'états réduit $R = (G, A_G, succ_G, \alpha 0)$ où A_G est l'ensemble des classes d'états accessibles dans R à partir de la classe d'états initiale α0, G la fonction qui permet de générer un ensemble de transitions et $succ_G$ est une fonction définie par :

$\forall \alpha = (M, F) \in Ac, \forall tf \in T, succ_G(\alpha, tf) \neq \emptyset \ ssi \ succ(\alpha, tf) \neq \emptyset.$

Si $succ_G(\alpha, tf) \neq \emptyset$ alors la classe d'états $\alpha' = succ_G(\alpha, tf)$ est calculée en appliquant le même algorithme pour CSCG avec la condition de tir suivante :

$F \wedge \bigwedge_{ti \in En(M) \cap G(\alpha)} tf \leq ti.$

Ils définissent les conditions suivantes pour étendre la méthode de Stubborn Set dans le contexte des TPN.

Soit α une classe d'états et w une séquence de transitions. On dénote 'stub' le Stubborn Set, l'ensemble de transitions vérifiant :

CD0 : $Fr(\alpha) \neq \emptyset \Leftrightarrow stub \cap Fr(\alpha) \neq \emptyset$.

CD1 : $\exists t \in stub \cap Fr(\alpha), \forall w \in (T - stub)^+, succ(\alpha, w) \neq \emptyset \Longrightarrow succ(\alpha, wt) \neq \emptyset.$

CD2 : $\forall t \in stub, \forall w \in (T - stub)+, succ(\alpha, wt) \neq \emptyset \Longrightarrow succ(\alpha, tw) = succ(\alpha, wt).$

La condition CD0 assure que le Stuborn set d'une classe d'états α est vide ssi α est un deadlock.

Les deux autres conditions CD1 et CD2 assurent que le franchissement des transitions à l'extérieur de stub n'a aucun effet sur les transitions à l'intérieur de stub.

Par la suite, les auteurs proposent une réécriture des conditions CD0, CD1 et CD2. Soit α ∈ Ac une classe d'états, G(α) est un Stubborn Set ssi il satisfait les conditions suivantes :

C0 : $Fr(\alpha) = \emptyset \Leftrightarrow \mathrm{G}(\alpha) \cap Fr(\alpha) = \emptyset$.

C1 : $\exists tf \in \mathrm{G}(\alpha) \cap Fr(\alpha), \forall w \in \left(T - \mathrm{G}(\alpha)\right)^+, succ(\alpha, w) \neq \emptyset \Longrightarrow succ(\alpha, wtf) \neq \emptyset.$

C2 : $\forall tf \in \mathrm{G}(\alpha), \forall w \in \left(T - \mathrm{G}(\alpha)\right)^+,$

$$succ(\alpha, wtf) \neq \emptyset \Longrightarrow succ(\alpha, wtf) \subseteq succ(\mathrm{succ}_{\mathrm{G}}(\alpha, tf), w).$$

C3 : $\forall tf \in \mathrm{G}(\alpha), \forall w \in T^{+}$,

$$succ(\mathrm{succ}_{\mathrm{G}}(\alpha, tf), w) \neq \emptyset \Longrightarrow \exists w' \equiv tfw, succ(\alpha, w') \neq \emptyset.$$

Les conditions C0 et C1 sont identiques aux conditions CD0 et CD1. Pour C2, cette condition permet de dire que la classe d'états accessible en franchissant la séquence wtf peut être couverte par le calcul du successeur par w, de la classe d'états successeur de α par tf.

Quant à la condition C3, elle assure que pour n'importe quelle séquence w franchie de $succ_G(\alpha$, tf), α possède au moins une séquence franchissable, équivalente à tfw.

Pour le calcul de G(α), les auteurs ont défini une condition suffisante appelée SC2 qui permet de satisfaire les conditions C1, C2 et C3.

SC2 est définie par : R ⊨ SC2 ssi ∀α = (M, F) ∈ A_{G}, ∀t ∈ G(α),

1 : $\forall p \in {}^{\circ}t, M(p) < pre(p,t) \wedge {}^{\circ}p \subseteq G(\alpha)$.

2 : $\forall p \in {}^{\circ}t, M(p) \geq pre(p,t) \wedge p^{\circ} \subseteq G(\alpha)$.

3 : $t \in En(M) - Fr(\alpha) \Longrightarrow \forall t' \in Fr(\alpha)\ s.t\ (F \wedge t' < t) \equiv F, t' \in G(\alpha)$.

4 : $t \in Fr(\alpha) \Longrightarrow ({}^{\circ}({}^{\circ}t) \cup (t^{\circ})^{\circ}) \subseteq G(\alpha)$.

Finalement, les auteurs présentent un algorithme pour calculer le Stubborn Set permettant de satisfaire les conditions C0 et SC2 (qui satisfait déjà C1, C2 et C3).

La figure ci-dessous montre cet algorithme.

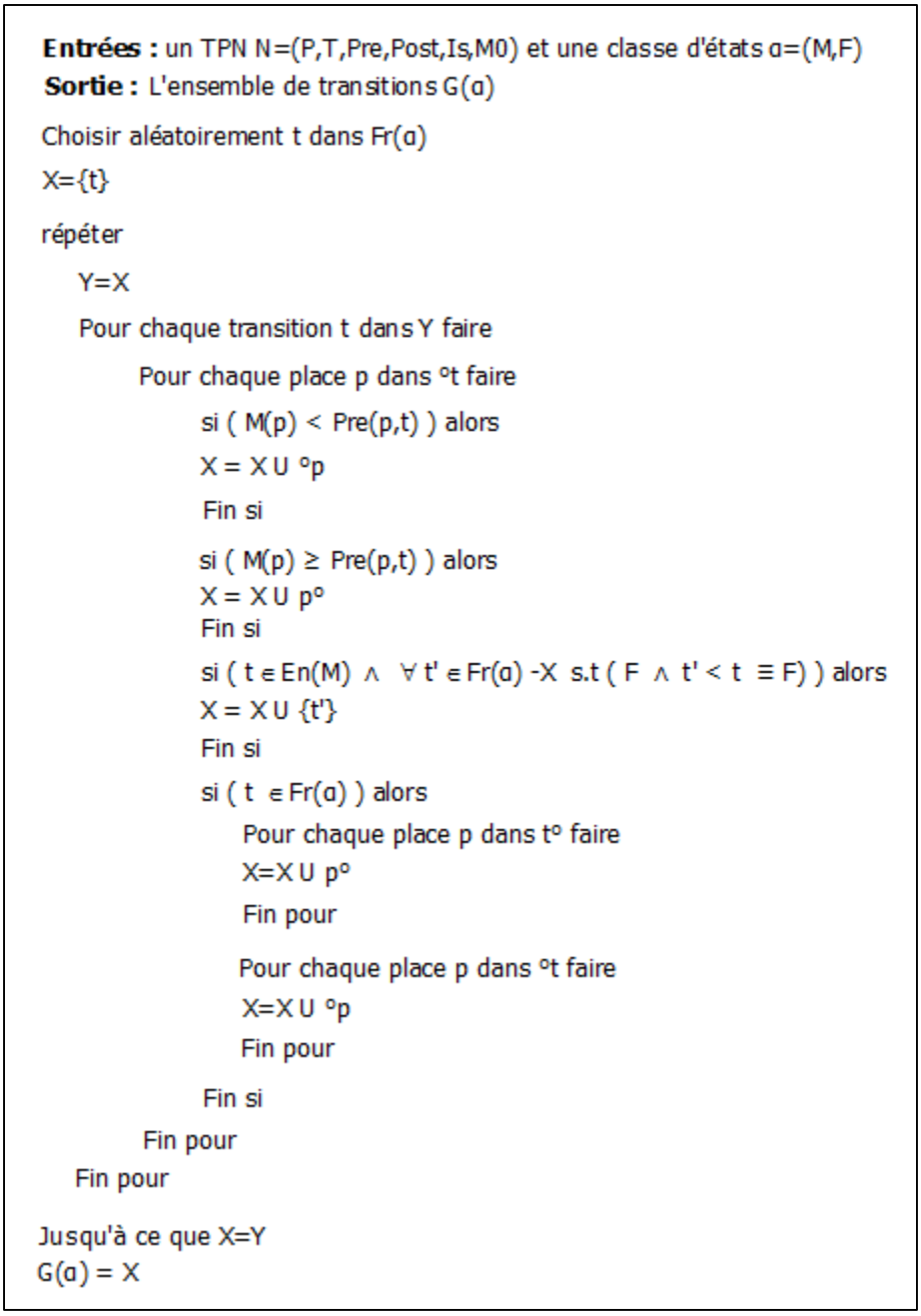

Figure 3-11 : Algorithme de calcul de G(α).

3.2.2 Persistent set et Sleep set

Les deux techniques Ensembles Persistants (Persistent Set) et Ensembles Dormants (Sleep Set) ont été développés par Godefroid (Godefroid, van Leeuwen, Hartmanis, Goos, & Wolper, 1996). Un ensemble de transitions PS est dit Persistent Set pour une classe d'états α, si toutes les séquences

de transitions possibles à tirer à partir de la classe α qui n'appartiennent pas au Persistent Set sont indépendantes à toutes les transitions dans PS.

À partir de chaque classe d'états, on ne tire que les transitions qui sont à l'intérieur de Persistent Set. Cette technique permet de préserver les deadlocks et les propriétés linéaires.

Quant à la technique Sleep Set, elle permet de garder l'historique des traces d'exécution déjà explorées. Ceci permet de ne pas ré-exécuter les traces équivalentes.

L'ensemble Sleep Set contient alors l'ensemble des traces à ne pas exécuter à partir de la classe d'états α. Cette technique ne permet pas un gain en termes de classes d'états, mais elle permet d'éviter les transitions qui permettent de revisiter les mêmes classes d'états. C'est pour ça qu'on combine cette technique avec d'autres techniques comme le Stubborn Set ou le Persistent Set.

3.3 Les outils

Dans cette partie, nous allons présenter quelques outils qui sont les plus utilisés dans la littérature pour la construction et la réduction de l'espace d'états. Ceci nous permet d'avoir une idée sur les fonctionnalités qu'ils offrent ainsi que les techniques et les méthodes utilisées lors de la phase de développement de notre outil.

3.3.1 PROD

C'est un outil qui permet d'analyser l'accessibilité pour un réseau de Petri simple (Rdp) (Varpaaniemi & Rauhamaa, 1992). Il a été développé par le laboratoire « Systèmes Digitaux » à l'université de technologie d'Helsinki en Finlande.

C'est un outil qui supporte plusieurs techniques de réduction du graphe d'accessibilité (de marquages) permettant d'atténuer le problème d'explosion combinatoire. Parmi ces techniques, on trouve : le Stubborn Set et le Sleep Set.

3.3.1.1 Analyseur Prod

PROD est un outil basé sur les lignes de commandes. Parmi les composants qu'il contient (Varpaaniemi, Heljanko, & Lilius, 1997), on cite :

-prod : la commande principale qui permet d'utiliser les autres composants.

-prpp c'est un programme qui permet à partir d'un fichier de description d'un réseau de Petri de générer un programme C. Une fois compilé et exécuté, ce programme génère le graphe d'accessibilité.

-probe : c'est un outil qui permet d'examiner et d'analyser le graphe d'accessibilité en offrant un ensemble de commandes permettant de le parcourir.

-Strong : un outil qui permet de calculer les composants fortement connectés dans un Rdp.

3.3.1.2 Fichier d'entrée

Le fichier d'entrée utilisé par PROD pour décrire un réseau de Petri est basé sur le langage C/Préprocesseur. Il est étendu par des directives de description de réseau.

Dans ce fichier (Kimmo Varpaaniemi, 1995) qui a pour extension .net, on trouve principalement les éléments suivants :

- Places : une place est définie par un nom, et trois valeurs qui indiquent la borne inférieure et supérieure du marquage ainsi que le marquage initial.

#place nom_de_la_place [**lo**(limite_minimale)] [**hi**(limite_maximale)] [**mk** (marquage_initial)]
-Transitions : une transition est définie par un nom et par 2 listes de places. In et out représentent respectivement la liste de places d'entrées et de sorties de la transition.

#trans nom_de_la_transition
[**in** {nom_place : input-expression;...}]
[**out** {nom_place : output-expression;...}]
#endtr
La figure 3-12 montre un exemple de fichier contenant la description d'un Rdp avec PROD.

```
#ifndef n
#define n 25
#endif
#place p_1 mk(<..>)
#place p lo(<.2.>) hi(<.n.>) mk(<.2..n.>)
#place q lo(<.1.>) hi(<.n.>)
#place q_2
#place q_21
#trans t1
    in  { p_1: <..>; }
    out { q: <.1.>; }
#endtr
#trans t2
    in  { p: <.2.>; q: <.1.>; }
    out { p_1: <..>; q_2: <..>; }
#endtr
#trans t3
    in  { p: <.3.>; q_2: <..>; }
    out { p: <.2.>; q: <.3.>; }
#endtr
#trans t21
    in  { p: <.21.>; q: <.20.>; }
    out { p: <.20.>; q_21: <..>; }
#endtr
#trans t22
    in  { p: <.22.>; q_21: <..>; }
    out { p: <.21.>; q: <.22.>; }
#endtr
#trans u
    in  { p: <.x.>; q: <.x - 1.>; }
    gate ((x >= 4) && (x <= 20)) || (x >= 23);
    out { p: <.x - 1.>; q: <.x.>; }
#endtr
```

Figure 3-12 : Exemple de fichier de description d'un Rdp dans PROD.

3.3.1.3 Fonctionnement

La commande prod File.init exécute le programme prpp sur un File.net (fichier contenant la description du réseau de Petri à analyser) et génère un fichier C : File.c avec d'autres fichiers de données.

Par la suite, on pourrait compiler ce fichier (File.c) et l'exécuter en utilisant des options qui sont fournies par l'outil PROD.

La figure 3-13 résume le fonctionnement de l'outil PROD.

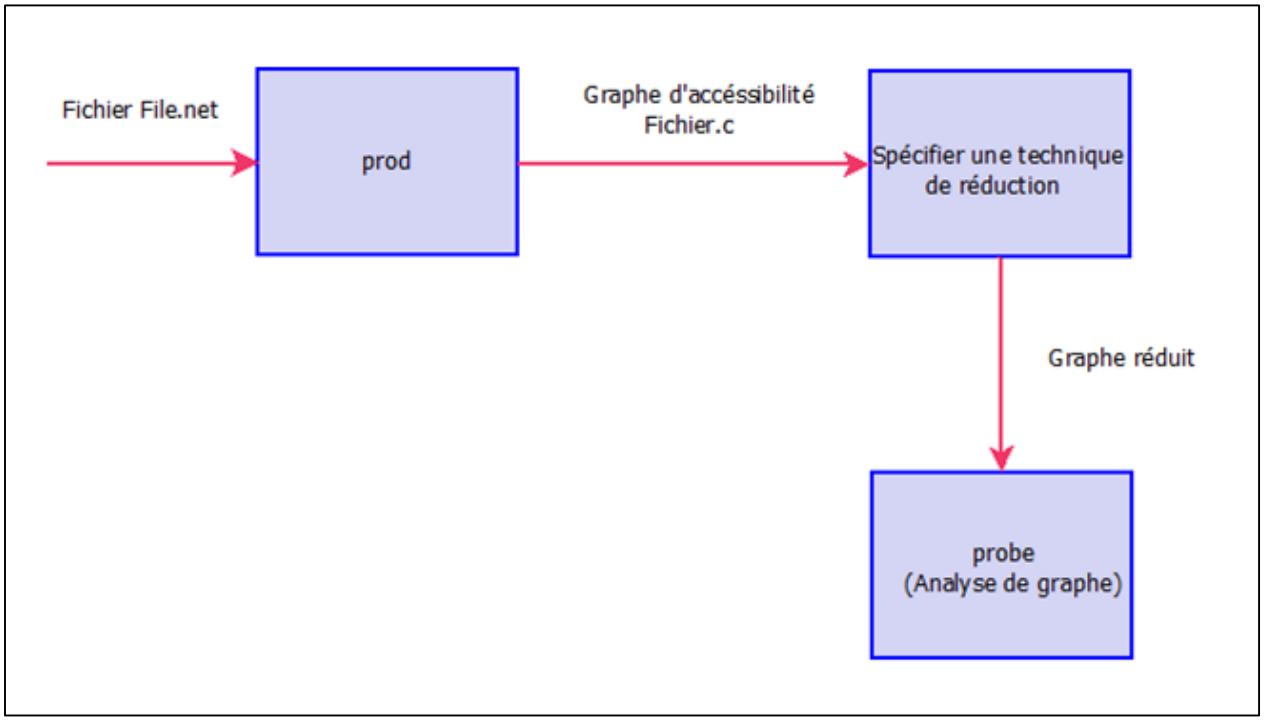

Figure 3-13 : Fonctionnement de PROD.

Parmi ces options, on trouve celles qui permettent de spécifier la technique de réduction à utiliser pour générer le graphe d'accessibilité.

-Stubborn Set : PROD calcule le Stubborn Set de deux manières différentes :

*) Une méthode incrémentale (option -s) : on commence par choisir une transition sensibilisée puis de construire le Stubborn Set au complet.

*) Une méthode basée sur la suppression (option -d) : cette méthode initialise le Stubborn Set avec l'ensemble de toutes les transitions sensibilisées puis supprime celles qui ne vérifient pas les conditions nécessaires pour qu'une transition appartienne à l'ensemble Stubborn Set.

-Sleep Sets (option –S) c'est une autre technique de réduction du graphe d'accessibilité. Elle utilise l'indépendance entre les transitions. Elle est souvent utilisée en combinaison avec d'autres techniques de réduction afin d'avoir de meilleurs résultats.

3.3.2 Tina

C'est une boîte à outils (http://projects.laas.fr/tina) riche en fonctionnalités qui permet d'éditer et d'analyser les réseaux de Petri simples (Rdp) et les réseaux de Petri temporels (TPN).

Cet outil a été développé dans le Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) à Toulouse, par les 2 groupes de recherche OLC et VerTICS.

