UNIVERSITÉ DE MONTRÉAL

ANALYSE ET VALIDATION FORMELLE DES SYSTÈMES TEMPS RÉEL

RACHID HADJIDJ
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIAE DOCTOR
(GÉNIE INFORMATIQUE)
FÉVRIER 2006

Your file Votre référence
ISBN: 978-0-494-16996-4
Our file Notre référence
ISBN: 978-0-494-16996-4

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

ANALYSE ET VALIDATION FORMELLE DES SYSTÈMES TEMPS RÉEL

présentée par: HADJIDJ Rachid

en vue de l'obtention du diplôme de: Philosophiae Doctor

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

M. PIERRE Samuel, Ph.D., président

Mme BOUCHENEB Hanifa, Doctorat, membre et directrice de recherche

Mme NICOLESCU Gabriela, Doct, membre

Mme DSSOULI Rachida, Ph.D., membre

REMERCIEMENTS

Je voudrais en premier remercier les professeurs Samuel Pierre, Rachida Dssouli, et Gabriela Nicolescu pour avoir accepté d'être les rapporteurs de cette thèse et de participer à mon jury.

Je tiens à remercier très chaleureusement ma directrice de thèse le professeur Hanifa Boucheneb, pour qui la réussite de cette thèse doit énormément. Je la remercie pour m'avoir guidé pendant mes années de thèse, pour sa confiance et pour toute l'attention qu'elle a porté à mon travail.

Merci également à toutes les personnes avec qui j'ai travaillé, ou que j'ai eu le plaisir de côtoyer pendant mes trois année de thèse. Plus particulièrement, Hamadou Sardaouna à qui je souhaite une bonne continuation dans ses travaux de thèse, le professeur Olivier Roux pour les discutions intéressantes durant sa visite au labo, ainsi que les membres du département Génie Informatique de l'école Polytechnique de Montréal pour le travail exemplaire qu'ils font.

Je remercie tous mes amis, en particulier Akram Benalia et sa femme d'avoir toujours été prêts à aider dans les moments difficiles, ainsi que Amar Bendou, Brahim Bessaa, Mourad Bouzegza, Tarek Ghazali et Mouhamed Chaouche pour leurs encouragements et soutien moral. Je n'oublierais pas aussi de remercier monsieur Said Boucheneb pour sa participation à la correction de ce document.

Enfin, merci énormément à ma femme, à mes parents et à toute ma famille pour leur soutien inconditionnel et constant.

RÉSUMÉ

Dans cette thèse, nous nous intéressons au modèle des réseaux de Petri temporels (modèle TPN) comme formalisme de modélisation des systèmes temps réels, pour lequel nous proposons un ensemble d'approches pour la vérification de ses propriétés temporisées et non temporisées. La technique de vérification retenue est le *model-checking*. Comme l'espace d'états du modèle TPN est généralement continu et infini, l'application du model-checking nécessite une étape supplémentaire d'abstraction afin de générer un espace d'état abstrait discret et fini. Cet espace doit aussi préserver les propriétés à vérifier. Pour les propriétés non temporisées, nos contributions sont plutôt des propositions qui visent à atténuer le problème de l'explosion des états. Nous commençons par définir un contexte formel pour caractériser un modèle d'états abstrait vis-à-vis des propriétés qu'il préserve, puis nous proposons un ensemble de modèles d'états abstraits pour le modèle TPN qui préservent ses propriétés d'atteignabilité, linéaires et de branchement. Ces propriétés peuvent par la suite être vérifiées efficacement avec des méthodes classiques de model-checking. Nous montrons par ailleurs que notre contexte formel est plus approprié que celui proposé dans la littérature pour caractériser les abstractions connues du modèle TPN. Pour les propriétés temporisées, nous proposons une logique temporelle et donnons sa sémantique formelle. Pour cette logique, nous développons une méthode de vérification à la volée, basée sur la méthode des classes, et prouvons sa décidabilité pour tous les modèles TPN bornés. Pour évaluer notre approche de vérification, nous la comparons à l'algorithme d'atteiganbilité implémenté dans l'outil UPPAAL. En moyenne, nous réussissons à vérifier des modèles TPN, en 7 fois moins de temps que l'outil UPPAAL et avec moins d'espace mémoire. Pour certaines propriétés, les réductions en temps de vérification dépassent les 3400, avec plus que 110 fois moins de ressources mémoire. Enfin, pour valider notre travail d'un point de vue pratique, nous avons développé un outil que nous avons appelé *RT-studio*, dans lequel nous

avons intégré toutes nos approches.

ABSTRACT

In this thesis, we are interested in the time Petri Net model (TPN model), for which we propose some approaches to verify its timed and untimed properties. The retained verification technique is *model-checking*. Since the TPN state space is in general continue and infinite, model-checking application requires an additional abstraction step to generate a discrete and finite abstract state space. This state space must also preserve the properties we want to verify. For untimed properties, our contributions are rather propositions to attenuate the state explosion problem. We start by defining a formal framework to characterize an abstract state model vis-a-vis the properties it preserves, then we propos some abstract state models for the TPN model that preserve its reachability, linear and branching properties. These properties can then be verified using classical model-checking methods. Furthermore, we show that our formal framework is more appropriate than the one proposed in the literature to characterize known abstractions for the TPN model. For timed properties, we propose a temporal logic, and give its formal semantic. For this logic, we propose a forward on-the-fly verification technique, based on the so called *state class method*, and prove its decidability for all bounded TPN Models. To evaluate our approach, we compare it to the reachability algorithm implemented in the tool UPPAAL. In average, we can verify TPN models 7 times faster than UPPAAl, with lesser memory usage. For certain properties, the verification is more than 3400 times faster, with 110 times lesser memory usage. To validate our approaches we implemented them in our tool called *RT-studio*.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

Abréviations

A-CSZG	Atomic Concrete State Zone Graph
$ACTL$	Universal Computation Tree Logic
$ACTL^*$	Universal Computation Tree Logic*
ASCG	Atomic State Class Graph
CC-CSZG	Convex combination Contracted Concrete State Zone Graph
CC-LSCG	Convex combination Contracted Linear State Class Graph
CTL	Computation Tree Logic
CTL^*	Computation Tree Logic*
CH-CSZG	*Convex Hull Contracted Concrete State Zone Graph*
CH-LSCG	Convex Hull Contracted Linear State Class Graph
CSZG	Concrete State Zone Graph
DBM	Difference Bound Matrix
$ELTL$	Existential Linear Temporal Logic
IC-LSCG	Inclusion Contracted Linear State Class Graph
IC-CSZG	Inclusion Contracted Concrete State Zone Graph
LSCG	Linear State Class Graph
LTL	Linear Temporal Logic
SSCG	Strong State Class Graph
TA	Timed Automata
$TCTL$	Timed Computation Tree Logic
$TCTL_{TPN}$	TCTL for the TPN model
TPN	Time Petri Net
ZBG	Zone Based Graph

Notations mathématiques

2^E	l'ensemble des sous ensembles de E
$[s]_\sim$	classe d'équivalence de s induite par $\sim$
E^n	l'ensemble des tuples à n dimensions sur l'ensemble E
$\|E\|$	cardinal de E
$\mathcal{M}_{m,n}(E)$	l'ensemble des matrices à m lignes et n colonnes sur E
$\mathbb{N}(\mathbb{N}^*)$	l'ensemble des entiers naturels (strictement positifs)
$\mathcal{P}$	partition
$\mathcal{P}_\sim$	partition induite par la relation d'équivalence $\sim$
$\mathbb{Q}(\mathbb{Q}^+)$	des nombres rationnels (positifs ou nuls)
$\mathbb{R}(\mathbb{R}^+)$	l'ensemble des nombres réels (positifs ou nuls)
$\sim, \approx$	relations d'équivalence

Systèmes de transitions

$s \rightarrow s'$	transition de s à s' $((s,s') \in \rightarrow)$
$s \xrightarrow{a} s'$	transition étiqueté par a (cas d'un système de transition étiqueté)
A	ensemble d'étiquettes
$(S, \rightarrow, s_0)$	Système de transition (S : ensemble d'états, $\rightarrow$: relation de transition, s_0 : état initial)
$\xrightarrow{*}$	fermeture réflexive et transitive $\rightarrow$
$\pi(s)$	l'ensemble des chemins d'exécution à partir de l'état s
ρ_s	chemin d'exécution à partir de l'état s
ρ	chemin d'exécution à partir de l'état initial s_0
$\rho_s(i)$	i-ème état sur le chemin d'exécution ρ_s
ρ_s^i	chemin d'exécution suffixe de ρ qui commence à l'états $\rho_s(i)$
$\breve{\rho}$	chemin d'exécution dense associé à un chemin d'exécution ρ
θ	délai temporel