3.3.2.1 Fonctionnalités offertes Par TINA

Parmi les plusieurs outils offerts par TINA, on cite :

nd : C'est un éditeur qui permet de modéliser un réseau de Petri (Rdp), un réseau de Petri temporel (TPN) ou encore un automate. Il permet de réaliser cette modélisation de deux façons : textuelle ou graphique.

La figure 3-14 représente l'éditeur graphique de TINA.

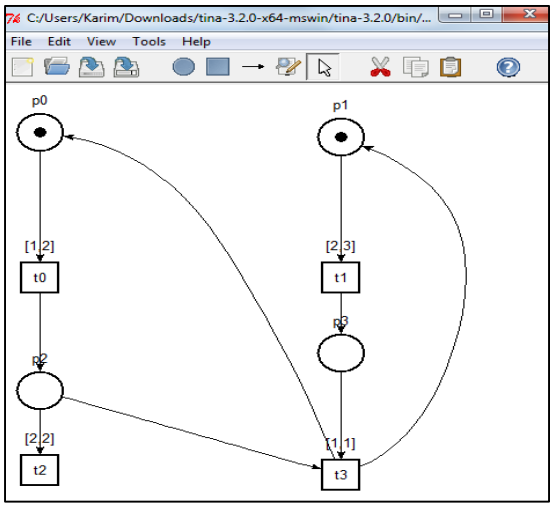

Figure 3-14 : L'outil nd.

tina : C'est l'outil qui permet l'abstraction et la génération de l'espace d'états d'un réseau de Petri simple ou temporel. Il prend comme entrée la description graphique ou textuelle d'un réseau.

Cet outil utilise plusieurs techniques afin de réaliser l'abstraction de l'espace d'états. On distingue les abstractions sur les :

-Réseaux de Petri simples (Rdp) : pour cela tina utilise les techniques de réduction d'ordre partiel pour atténuer le problème d'explosion combinatoire, à savoir (Berthomieu*, Ribet, & Vernadat, 2004) :

- Les ensembles persistants : C'est une technique qui a été développé par Godefroid. À partir de chaque état, on ne considère qu'un sous-ensemble de transitions sensibilisées. Les transitions appartenant à ce sous-ensemble sont toutes indépendantes à celles qui sont à l'extérieur.
- Graphe de pas couvrants : C'est un graphe constitué d'états et relié par des pas. Ces pas sont des ensembles de transitions qui sont indépendants.
- Graphe de pas persistants : Ce graphe permet de se focaliser sur la préservation des états de blocage dans la technique de graphe de pas couvrants. C'est une technique qui combine les ensembles persistants avec les pas couvrant. En effet, à partir de chaque classe d'états, les pas de transitions sont choisis uniquement sur les ensembles persistants.

-Réseaux de Petri temporels : pour cela, tina se base sur les classes d'états à savoir :

- LSCG : Classes d'états linéaires. C'est une technique basée sur l'abstraction SCG.

- SSCG : Classes d'états linéaires fortes. Les classes linéaires fortes sont définies par un marquage et un système d'inéquations basé sur les horloges. Cette technique est plus complexe que le LSCG.

- ASCG : Graphe des classes d'états atomiques. C'est une technique basée sur le partitionnement du graphe des classes linéaires jusqu'à l'obtention des classes stables (atomiques).

Une classe est dite stable si tous les successeurs des états contenus dans une classe CL sont tous contenus dans une même classe.

Dans les figures 3.15 et 3.16, nous présentons un exemple d'utilisation de la commande tina avec un réseau de Petri temporel (figure 3-14) ainsi que les résultats obtenus.

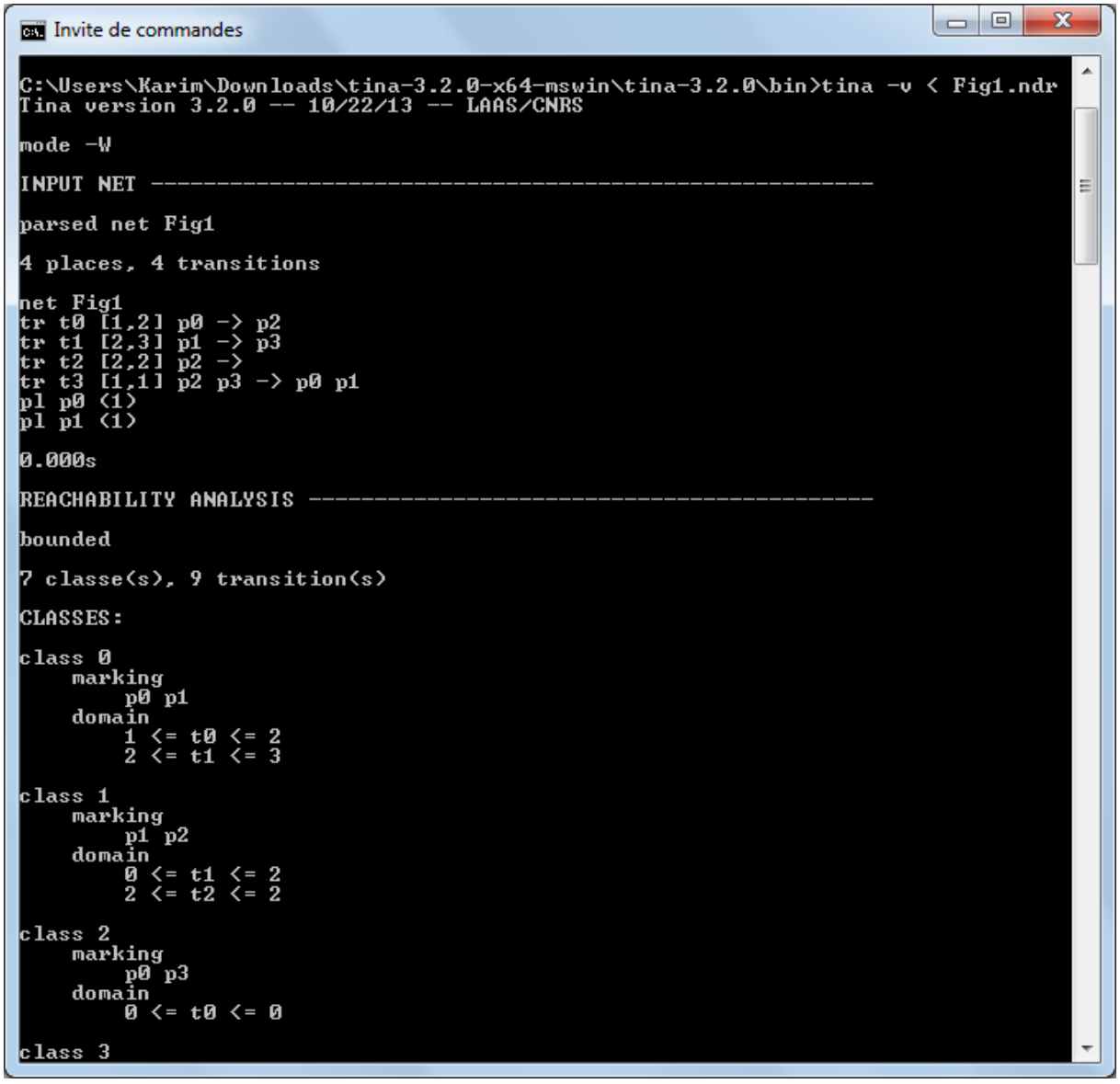

Figure 3-15 : Exemple d'utilisation de tina.

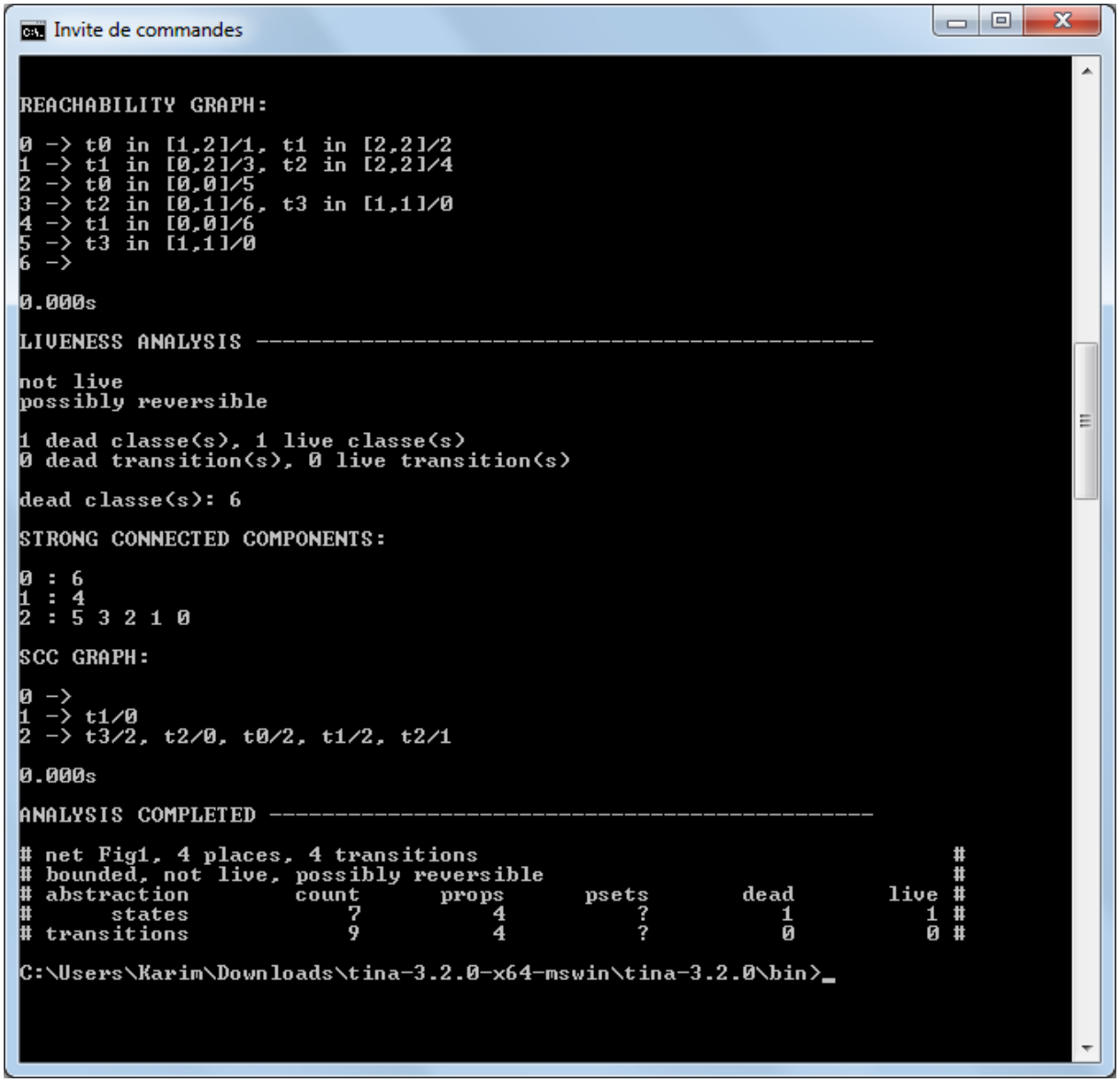

Figure 3-16 : Suite de l'exemple d'utilisation de tina.

selt : C'est l'outil de vérification et d'analyse. Il utilise comme entrée le graphe (de marquages pour le cas des réseaux de Petri simples ou le graphe de classes d'états pour les réseaux temporels) réduit suite à une abstraction.

sift : C'est un outil qui permet d'explorer et d'analyser les réseaux de Petri Rdp ainsi que les TPN d'une manière efficace, même s'il offre moins d'options que les autres outils de TINA. Il opère sur une description textuelle ou graphique de réseau.

Tous les outils de TINA peuvent être exécutés d'une façon indépendante ou combinée.

3.3.2.2 Fichier en entrée :

Plus qu'une dizaine de types de fichiers en entrée sont supportés par TINA. On trouve :

.net : c'est le format utilisé pour modéliser les réseaux de Petri temporel d'une façon textuelle.

.ndr : c'est le format utilisé pour modéliser les réseaux de Petri temporel d'une façon graphique.

.adr : c'est le format produit par l'éditeur graphique lorsqu'on représente les automates.

La figure 3-17 montre la grammaire utilisée dans TINA pour modéliser les TPN avec le format .ndr.

Grammar :

```
.ndr            ::= (<trdesc>|<pldesc>)* (<edgedesc>|<prdesc>)* <netdesc>
trdesc          ::= 't' <xpos> <ypos> <transition> <eft> <lft> <anchor>
                  | 't' <xpos> <ypos> <transition> <anchor> <eft> <lft> <anchor> <label> <anchor>
pldesc          ::= 'p' <xpos> <ypos> <place> <marking> <anchor> {<label> <anchor>}
edgedesc        ::= 'e' <place> <transition> {<arckind>}<weight> <anchor>
                  | 'e' <place> <ang> <rad> <transition> <ang> <rad> {<arckind>}<weight> <anchor>
                  | 'e' <transition> <place> <weight> <anchor>
                  | 'e' <transition> <ang> <rad> <place> <ang> <rad> <weight> <anchor>
prdesc          ::= 'e' <transition> <transition> 1 <anchor>
                  | 'e' <transition> <ang> <rad> <transition> <ang> <rad> 1 <anchor>
netdesc         ::= 'h' <net> {<nodesize> {<bgcolor>}}
eft             ::= {-}INT
lft             ::= {-}INT | 'w'
weight, marking ::= INT{'K'|'M'}
arckind         ::= '?'                              // test (read) arc
                  | '?-'                             // inhibitor
                  | '!'                              // stopwatch
                  | '?-'                             // stopwatch-inhibitor
xpos, ypos, rad ::= FLOAT
ang             ::= UFLOAT
net, place, transition, label   ::= ANAME | '{'QNAME'}'
anchor          ::= 'n' | 'nw' | 'w' | 'sw' | 's' | 'se' | 'e' | 'ne' | 'c'
FLOAT           ::= unsigned float (without exponent)
UFLOAT          ::= unsigned float between 0 and 1 (without exponent)
INT             ::= unsigned integer
ANAME           ::= see notes below
QNAME           ::= see notes below
<nodesize>      ::= 'small' | 'normal' | 'large'
<bgcolor>       ::= any tcl-tk color
```

Figure 3-17 : Structure générale du format .ndr.

La figure ci-dessous montre le fichier généré d'un exemple de réseau de Petri temporel saisi d'une façon graphique.

```
p 30.0 325.0 p2 0 n
t 30.0 410.0 t2 2 2 n
t 30.0 205.0 t0 1 2 n
p 30.0 50.0 p0 1 n
t 290.0 415.0 t3 1 1 n
p 290.0 285.0 p3 0 n
t 290.0 205.0 t1 2 3 n
p 290.0 55.0 p1 1 n
e p0 t0 1 n
e t0 p2 1 n
e p2 t2 1 n
e p2 t3 1 n
e p1 t1 1 n
e t1 p3 1 n
e p3 t3 1 n
e t3 0.2999484159 119.8540779 p0 0.9697082721 111.0045044 1 n
e t3 0.07906989483 94.41398202 p1 0.9282420103 139.9892853 1 n
h Fig1
```

Figure 3-18 : Représentation .ndr du TPN de la figure 3-14.

3.3.3 Romeo

C'est un outil permettant la modélisation et la vérification des différents types de réseaux de Petri à savoir les réseaux de Petri temporels (TPN), les réseaux de Petri à chronomètres (Stopwatch Petri Nets ou SwPN) ainsi que les réseaux TPN et SwPN paramétrés.

Il a été développé à l'Institut de Recherche en Communication et Cybernétique de Nantes au sein de l'équipe « Systèmes Temps-Réel » (http://romeo.rts-software.org/).

L'outil se présente sous forme graphique développée en tcl/tk (tcl ("Tcl," n.d.) : tool command language est un langage de script et tk est une bibliothèque permettant de créer et gérer des interfaces graphiques portables d'une façon simple). Il est composé de deux modules de calculs qui sont Gpn et Mercutio développés en C++.

3.3.3.1 Fonctionnalités

Parmi les fonctionnalités offertes Par Romeo, on cite :

-La simulation des réseaux. La figure 3-19 montre l'interface graphique « TPN simulator » de l'outil Romeo. Elle montre aussi l'interface graphique qui permet de modéliser les réseaux de Petri.

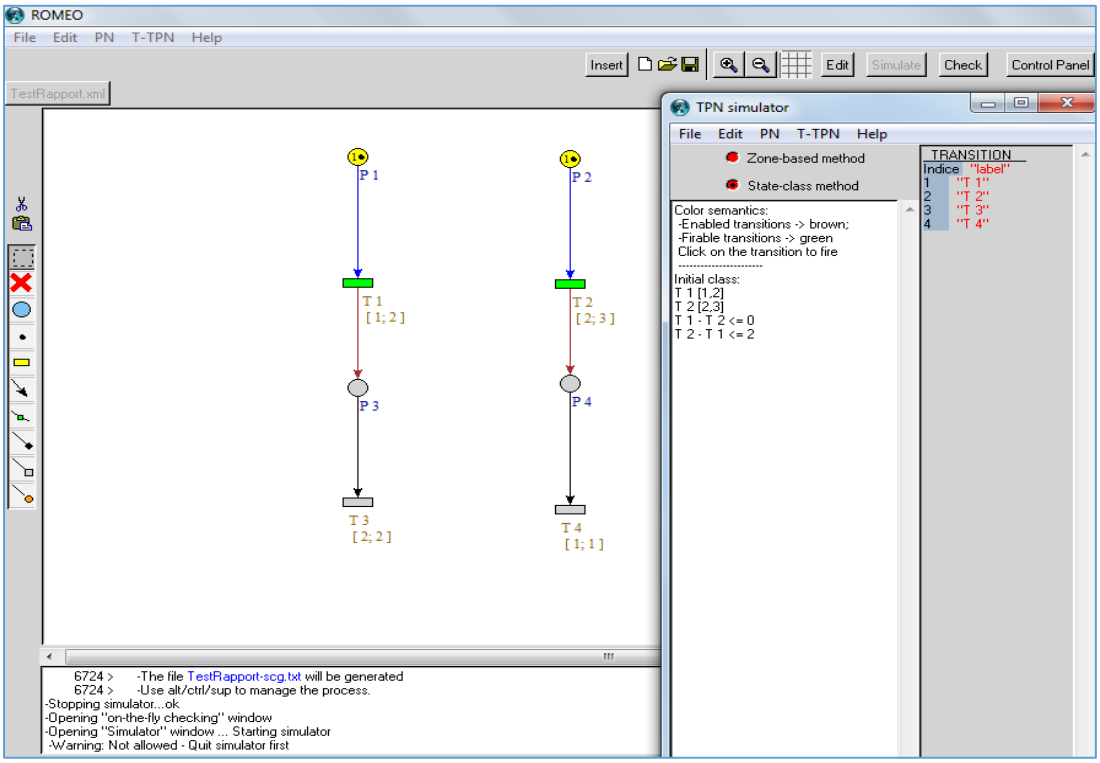

Figure 3-19 : L'éditeur et le simulateur de ROMEO.