Polyèdres, hyperplans, zones et matrices de bornes

$(c, \prec)$	borne sur $\mathbb{Q}$ ($c \in \mathbb{Q}$ et $\prec \in \{<, \leq\}$)
$(c_{i,j}, \prec_{i,j})$	borne de rang (i, j)
$(x \prec c)$	contrainte atomique simple ($c \in \mathbb{Q} \cup \{\infty, -\infty\}$, x variable sur $\mathbb{Q}$)
$(x - y \prec c)$	contrainte atomique triangulaire ($c \in \mathbb{Q} \cup \{\infty, -\infty\}$, x, y variables sur $\mathbb{Q}$)
B	matrice de bornes
$\mathcal{C}(X)$	l'ensemble des contraintes atomiques sur X
$Dom(F)$	domaine de F ($Dom(F) = \{V \in \mathbb{R}^X$ $t.q.$ $F(V)$ est vrai$\}$)
F	formule (système d'inéquations linéaires sur un ensemble de variables)
$\mathbb{R}^X$	ensemble de toutes les valuations sur X
$Sup(x, D)$	suppremum de x dans le domaine D
$V : X \rightarrow \mathbb{R}^+$	fonction de valuation sur X
$X = \{x_1, ..., x_n\}$	ensemble de variables
$\mathcal{Z}(X)$	l'ensemble de tous les polyèdres convexes (zones) sur X
$\prec$	operateur de comparaison, $\prec \in \{<, =, \leq, >, \geq\}$

Logiques temporelles

I	intervalle temporel
I_r	intervalle temporel de la forme $[0, c]$ ou $[0, c[$ ($c \in \mathbb{Q}^+ \cup \{\infty\}$)
$\mathcal{M}$	modèle (système de transitions + fonction de valuation d'états)
PV	ensemble de variables propositionnelles (propositions atomiques).
U,X	opérateurs modaux d'états *Until* et *Next*
$U_I, \Diamond_I, \Box_I$	version temporisée des opérateurs U, $\Diamond$ et $\Box$
φ, ψ, ϕ	formules de logique temporelle
$\wp$	proposition atomique
$\forall, \exists$	quantificateur de chemins universelle et existentiel
$\models$	relation de satisfaction
$\Diamond, \Box$	opérateurs modaux d'états exprimant la fatalité et la globalité

$\rightsquigarrow_{I_r}, \mapsto_I$	opérateurs modaux de la réponse bornée et de la première réponse bornée

Modèle TPN

$\downarrow I$	borne inférieure de l'intervalle I
$\uparrow I$	borne supérieure de l'intervalle I
m	marquage
m_0	marquage initial
(m, Id)	état intervalle
(m, V)	état horloge
$norm_k$	opération de normalisation sur les zones d'états
p	place
$pred(\alpha, t)$	prédécesseur de α par t ($pred(\alpha,t) \stackrel{def}{=} \{\sigma \mid \exists \sigma' \in \alpha,\ \sigma \overset{t}{\rightsquigarrow} \sigma'\}$)
$pred(\alpha, t, \alpha')$	prédécesseur de α par t dans α' ($pred(\alpha,t,\alpha') \stackrel{def}{=} pred(\alpha,t) \cap \alpha'$)
$succ(\alpha, t)$	successeur de α par t ($succ(\alpha,t) \stackrel{def}{=} \{\sigma \mid \exists \sigma' \in \alpha,\ \sigma' \overset{t}{\rightsquigarrow} \sigma\}$)
t	transition
$tmin(t)$	délais de franchissement minimal de t
$tmax(t)$	délais de franchissement maximal de t
AE	condition $AE : (\alpha \overset{t}{\rightarrow\!\!\!\!\!\rightarrow} \alpha') \Leftrightarrow (\forall \sigma \in \alpha, \exists \sigma' \in \alpha', \sigma \overset{t}{\rightsquigarrow} \sigma')$
AE_r	condition $AE_r : (\alpha \overset{t_f}{\rightarrow\!\!\!\!\!\rightarrow} \alpha') \Rightarrow (\forall \sigma \in \alpha, \exists \sigma' \in \alpha', \sigma \overset{t_f}{\rightsquigarrow} \sigma')$
$(C, \twoheadrightarrow, \alpha_0)$	modèle d'états abstraits
$En(m)$	l'ensemble des transitions sensibilisées dans le marquage m
EE	condition $EE : (\alpha \overset{t}{\rightarrow\!\!\!\!\!\rightarrow} \alpha') \Leftrightarrow (\exists \sigma \in \alpha, \exists \sigma' \in \alpha', \sigma \overset{t}{\rightsquigarrow} \sigma')$
EE'	condition $EE' : (\alpha \overset{t_f}{\rightarrow\!\!\!\!\!\rightarrow} \alpha') \Rightarrow (\exists \sigma \in \alpha, \exists \sigma' \in \alpha', \sigma \overset{t_f}{\rightsquigarrow} \sigma')$
EE_r	condition $EE_r : SG \wedge EE'$
EA	condition $EA : (\alpha \overset{t}{\rightarrow\!\!\!\!\!\rightarrow} \alpha') \Leftrightarrow (\forall \sigma' \in \alpha', \exists \sigma \in \alpha, \sigma \overset{t}{\rightsquigarrow} \sigma')$
EA_r	condition $EA_r : (\alpha \overset{t_f}{\rightarrow\!\!\!\!\!\rightarrow} \alpha') \Rightarrow (\forall \sigma' \in \alpha', \exists \sigma \in \alpha, \sigma \overset{t_f}{\rightsquigarrow} \sigma')$
$\breve{F}_{(F1,..,Fn)}$	couverture convexe des formules F_i ($i = \overline{1,n}$)
Id	fonction délais ($Id : T \rightarrow \mathbb{Q}^+_{[\,]}$)

Is	$Is : T \to \mathbb{Q}^+_{[\,]}$, associe à chaque transition t un intervalle statique de franchissement
$\mathcal{N}$	modèle TPN
P	ensemble fini de places
$Post$	fonction d'incidence arrière ($Post : P \times T \to \mathbb{N}$)
Pre	fonction d'incidence avant ($Pre : P \times T \to \mathbb{N}$)
SG	condition $SG : (\sigma \overset{t_f}{\rightsquigarrow} \sigma') \Rightarrow (\forall \alpha \in C \ t.q. \ \sigma \in \alpha, \exists \alpha' \in C, (\sigma' \in \alpha' \wedge \alpha \overset{t_f}{\twoheadrightarrow} \alpha'))$
$(\mathcal{S}, \to, s_0)$	espace des états du modèle TPN
T	ensemble fini de transitions
T_∞	ensemble des transitions avec intervalles statiques de franchissement non bornés ($T_\infty = \{t \in T \mid tmax(t) = \infty\}$)
U	condition $U : (\alpha \overset{t}{\twoheadrightarrow} \alpha') \Leftrightarrow (\forall \sigma \in noyau(\alpha), \exists \sigma' \in noyau(\alpha'), \sigma \overset{t}{\rightsquigarrow} \sigma')$, où $noyau(\alpha) \subseteq \alpha$, $\forall \alpha \in C$, et $\sigma_0 \in \alpha_0$
U_r	condition $U_r : (\alpha \overset{t_f}{\twoheadrightarrow} \alpha') \Rightarrow (\forall \sigma \in noyau(\alpha), \exists \sigma' \in noyau(\alpha'), \sigma \overset{t_f}{\rightsquigarrow} \sigma')$, où $noyau(\alpha) \subseteq \alpha \vee noyau(\alpha) \neq \emptyset$, $\forall \alpha \in C$, et $\sigma_0 \in noyau(\alpha_0)$
V	fonction de valuation ($V : T \to \mathbb{R}^+$)
$(Z, \twoheadrightarrow, \alpha_0)$	graphe de zones d'états concrets
α	état abstrait, classe d'états ou zone d'états
α_0	état abstrait initial
σ	état concret
$(\Sigma, \rightsquigarrow, \sigma_0)$	espace des états concrets du modèle TPN
$\rightsquigarrow$	relation de transition dans l'espace des états concrets
$\twoheadrightarrow$	relation de transition abstraite
$\prec_F^{x-y}$	$\leq$ ou $<$ selon que $x - y$ atteigne ou non son supremum dans le domaine de F
$\doteq$	équivalence par horloges

INTRODUCTION

La technologie informatique, qui envahie notre vie de tous les jours, est aussi utilisée pour contrôler des fonctions critiques dans des systèmes qui mettent en jeu des vies humaines et des coûts financiers importants. Les exemples les plus communément cités sont : les centrales nucléaires, les procédés industriels, les avions, les instruments médicaux, les protocoles de télécommunications et de commerce électroniques, etc. L'assurance de la qualité de fonctionnement de ces systèmes est donc indispensable.

L'analyse formelle des système temps réel

Il existe principalement quatre techniques pour assurer la correction des systèmes informatiques : la simulation, le test, la vérification déductive et le model-checking. La simulation et le test sont les techniques de vérification les plus souvent utilisées. La simulation est appliquée généralement sur un modèle abstrait du système à vérifier (Banks, 2000), alors que le test est effectué sur le système lui-même (Rusu *et al.*, 2005). Malheureusement, dans le cas de systèmes complexes, ces deux techniques ne peuvent couvrir qu'une partie limitée du comportement. La vérification déductive utilise un ensemble d'axiomes et de règles de déduction pour vérifier un système. Le processus de vérification peut être très complexe et très long. L'intervention d'experts qualifiés est aussi très souvent nécessaire. Cette technique est donc très rarement utilisée.

Le model-checking, introduit en 1980 par Queille et Sifakis, et par Clarke et Emmerson, est une technique automatique de vérification pour les systèmes d'états finis. Cette technique qui ne se veut pas générale, diffère des autres techniques par le fait qu'elle est complètement algorithmique et d'une complexité de mise en oeuvre

peu élevée. Elle consiste à effectuer une recherche exhaustive dans l'espace des états du système à vérifier pour déterminer la validité ou non d'une spécification. Ainsi, il a été possible de détecter beaucoup d'erreurs de conception dans des systèmes critiques. Pour toutes ces raisons, le model-checking est adopté de nos jours comme une procédure standard pour garantir la qualité des systèmes réactifs.

D'une manière générale, la vérification d'un système comporte trois étapes qui sont : la description formelle du comportement du système (la modélisation), la spécification des propriétés attendues (la spécification) et la preuve de la conformité des propriétés du modèle vis à vis de celles souhaitées (la validation). Le model-checking revient à vérifier qu'une propriété, généralement exprimée dans une logique temporelle telle que LTL ou CTL^*, est satisfaite par un modèle mathématique qui capture tous les comportements pertinents du systeme.

Les techniques de model-checking appliquées aux systèmes d'états finis discrets sont bien établies depuis déjà quelques décennies. Elles ont servi à valider avec grand succès des protocoles de télécommunication et des circuits électroniques très complexes (Clarke et Emerson, 1981; Queille et Sifakis, 1981). Récemment, l'intérêt dans le domaine de la vérification automatique s'oriente vers les systèmes temps réel. De par leurs fonctions, ces systèmes doivent impérativement satisfaire des contraintes temporelles strictes. Valider un système temps réel consiste à garantir, d'une part, sa correction algorithmique (validation fonctionnelle), et d'autre part, que le système sera toujours capable de réagir dans les bons délais, indépendamment du contexte (validation temporelle).

Plusieurs modèles formels, qui intègrent de différentes manières le paramètre temps, ont été proposés dans la littérature. Parmi les plus étudiés et les plus utilisés, on retrouve le modèle des automates temporisés (*Timed Automatat* (TA)), et plusieurs modèles de réseaux de Petri temporisés (*Timed Petri Net* (TPN)) : Ramchandani (Ramchandani, 1974), Sifakis (Sifakis, 1977), Merlin (Merlin et Farber, 1976), André

(André, 1989), Van Der Aalst (Aalst, 1993) et Diaz (Diaz et Senac, 1994).

Problématique

Dans la pratique, les techniques de model-checking se heurtent au problème d'explosion des états. Pour les systèmes temps réel ce problème est beaucoup plus sévère. L'aspect continu du temps rend souvent l'espace des états de tels systèmes infini et non dénombrable. Les méthodes qui traitent cette problématique sont nombreuses dans la littérature, mais n'ont malheureusement pas encore atteint la maturité escomptée. Parmi les techniques développées, on retrouve les méthodes symboliques basées soit sur l'une des variantes des graphes de décision, ou le model-cheking borné (Moskewicz *et al.*, 2001) et les résolveurs SAT (Penczek *et al.*, 2002). Quant aux méthodes les plus connues et les plus utilisées, elles se basent généralement sur la construction de modèles d'états abstraits qui préservent les propriétés des systèmes à vérifier. L'abstraction permet de capturer des espaces d'états généralement infinis dans des structures discrètes et finies, de telle sorte que les techniques définies dans le cadre discret puissent être appliquées. Le niveau de réduction que procure une abstraction dépend souvent des propriétés à préserver, mais aussi de la technique d'abstraction utilisée. Dans ce contexte, on a souvent recours à l'utilisation de structures de données adaptées. Parmi les représentations les plus couramment utilisées, on retrouve les matrices de bornes (*Difference Bound Matrix* (DBM)) (Dill, 1989), les diagrammes de décision booléens (*Boolean Decision Diagram* (BDD)) (Behrmann *et al.*, 1999), les diagrammes de différence d'horloges (*Clock Difference Diagram* (CDD)) (Behrmann *et al.*, 2002; Paige et Tarjan, 1987), etc. D'autres techniques appliquent des méthodes d'ordre partiel qui visent à supprimer les entrelacements inutiles d'actions (Godefroid et Wolper, 1994; Valmari, 1992), ou utilisent l'abstraction et la symétrie (Alur et Dill, 1990; Bobbio et Horvath, 2001; Clarke *et al.*, 1993; Clarke *et al.*, 1994; Emerson et Sistla, 1996).

Plusieurs techniques de construction de modèles abstraits existent pour les automates temporisés (Alur *et al.*, 1993; Alur *et al.*, 1992; Bouajjani *et al.*, 1997; Dembinski *et al.*, 2002; Polrola *et al.*, 2003; Tripakis et Yovine, 2001) et les différents modèles de réseaux de Petri temporisés (Berthomieu et Diaz, 1991; Berthomieu et Vernadat, 2003; Gardey *et al.*, 2003; Lilius, 1999; Okawa et Yoneda, 1997; Penczek et Polrola, 2001; Virbitskaite et Pokozy, 1999; Yoneda et Ryuba, 1998). Il existe aussi des techniques pour la transformation structurelle d'un modèle formel dans un autre afin d'exploiter des techniques déjà développées (Cortés *et al.*, 2002; Haar *et al.*, 2000; Hulgaard et Burns, 1995; Lime et Roux, 2003; Polrola et Penczek, 2004; Sifakis et Yovine, 1996), ou des adaptations de techniques existantes pour certains modèles mais qui ne le sont pas pour d'autres (Gardey *et al.*, 2003; Okawa et Yoneda, 1997; Virbitskaite et Pokozy, 1999). Cependant, les techniques proposées pour le modèle des réseaux de Petri temporisés, n'ont pas connu des développements comparables à ceux des automates temporisés. *La méthode des classes*, introduite par Berthomieux et Menasche en 1983 (Berthomieu et Menasche, 1983), reste à ce jour la méthode de référence en matière de vérification des propriétés d'atteignabilité et temporelles du modèle TPN. En dépit de ses avantages indéniables, cette méthode a le désavantage de générer des modèles d'états abstraits très grands pour des modèles relativement petits. Quant au problème de vérification de propriétés temporisées du modèle TPN, il reste à ce jour non traité d'une manière satisfaisante. Parmi les approches proposées, la *méthode des observateurs* (Toussaint *et al.*, 1997) modifie le modèle TPN selon la propriété à vérifier en lui greffant des places et des transitions qu'on qualifie d'*observateur*. Une propriété temporisée dans le modèle initial devient alors une propriété d'atteignabilité dans le modèle transformé, qu'on vérifie avec la méthode des classes. Malheureusement, cette technique se limite à des propriétés temporisées très particulières, et ne permet de couvrir qu'une classe assez réduite de propriétés exprimables dans la logique $TCTL$. Les autres techniques connues se basent généralement sur la traduction des TPNs en des automates temporisés

(Alur et Dill, 1990), afin d'utiliser les outils de model-checking existants (Cassez et Roux, 2003; Cortés *et al.*, 2002; Gu et Shin, 2002; Lime et Roux, 2003; Okawa et Yoneda, 1997). Un modèle TPN est généralement traduit en un ensemble d'automates temporisés dont la composition parallèle est sémantiquement équivalente au modèle d'origine. Le model-checking est par la suite effectué sur l'automate temporisé, puis les résultats sont interprétés de nouveau sur le modèle TPN. Malgré leur pertinence, ces techniques sont exposées à la complexité exponentielle en nombre d'horloges, inhérente aux automates temporisés et à la difficulté d'interpréter les propriétés et les résultats obtenus sur le modèle d'origine.