-La génération de l'espace d'états d'un réseau de Petri en plusieurs formats possibles à savoir le graphe des classes d'états, l'automate des classes d'états et le graphe des zones.

-La traduction d'un réseau de Petri temporel (TPN) en un ensemble d'automates temporisés.

-La vérification des propriétés en appliquant le modèle Cheking.

La figure 3-20 montre l'interface qui permet de vérifier les propriétés sur les réseaux.

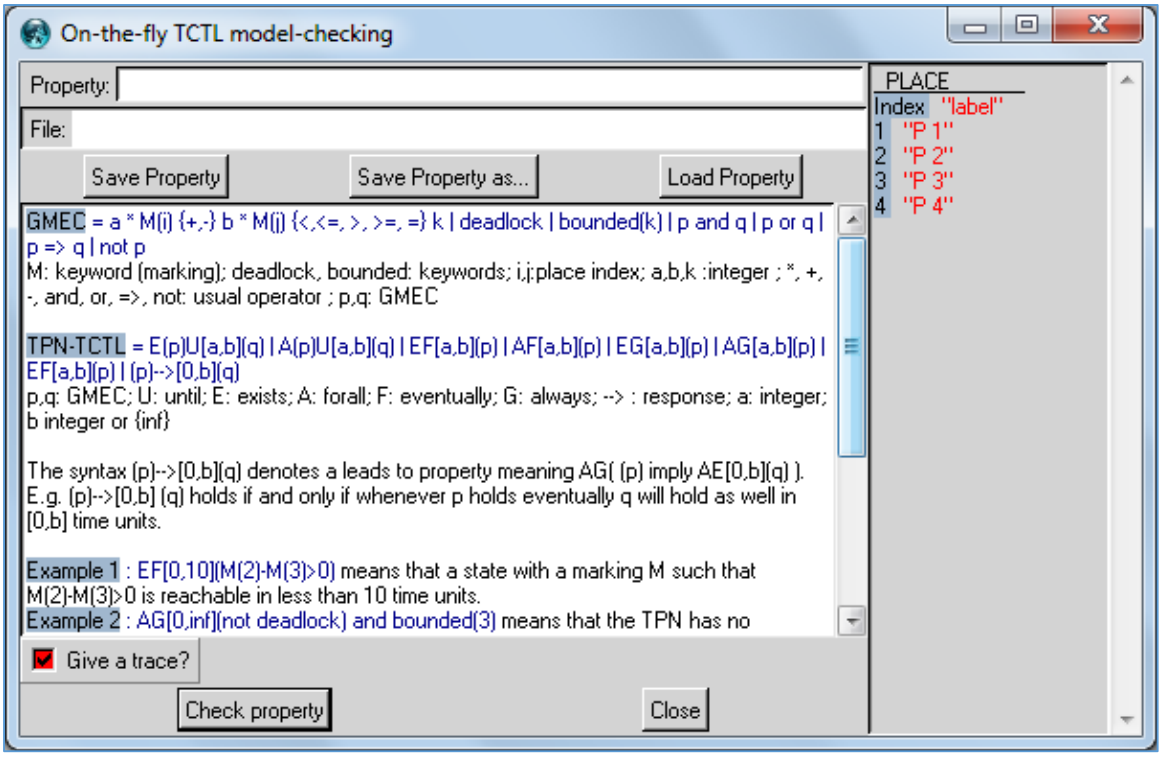

Figure 3-20 : Vérificateur de propriétés de ROMEO.

3.3.3.2 Architecture

ROMEO propose un éditeur graphique pour modéliser les différents types de réseaux de Petri qui sont à vérifier. Il permet aussi, en entrée, de lire la structure du réseau sous format XML ou même sous le format .ndr définie dans l'outil TINA.

La figure 3-21 montre un exemple de la structure d'un réseau TPN sous format XML.

```
<?xml version="1.0" encoding="UTF-8"?>
- <TPN name="C:/Users/Karim/Desktop/TestRapport.xml">
  - <place lft="inf" eft="0" initialMarking="1" label="P 1" id="1">
    - <graphics color="0">
        <position y="102.999996" x="259.0"/>
        <deltaLabel deltay="10" deltax="10"/>
      </graphics>
      <scheduling omega="0" gamma="0"/>
    </place>
  - <transition lft="1" eft="1" label="T 4" id="4" obs="1" lft_param="1"
    eft_param="1">
    - <graphics color="3">
        <position y="435.999996" x="432.0"/>
        <deltaLabel deltay="10" deltax="10"/>
      </graphics>
    </transition>
  - <arc weight="1" type="PlaceTransition" transition="2" place="2">
      <nail ynail="0" xnail="0"/>
      <graphics color="0"> </graphics>
    </arc>
```

Figure 3-21 : Structure d'un réseau TPN sous format XML.

Pour l'analyse des réseaux de Petri, ROMEO permet de réaliser deux types d'analyses (Guillaume Gardey, 2005) :

-L'analyse in-line avec le calcul de graphe d'accessibilité (de marquages) et la fonctionnalité de simulation.

-L'analyse off-line avec l'abstraction de l'espace d'états et la traduction de la structure d'un TPN en plusieurs automates temporisés.

ROMEO offre deux abstractions à savoir le SCG (State Class Graph) et le ZBG (Zone Based Graph) qui est une méthode d'abstraction basée sur les horloges.

En ce qui concerne la vérification des propriétés, ROMEO propose des modèles Cheking pour vérifier les réseaux de Petri temporels bornés et SwPN simples et paramétrés. La figure 3-22 décrit l'architecture de ROMEO.

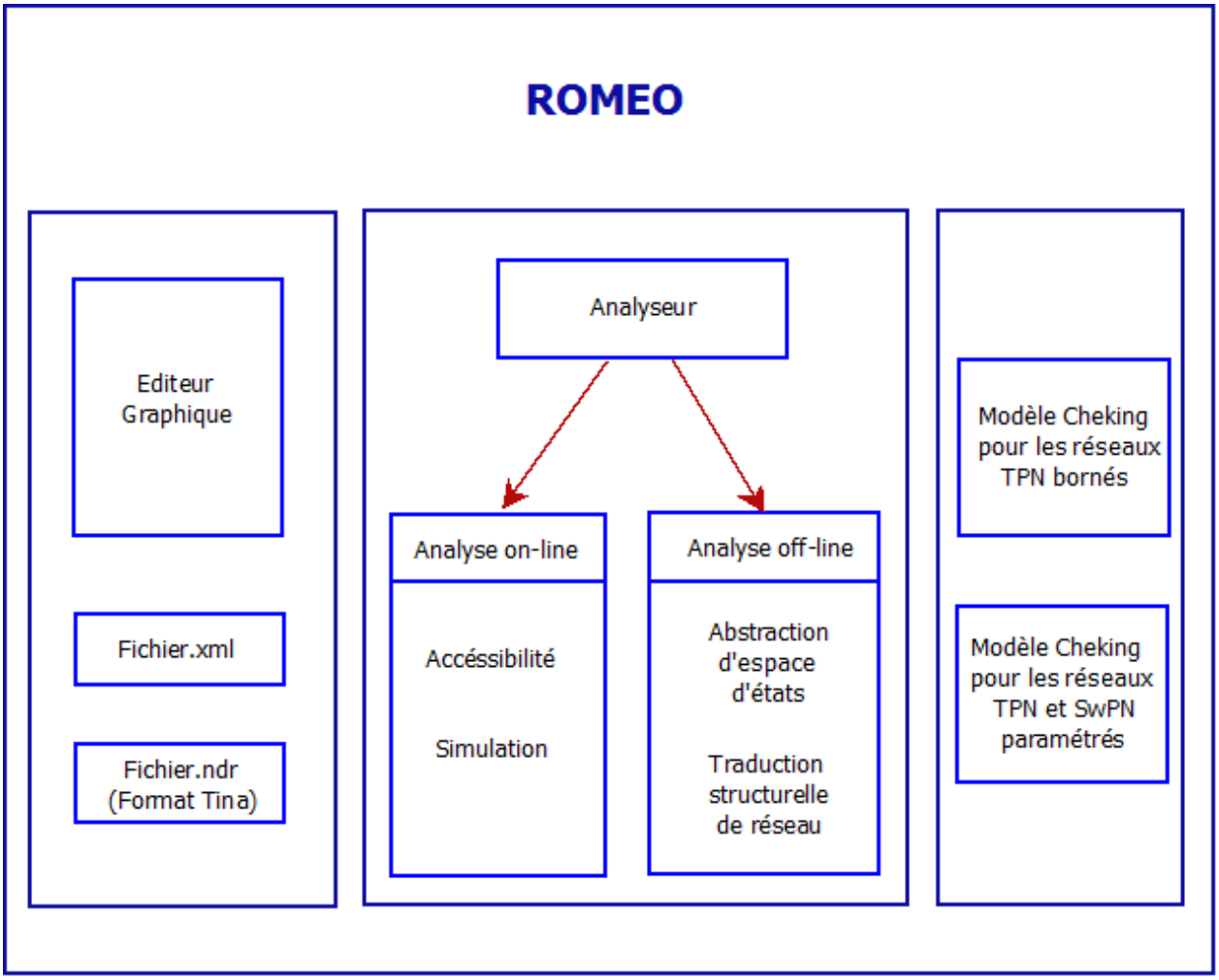

Figure 3-22 : Architecture de ROMEO.

Dans ce chapitre, nous avons présenté des techniques d'abstractions pour les réseaux de Petri temporel. Nous avons vu que la méthode CSCG donne de meilleurs résultats par rapport à la méthode SCG.

D'autre part, on a vu certaines techniques de réduction d'ordre partiel, nous nous sommes concentrés sur la méthode « Stubborn Set » vu que dans la littérature elle a donné de meilleurs résultats par rapport aux autres techniques.

Nous avons vu aussi une extension de la méthode « Stubborn Set » pour les réseaux de Petri temporel avec la méthode basée sur le calcul de G(α). Toutefois, cette méthode ne tient pas compte de la structure du TPN.

Enfin, nous avons présenté certains outils qui sont les plus utilisés dans la littérature pour construire le graphe d'états que ce soit pour les réseaux de Petri simples ou temporels. Nous avons détaillé leur fonctionnement, leur architecture pour tirer profit dans la phase de développement de notre outil. Sachant qu'aucun de ces outils n'applique les techniques de réduction d'ordre partiel dans le contexte de réseaux de Petri temporels, c'est ce que nous allons réaliser dans ce présent mémoire.

En effet, nous avons développé un outil qui permet de réaliser cette tâche en utilisant certaines fonctionnalités des outils présentés.

Dans le prochain chapitre, nous allons définir notre propre technique de réduction d'ordre partiel dans le contexte des TPN.

CHAPITRE 4 IMPLÉMENTATION DE G'

Durant le chapitre « revue de littérature », nous avons vu des méthodes d'abstraction de l'espace d'états ainsi que des techniques de réduction d'ordre partiel.

Dans ce présent chapitre, nous allons présenter notre propre technique. Elle est basée sur la méthode d'abstraction CSCG et inspirée de la technique de réduction d'ordre partiel Stubborn Set.

Notre technique va prendre en considération les intervalles de tirs statiques et dynamiques des transitions.

De plus, nous allons présenter l'algorithme qui nous permettra de calculer l'ensemble de transition G'(α) à tirer à partir de chaque classe d'états α.

4.1 Explication de G'(α)

La méthode d'abstraction utilisée est le CSCG. Comme nous l'avons vu dans le chapitre « revue de littérature », cette méthode d'abstraction permet d'avoir moins de classes d'états et d'arcs que la méthode SCG, de plus CSCG est adaptée pour les TPN où les contraintes sur les transitions sont exprimées par des intervalles.

Pour G'(α), l'idée est de sélectionner à partir de chaque classe d'états α un ensemble réduit de transitions. Ceci permet d'atténuer le problème d'explosion combinatoire. C'est une technique de réduction d'ordre partiel qui prend en considération la structure du TPN ainsi que de la classe d'états courante, c'est-à-dire le marquage et les intervalles de tirs des transitions sensibilisées.

Comme nous l'avons vu dans le chapitre précédent, les techniques de réduction d'ordre partiel se basent sur la notion de l'indépendance entre les transitions.

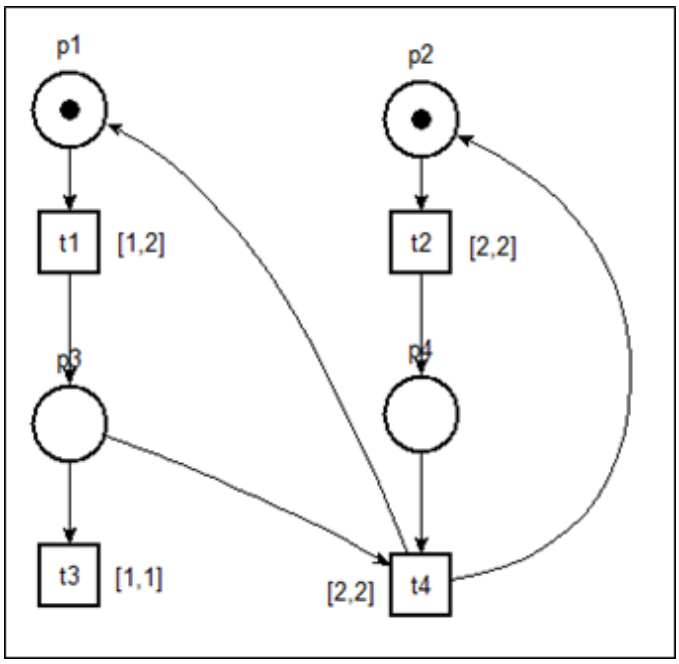

Figure 4-1 : Exemple de TPN.

Considérons le cas du TPN de la figure 4-1. En appliquant les règles de calculs pour obtenir le graphe de classes d'états avec l'algorithme CSCG (vu dans le chapitre « revue de littérature ») nous obtenons les deux classes d'états $\alpha 3 = (P3 + P4, -2 \leq t3 - t4 \leq -1)$ et $\alpha 4 = (P3 + P4, t3 - t4 = -1)$ en franchissant respectivement les séquences t1t2 et t2t1.

Les deux transitions t1 et t2 sont indépendantes et normalement les deux séquences t1t2 et t2t1 devraient nous mener vers la même classe d'états. Ceci est vrai dans le cas des réseaux de Petri simples qui ne tiennent pas compte de la notion du temps, mais dans le contexte des TPN on obtient deux classes d'états différentes. Bien que les marquages soient les mêmes dans α3 et α4, leurs contraintes temporelles sont différentes.

Pour pallier ce problème, nous allons utiliser les réductions d'ordre partiel basées sur la notion de Posets. Ceci permet de relaxer les contraintes de tirs des transitions et forcer la relation d'indépendance en fixant partiellement l'ordre de tir des transitions.

Soit $A = (Ac, succ, \alpha 0)$ un espace d'états généré par la méthode CSCG d'un TPN où α0 représente la classe d'états initiale, $succ$ la fonction qui permet de calculer le successeur d'une classe d'états α et Ac l'ensemble des classes d'états accessible à partir de α0 en appliquant la fonction $succ$.

Nous définissions l'espace d'états réduit comme étant $R = (G', C_{G'}, succ_{G'}, \alpha 0)$ avec α0 la classe d'états initiale, $C_{G'}$ l'ensemble des classes d'états accessibles dans R, G' la fonction qui permet de générer un ensemble de transitions et $succ_{G'}$ la fonction définie par :

$\forall \alpha = (M, F) \in Ac, \forall tf \in T, succ_{G'}(\alpha, tf) \neq \emptyset \; ssi \; succ(\alpha, tf) \neq \emptyset.$

Si $succ_{G'}(\alpha, tf) \neq \emptyset$ alors la classe d'états $\alpha' = succ_{G'}(\alpha, tf)$ est calculée en appliquant le même algorithme pour CSCG avec la condition de tir suivante :

$F \wedge \bigwedge_{ti \in En(M) \cap G'(\alpha)} tf \leq ti.$

4.2 Calcul de G'(α)

En franchissant une transition t_j, les transitions en conflit avec t_j dans le marquage M seront désensibilisées et d'autres transitions pourront être nouvellement sensibilisées.

Soit t_i une transition nouvellement sensibilisée suite au franchissement de t_j, vu qu'elle est nouvellement sensibilisée son intervalle de tir est initialisé à son intervalle statique de tir, c'est-à-dire $I(ti) = [\downarrow Is(ti), \uparrow Is(ti)]$ et du coup la borne inférieure du délai de tir de la transition t_i relativement au franchissement de tj n'est que $\downarrow Is(ti)$.

Nous définissons dans ce qui suit la matrice carrée L qui représente la borne inférieure des délais de tirs de l'ensemble T des transitions d'un TPN où $\forall ti, tj \in T$ nous avons :

$Lij = 0 \; si \; ti = tj.$

$Lij = \downarrow Is(ti) \; si \; ti \neq tj \wedge tj \in {}^\circ({}^\circ ti).$

$Lij = \infty \; sinon.$

Soit la matrice L' qui représente la forme canonique de la matrice L en appliquant l'algorithme du plus court chemin Floyd-Warshall.

La figure 4-2 contient les matrices L et L' relatives au TPN de la figure 4-1.

L	t1	t2	t3	t4
t1	0	∞	∞	1
t2	∞	0	∞	2
t3	1	∞	0	∞
t4	2	2	∞	0

L'	t1	t2	t3	t4
t1	0	3	∞	1
t2	4	0	∞	2
t3	1	4	0	2
t4	2	2	∞	0

Figure 4-2 : Matrices L et L' relatives au TPN de la figure 4-1.

L'ij représente la borne inférieure du délai de tir de la transition ti relativement au franchissement de tj, dans le cas où ti n'est pas sensibilisée quand tj est franchi.

Lorsque L'ij=infini ça veut dire qu'il n'y a pas de chemin direct entre tj et ti et donc ti ne peut pas être sensibilisée directement ou indirectement par tj.

Par exemple, la valeur 4 de L'21 est la borne inférieure de délai de tir de t2 relativement au franchissement de t1 dans le cas où t2 n'est pas sensibilisée au moment de franchissement de t1. En effet, ça correspond à la situation où t1 sensibilise t4 qui à son tour sensibilise t2. C'est-à-dire que $L'21 = L24 + L41 = \downarrow Is(t2) + \downarrow Is(t4)$.

Soit α = (M, F) une classe d'états, ti et tj deux transitions franchissables à partir de α. Soient Mi et Mj les marquages successeurs de M respectivement par ti et tj. Les deux transitions ti et tj sont Effet-Independent dans α, noté $ti \,||\alpha\, tj$ ssi leurs effets sont indépendants de leurs ordres de tirs.

Formellement : $CF(M, ti) = CF(Mj, ti) \wedge CF(M, tj) = CF(Mi, tj) \wedge$

$$Nw(M, ti) = Nw(Mj, ti) \wedge Nw(M, tj) = Nw(Mi, tj).$$

À noter que la relation Effet-Independent est une relation symétrique, c'est-à-dire $ti \,||\alpha\, tj$ ssi $tj \,||\alpha\, ti$.

Pour construire le G'(α), nous commençons par choisir une transition franchissable puis nous ajoutons au fur et à mesure les transitions qui pourront affecter directement ou indirectement les transitions qui sont l'intérieur de G'(α) jusqu'à l'obtention d'un point fixe.

G'(α) est le plus petit ensemble de transition de En(M) qui satisfait les conditions suivantes (Boucheneb, Barkaoui, & Weslati, 2014) :

$C0 : Fr(\alpha) \neq \emptyset \Leftrightarrow G'(\alpha) \cap Fr(\alpha) \neq \emptyset.$

$C1 : \forall ti \in G'(\alpha), CF(M, ti) \subseteq G'(\alpha).$

$C2 : \forall ti \in G'(\alpha), \forall tj \in En(M), \forall tk \in Adj(ti) - En(M), L'kj - \downarrow Is(tk) \leq Dij$

$\Rightarrow tj \in G'(\alpha).$

$C3 : \forall ti \in G'(\alpha), \forall tj \in Fr(\alpha), not(ti \,||_{\alpha}\, tj) \Rightarrow tj \in G'(\alpha).$

$C4 : \forall ti \in G'(\alpha), \forall tj \in En(M), ti \notin Fr(\alpha) \wedge tj \in Fr(\alpha) \Rightarrow tj \in G'(\alpha).$

La condition C0 assure que G'(α) est vide ssi nous avons un inter-blocage (deadlock).

C1 veut dire qu'il n'existe aucune transition à l'extérieur de G'(α), pour le marquage M, en conflit avec une transition de G'(α). D'où le franchissement d'une transition à l'extérieur de G'(α) ne pourra en aucun cas désensibiliser une transition à l'intérieur de G'(α).

C2 assure le faite que durant la sensibilisation de n'importe quelle transition ti appartenant à G'(α), il n'y aurait pas de transition tj à l'extérieur de G'(α) qui pourrait sensibiliser directement ou indirectement une transition qui soit adjacente à ti (à rappeler que deux transitions sont adjacentes si elles ont des places en commun que ce soit en entrée ou bien en sortie).

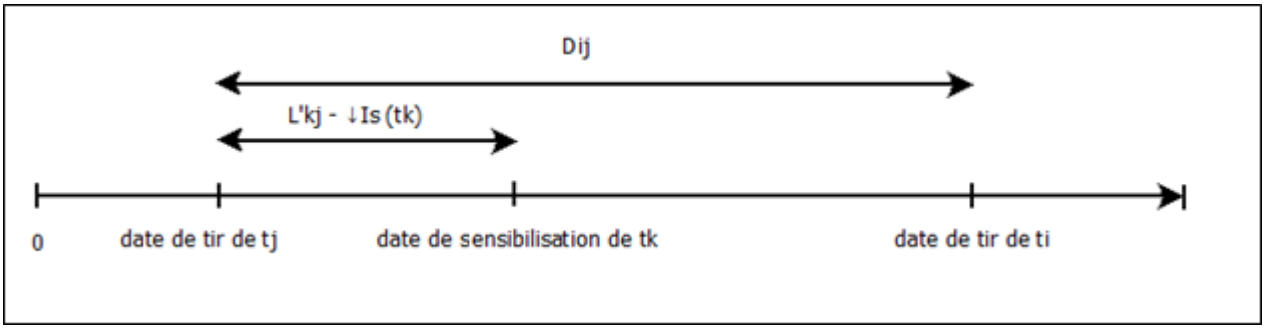

Figure 4-3 : Schématisation de la condition C2.

Si tj sensibilise la transition tk (qui est adjacente à ti), alors tj doit appartenir à G'(α). Cette sensibilisation peut se réaliser selon deux cas possibles : tk peut-être sensibilisée avant ou après le franchissement de ti.

La première variante de G'(α) est G'1(α). Dans cette variante, on considère le cas où tk est sensibilisée avant le franchissement de ti.

La deuxième variante est G'2(α) qui considère le cas où tk peut être sensibilisé avant ou après le franchissement de ti.

Cette version est plus relaxée que G'1(α). Effet, au lieu de vérifier $L'kj - \downarrow Is(tk) \leq Dij$, on s'assure que $L'kj - \downarrow Is(tk) \neq \infty$. Par conséquent G'2(α) est moins contractée que G'1(α).

Les conditions C2 et C3 impliquent que l'effet de ti ne pourra pas être affecté par le franchissement d'une transition tj se trouvant à l'extérieur de G'(α).

Finalement, la condition C4 indique qu'il n'existe pas une transition tj à l'extérieur de G'(α) qui doit être franchie avant quelques transitions appartenant à G'(α).

La figure 4-3 contient l'algorithme conçu pour l'obtention de l'ensemble de transition G'(α) à franchir à partir de la classe d'états α en se basant sur les conditions ci-dessus.

Entrée : une classe d'états α
Sortie : l'ensemble de transitions G'(α)
Choisir aléatoirement une transition tf de l'ensemble Fr(α)
X=tf
Y=∅
Tant que (X ≠ Y)
Y = X
Pour chaque ti dans X faire
    Pour chaque tj dans En(M) faire
        si ((CF(M,ti) U Nw(M,ti)) ∩ (CF(M,tj) U Nw(M,tj)) ≠ ∅)
        X = X U tj
        Fin si
        si (il existe tl dans (°ti U ti°)° - En(M) s.t L'lj - ↓Is(tl) ≤ Dij)
        X = X U tj
        Fin si
        si (il existe tk dans (°tj U tj°)° - En(M) s.t L'ki - ↓Is(tk) ≤ Dij)
        X = X U tj
        Fin si
        si (il existe tn dans X s.t (F ∧ tj<tn) ≡ F)
        X = X U tj
        Fin si
    Fin Pour
Fin Pour
Fin Tant que
G'(α) = X

Figure 4-4 : Algorithme de construction de G'(α).

Dans ce chapitre, nous avons présenté notre technique de réduction d'ordre partiel. Cette méthode est une extension de la technique Stubborn Set dans le contexte temporel, tout en prenant en considération la structure du TPN, c'est-à-dire les délais statiques des transitions constituant le TPN ainsi que de la classe d'états courante.

Nous avons aussi proposé un algorithme pour calculer les transitions à franchir de chaque classe d'états. Comme nous l'avons indiqué dans les objectifs de recherches au chapitre d'introduction, nous avons participé au développement de cette technique. Par conséquent, il y en a des parties que nous n'avons pas réalisé comme les preuves qui montrent la complétude de G'(α).

Dans le prochain chapitre, nous présentons l'outil développé pour exécuter notre algorithme. Ceci nous permet de le tester sur différents types de TPN ainsi d'évaluer sa performance en le comparant avec d'autres techniques de réduction d'ordre partiel ou les méthodes d'abstraction déjà vues dans le chapitre « revue de littérature ».

CHAPITRE 5 IMPLÉMENTATION ET RÉSULTATS EXPÉRIMENTAUX

Après la présentation de l'algorithme G'(α) dans le chapitre précèdent, nous allons nous concentrer sur l'implémentation de cet algorithme avec ses deux variantes ainsi que l'algorithme G(α) déjà vu au chapitre « revue de littérature ».

Cette implémentation est réalisée à l'aide d'un outil que nous avons développé, appelé « INKA ». Nous allons détailler son fonctionnement, son architecture ainsi que les fonctionnalités qu'il offre.

Dans un second temps, nous présentons et nous analysons les résultats obtenus suite à l'exécution des différents TPN avec notre outil. Il s'agit des TPN variés, certains sont simples et d'autres sont complexes. Ces derniers sont des TPN de références qui sont fortement connectés et qui sont proposés dans la conférence annuelle Petri nets pour comparer les performances des outils de vérification.

5.1 Présentation

Nous avons choisi le langage C pour implémenter notre outil. Notre outil est portable sur différentes plates-formes. En effet, nous l'avons testé avec divers systèmes d'exploitation à savoir Windows, Ubuntu (Linux) et Mac OS, et avec différents compilateurs comme DevC++ et Visual Studio2013.

L'exécution de l'outil ne nécessite pas d'ordinateurs sophistiqués, un ordinateur ordinaire est suffisant.

5.2 Architecture globale

L'outil « INKA » lit la structure d'un réseau de Petri temporel (TPN) et par la suite génère le graphe de classes d'états réduit suivant la technique choisie par l'utilisateur.

La figure suivante présente d'une façon globale l'architecture de notre outil « INKA ».

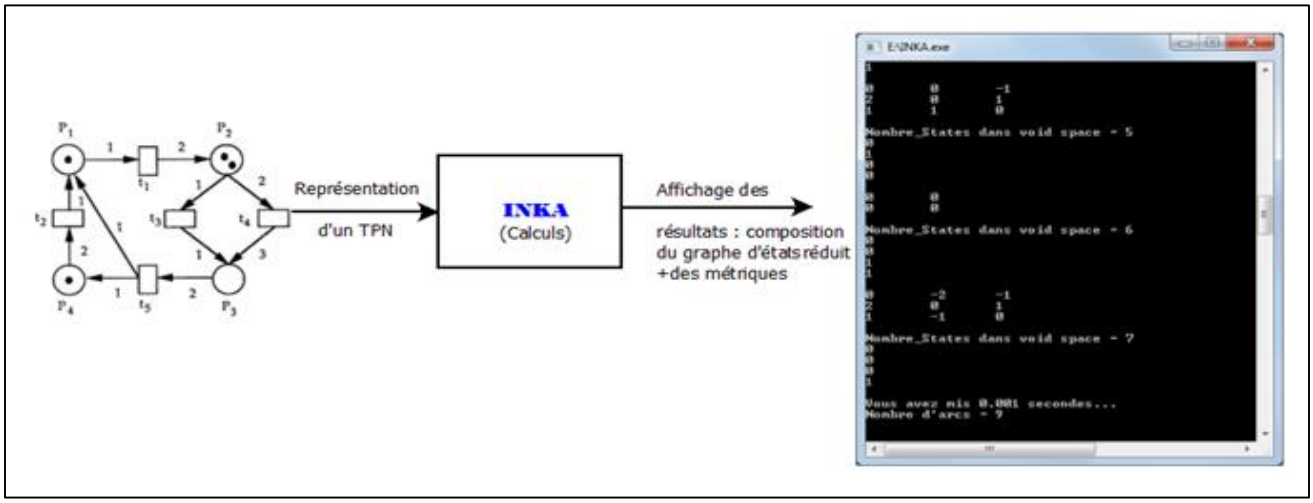

Figure 5-1 : Architecture globale de l'outil INKA.

5.3 Fonctionnement

Nous allons maintenant détailler notre outil afin de bien comprendre son fonctionnement. Dans la figure ci-dessous, nous montrons l'architecture détaillée.

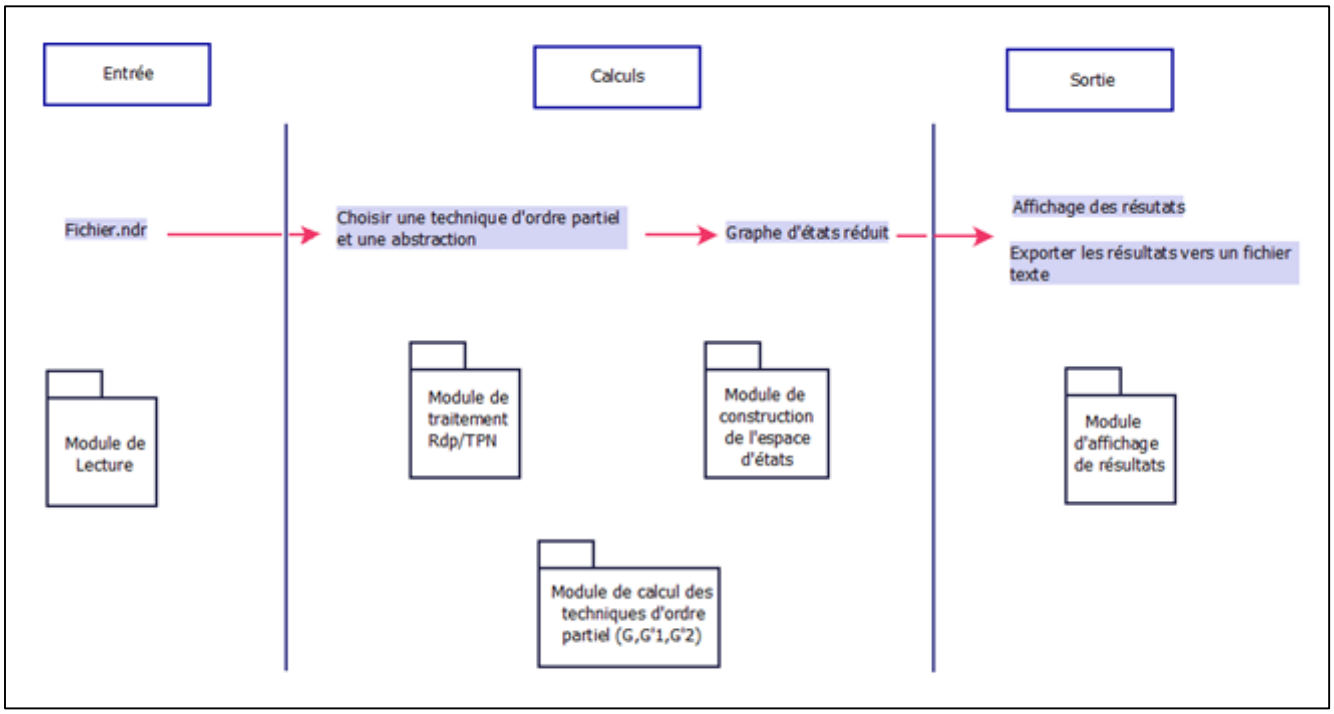

Figure 5-2 : Architecture détaillée de l'outil INKA.

Comme entrée nous avons opté pour le format « .ndr » (vu au chapitre de « revue de littérature » dans la partie où nous avons présenté l'outil TINA) pour représenter la structure du réseau de pétri temporel. Nous avons choisi ce format pour sa simplicité où on peut définir les places avec leurs jetons, les transitions avec leurs intervalles de temps ainsi que les arcs reliant une place à une transition et vice-versa en spécifiant leurs poids.

Pour modéliser un TPN, nous pouvons nous servir de la représentation graphique offerte aussi par la boîte à outils TINA qui permet de générer par la suite le fichier « .ndr ».

Une fois que le fichier d'entrée est spécifié par l'utilisateur, notre outil appelle les méthodes adéquates contenues dans le module de lecture afin d'initialiser les structures de données avec les informations du TPN décrites dans le fichier « .ndr ».

On construit alors la liste des places, des transitions et la classe d'états initiale $\alpha 0 = (M0, D0)$ avec M0 le marquage initial et D0 la matrice qui représente la forme canonique de la matrice de borne initiale (DBM) des domaines de tirs des transitions sensibilisées à la classe $\alpha 0$.

Pour les structures de données, nous avons eu recours à la structure de graphe qui est composée de plusieurs classes d'états. Chaque classe d'états est constituée d'un marquage M et d'une matrice D.

Un marquage est constitué de l'ensemble de places présentes dans le TPN, où chaque place est caractérisée par un numéro, un nombre de jetons qu'elle contient ainsi que deux listes : l'une pour spécifier les transitions en entrées et l'autre pour les transitions en sorties. Pour la matrice D, sa dimension est égale au nombre de transitions sensibilisées plus 1 pour tenir compte de la ligne et la colonne 0.

Tandis que les transitions sont définies par un numéro, la borne minimale et maximale de l'intervalle de tir. Elles contiennent aussi 2 listes pour spécifier les places en entrée et en sortie.

La figure 5-3 montre les différentes structures de données utilisées dans notre outil.