Notre contribution

Dans notre thèse, nous considérons le modèle réseaux de Petri temporels (Merlin et Farber, 1976) (la notation TPN réfère à ce modèle dans le reste de ce document) comme formalisme de modélisation des systèmes temps réel et proposons un ensemble de techniques pour la vérification par model checking de ses propriétés temporisées et non temporisées. Ce modèle étant infini et à branchements inifinis, la vérification de propriétés par model checking doit passer par des abstractions. Le but d'une abstraction est de construire un modèle abstrait fini qui a les mêmes propriétés (du moins celles à vérifier) que le modèle de départ. Ainsi, les propriétés peuvent être vérifiées sur le modèle abstrait fini obtenu. Nous proposons un ensemble de conditions suffisantes pour caractériser un modèle d'états abstraits vis-à-vis des propriétés qu'il préserve. Nous montrons que nos conditions sont beaucoup plus souples que celles proposées dans (Penczek et Polrola, 2001), et plus appropriées pour caractériser les différentes abstractions du modèle TPN proposées dans la littérature. Nous proposons ensuite une autre abstraction pour le modèle TPN qui préserve ses propriétés d'atteignabilité et linéaires. Nous appelons cette abstraction *graphe des zones d'états concrets* (*Concrete State Zone Graph* (CSZG)). Pour

forcer la terminaison de la construction du CSZG dans le cas d'un modèle TPN borné avec des intervalles de franchissement non bornés, nous définissons une équivalence entre états, et une opération d'approximation (appelée *k-approximation* ou *k-normalisation* (Pettersson, 1999)) sur les zones d'états. Nous montrons que cette approximation préserve les séquences de franchissement et les propriétés linéaires du modèle TPN. Nous prouvons par la suite que le CSZG est fini ssi le modèle TPN est borné. Afin de contracter davantage le CSZG, nous proposons trois abstractions : par *inclusion* (Boucheneb et Hadjidj, 2005), par *combinaison convexe* (Hadjidj et Boucheneb, 2005a) et par *couverture convexe*. Les deux premières abstractions préservent les propriétés d'atteignabilité du modele TPN et ont l'avantage d'être beaucoup plus compactes et plus rapidement calculables que le CSZG (leur calcul se fait directement, sans passer au préalable par le calcul du CSZG). La troisième abstraction est généralement plus compacte et plus rapide à calculer que les deux premières, mais ne préserve pas forcément les propriétés d'atteignabilité. Elle permet cependant de définir des conditions suffisantes pour vérifier la bornitude et l'atteignabilité des marquages du modèle TPN. D'une manière similaire, nous montrons que les trois abstractions du CSZG s'appliquent aussi au graphe des classe linéaires (LSCG) (Berthomieu et Menasche, 1982; Berthomieu et Vernadat, 2003) et produisent des contractions avec des propriétés très intéressantes. Par ailleurs, nous montrons que les abstractions du CSZG peuvent être utilisées pour construire par raffinement des abstractions qui préservent les propriétés de branchement. Pour accélérer la construction de telles abstractions, nous proposons un algorithme de raffinement combiné avec l'abstraction par inclusion. La terminaison de cet algorithme est garantie pour tous les modèles TPN bornés. Afin d'améliorer davantage les performances, nous montrons comment réduire la complexité de calcul de chaque zone d'états de $O(n^3)$ à $O(n^2)$, n étant le nombre de transitions sensibilisées dans la zone d'états. Sur le plan pratique, les tests montrent qu'il est plus avantageux de raffiner une contraction du CSZG que le CSZG lui même. En effet, le temps et

l'espace nécessaires à la construction sont réduits d'un facteur pouvant dépasser 10 dans certains cas.

Pour vérifier les propriétés temporisées du modèle TPN nous considérons une sous-classe de la logique temporisée $TCTL$, et définissons sa sémantique formelle dans le contexte du modèle TPN. Nous proposons ensuite une technique de vérification plus efficace que celles utilisées dans la littérature, et prouvons sa décidabilité pour les modèles TPN bornés. Enfin, nous comparons notre technique de vérification avec l'algorithme implanté dans l'outil UPPAAL (Behrmann *et al.*, 2002), et montrons ses point forts.

Pour valider notre travail d'un point de vue pratique, nous avons développé un outil que nous avons appelé *RT-studio*, dans lequel nous avons intégré toutes nos approches.

Plan de la thèse

La suite de ce document est organisée de la manière suivante :
Le chapitre 1 présente les concepts de base utilisés dans le domaine de la vérification formelle des systèmes temps réel, ainsi que les logiques temporelles considérées dans cette thèse, à savoir, les logiques CTL^* et $TCTL$. Le chapitre 2 est consacré aux réseaux de Petri temporels (modèle TPN), et aux techniques les plus connues de construction de modèles d'états abstraits pour ce modèle. Le chapitre 3 propose une définition générale pour les modèles d'états abstraits ainsi qu'un ensemble de conditions suffisantes pour caractériser les propriétés qu'ils préservent. Le chapitre 4 est dédié à la construction du graphe des zones d'états concrets ainsi qu'à celles de ses trois contractions : par inclusion, combinaison convexe et couverture convexe. Nous montrons également à ce niveau comment raffiner les graphes obtenus afin de restaurer les propriétés de branchement. Le chapitre 5 est consacré à la

vérification des propriétés temporisées du modèle TPN. Nous définissons dans ce chapitre la logique temporelle temporisée $TCTL_{TPN}$ considérée dans cette thèse, et développons un algorithme de model-checking pour ces propriétés. Au chapitre 6, nous présentons brièvement notre outil *RT-studio*, et reportons les principaux résultats expérimentaux obtenus. Finalement, nous terminons ce document par une conclusion et quelques perspectives aux travaux présentés dans cette thèse.

CHAPITRE 1

NOTIONS PRÉLIMINAIRES

Nous désignons par $\mathbb{N}$, $\mathbb{Q}$ et $\mathbb{R}$ respectivement l'ensemble des entiers naturels, des nombres rationnels et des nombres réels. $\mathbb{N}^*$ est l'ensemble des entiers naturels strictement positifs. $\mathbb{R}^+$ et $\mathbb{Q}^+$ sont respectivement l'ensemble des nombres réels et rationnels positifs ou nuls. Soient $n \in \mathbb{N}$ et E un ensemble quelconque. E^n dénote l'ensemble des tuples à n dimensions sur E. 2^E dénote l'ensemble des sous ensembles de E. Soit $(m, n) \in \mathbb{N}^2$, $\mathcal{M}_{m,n}(E)$ est l'ensemble des matrices à m lignes et n colonnes sur E. Si E est un ensemble fini, le cardinal de E est noté $|E|$. Si E est un ensemble infini, nous écrirons $|E| = \infty$.

1.1 Système de transitions

Un *système de transitions* est une structure définie par un triplet $(S, \rightarrow, s_0)$, où S est un ensemble d'états, $s_0 \in S$ est l'état initial, et $\rightarrow$ une relation de transitions entre les états de S. La notation $s \rightarrow s'$ est utilisée pour spécifier que $(s, s') \in \rightarrow$. Un système de transitions est dit *fini* ssi $\rightarrow$ est finie. Dans le cas contraire, le système est dit *infini*. Un système de transitions est dit *à branchements infinis* ssi il existe $s \in S$ tel que $|\{s' \in S | s \rightarrow s'\}| = \infty$ (i.e., s a une infinité, éventuellement non dénombrable, de successeurs). L'ensemble des *états atteignables* du système de transitions est $\{s | s_0 \stackrel{*}{\rightarrow} s\}$, où $\stackrel{*}{\rightarrow}$ est la fermeture réflexive et transitive de $\rightarrow$. Un *chemin d'exécution*, à partir de l'état $s \in S$, est une séquence maximale $\rho_s = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 .....$, où $s = s_1$. Lorsque l'état initial d'un chemin n'est pas explicitement spécifié, c'est l'état initial du système de transitions qui sera considéré (i.e., $\rho = \rho_{s_0}$). Pour un chemin d'exécution ρ_s, $\rho_s(i)$ dénote son i-ème

état, les états étant numérotés à partir de *un*. Un état s est donc atteignable, s'il existe un chemin d'exécution ρ et $i \in \mathbb{N}^*$ tel que $s = \rho(i)$. ρ_s^i dénote le chemin suffixe de ρ_s qui commence à l'états $\rho_s(i)$. L'ensemble des chemins d'exécution à partir d'un état s est noté $\pi(s)$.

1.2 Système de transitions étiqueté

Soit A un ensemble d'étiquettes. Un système de transitions $(S, \rightarrow, s_0)$ est dit *étiqueté* par l'ensemble A, si la relation de transition $\rightarrow$ est étiquetée par A. i.e., $\rightarrow \subseteq (S \times A \times S)$. La notation $s \xrightarrow{a} s'$ est utilisée pour spécifier que $(s, a, s') \in \rightarrow$. Nous utiliserons aussi la notation $s \rightarrow s'$ pour spécifier que $\exists a \in A \ t.q. \ (s, a, s') \in \rightarrow$.

1.3 Système de transitions temporisé

Un *système de transitions temporisé* sur l'ensemble d'actions A est un système de transitions étiqueté par $A \cup \mathbb{R}^+$, tel que $A \cap \mathbb{R}^+ = \emptyset$. Une transition $s \xrightarrow{a} s'$ où $a \in A$ est dite *discrète* alors qu'une transition $s \xrightarrow{\theta} s'$ où $\theta \in \mathbb{R}^+$ est dite *temporelle* de θ unités de temps. Un chemin d'exécution à partir de l'état $s \in S$, est une séquence maximale alternée $\rho = s_1 \xrightarrow{\theta_1} s_1 + \theta_1 \xrightarrow{a_1} s_2 \xrightarrow{\theta_2} s_2 + \theta_2 \xrightarrow{a_2} .....$, avec $s = s_1$, $a_i \in A$, et $\theta_i \in \mathbb{R}$. $s + \theta$ dénote l'état atteignable à partir de s après une progression de θ unités de temps. Un état s est dit atteignable s'il existe un chemin d'exécution ρ et $i \in \mathbb{N}^*$ tel que $s = s_i + \delta$, et $0 \leq \delta \leq \theta_i$.