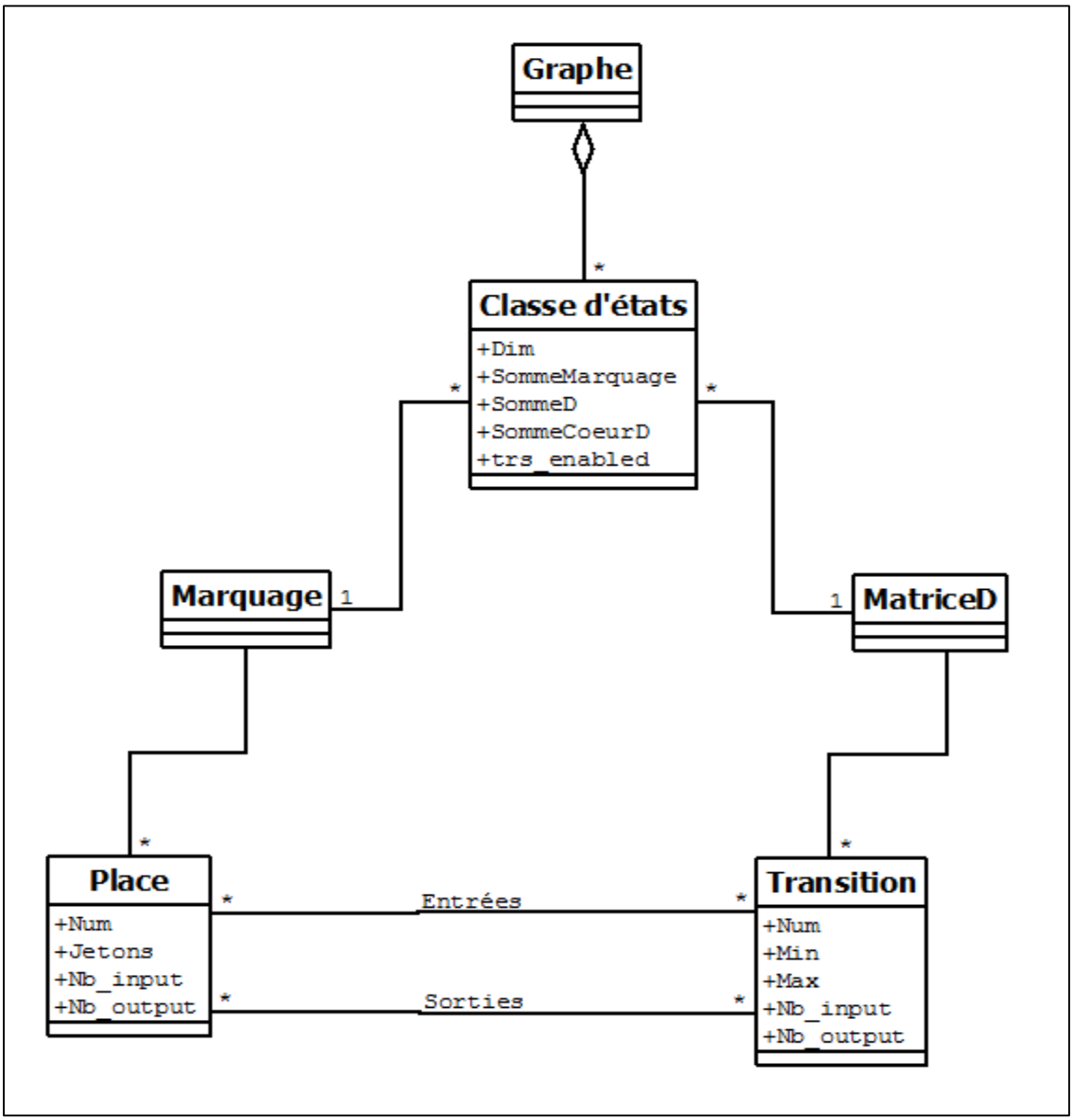

Figure 5-3 : Structures de données utilisées dans l'outil INKA.

Une fois que l'outil a construit les structures de données, nous demandons à l'utilisateur de spécifier une technique de réduction d'espace d'états. On distingue :

1-G(α) : une extension de la technique Stubborn Set pour les TPN.

2- G'1(α) : une extension de la technique Stubborn Set pour les TPN en tenant compte de la structure du TPN et de l'état courant.

3- G'2(α) : G'1(α) relaxée.

Pour chaque technique de réduction d'ordre partiel, on pourrait utiliser des abstractions tirées de la revue de littérature à savoir le SCG et le CSCG. Comme nous avons vu dans le chapitre 2, la seule différence entre ces deux méthodes d'abstraction est qu'une classe d'états définie par $\alpha = (M, D)$ peut être ajouté au graphe réduit suite à une comparaison de marquage et de la matrice D des autres classes du graphe : pour le SCG on compare la totalité de la matrice D, pour le CSCG on ignore la ligne et la colonne 0 de la matrice D. Évidemment avec l'abstraction CSCG on obtiendra moins de classes d'états dans le graphe final.

L'espace d'états est donc calculé à l'aide des méthodes d'abstraction (SCG/CSCG) plus l'utilisation d'une technique de réduction d'ordre partiel. En effet, les techniques de réduction d'ordre partiel permettent de choisir un nombre limité de transitions à franchir à partir d'une classe d'états, tandis que les abstractions permettent d'avoir une représentation finie de l'espace d'états.

La figure 5-4 résume la méthode d'obtention d'un graphe de classes d'états réduit.

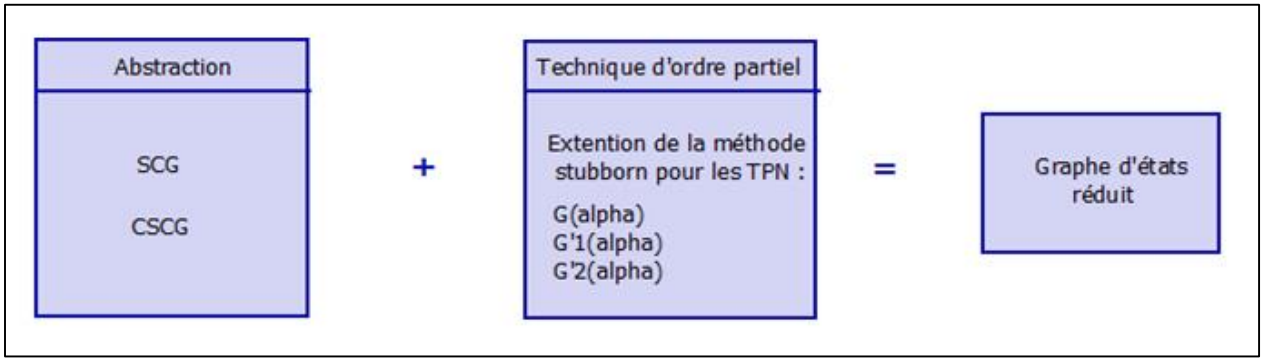

Figure 5-4 : Comment obtenir un graphe de classes d'états réduit.

Finalement, le module d'affichage de résultats permet de lister toutes les classes d'états du graphe réduit, en spécifiant pour chaque classe son marquage ainsi que sa matrice de bornes D. De plus, l'outil permet d'afficher certaines métriques comme le nombre total de classes d'états retenus dans le graphe réduit, le nombre d'arcs qui représente le nombre total de classes calculées et aussi le temps total d'exécution en secondes pour générer le graphe de classes d'états.

On pourrait exporter le graphe ainsi que les différentes métriques vers un fichier texte.

5.4 Méthodologie de développement

Durant le développement de notre outil, nous avons opté pour une approche itérative et incrémentale.

Chaque itération est composée des phases suivantes : spécification et analyse des besoins, conception, développement et test.

Une itération correspond à une incrémentation dans le sens où à la fin de chaque itération nous obtenons une portion de l'outil qui est exécutable d'où l'enrichissement des fonctionnalités déjà existantes.

Au début de chaque itération, nous spécifions et nous analysons les nouvelles fonctionnalités à développer et durant le développement nous planifions des révisions afin de nous assurer que nous sommes en train de développer le bon outil et que toutes les contraintes sont tout à fait respectées.

Durant la phase de test, nous essayons de nous assurer du bon fonctionnement des nouvelles portions ajoutées, de plus nous essayons de réaliser des tests d'intégration ainsi que des tests de régression pour voir l'impact des nouvelles fonctionnalités sur les anciennes.

Dans cette approche itérative et incrémentale, nous nous sommes inspirés des méthodes agiles.

Concernant les itérations, nous avons eu recours à cinq itérations, chacune s'est déroulée en moyenne entre un mois et un mois et demi.

La première itération était conçue pour définir les structures de données, lire le fichier contenant la spécification du TPN (places, transitions et arcs) sous format texte ou Tina et construire toutes les structures afin de réaliser les opérations demandées.

Dans la deuxième itération, nous avons développé les opérations élémentaires pour les réseaux de Petri simple. Nous citons comme exemple les opérations relatives à la détection des transitions sensibilisées, au calcul du nouveau marquage suite au franchissement d'une transition, à l'affichage des classes d'états, etc.

La troisième itération était conçue pour l'implémentation des algorithmes d'abstraction SCG et CSCG. Les importantes opérations développées durant cette itération sont : le successeur d'une classe d'états par une transition, la fonction qui permet de vérifier si une classe existe déjà dans le graphe ou non et quelques opérations relatives aux TPN comme les fonctions qui permettent de

déterminer l'ensemble de transitions franchissables dans une classe d'états, les transitions nouvellement sensibilisées ou encore les transitions qui sont en conflits à une transition définie.

Dans la quatrième itération, nous nous sommes concentrés sur la réalisation de la technique basée sur le calcul de G(α).

Et finalement, la dernière itération était réalisée pour l'implémentation de la technique basée sur le calcul de l'ensemble de transition G'(α) avec ses deux variantes.

Le schéma 5.5 résume les itérations utilisées pour le développement de l'outil INKA.

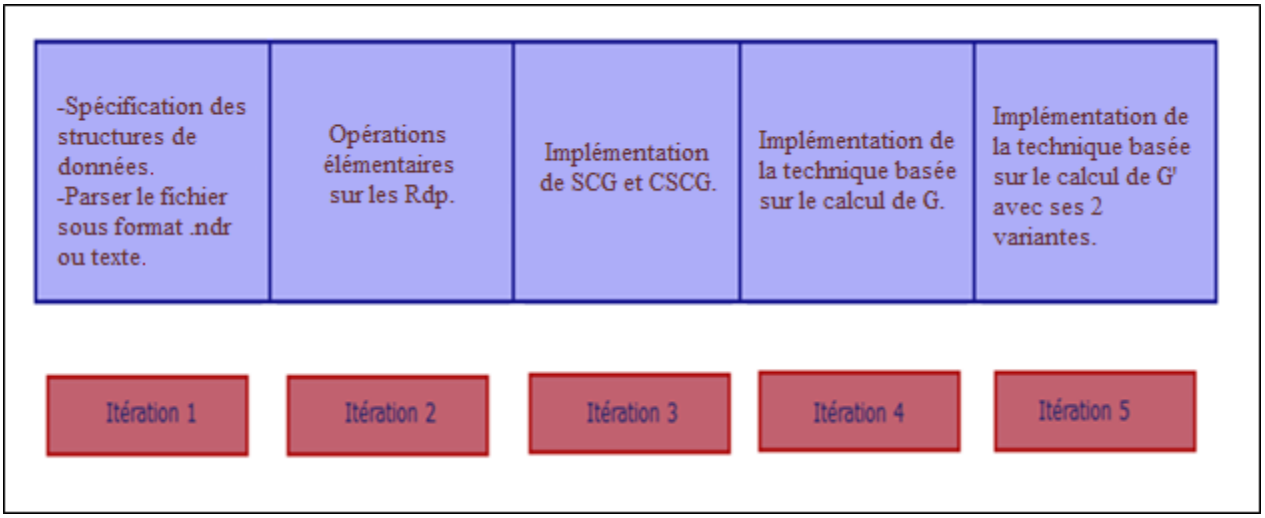

Figure 5-5 : Ensemble des itérations réalisées pour l'implémentation de l'outil INKA.

5.5 Optimisation

Dans le développement de notre outil, nous avons eu recours à l'utilisation des techniques permettant la réduction du temps des calculs, la minimisation des ressources utilisées et donc l'amélioration des performances de notre outil.

Parmi les techniques utilisées, on trouve celle qui permet de calculer la matrice D' du successeur de la classe d'états α (avec α= (M, D) et α'= (M', D')). En effet, pour le calcul de D', ça nécessite un algorithme de calcul de complexité ≅ o(n^3). Avec la technique utilisée (Boucheneb & Mullins, 2003) on calcule D' avec une complexité ≅ o(n^2). Il ne s'agit pas d'une contribution réalisée au cours de nos travaux de recherches mais un choix fait afin d'avoir des algorithmes de complexités réduites.

La figure ci-dessous montre l'algorithme utilisé pour performer les calculs.

```
Classe fonction successeur (Classe(M,D), Transition tf)

{
    Classe(M',D')
    M'=M - Pre(tf) + Post (tf)

    D'[0,0] = 0
    Pour chaque transition t sensibilisée pour M'  (i.e. Pre(t) ≤ M' )
    {
        D'[t,t] = 0

        Si t est nouvellement sensibilisée (i.e. ¬(Pre(t) ≤ M - Pre(tf)))
        alors {
                D'[t,0] = - tmin(t)
                D'[0,t] =   tmax(t)
              }

        sinon {
                D'[0,t] = D [tf,t]
                D'[t,0] = 0

                pour chaque transition t' sensibilisée pour M (i.e. Pre(t') ≤ M)
                                D'[t,0] = Min ( D'[t,0] , D [t,t'] )

              }

    }

    Pour chaque transition t sensibilisée pour M' (i.e. Pre(t) ≤ M' )
    {
        Pour chaque transition t' sensibilisée pour M' telle que t ≠ t'
        si t ou t' est nouvellement sensibilisée
        alors
              D'[t,t'] = D'[t,0] + D'[0,t']
        sinon
              D'[t,t'] = Min (D [t,t'] , D'[t,0] + D'[0,t'] =)
    }

Retourner (M',D')

} //Fin de la fonction
```

Figure 5-6 : Algorithme utilisé pour le calcul du successeur d'une classe d'états.

Une autre méthode utilisée pour réduire les temps de calculs est celle qui permet d'effectuer la comparaison entre deux classes d'états. En effet, deux classes sont équivalentes si elles ont le même marquage et la même matrice D (pour l'abstraction SCG, on compare la totalité de la matrice D et

pour l'abstraction CSCG on compare seulement le cœur de la matrice D c'est-à-dire sans tenir compte de la ligne et la colonne 0). Pour réaliser la comparaison, il faut comparer élément par élément : avec des milliers de classes d'états, ça va être couteux en termes du temps des calculs. Pour cela, nous avons opté pour la définition de quelques mesures utiles afin de différencier deux classes d'états. Ces mesures sont la somme du marquage, la somme de la matrice D et la somme du cœur de la matrice D qui sont calculées lors de la création des classes d'états. Au lieu, de comparer le marquage et la matrice D de deux classes d'états élément par élément, nous comparons les mesures de ces deux classes : si les mesures sont différentes donc il ne s'agit pas de deux classes équivalentes et si les mesures sont égales, c'est en ce moment-là seulement que nous comparons les deux matrices D élément par élément. Ceci nous a permis de réduire d'une façon importante le nombre de comparaisons de deux classes d'états (élément par élément) lors de l'ajout d'une classe au graphe de classes d'états.

De plus, nous avons essayé d'optimiser le code en réalisant une bonne décomposition des modules, des méthodes et des structures de données adéquates pour favoriser la réutilisation du code.

5.6 Exemple d'illustration

Nous allons considérer le réseau de Petri temporel de la figure 5-7 pour calculer le graphe de classes d'états avec une technique de réduction d'ordre partiel. Au début, nous commençons par modéliser le TPN. Comme nous avons mentionné avant; le format utilisé est le format .ndr généré par l'éditeur graphique de la boîte à outils TINA.

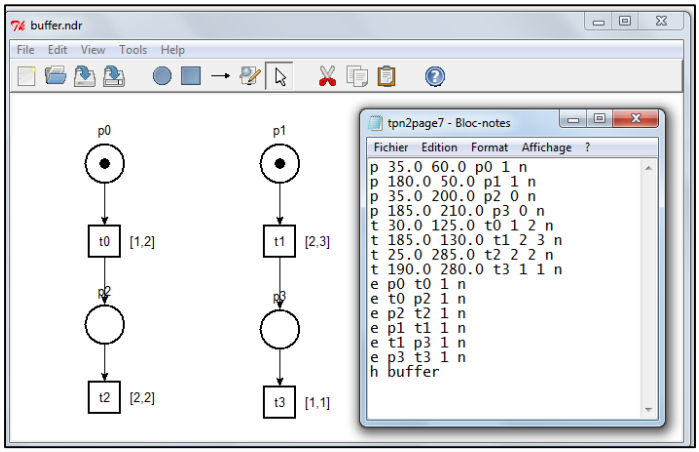

Figure 5-7 : Exemple d'un TPN.

Ensuite, nous choisissons une technique de réduction parmi les trois techniques offertes par notre outil.

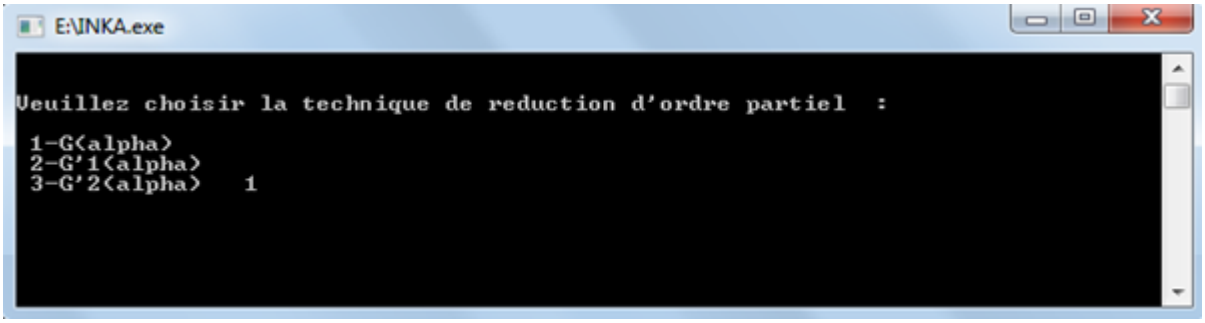

Figure 5-8 : Interface du choix de la technique de réduction d'ordre partiel.