1.4 Relation d'équivalence et partition

Une relation binaire $\sim$ sur un ensemble S est un sous ensemble de $(S \times S)$. $\sim$ est *réflexive* ssi $s \sim s$, $\forall s \in S$. $\sim$ est *symétrique* ssi $s \sim s' \Rightarrow s' \sim s$, $\forall s, s' \in S$. $\sim$ est

transitive ssi $s \sim s' \wedge s' \sim s" \Rightarrow s \sim s"$, $\forall s, s', s" \in S$. Etant données deux relations $\sim$ et $\sim'$, $\sim$ est dite *plus forte* que $\sim'$ ssi $\sim \subseteq \sim'$ ($\sim'$ induit une partition plus petite que celle induite par $\sim$).

Une *partition* de l'ensemble S est un ensemble $\mathcal{P} \subseteq 2^S$ tel que :
- $\forall P_1, P_2 \in \mathcal{P}$, $P_1 \cap P_2 = \emptyset$,
- $\forall s \in S$, $\exists P \in \mathcal{P}$ telle que $s \in P$.

La relation $\sim$ est une *relation d'équivalence* sur S ssi elle est réflexive, symétrique et transitive. Dans ce cas, $\sim$ induit sur S une *partition* $\mathcal{P}_\sim$, telle que pour chaque $P \in \mathcal{P}_\sim$, $s_1, s_2 \in P$ ssi $s_1 \sim s_2$. Chaque élément de $\mathcal{P}_\sim$ est appelé *classe d'équivalence.* Pour $s \in S$, la classe d'équivalence de s par $\sim$, notée $[s]_\sim$, est la classe d'équivalence $P \in \mathcal{P}_\sim$ telle que $s \in P$.

1.5 Simulation et Bisimulation

Soient $\mathcal{S} = (S, \rightarrow, s_0)$ et $\mathcal{S}' = (S', \rightarrow', s'_0)$ deux systèmes de transitions étiquetés par un ensemble d'étiquettes A. Soit $\approx \subseteq (S \times S')$ une relation binaire sur l'ensemble des états des deux systèmes. La relation $\approx$ est une *simulation* ssi $\forall (s, s') \in (S \times S')$ tel que $s \approx s'$, la condition suivante est vérifiée : $s \xrightarrow{a} s_1 \Rightarrow \exists s'_1$ *t.q.* $s' \xrightarrow{a} s'_1 \wedge s_1 \approx s'_1$.

La relation $\approx$ est une *bisimulation* ssi $\approx$ et son inverse sont des simulations.

On dit que le système $\mathcal{S}$ *simule* le système $\mathcal{S}'$ ssi il existe une relation de simulation $\approx$ telle que $(s_0, s'_0) \in \approx$. $\mathcal{S}$ et $\mathcal{S}'$ sont dits *bisimilaires* ssi il existe une relation de bisimulation $\approx$ telle que $(s_0, s'_0) \in \approx$.

1.6 Polyèdres, hyperplans et zones

Soit $X = \{x_1, ..., x_n\}$ un ensemble de variables sur $\mathbb{R}$. Une *valuation* sur X est une fonction $V : X \rightarrow \mathbb{R}^+$. L'ensemble de toutes les valuations sur X est noté $\mathbb{R}^X$. La valuation qui associe à chaque variable $x \in X$ la valeur zero est notée 0. Notons qu'une valuation V sur X peut être vue comme un tuple sur $\mathbb{R}^n$. Dans ce cas, $\mathbb{R}^X$ sera confondu avec $\mathbb{R}^n$. Une *contrainte atomique* sur X est une inéquation linéaire de la forme $(x - y \prec c)$ ou $(x \prec c)$, avec $c \in \mathbb{Q} \cup \{\infty, -\infty\}$, $\prec \in \{<, =, \leq, >, \geq\}$ et $x, y \in X$. $\mathcal{C}(X)$ dénote l'ensemble des contraintes atomiques sur X. Une contrainte atomique du type $(x \prec c)$ est dite *simple*. Une contrainte atomique du type $(x - y \prec c)$ est dite *triangulaire*. Un système d'inéquations linéaires sur X est aussi appelé *formule*. Le domaine d'une formule F, noté $Dom(F)$, est l'ensemble des valuations sur X qui satisfont F (i.e., $Dom(F) = \{V \in \mathbb{R}^X \ t.q. \ F(V) \text{ est vrai}\}$). Nous confondrons souvent dans la suite de ce document une formule F avec son domaine. Une formule F sera dite *consistante* ssi le système d'inéquations qu'elle représente possède au moins une solution (i.e., un tuple de valeurs qui satisfont en même temps toutes les contraintes de F).

Un *hyperplan* H sur X est un ensemble de valuations qui satisfont une contrainte atomique sur X. Un *polyèdre* sur X est la composition par union et intersection d'un nombre fini d'hyperplans sur X. La formule associée à un polyèdre est donc la composition par disjonction et conjonction des formules de ses hyperplans.

Un polyèdre P est dit *convexe* ssi pour toutes valuations $v_1, v_2 \in P$ et $\lambda \in \mathbb{R}$ t.q. $0 < \lambda < 1$, $\lambda v_1 + (1 - \lambda)v_2 \in P$. Un polyèdre convexe est aussi appelé une *zone*. L'ensemble de toutes les zones sur X est noté $\mathcal{Z}(X)$. Il est facile de montrer qu'un polyèdre est convexe (i.e., une zone) ssi il peut être défini comme une intersection d'un nombre fini d'hyperplans. La formule associée à une zone est donc une conjonction de contraintes atomiques. De la définition d'un polyèdre, nous pouvons déduire

qu'un polyèdre non convexe est toujours l'union d'un nombre fini de zones.

1.7 Matrice de Bornes et graphe de contraintes

Une *borne* est une paire $(c, \prec)$ où $c \in \mathbb{Q} \cup \{\infty, -\infty\}$ et $\prec \in \{<, \leq\}$. Nous définissons un ordre total sur les opérateurs '$<$' et '$\leq$', tel que '$<$' $<$ '$\leq$', et sur les bornes comme suit :

- $(c, \prec) = (c', \prec')$ ssi $c = c'$ et $\prec = \prec'$,
- $(c, \prec) < (c', \prec')$ ssi $c < c'$ ou $(c = c'$ et $\prec < \prec')$,
- $(c, \prec) \leq (c', \prec')$ ssi $(c, \prec) = (c' \prec')$ ou $(c, \prec) < (c', \prec')$.

La *somme* de deux bornes $(c, \prec)$ et $(c', \prec')$, notée $(c, \prec) + (c', \prec')$ est la borne $(c+c', min(\prec, \prec'))$. Le *complément* d'une borne $(c, \prec)$ est la borne $(c', \prec')$ telle que $(c, \prec) + (c', \prec') = (0, <)$. À titre d'exemple, le complément de $(3, <)$ est $(-3, \leq)$.

1.7.1 Matrices de bornes

Soit $X = \{x_1, ..., x_n\}$ un ensemble de variables sur $\mathbb{R}$. La formule F associée à une zone Z sur X est une conjonction de contraintes atomiques sur X. Si nous ajoutons une variable x_0 à l'ensemble X telle que $x_0 = 0$ (x_0 est parfois notée 0 aussi), chaque contrainte atomique qui apparaît dans F peut être écrite sous forme $x_i - x_j \prec_{i,j} c_{i,j}$, où $c_{i,j} \in \mathbb{Q} \cup \{\infty, -\infty\}$, $\prec_{i,j} \in \{<, \leq\}$ et $x_i, x_j \in X \cup \{x_0\}$. F peut être alors représentée par une matrice B d'ordre $n + 1$ telle que $b_{i,j}$ est la borne $(c_{i,j}, \prec_{i,j})$. B est appelée *matrice de bornes* (Dill, 1989) (*Difference Bound Matrix* (DBM)).