Puis, nous sélectionnons la méthode d'abstraction à utiliser.

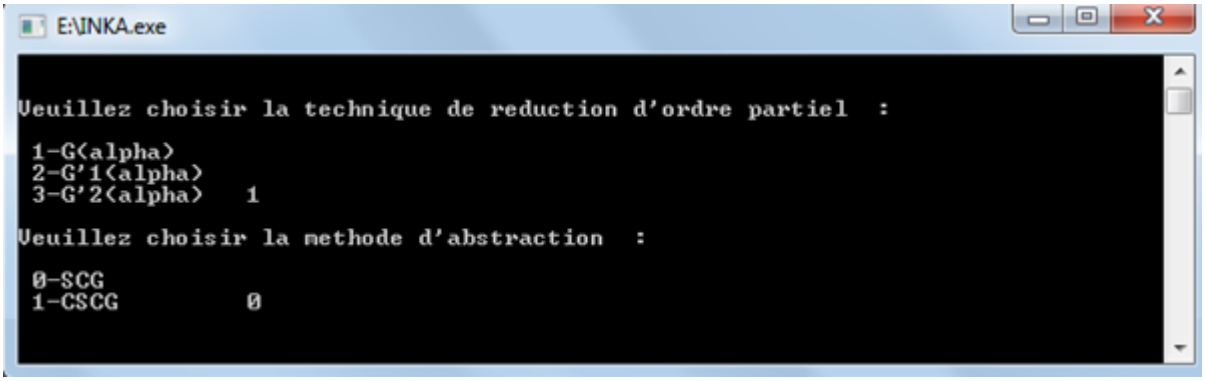

Figure 5-9 : Interface du choix de la méthode d'abstraction.

Enfin, nous visualisons les classes d'états contenus dans le graphe de classes d'états ainsi que les métriques suivantes : le temps de calcul, le nombre de classes d'états et le nombre d'arcs.

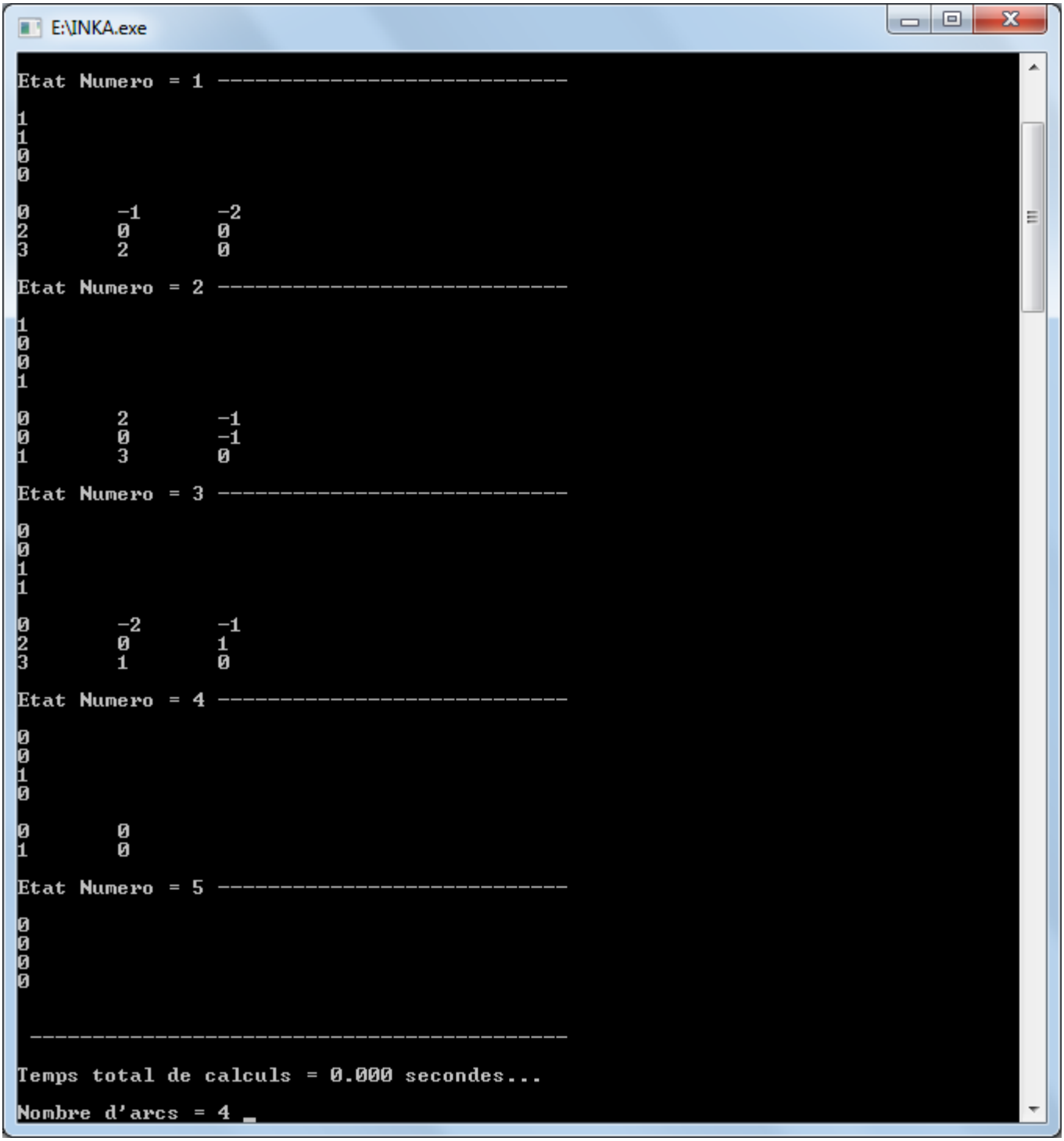

Figure 5-10 : Interface de l'affichage des classes d'états d'un graphe avec différentes métriques.

5.7 Les TPN étudiés

Pour vérifier les différentes techniques de réduction d'ordre partiel et voir le gain tiré en les appliquant, nous avons choisi quatre exemples de réseaux de Petri temporels : deux TPN simples, deux TPN qui ont été proposés dans l'événement annuel « Model Checking Contest » et un TPN de référence modélisant le processus d'attribution de pensions aux personnes à mobilité réduite.

Le tableau 5.1 contient les informations relatives à chaque TPN en termes de places, de transitions ainsi que d'arcs reliant les places aux transitions et vice-versa.

Tableau 5-1: Informations sur les TPN étudiés.

	Nombre de places	Nombre de transitions	Nombre d'arcs
TPN 1	4	4	9
Duplication	8	8	12
FMS	22	20	49
TPN de référence	36	22	64
HC	26	18	51

La figure suivante contient le premier TPN simple.

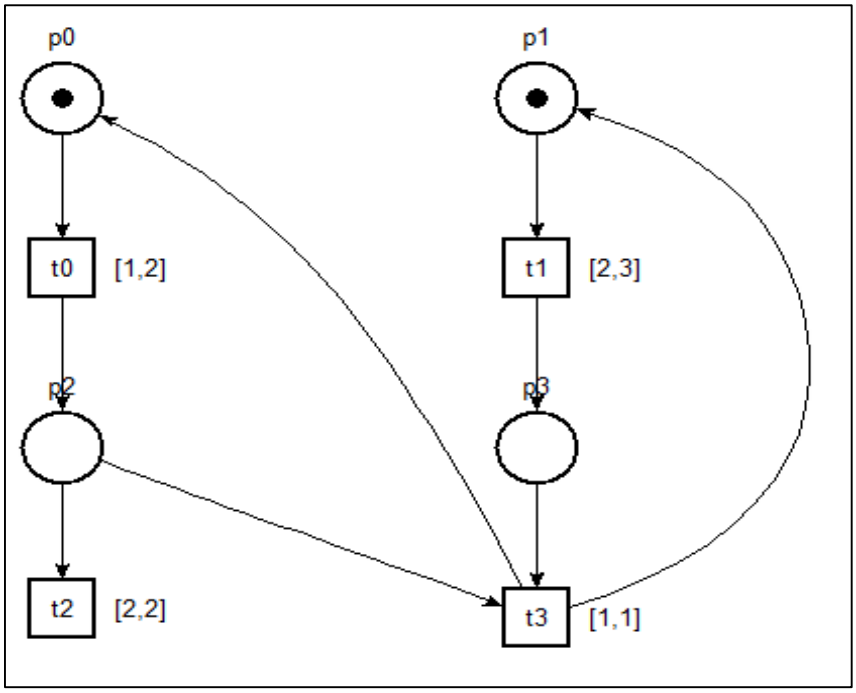

Figure 5-11 : TPN1.

La figure 5-12 représente le deuxième TPN. C'est un exemple simple, mais qui nous permet de tirer des conclusions importantes dans le cas où nous avons des branches indépendantes. Nous revenons à ces conclusions dans la section « Commentaires des résultats ».

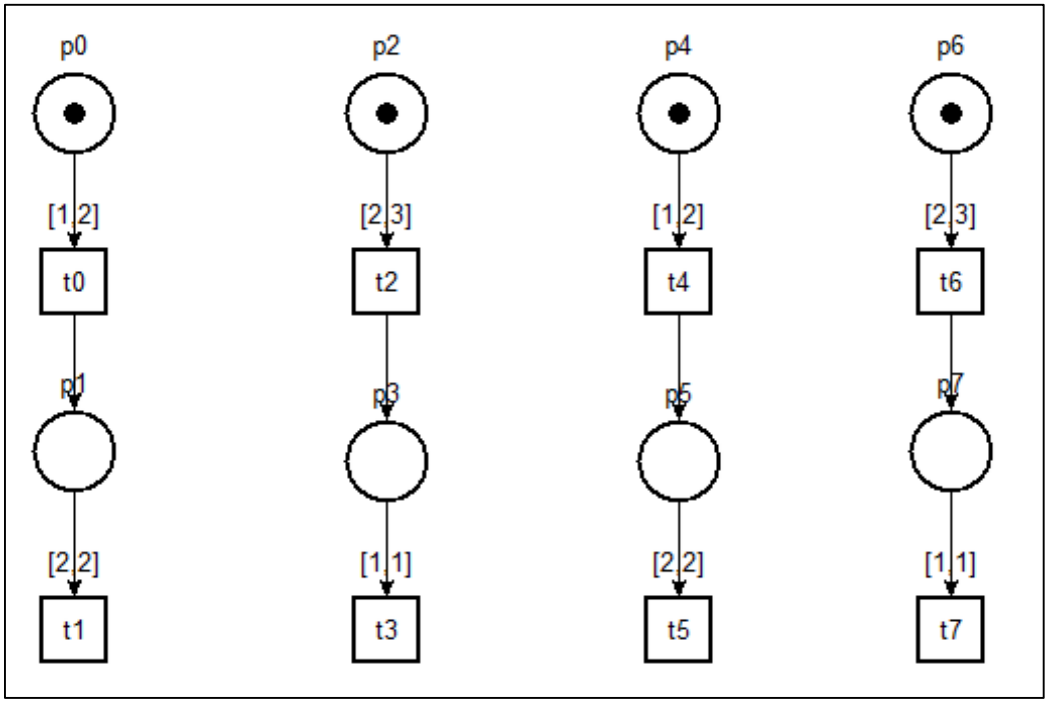

Figure 5-12 : TPN « Duplication ».

La figure 5-13 contient le TPN FMS (Flexible Manufacturing System). Il a été proposé par Lom-Messan Hillah (Hillah, 2011) pour l'édition 2011 du MCC (Model Checking Contest). C'est un TPN de référence et qui est fortement connecté.

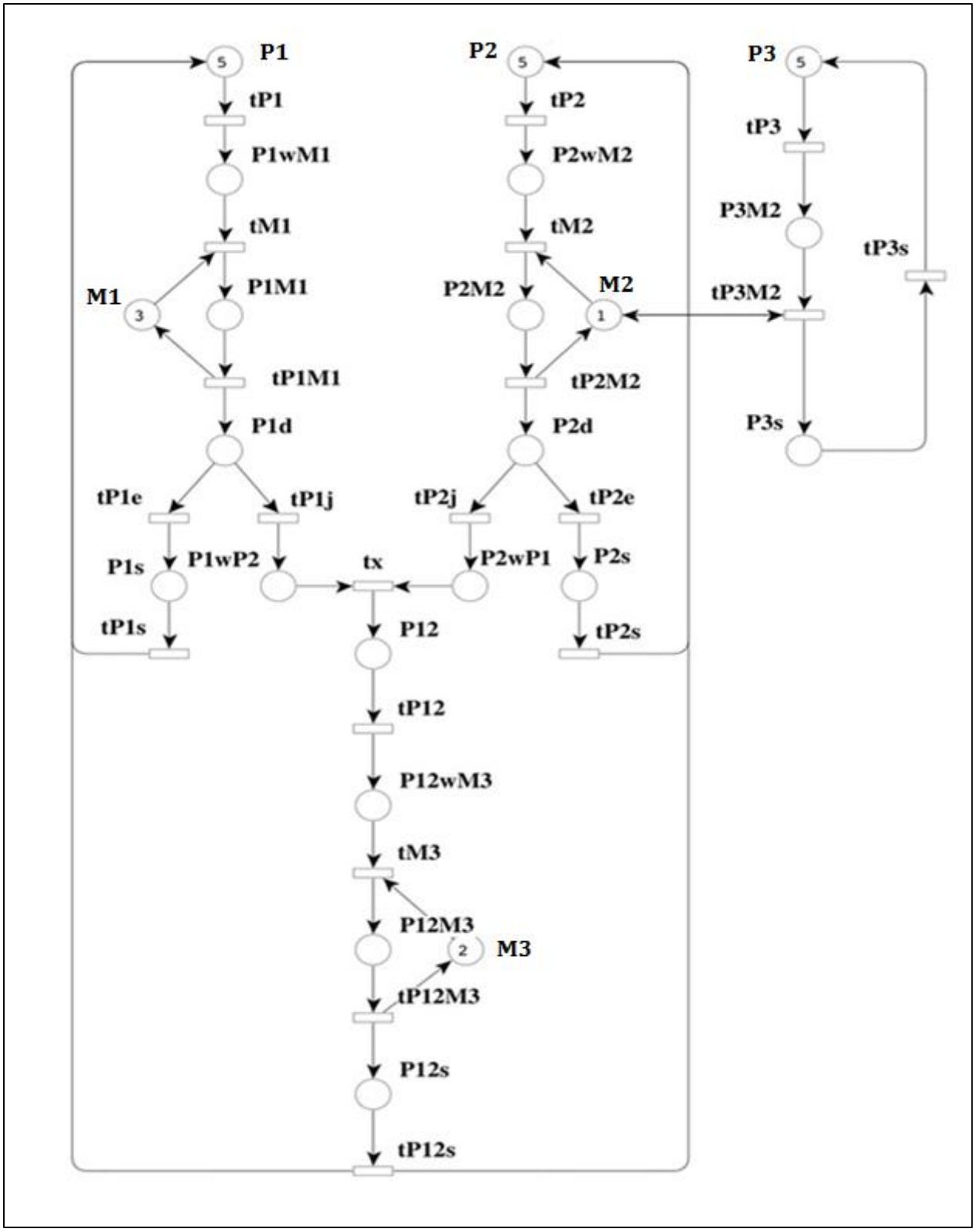

Figure 5-13 : TPN « FMS ».

La figure 5-14 montre un autre TPN de référence (Sbaï, Barkaoui, & Boucheneb, 2014). Il s'agit d'un processus permettant aux personnes à mobilité réduite d'attribuer des pensions. Ce processus est composé de quatre sous processus qui sont : le demandeur, la préfecture, la mairie et l'unité médicale.

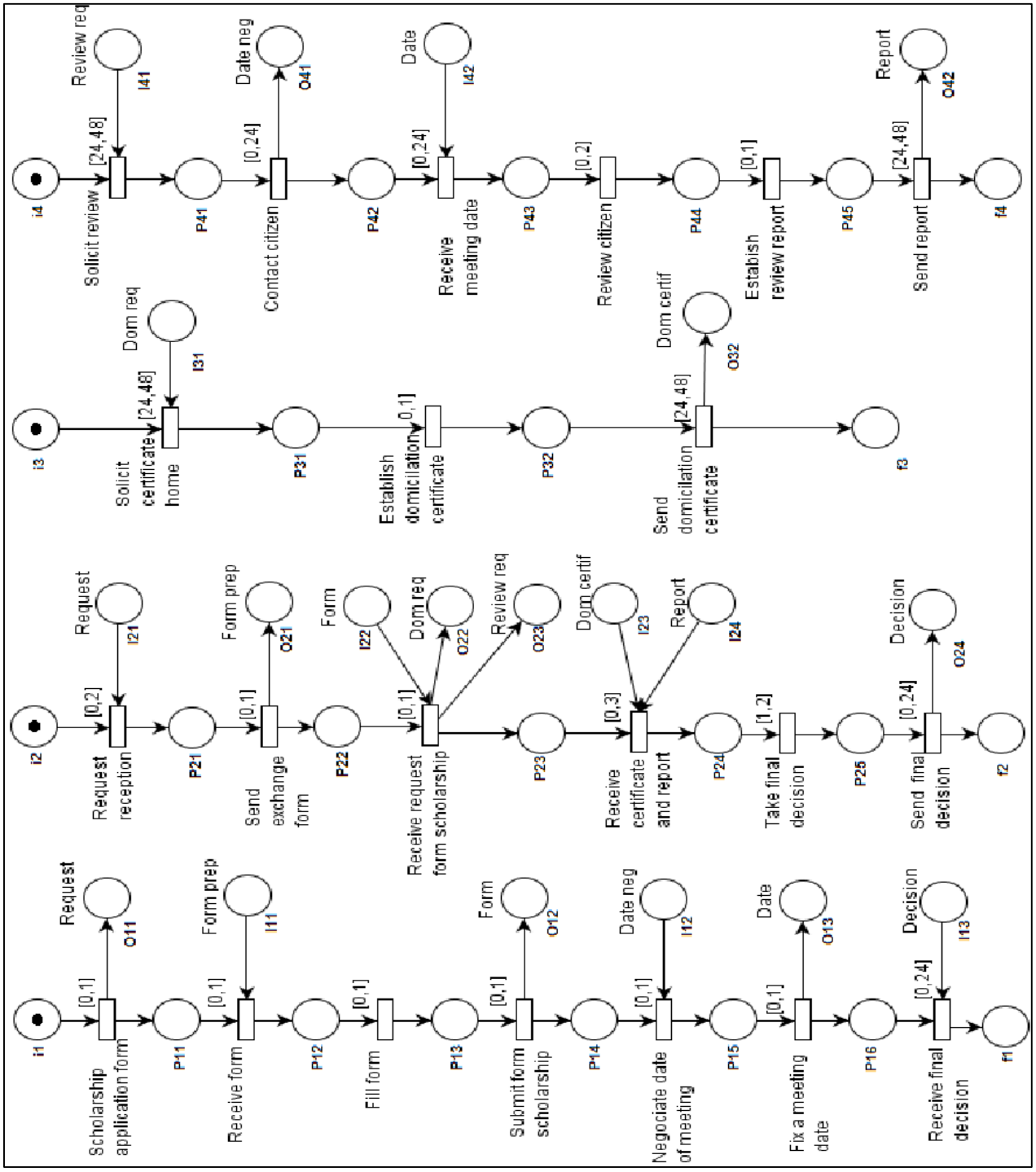

Figure 5-14 : TPN de référence modélisant le processus d'attribution des pensions.

La figure 5-15 contient le TPN HC (House Construction). Il a été proposé par Fabrice Kordon (Kordon, 2013) pour l'édition 2013 du MCC (Model Checking Contest). C'est un TPN de référence et qui est fortement connecté.

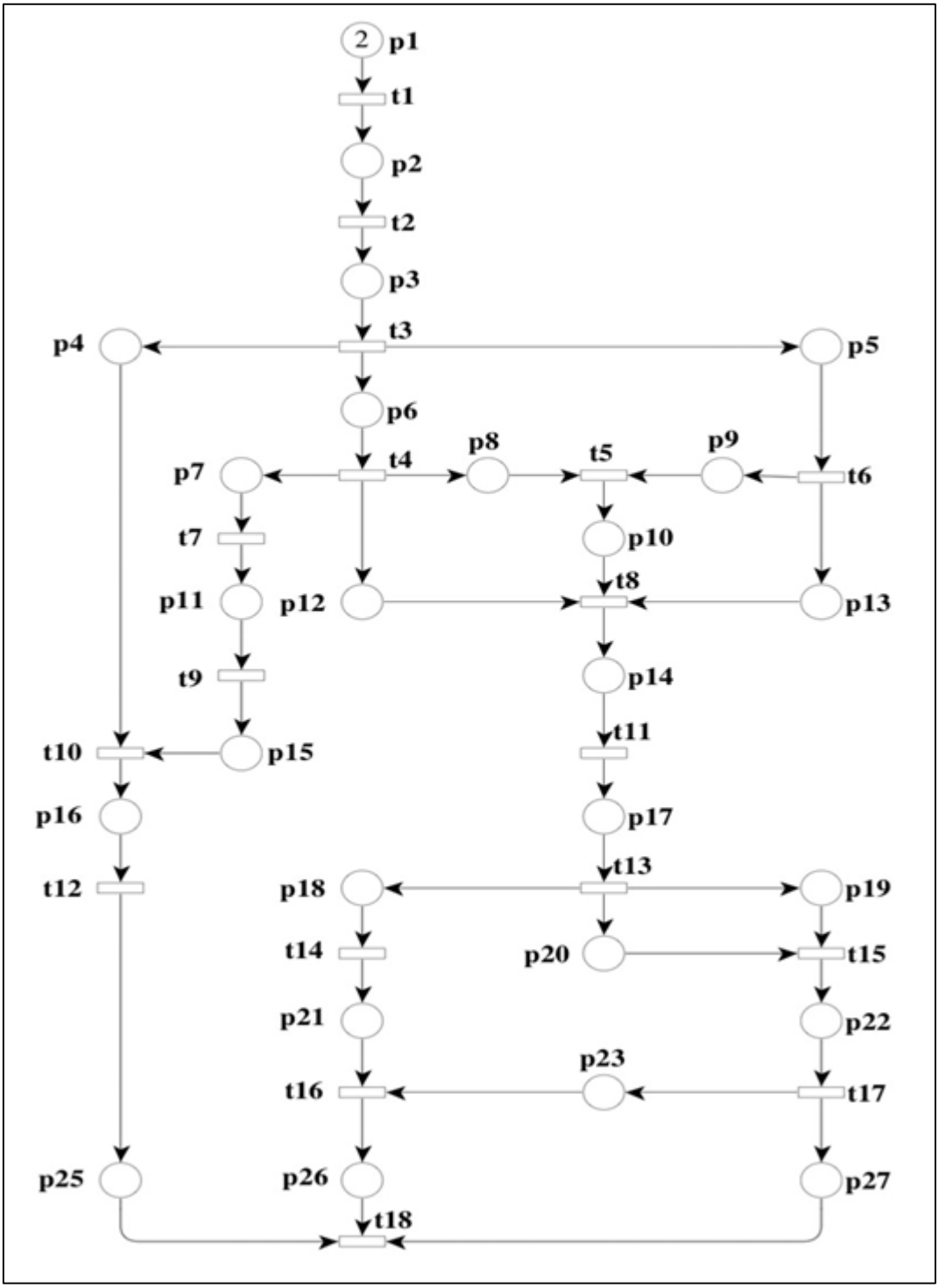

Figure 5-15 : TPN « HC ».

5.8 Résultats

Pour les résultats, nous avons spécifié un tableau par TPN. Pour chaque réseau nous avons varié les nombres de jetons contenus dans les places et nous avons utilisé différents types d'abstractions (SCG et CSCG) ainsi que différentes techniques de réduction d'ordre partiel (G(α), G'1(α) et G'2(α)). À noter que la méthode d'abstraction combinée avec les techniques de réduction d'ordre partiel est la CSCG.

Les tableaux 5.2, 5.3, 5.4, 5.5 et 5.6 contiennent respectivement les résultats suite à l'exécution des algorithmes de réduction d'ordre partiel sur les TPN : TP1, duplication, FMS, le TPN de référence et HC. Chaque variation de jetons, contient 3 lignes qui correspondent en ordre aux : nombre de classes d'états, le temps total d'exécution en secondes et le nombre d'arcs.

La colonne « Gain » permet de représenter le gain en pourcentage de notre technique G'1 par rapport à la méthode d'abstraction CSCG en termes de classes d'états et d'arcs. Prenons l'exemple du gain au niveau des classes d'états dans le tableau 5-2 ci-dessous, dans le cas où nous avons 5 jetons. Avec l'abstraction CSCG nous avons 407 classes tandis qu'avec G'1 nous avons obtenu 250 classes d'états et du coup le gain est : [(407-250)/407]*100 = 38,58%. De même pour le calcul du gain au niveau d'arcs est : [(825-397)/825]*100 = 51,88%.

Rappelons que notre technique basée sur le calcul de G'(α) (avec ses deux variantes) est une technique de réduction d'ordre partiel basée sur l'abstraction CSCG et l'ensemble G'(α) est complet.

À noter aussi que pour le TPN de la figure 5-11 la variation des jetons correspond aux places P0 et P1, c'est-à-dire lorsqu'il y a 1 dans la colonne 'Jetons' ça implique qu'il existe un jeton dans la place P0 et un jeton dans la place P1.

De même pour le TPN de la figure 5-12, la variation de jetons correspond aux places P0, P2, P4 et P6.

Tableau 5-2: Résultats de l'exécution du TPN1 avec l'outil INKA.

Jetons		SCG	CSCG	G	G'2	G'1	Gain (%)
1	Classe d'états	7	7	7	7	7	0
	T-exécution	0 s	0 s	0 s	0 s	0.001 s	
	Arcs	9	9	9	9	9	0
5	Classe d'états	453	407	394	255	250	38.58
	T-exécution	0.008 s	0.006 s	0.012 s	0.015 s	0.011 s	
	Arcs	902	825	777	416	397	51.88
10	Classe d'états	2226	2076	2072	1198	1195	42.46
	T-exécution	0.11 s	0.094 s	0.111 s	0.086 s	0.083 s	
	Arcs	4739	4470	4460	2005	1990	55.49
20	Classe d'états	9907	9396	9293	5551	5383	42.71
	T-exécution	2.029 s	1.822 s	1.903 s	0.796 s	0.763 s	
	Arcs	21564	20611	20254	9452	9183	55.45
50	Classe d'états	66106	63316	62412	36865	36438	42.46
	T-exécution	178.479 s	152.931 s	181.474 s	46.374 s	43.983 s	
	Arcs	145 579	140 170	139 450	63 392	62933	55.11

Tableau 5-3: Résultats de l'exécution du TPN 'Duplication' avec l'outil INKA.

Jetons		SCG	CSCG	G	G'2	G'1	Gain (%)
1	Classe d'états	230	101	9	9	9	91.08
	T-exécution	0.009 s	0.006 s	0 s	0.001 s	0.001 s	
	Arcs	506	436	8	8	8	98.17
2	Classe d'états	3510	3099	60	17	17	99.46
	T-exécution	1.446 s	1.172 s	0.002 s	0.002 s	0.002 s	
	Arcs	10996	9632	70	16	16	99.84
3	Classe d'états	44515	36477	808	25	25	99.94
	T-exécution	894.244 s	542.140 s	0.067 s	0.003 s	0.003 s	
	Arcs	176854	144400	1110	24	24	99.98
4	Classe d'états	_	_	11043	33	33	-
	T-exécution			13.915 s	0.005 s	0.005 s	
	Arcs			16303	32	32	
5	Classe d'états	_	_	101470	41	41	-
	T-exécution			1464.329 s	0.006 s	0.006 s	
	Arcs			159498	40	40	
10	Classe d'états	_	_	_	81	81	-
	T-exécution				0.012 s	0.012 s	
	Arcs				80	80	
50	Classe d'états	_	_	_	401	401	-
	T-exécution				0.065 s	0.065 s	
	Arcs				400	400	
1000	Classe d'états	_	_	_	8001	8001	-
	T-exécution				1.7 s	1.7 s	
	Arcs				8000	8000	

Tableau 5-4: Résultats de l'exécution du TPN de référence avec l'outil INKA.

Jetons		SCG	CSCG	G	G'2	G'1	Gain (%)
1 i1	Classe d'états	73	62	26	23	23	62.91
1 i2	T-exécution	0.002 s	0.002 s	0.002 s	0.011 s	0.011 s	
1 i3							
1 i4	Arcs	104	93	28	22	22	76.35
5 i1	Classe d'états	111	96	58	58	58	39.59
1 i2	T-exécution	0.004 s	0.003 s	0.006 s	0.055 s	0.045 s	
1 i3							
1 i4	Arcs	170	151	78	82	82	45.70
10 i1	Classe d'états	181	161	94	124	124	22.99
1 i2	T-exécution	0.005 s	0.006 s	0.009 s	0.121 s	0.105 s	
1 i3							
1 i4	Arcs	275	246	125	179	179	27.24
2 i1	Classe d'états	8553	7565	7071	833	773	89.79
2 i2	T-exécution	36.330 s	21.879 s	19.872 s	0.798 s	0.849 s	
2 i3							
2 i4	Arcs	20881	19004	12864	1204	1121	94.11
5 i1	Classe d'états	8770	7754	7536	4996	3359	56.69
2 i2	T-exécution	44.977 s	26.844 s	34.690 s	14.643 s	7.248 s	
5 i3							
5 i4	Arcs	21319	19413	16138	7638	4977	74.37
5 i1	Classe d'états	18687	16126	16082	8839	8796	45.46
5 i2	T-exécution	114.853 s	86.650 s	67.743 s	61.503 s	82.798 s	
5 i3							
1 i4	Arcs	47649	42555	41544	25556	25496	40.09
20 i1	Classe d'états	10671	9535	5085	4919	4418	53.67
2 i2	T-exécution	37.805 s	31.273 s	3.183 s	17.337 s	9.495 s	
20 i3							
20 i4	Arcs	25057	23020	8674	7966	7398	67.87

Tableau 5-5: Résultats de l'exécution du TPN 'FMS' avec l'outil INKA.

Jetons		SCG	CSCG	G	G'2	G'1	Gain (%)
1P1, 1P2, 1P3, 3M1, 1M2, 3M3	Classe d'états	3211	2894	2790	1731	1661	42.61
	T-exécution	0.397 s	1.135 s	1.12 s	1.66 s	1.555 s	
	Arcs	6735	6175	6022	2642	2227	63.94
1P1, 1P2, 2P3, 3M1, 1M2, 3M3	Classe d'états	14580	5437	13009	5106	4807	11.59
	T-exécution	1.588 s	4.575 s	6.7 s	8.04 s	7.904 s	
	Arcs	6020	13188	5322	7323	6796	48.47
1P1, 1P2, 5P3, 3M1, 1M2, 3M3	Classe d'états	35643	32753	31986	29220	24872	24.07
	T-exécution	496.79 s	327.709 s	144.96 s	329.87 s	320.82 s	
	Arcs	107116	98779	97812	87122	83989	14.98
1P1, 1P2, 5P3, 5M1, 5M2, 5M3	Classe d'états	7184	6007	5980	5120	4992	16.9
	T-exécution	2.53 s	6.780 s	7.81 s	10.07 s	9.72 s	
	Arcs	18982	15994	15792	13398	12730	20.41
2P1, 1P2, 1P3, 3M1, 1M2, M3	Classe d'états	57774	52080	51910	46981	44570	14.43
	T-exécution	722.07 s	631.671 s	666.38 s	899.22 s	883.63 s	
	Arcs	153503	139991	135180	120008	111339	20.47
1P1, 2P2, 2P3, 3M1, M2, M3	Classe d'états	42655	38825	35697	27776	26543	31.64
	T-exécution	576.58 s	493.935 s	502.65 s	502.07 s	483.17 s	
	Arcs	125607	114255	112912	99299	97150	14.98
1P1, 3P2, 1P3, 3M1, 1M2, M3	Classe d'états	46619	41748	40912	34798	32987	20.99
	T-exécution	647.30 s	479.699 s	521.76 s	524.898 s	522.373 s	
	Arcs	131215	118627	114277	98383	92242	22.25
1P1, 4P2, 1P3, 3M1, 1M2, 3M3	Classe d'états	80949	72292	60881	58222	55177	23.68
	T-exécution	2267.04 s	1683.05 s	1703.48 s	1803.29 s	1783.66 s	
	Arcs	240986	216243	208449	135887	130647	39.59

Tableau 5-6 : Résultats de l'exécution du TPN 'HC' avec l'outil INKA.

Jetons		SCG	CSCG	G	G'2	G'1	Gain (%)
100	Classe d'états	300	300	300	200	200	33.33
	T-exécution	0.002 s	0.008 s	0.008 s	0.005 s	0.01 s	
	Arcs	398	398	398	201	201	49.49
1,000	Classe d'états	3000	3000	3000	2000	2000	33.33
	T-exécution	0.079 s	0.092 s	0.6 s	0.72 s	0.59 s	
	Arcs	3998	3998	3998	2001	2001	49.95
10,000	Classe d'états	30,000	30,000	30,000	20,000	20,000	33.33
	T-exécution	17.611 s	18.422 s	15.91 s	11.055 s	12.896 s	
	Arcs	39,998	39,998	39,998	20,001	20,001	49.99
100,000	Classe d'états	300,000	300,000	300,000	200,000	200,000	33.33
	T-exécution	1615.63 s	1245.17 s	1358.22 s	990.87 s	920.67 s	
	Arcs	399,998	399,998	399,998	200,001	200,001	49.99

5.9 Commentaires des résultats

En observant les tableaux des résultats, nous remarquons bien que l'algorithme CSCG permet d'obtenir de meilleurs résultats par rapport à l'algorithme SCG.

Avec CSCG, nous avons moins de classes d'états, moins d'arcs et dans la plupart des cas moins de temps d'exécution par rapport au SCG. Ceci est tout à fait logique, car la différence entre ces deux algorithmes réside dans le faite que CSCG est une version contractée de SCG.

Comme nous l'avons vu dans le chapitre « revue de littérature », lorsqu'on compare deux classes d'états, dans CSCG on compare leurs matrices D sans tenir compte de la ligne et de la colonne 0 et du coup avec CSCG nous obtenons moins de classes d'états et d'arcs par rapport au SCG qui compare la totalité des matrices de deux classes. À noter que cette comparaison permet l'ajout ou non d'une classe au graphe final.

Pour la méthode G(α), nous constatons un léger gain. En effet, si le TPN est très connecté (comme c'est le cas pour le TPN1, FMS, le cas réel et HC) l'ensemble de transition à tirer à partir de chaque classe d'états avec cette technique va converger vers Fr (α) qui représente la liste de transition à tirer à partir d'une classe d'états utilisée dans les algorithmes CSCG et SCG.

En effet, dans cette technique basée sur G(α), nous construisons Fr (α) et G(α) puis nous calculons les successeurs de la classe d'états courante avec les transitions qui appartiennent à l'intersection entre Fr (α) et G(α). Alors si G(α) va être plus au moins identique à Fr (α) nous aurons à peu près les mêmes résultats que pour le CSCG et du coup le gain ne sera pas important.

Pour la technique basée sur G'(α) avec ses deux variantes G'1(α) et G'2(α), nous remarquons un gain aussi important variant dans la plupart des cas de 20 % à 99 %.

En effet, pour la technique utilisant le G'(α), l'ensemble de transitions à franchir à partir d'une classe d'états est plus compact que celui de G(α).