En général, une zone sur X peut être représentée par une infinité de formules. Cependant, il existe une unique formule en *forme canonique* pour chaque zone. L'existence de cette forme, calculable en $O(n^3)$, où $n = |X|$, simplifie considérable-

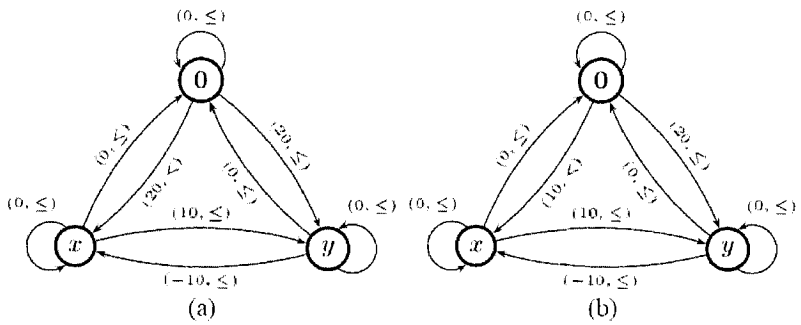


FIGURE 1.1 Graphe de contraintes : (a) sous forme non canonique, (b) sous forme canonique

ment le test d'égalité et d'inclusion entre zones, ainsi que le calcul de l'intersection. La complexité de ces opérations est $O(n^2)$.

1.7.2 Graphes de contraintes

Soient Z une zone sur $X = \{x_1, ..., x_n\}$ et F sa formule. Le calcul de la forme canonique de Z consiste à mettre chaque contrainte atomique de F dans sa forme la plus restreinte. Ceci revient à remplacer, dans chaque contrainte atomique $x_i - x_j \prec_{i,j} c_{i,j}$ de F, la constante $c_{i,j}$ par la constante la plus petite en valeur absolue, sans affecter le domaine de F. Pour résoudre ce problème, on associe à la formule F une interprétation sous forme de graphe, appelé *graphe de contraintes*. Le graphe de contraintes G associé à F est un graphe orienté et valué, où les variables de $X \cup \{x_0\}$ sont des sommets (la variable x_0 est représentée par le sommet 0) et les contraintes atomiques de F sont des arcs. À chaque contrainte atomique $x_i - x_j \prec_{i,j} c_{i,j}$, est associé un arc du sommet x_j au sommet x_i ayant comme poids la borne $(c_{i,j}, \prec_{i,j})$. Cette borne est interprétée comme la borne supérieure de la distance du sommet x_j au sommet x_i.

Soient F une formule et G le graphe de contraintes qui lui est associé :

- F est consistante ssi, G n'admet aucun cycle négatif,
- Si F est consistante, le poids du plus court chemin du noeud x_j au noeud x_i dans G est égal à $Sup(x_i - x_j, F)$ ($Sup(x_i - x_j, F)$ étant le supremum de $(x_i - x_j$ dans le domaine de F).

De ces résultats, on déduit que le calcul de la forme canonique de F revient à calculer le graphe des plus courts chemins sur G. L'algorithme de *Floyd-Warshall* (Floyd, 1962) est une référence dans ce contexte, avec une complexité de $O(n^3)$, où $n = |X|$.

Comme exemple d'illustration, soit Z la zone sur $X = \{x, y\}$ caractérisée par la formule $F = (0 \leq x < 20 \ \wedge \ 0 \leq y \leq 20 \ \wedge \ -10 \leq x - y \leq -10)$. La figure 1.1 (a) montre le graphe des contraintes associé à F. Le même graphe, après le calcul des plus courts chemins, est montré dans la figure 1.1 (b). La forme canonique de F est donc $(0 \leq x < 10 \ \wedge \ 0 \leq y \leq 20 \ \wedge \ -10 \leq x - y \leq -10)$. Sa matrice de bornes est :

$$\begin{array}{c} \\ o \\ x \\ y \end{array} \begin{array}{c} \begin{array}{ccc} o & x & y \end{array} \\ \begin{pmatrix} (0,\leq) & (0,\leq) & (0,\leq) \\ (10,<) & (0,\leq) & (-10,\leq) \\ (20,\leq) & (10,\leq) & (0,\leq) \end{pmatrix} \end{array}$$

1.8 Formalismes pour la spécification de propriétés

Les logiques temporelles sont parmi les formalismes mathématiques les plus utilisés pour spécifier des propriétés sur les comportements des systèmes. Ces propriétés peuvent être classées de la manière suivante :

- **Propriété d'atteignabilité** (*Reachability property*) : indique qu'un état du système peut être atteint.
- **Propriété de sûreté** (*Safety property*) : exprime que sous certaines conditions,

un évènement ne peut jamais se produire. Notons que la négation d'une propriété d'atteignabilité est une propriété de sûreté.

- **Propriété de vivacité** (*Liveness property*) : exprime que sous certaines conditions un évènement finira par se produire.
- **Propriété d'absence de blocage** (*Deadlock free property*) : exprime que le système ne se retrouvera jamais dans une situation où il ne pourra plus progresser.
- **Propriété d'équité** (*Fairness property*) : exprime que, sous certaines conditions, un évènement se produira (ou ne se produira pas) infiniment souvent.

Les logiques temporelles peuvent elles aussi être classées en deux catégories : *non temporisées* et *temporisées*. Les logiques non temporisées permettent d'exprimer des propriétés sur l'ordre d'occurrence des événements dans un système, sans quantifier le temps qui les sépare. Dans les logiques temporisées, le temps est explicitement quantifié. Les logiques temporelle non temporisées que nous considérons dans ce document sont la logique temporelle arborescente CTL^* (*Computation Tree Logic**) et ses sous classes LTL et CTL. Pour les logiques temporisées, nous considérons plus particulièrement la logique temporisée $TCTL$.

1.8.1 La logique temporelle CTL^* et ses sous classes

Soit $PV = \{\wp_1, \wp_2 ...\}$ un ensemble de variables propositionnelles (appelées aussi *propositions atomiques*). Les formules de la logique CTL^* sont données par la grammaire suivante :

$$\varphi_e := \wp \mid \neg\varphi_e \mid \varphi_e \wedge \varphi_e \mid \varphi_e \vee \varphi_e \mid \forall\varphi_c \mid \exists\varphi_c$$

$$\varphi_c := \varphi_e \mid \neg\varphi_c \mid \varphi_c \wedge \varphi_c \mid \varphi_c \vee \varphi_c \mid X\varphi_c \mid \varphi_c \, U \, \varphi_c$$

Dans la grammaire, $\wp \in PV$ est une variable propositionnelle, φ_e définit les formules qui expriment des propriétés sur des états, et φ_c définit des formules qui

expriment des propriétés sur des chemins d'exécution. Les opérateurs *modaux* $\forall$ et $\exists$ (notés aussi A et E) sont des quantificateur de chemins, U et X sont des opérateurs d'états. Les formules de la logique CTL^* sont interprétées sur les états et les chemins d'exécution d'un modèle $\mathcal{M} = (\mathcal{S}, \mathcal{V})$, où $\mathcal{S} = (S, \rightarrow, s_0)$ est un système de transitions et $\mathcal{V} : S \rightarrow 2^{PV}$ est une fonction de valuation d'états qui associe à chaque états l'ensemble des propositions atomiques qu'il satisfait.

Soient $s \in S$ un état de $\mathcal{S}$, $\pi(s)$ l'ensemble des chemins d'exécution à partir de s, et $\rho \in \pi(s)$ un chemin d'exécution à partir de s avec ρ^i le chemin suffixe de ρ qui commence à l'état i (les états étant numérotés dans ρ à partir de *un*). La sémantique formelle de CTL^* est donnée par la relation de satisfaction $\models$ définie comme suit : (l'expression $\mathcal{M}, x \models \varphi$ se lit : x satisfait la propriété φ dans le modèle $\mathcal{M}$)

- $\mathcal{M}, s \models \wp$ *ssi* $\wp \in \mathcal{V}(s)$,
- $\mathcal{M}, x \models \neg\varphi$ *ssi* $\mathcal{M}, x \not\models \varphi$, *pour* $x \in \{s, \rho\}$,
- $\mathcal{M}, x \models \varphi \vee \psi$ *ssi* $\mathcal{M}, x \models \varphi$ *ou* $\mathcal{M}, x \models \psi$, *pour* $x \in \{s, \rho\}$,
- $\mathcal{M}, x \models \varphi \wedge \psi$ *ssi* $\mathcal{M}, x \models \varphi$ *et* $\mathcal{M}, x \models \psi$, *pour* $x \in \{s, \rho\}$,
- $\mathcal{M}, s \models \forall\varphi$ *ssi* $\forall\rho \in \pi(s)$, $\mathcal{M}, \rho \models \varphi$,
- $\mathcal{M}, s \models \exists\varphi$ *ssi* $\exists\rho \in \pi(s)$, $\mathcal{M}, \rho \models \varphi$,
- $\mathcal{M}, \rho \models \varphi$ *ssi* $\mathcal{M}, \rho(1) \models \varphi$, pour une formule d'état φ,
- $\mathcal{M}, \rho \models X\varphi$ *ssi* $\mathcal{M}, \rho^2 \models \varphi$,
- $\mathcal{M}, \rho \models \psi U\varphi$ *ssi* $(\exists j \geq 1)$ $(\mathcal{M}, \rho^j \models \varphi$ *et* $(\forall 1 \leq i < j), \mathcal{M}, \rho^i \models \psi)$.

On dit que le modèle $\mathcal{M}$ satisfait la formule φ, écrit $\mathcal{M} \models \varphi$, ssi $\mathcal{M}, s_0 \models \varphi$.

Pour simplifier l'écriture des formules CTL^*, certaines abréviations sont utilisées :

- $\exists\Diamond\varphi = \exists(trueU\varphi)$,
- $\forall\Diamond\varphi = \forall(trueU\varphi)$,

- $\exists\Box\varphi = \neg\forall\Diamond\neg\varphi$,
- $\forall\Box\varphi = \neg\exists\Diamond\neg\varphi$.

Les sous classes de la logique CTL^* les plus connues sont :

- $ACTL^*$ (*Universal* CTL^*) : Les formules temporelles sont restreintes à celles où l'opérateur de négation $\neg$ n'apparaît que dans des sous formules sans opérateurs modaux $(\forall, \exists, U, X)$.
- CTL (*Computation Tree Logic*) : Les formules temporelles sont restreintes à des combinaisons booléennes positives (i.e., sans l'opérateur de négation $\neg$) de $\forall(\varphi U\psi)$, $\exists(\varphi U\psi)$, $AX\varphi$, et $EX\varphi$ seulement.
- $ACTL$ (*Universal* CTL) : Les formules temporelles sont restreintes à des combinaisons booléennes positives de $\forall(\varphi U\psi)$ et $AX\varphi$ seulement.
- $ECTL$ (*Existential* CTL) : Les formules temporelles sont restreintes à des combinaisons booléennes positives de $\exists(\varphi U\psi)$ et $EX\varphi$ seulement.
- LTL (*Linear Time Logic*) : Seules les formules de la forme $\forall\varphi$ sont autorisées, où φ ne contient pas les quantificateurs de chemin $\forall$ et $\exists$.

1.8.2 La logique temporelle temporisée $TCTL$

La logique temporelle $TCTL$ est une extension de la logique CTL sans l'opérateur X, où l'opérateur modal U est indexé par un intervalle de temps pour spécifier des restrictions temporelles sur les formules. La syntaxe de $TCTL$ est donnée par la grammaire suivante :

$$\varphi \ := \ \wp \mid \neg\varphi \mid \varphi \ \wedge \ \varphi \mid \varphi \ \vee \ \varphi \mid \forall(\varphi \ U_I \ \varphi) \mid \exists(\varphi \ U_I \ \varphi)$$

Dans la grammaire, l'indexe I est un intervalle non vide dans $\mathbb{R}^+$, à bornes dans $\mathbb{Q}^+ \cup \{\infty\}$.

Les formules de la logique $TCTL$ sont interprétées sur les états d'un modèle $\mathcal{M} = (\mathcal{S}, \mathcal{V})$, où $\mathcal{S} = (S, \rightarrow, s_0)$ est un système de transitions temporisé et $\mathcal{V} : S \rightarrow 2^{PV}$ est une fonction de valuation d'états. Soient $s \in S$ un état de $\mathcal{S}$, $\pi(s)$ l'ensemble des chemins d'exécution à partir de s et $\rho = s_1 \xrightarrow{\theta_1} s_1 + \theta_1 \xrightarrow{a_1} s_2 \xrightarrow{\theta_2} s_2 + \theta_2 \xrightarrow{a_2} .....$ un chemin d'exécution tel que $\rho \in \pi(s)$ (i.e., $s_1 = s$). Pour interpréter une formule $TCTL$ sur un chemin d'exécution, nous introduisons la notion de *chemin d'exécution dense*, d'un chemin d'exécution ρ, noté $\breve{\rho}$. $\breve{\rho} : \mathbb{R}^+ \rightarrow S$ est une application définie par : $\breve{\rho}(r) = s_i + \delta$ où $r = \sum_{j=1}^{i} \theta_j + \delta$, $i \geq 1$ et $0 \leq \delta < \theta_i$.

La sémantique formelle de $TCTL$ est donnée par la relation de satisfaction $\models$ définie comme suit :

- $\mathcal{M}, s \models \wp$ *ssi* $\wp \in \mathcal{V}(s)$,
- $\mathcal{M}, s \models \neg\wp$ *ssi* $\wp \notin \mathcal{V}(s)$,
- $\mathcal{M}, s \models \varphi \vee \psi$ *ssi* $\mathcal{M}, s \models \varphi$ *ou* $\mathcal{M}, s \models \psi$,
- $\mathcal{M}, s \models \varphi \wedge \psi$ *ssi* $\mathcal{M}, s \models \varphi$ *et* $\mathcal{M}, s \models \psi$,
- $\mathcal{M}, s \models \forall(\varphi U_I \psi)$ *ssi* $\forall \rho \in \pi(s), \exists r \in I, (\mathcal{M}, \breve{\rho}(r) \models \psi \wedge (\forall r' < r, \mathcal{M}, \breve{\rho}(r') \models \varphi))$,
- $\mathcal{M}, s \models \exists(\varphi U_I \psi)$ *ssi* $\exists \rho \in \pi(s), \exists r \in I, (\mathcal{M}, \breve{\rho}(r) \models \psi \wedge (\forall r' < r, \mathcal{M}, \breve{\rho}(r') \models \varphi))$.

Pour simplifier l'écriture des formules $TCTL$, des abréviations similaires à celles introduites pour la logique CTL^* sont utilisées, mais indexées par un intervalle temporel :

- $\exists\Diamond_I \varphi = \exists(true U_I \varphi)$,
- $\forall\Diamond_I \varphi = \forall(true U_I \varphi)$,
- $\exists\Box_I \varphi = \neg\forall\Diamond_I \neg\varphi$,
- $\forall\Box_I \varphi = \neg\exists\Diamond_I \neg\varphi$,
- $\varphi \rightsquigarrow_{I_r} \psi = \forall\Box(\varphi \Rightarrow \forall\Diamond_I \psi)$.

Lorsque l'intervalle I est omis, l'intervalle $[0, +\infty[$ est alors considéré.

CHAPITRE 2

LES RÉSEAUX DE PETRI TEMPORELS (MODÈLE TPN)

Dans ce chapitre nous donnons la définition des réseaux de Petri simple puis celle des réseaux de Petri temporels (modèle TPN) et leur sémantique. Nous introduisons par la suite la notion de modèle d'états abstraits et décrivons quelques approches de construction proposées dans la littérature.

2.1 Les réseaux de Petri simples

La théorie des réseaux de Petri offre un contexte général pour la modélisation des systèmes concurrents. Elle a été introduite par C.A. Petri en 1962, afin de modéliser la composition et la synchronisation d'automates évoluant en parallèle et pallier l'incapacité des automates à modéliser naturellement des comportements parallèles.

Définition 2.1.1 : ***Réseau de Petri simple***

Un réseau de Petri est un tuple $(P, T, Pre, Post, m_0)$*, où :*

- P *est un ensemble fini de places,*
- T *est un ensemble fini de transitions* $(P \cap T = \emptyset)$*,*
- Pre *et* $Post$ *sont les fonctions d'incidence avant et arrière :* $P \times T \rightarrow \mathbb{N}$*,*
- m_0 *est le marquage initial,* $m_0 : P \rightarrow \mathbb{N}$*.*

Intuitivement, un réseau de Petri est un graphe *biparti*[1] orienté avec deux types

[1]Un graphe biparti est un graphe dans lequel les sommets sont répartis en deux groupes, tel qu'aucune arête ne relie deux sommets d'un même groupe.

de sommets : les *places* et les *transitions*. D'une manière générale, les places représentent des conditions ou des ressources. L'état de vérité d'une condition ou la disponibilité d'une ressource est matérialisé dans une place par des jetons. Les transitions représentent des actions ou des événements qui peuvent changer l'état de vérité des conditions ou l'état de disponibilité des ressources.

Soient m un marquage et t une transition. t est dite *sensibilisée* pour m ssi tous les jetons requis pour franchir t sont présents dans m, i.e., $\forall p \in P, m(p) \geq Pre(p,t)$. L'ensemble de toutes les transitions sensibilisées dans m est dénoté par $En(m)$. Le franchissement d'une transition $t \in En(m)$ à partir du marquage m mène au marquage m', tel que $\forall p \in P, m'(p) = m(p) - Pre(p,t) + Post(p,t)$. Un marquage m est dit *atteignable* ou *accessible*, s'il existe une séquence de transitions $t_1, t_2, ..., t_n$ telle que m est atteint après le franchissement consécutif des transitions $t_1, t_2, ..., t_n$ à partir du marquage initial m_0. Un réseau de Petri est dit *borné* si l'ensemble de ses marquages accessibles est fini.