G'(α) exclut toutes les transitions qui sont en conflits ou bien dépendantes avec une transition se trouvant à l'intérieur de G'(α) si l'écart des délais de franchissement entres ces transitions est très grand, c'est-à-dire qu'il y a toujours une transition qui va être franchie avant celle qui est en conflit ou dépendante à la transition se trouvant dans G'(α).

Par conséquent, le gain en classes d'états et en arcs est plus considérable en appliquant l'algorithme G'(α). À rappeler que G'2(α) est moins contractée que G'1(α) et donc cette variante contient plus de classes d'états et d'arcs que la variante G'1(α).

Dans certains cas, nous avons obtenu des résultats très proches entre les différentes techniques, mais avec un temps d'exécution plus grand pour les techniques basées sur G(α), G'1(α) et G'2(α) par rapport à SCG et CSCG.

Ceci est justifié par le fait que G(α) et G'(α) avec ses deux variantes aient plus de calculs à faire, des conditions à vérifier et surtout la construction de l'ensemble de transitions à tirer à partir de chaque classe d'états. Et donc, si le nombre de classes d'états et d'arcs de ces trois techniques sont vraiment proches de ceux de SCG et CSCG, ce temps de calculs de plus va être apparent et du coup ces deux méthodes vont avoir moins de temps d'exécution.

Autre paramètre important que nous devons prendre en considération est que dans les trois techniques G(α), G'1(α) et G'2(α), nous commençons toujours par choisir une transition franchissable d'une façon aléatoire parmi toute la liste des transitions franchissables. Puis nous construisons l'ensemble de transitions à tirer d'une classe d'états α, par conséquent les classes et les arcs vont être influencés par ce choix aléatoire, d'où ce choix-là est décisif dans la réduction de l'espace d'états.

Dans l'exemple « Duplication », nous remarquons très bien le gain énorme en classes d'états, en arcs et en temps d'exécution surtout pour la méthode G'(α).

À noter que dans cet exemple nous avons mis des tirets pour mentionner que le nombre de classes d'états ou d'arcs a dépassé un million et que le temps d'exécution a dépassé une heure (3600 s).

Ce TPN nous permet de tirer une conclusion importante pour la technique développée à base de G'(α) : lorsque le TPN n'est pas fortement connecté (ici on a des branches indépendantes), le gain est très important.

Cette conclusion n'est pas spécifique uniquement à notre approche mais il s'agit d'un résultat obtenu en appliquant les techniques de réductions d'ordre partiel.

Les trois TPN FMS, le TPN de référence et HC sont fortement connectés. Dans certains cas nous avons obtenu un gain considérable, mais pas aussi grand comme pour l'exemple « Duplication » qui a dépassé les 90 %.

Dans ce chapitre nous avons détaillé la structure de notre outil INKA que nous avons implémenté pour tester l'algorithme G'(α) avec ses deux variantes ainsi que l'algorithme G(α).

Les résultats ont montré que notre algorithme G'(α) peut permettre d'obtenir un gain considérable par rapport aux autres techniques : plus le réseau TPN est moins connecté plus le gain en terme de classes d'états, d'arcs et de temps d'exécution est important.

Notre approche souffre de certaines limitations surtout en ce qui concerne notre outil développé. En effet, notre approche ne peut pas résoudre le problème d'explosion combinatoire mais elle permet de l'atténuer.

CHAPITRE 6 CONCLUSION

Dans ce présent mémoire, nous nous sommes intéressés à la vérification des réseaux de Petri temporels en utilisant une approche énumérative. Pour cela, nous avons développé une technique d'ordre partiel permettant des réductions importantes de l'espace d'états d'un réseau de Petri temporel ayant des comportements concurrents.

Cette technique se base sur l'abstraction CSCG et étend la méthode Stubborn Set tout en prenant en compte les paramètres temporels statiques et dynamiques des TPN. Il n'existe pas, à notre connaissance de techniques d'ordre partiel pour les TPN qui prennent en compte les paramètres temporels des TPN.

Ceci nous a permis de réduire le nombre de transitions à explorer à partir de chaque classe d'états et d'atténuer ainsi le problème d'explosion combinatoire.

Afin de tester l'efficacité de notre technique d'ordre partiel, nous avons développé un outil qui permet de construire le SCG, le CSCG, le graphe réduit résultant de notre technique ainsi que quelques graphes réduits obtenus par d'autres approches. Nous avons testé plusieurs modèles, incluant quelques modèles de références. Ces derniers sont utilisés pour confronter des outils de vérification, dans des compétitions organisées au sein de la conférence annuelle Petri nets.

Nous avons essayé d'optimiser cet outil en nous basant sur des algorithmes de complexités réduites comme dans le calcul de successeur d'une classe d'états, d'une bonne décomposition des modules et avec un bon choix de structures de données.

En parcourant les résultats, nous constatons que notre technique a donné dans la majorité des cas les meilleurs résultats en termes de nombre de classes d'états, d'arcs et de temps total de calcul. En effet, dans la majorité des cas, le gain a varié entre 20% et 90%.

Nos travaux de recherche ont fait l'objet d'un article présenté dans la conférence FORMATS 2014 (Boucheneb, Barkaoui, & Weslati, 2014).

Limitations

Comme tout travail humain, nos travaux de recherche dans ce présent mémoire contiennent certaines limitations surtout en ce qui concerne notre outil développé.

Parmi ces limites, nous citons :

-Notre outil cesse de fonctionner lorsqu'il s'agit d'un réseau de Petri temporel très compliqué et très connecté avec un grand nombre de jetons. C'est le cas où nous obtenons un espace d'états composés de plusieurs millions de classes d'états après des heures de calculs. En effet, notre technique ne résout pas le problème d'explosion combinatoire mais permet de l'atténuer.

-L'utilisation de la représentation de « TINA » seulement pour la modélisation des réseaux de Petri temporels. En effet, notre outil lit en entrée la spécification d'un TPN sous format TINA .txt ou .ndr. Ça sera plus flexible de varier les formats de fichiers pris en compte en entrée comme par exemple prendre en compte le format de fichier généré par l'outil Romeo.

-Absence de la fonctionnalité de représentation graphique d'un TPN dans notre outil. En effet, on génère le fichier contenant la spécification du réseau de Petri temporel étudié avec d'autres éditeurs graphiques qu'on les lit par la suite dans notre outil. Ça sera plus intéressant d'ajouter une partie pour modéliser les TPN directement dans notre outil.

-Absence des interfaces graphiques. En effet, notre outil s'exécute sur le Shell ou bien à l'aide d'un exécutable sur Windows. Comme amélioration, nous pourrions développer des interfaces graphiques avec Swing en Java par exemple et utiliser les modules de calculs écrits en C. Nous nous baserons sur le concept de JNI (Java Native Interface) qui nous permet de faire le lien entre les interfaces graphiques écrites en Java et les modules de traitement écrits en C ou en C++.

Travaux futurs

Face à une rude concurrence du marché, les entreprises accordent une importance considérable à la gestion de leurs activités (processus métiers). L'objectif global de ces entreprises est de fournir, à moindre coût, des services ou des produits de très haute qualité. L'automatisation des processus métiers, de plus en plus complexes, constitue un défi incontournable pour les entreprises soucieuses de leur efficacité, flexibilité et compétitivité.

Parmi les technologies utilisées pour gérer les processus métiers, nous trouvons les Workflows (SBAÏ, 2010).

Les Workflows sont utilisés, de nos jours, d'une façon importante pour gérer les processus métiers au sein des entreprises. Ils permettent l'automatisation d'une partie ou d'un ensemble complet d'un processus métier, durant lequel des données, des documents ou des tâches sont transférés d'un

acteur à un autre dans le but d'effectuer des actions. Ces dernières sont spécifiques selon le profil de l'acteur.

Une panoplie d'outils et de modèles de processus a donné naissance au développement de plusieurs systèmes de gestions de Workflows. Bien qu'il y ait un énorme effort dans les fonctionnalités offertes par ces systèmes de gestion de Workflows, toutefois, ils restent limités quant à la vérification systématique de certaines propriétés telles que la cohérence (Van der Aalst, 1997) des Workflows et la vivacité.

Un workflow est dit cohérent (sound) si :

- Chaque instance du Workflow se termine correctement,
- Il n'y a pas d'inter-blocage, et
- Il n'y a pas de tâches inutiles (toutes les tâches ont une chance de se réaliser).

Pour la vivacité, il s'agit de vérifier l'absence de situations globales ou locales de blocage.

La vérification des Workflows s'appuie sur le formalisme de réseaux de pétri temporels(TPN) où les tâches (activités) des Workflows sont modélisées par les transitions et les dépendances causales sont représentées par les places et les arcs.

Dans la littérature, la vérification des Workflows se base sur deux principales approches (Peter Bachmann & Popova-Zeugmann, 2010) (van der Aalst et al., 2011) :

- approches structurelles qui visent à déterminer des liens entre la structure des TPN et les propriétés à vérifier.

- approches énumératives (dynamiques) qui calculent l'espace des états puis parcourent cet espace afin de vérifier les propriétés. Ces approches souffrent du problème d'explosion combinatoire. Pour pallier ce problème, les techniques de réduction d'ordre partiel qui visent à réduire l'espace d'états ont prouvé leur efficacité dans la vérification de diverses propriétés. Les techniques de réduction d'ordre partiel proposées dans la littérature sont cependant moins appropriées pour vérifier la cohérence des Workflows.

Nous allons essayer alors d'améliorer notre technique pour réduire davantage l'espace d'états à explorer ainsi que d'étendre et d'adapter notre technique au contexte des Workflows.

BIBLIOGRAPHIE

Berthomieu, B., & Diaz, M. (1991). Modeling and verification of time dependent systems using time Petri nets. *Software Engineering, IEEE Transactions on, 17*(3), 259-273. doi: 10.1109/32.75415

Berthomieu, B., & Vernadat, F. (2003). State Class Constructions for Branching Analysis of Time Petri Nets. Dans H. Garavel & J. Hatcliff (Édit.), *Tools and Algorithms for the Construction and Analysis of Systems* (vol. 2619, p. 442-457): Springer Berlin Heidelberg.

Berthomieu*, B., Ribet, P.-O., & Vernadat, F. (2004). The tool TINA–construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research, 42*(14), 2741-2756.

Boucheneb, H., & Barkaoui, K. (2013). Reducing Interleaving Semantics Redundancy in Reachability Analysis of Time Petri Nets. *ACM Trans. Embed. Comput. Syst., 12*(1), 1-24. doi: 10.1145/2406336.2406343

Boucheneb, H., & Barkaoui, K. (2014). Stubborn sets for time Petri nets. ACM Transactions in Embedded Computing Systems (TECS).

Boucheneb, H., Barkaoui, K., & Weslati, K. (2014). Delay-Dependent Partial Order Reduction Technique for Time Petri Nets. Dans A. Legay & M. Bozga (Édit.), *Formal Modeling and Analysis of Timed Systems* (vol. 8711, p. 53-68): Springer International Publishing.

Boucheneb, H., Gardey, G., & Roux, O. H. (2009). TCTL model checking of time Petri nets. *Journal of Logic and Computation, 19*(6), 1509-1540.

Boucheneb, H., & Hadjidj, R. (2006). CTL* model checking for time Petri nets. *Theoretical Computer Science, 353*(1), 208-227.

Boucheneb, H., & Mullins, J. (2003). Analyse des réseaux temporels: Calcul des classes en O (n2) et des temps de chemin en O (m× n). *TSI. Technique et science informatiques, 22*(4), 435-459.

Boucheneb, H., & Rakkay, H. (2008). A More Efficient Time Petri Net State Space Abstraction Useful to Model Checking Timed Linear Properties. *Fundam. Inf., 88*(4), 469-495.

Clarke, E. M., Grumberg, O., & Peled, D. (1999). *Model checking*: MIT press.

Dams, D., Gerth, R., Knaack, B., & Kuiper, R. (1998). Partial-order Reduction Techniques for Real-time Model Checking. *Formal Aspects of Computing*, *10*(5-6), 469-482. doi: 10.1007/s001650050028

Dill, D. (1990). Timing assumptions and verification of finite-state concurrent systems. Dans J. Sifakis (Édit.), *Automatic Verification Methods for Finite State Systems* (vol. 407, p. 197-212): Springer Berlin Heidelberg.

Evangelista, S., & Pradat-Peyre, J.-F. (2006). On the Computation of Stubborn Sets of Colored Petri Nets. Dans S. Donatelli & P. S. Thiagarajan (Édit.), *Petri Nets and Other Models of Concurrency - ICATPN 2006* (vol. 4024, p. 146-165): Springer Berlin Heidelberg.

Godefroid, P., van Leeuwen, J., Hartmanis, J., Goos, G., & Wolper, P. (1996). *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem* (vol. 1032): Springer Heidelberg.

Guillaume Gardey, D. L., Morgan Magnin, Olivier (H.) Roux. (2005). Romeo: A Tool for Time Petri Nets Analysis. Tiré de http://romeo.rts-software.org/papers/cav-tool-05/romeo-cavtool-slides-2005.pdf

Hillah, L.-M. (2011). Picture of The Model. Tiré de http://mcc.lip6.fr/2013/index.php?N1=5&N2=7

Ismail Berrada, R. C., Patrick Félix. (2005). Prototype de génération de test d'interopérabilité pour les systèmes temporisés. Tiré de http://www-verimag.imag.fr/PROJECTS/DCS/AVERROES/Fournitures/Docs/F3.2.2.pdf

J-C.Geffroy. (2002). Introduction aux Réseaux de Petri. Tiré de http://cyranac.free.fr/pub/cours/cnamMOCA/CH10.pdf

Kimmo Varpaaniemi, J. H., Kari Hiekkanen, Tino Pyssysalo. (1995). PROD REFERENCE MANUAL. Tiré de http://move.lip6.fr/software/DOWNLOADS/CPN-AMI/B13.pdf

Kordon, F. (2013). Picture of The Model. Tiré de http://mcc.lip6.fr/2013/index.php?N1=5&N2=9

Merlin, P. M., & Farber, D. J. (1976). Recoverability of Communication Protocols--Implications of a Theoretical Study. *Communications, IEEE Transactions on*, *24*(9), 1036-1043. doi: 10.1109/TCOM.1976.1093424

Morère, Y. (2012). Cours de réseau de Petri. Tiré de http://www.morere.eu/IMG/pdf/cours_petri2.pdf

Partial order reduction. (n.d.). Dans *Wikipedia*. Consulté 25 août 2013, tiré de http://en.wikipedia.org/wiki/Partial_order_reduction

Penczek, W., & Półrola, A. (2001). Abstractions and Partial Order Reductions for Checking Branching Properties of Time Petri Nets. Dans J.-M. Colom & M. Koutny (Édit.), *Applications and Theory of Petri Nets 2001* (vol. 2075, p. 323-342): Springer Berlin Heidelberg.

Peter Bachmann, J., & Popova-Zeugmann, L. (2010). Time-independent Liveness in Time Petri Nets. *Fundamenta Informaticae, 102*(1), 1-17.

Réseau de Petri (n.d.). Dans *Wikipedia*.Consulté le 27 juillet 2013 Tiré de http://fr.wikipedia.org/wiki/Réseau_de_Petri

S.Vialle. (2007). 3-Les Réseaux de Petri. Tiré de http://www.metz.supelec.fr/metz/personnel/vialle/course/CNAM-ACCOV-NFP103/extern-doc/RdP/Introduction-RdP.pdf

Sami Evangelista, J.-F. c. P.-P. (2006). On the Computation of Stubborn Sets of Colored Petri Nets. Tiré de https://lipn.univ-paris13.fr/~evangelista/biblio-sami/doc/ICATPN-2006-slides.pdf

SBAÏ, Z. (2010). *Contribution à la modélisation et à la vérification de processus workflow*. (Doctoral dissertation, Conservatoire national des arts et metiers - CNAM, France). Tiré de https://tel.archives-ouvertes.fr/file/index/docid/553383/filename/TheseZohraSbaA_.pdf

Sbaï, Z., Barkaoui, K., & Boucheneb, H. (2014). Compatibility Analysis of Time Open Workflow Nets.

Sloan, R., & Buy, U. (1997). Stubborn Sets for Real-Time Petri Nets. *Formal Methods in System Design, 11*(1), 23-40. doi: 10.1023/A:1008629725384

Système temps réel. (n.d.). Dans *Wikipedia*.Consulté le 22 juillet 2013, tiré de http://fr.wikipedia.org/wiki/Système_temps_réel

Tcl. (n.d.). Dans *Wikipedia*.Consulté le 14 janvier 2014, tiré de http://en.wikipedia.org/wiki/Tcl

V.Augusto. (2012). Réseaux de Petri. Tiré de http://www.emse.fr/~augusto/enseignement/icm/gis1/UP2-2-RdP-slides.pdf

Valmari, A. (1991). Stubborn sets for reduced state space generation. Dans *Advances in Petri Nets 1990* (p. 491-515): Springer.

Valmari, A. (1992). A stubborn attack on state explosion. *Formal Methods in System Design, 1*(4), 297-322. doi: 10.1007/BF00709154

Valmari, A. (1994). State of the art report: Stubborn sets. *Petri Net Newsletter, 46* 6-14.

Van der Aalst, W. M. (1997). Verification of workflow nets. Dans *Application and Theory of Petri Nets 1997* (p. 407-426): Springer.

van der Aalst, W. M., van Hee, K. M., ter Hofstede, A. H., Sidorova, N., Verbeek, H., Voorhoeve, M., & Wynn, M. T. (2011). Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing, 23*(3), 333-363.

Varpaaniemi, K., Heljanko, K., & Lilius, J. (1997). *Prod 3.2 An advanced tool for efficient reachability analysis.* Communication présentée à Computer Aided Verification (p. 472-475).

Varpaaniemi, K., & Rauhamaa, M. (1992). The stubborn set method in practice. Dans K. Jensen (Édit.), *Application and Theory of Petri Nets 1992* (vol. 616, p. 389-393): Springer Berlin Heidelberg.