2.2 Les réseaux de Petri temporels (le modèle TPN)

Le modèle réseaux de Petri a été ensuite étendu pour prendre en compte des contraintes temporelles, afin de modéliser des systèmes temps réels. Les nombreuses extensions proposées peuvent être classées selon la forme des contraintes temporelles (intervalles (Merlin et Farber, 1976; Walter, 1983) ou constantes (Ramchandani, 1974)), ou selon les parties du réseau qui sont contraintes (places (Coolahan et Roussopoulos, 1983), transitions (Merlin et Farber, 1976; Ramchandani, 1974) ou arcs (Abdulla et Nylén, 2001; Hanisch, 1993; Walter, 1983)). Chaque extension associe aussi une interprétation spécifique aux contraintes temporelles. Lorsqu'une contrainte temporelle est associée à une transition, elle est considérée soit comme la durée de franchissement ou le temps de sensibilisation de la transition. Si elle

est associée à une place, elle spécifie généralement le temps de séjour d'un jeton dans la place avant qu'il ne devienne disponible (Coolahan et Roussopoulos, 1983). Une contrainte associée à un arc rentrant dans une transition indique l'intervalle de temps pendant lequel un jeton peut être consommé à travers l'arc (Hanisch, 1993). Lorsque la contrainte est associée à un arc sortant d'une transition, elle indique quand est ce qu'un jeton produit à travers cet arc deviendra disponible (Walter, 1983).

Les réseaux de Petri étendus au facteur temps peuvent être aussi classés selon les règles de franchissement qu'ils utilisent : *faible*, *au plutôt forte*, *au plus tard forte*. La règle de franchissement *faible* signifie que le temps qui s'écoule entre la sensibilisation d'une transition et son franchissement n'est pas déterminé (Tsai *et al.*, 1995). La règle de franchissement *au plutôt forte* impose qu'une transition doit être franchie dès que possible (i.e., elle est sensibilisée et les contraintes temporelles sont satisfaites) (Hanisch, 1993). La règle de franchissement *au plus tard forte* indique qu'une transition doit être franchie dans une période de temps spécifiée depuis sa sensibilisation, à moins qu'elle ne soit désensibilisée par un autre franchissement (Merlin et Farber, 1976).

L'extension des réseaux de Petri la plus connue, et la plus utilisée, est celle proposée par Merlin et Farbre en 1976 appelée *réseaux de Petri temporels* (Merlin et Farber, 1976). Ce modèle, qui a la puissance d'expression des *machines de Turing* (Ohta et Tsui, 2000), est un réseau de Petri simple où chaque transition est associée avec *intervalle de franchissement*. Les bornes de cet intervalle sont les temps d'attente *minimal* et *maximal* avant le franchissement de la transition (les durées de sensibilisation minimale et maximale). Ce franchissement est instantané selon la règle au plus tard forte. Le modèle réseaux de Petri temporels constitue un bon compromis entre la puissance de modélisation et la complexité d'analyse. Dans le reste de ce document, nous nous intéresserons plus particulièrement à ce modèle.

Soit $\mathbb{Q}^+_{[\,]}$ l'ensemble des intervalles non vides de $\mathbb{R}^+$ dont les bornes sont dans $\mathbb{Q}^+ \cup \{\infty\}$. Pour $I \in \mathbb{Q}^+_{[\,]}$, $\downarrow I$ et $\uparrow I$ dénotent respectivement la *borne inférieure* et la *borne supérieure* de I.

Définition 2.2.1 : *Réseau de Petri temporel*

*Un réseau de Petri temporel (*Time Petri Net *(TPN)) est un tuple* $(P, T, Pre, Post, m_0, Is)$, *où :*

- P *est un ensemble fini de places,*
- T *est un ensemble fini de transitions* ($P \cap T = \emptyset$),
- Pre *et* $Post$ *sont les fonctions d'incidence avant et arrière :* $P \times T \rightarrow \mathbb{N}$,
- m_0 *est le marquage initial,* $m_0 : P \rightarrow \mathbb{N}$,
- $Is : T \rightarrow \mathbb{Q}^+_{[\,]}$ *associe à chaque transition* t *un intervalle* $[\downarrow Is(t), \uparrow Is(t)]$ *appelé* intervalle statique de franchissement *de* t. $tmin(t) = \downarrow Is(t)$ *et* $tmax(t) = \uparrow Is(t)$ *sont respectivement les* délais de franchissement minimal *et* maximal *de* t.

A titre d'exemple, la figure 2.1 est la représentation graphique du réseau de Petri temporel défini par[2] :

- $P = \{p_0, p_1, p_2\}$,
- $T = \{t_0, t_1, t_2\}$,
- $Pre = (p_0, t_0) + (p_1, t_1) + (p_2, t_2)$,
- $Post = (p_0, t_0)$,
- $m_0 = p_0 + p_1 + p_2$,
- $Is = \{(t_0, [1, 2]), (t_1, [2, \infty]), (t_2, [2, \infty])\}$.

[2] Pre, $Post$ et m_0 sont des multi-ensembles représentés par leurs sommes formelles.

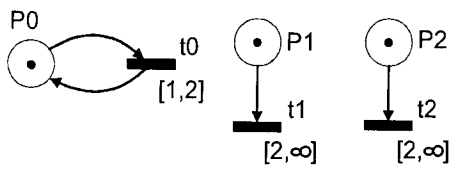


FIGURE 2.1 Un modèle TPN avec des intervalles de franchissement statiques non bornés

2.2.1 Etat du modèle TPN

On retrouve dans la littérature deux caractérisations distinctes de l'état du modèle TPN. La première, appelée *caractérisation intervalle* (Berthomieu et Menasche, 1983), définit l'état du modèle par un couple (m, Id) combinant un marquage m et une *fonction délais* Id. $Id : T \rightarrow \mathbb{Q}^{+}_{[\,]}$ associe à chaque transition sensibilisée dans m un *intervalle de franchissement*. Lorsqu'une transition t devient sensibilisée, son intervalle de franchissement est initialisé à son intervalle de franchissement statique $Is(t)$. Les bornes de cet intervalle décroissent avec le temps, jusqu'au franchissement ou la désensibilisation de la transition. t peut être franchie si la borne inférieure de son intervalle atteint la valeur 0. Mais elle doit être franchie, sans aucun autre délai supplémentaire, lorsque la borne supérieure de son intervalle atteint la valeur 0. Avec cette caractérisation, l'état initial du modèle TPN est le couple (m_0, Id_0), où m_0 est le marquage initial et Id_0 est tel que $Id_0(t) = Is(t), \forall t \in En(m_0)$.

L'état du modèle *TPN* évolue soit par progression du temps ou par franchissement de transitions. La notation $(m, Id) \xrightarrow{t_f} (m', Id')$ dénote le fait que l'état (m', Id') est immédiatement atteignable depuis l'état (m, I) par franchissement de la transition t_f. i.e. :

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

- $Id'(t) = Is(t)$, si t est nouvellement sensibilisée par le franchissement de t_f,
- $Id'(t) = Id(t)$, sinon.

La notation $(m, I) \xrightarrow{\theta} (m', I')$ dénote le fait que l'état (m', I') est atteignable depuis l'état (m, I) par progression de θ unités de temps. i.e. :

- $\exists \theta \in \mathbb{R}^+$, $(\bigwedge_{t \in En(m)} \theta \leq \uparrow Id(t)$,
- $m' = m$,
- $\forall t \in En(m')$, $I'(t) = [max(\downarrow Id(t) - \theta, 0), \uparrow Id(t) - \theta]$.

Dans la deuxième caractérisation, appelée *caractérisation horloge* (Penczek et Polrola, 2001;Penczek et Polrola, 2004;Yoneda et Ryuba, 1998), une *horloge* est associée à chaque transition pour mesurer le temps écoulé depuis sa sensibilisation. L'état du modèle est défini par un couple (m, V), combinant un marquage m et une valuation d'horloges V sur $En(m)$. La fonction V associe à chaque transition sensibilisée dans m la valeur de son horloge. Lorsqu'une transition t devient sensibilisée, son horloge est initialisée à la valeur 0. La valeur de cette horloge augmente avec le temps jusqu'au franchissement ou la désensibilisation de t. t peut être franchie si la valeur de son horloge est à l'intérieur de son intervalle statique de franchissement $Is(t)$. Elle doit être franchie immédiatement sans aucun délai supplémentaire, si son horloge a atteint $tmax(t)$. Avec cette caractérisation, l'état initial du modèle TPN est le couple (m_0, V_0) où m_0 est le marquage initial et V_0 est telle que $V_0(t) = 0, \forall t \in En(m_0)$. L'état du modèle TPN évolue soit par progression du temps ou par franchissement de transitions.

La notation $(m, V) \xrightarrow{t_f} (m', V')$ dénote le fait que l'état (m', V') est immédiatement atteignable depuis l'état (m, V) par franchissement de la transition t_f. i.e. :

- $t_f \in En(m)$,
- $tmin(t_f) \leq V(t_f) \leq tmax(t_f)$,
- $\forall p \in P, m'(p) = m(p) - Pre(p, t) + Post(p, t)$,

- $\forall t' \in En(m')$, $V'(t') = V(t')$ si t' n'est pas nouvellement sensibilisée dans m' (i.e., $\forall p \in P, Pre(p,t') + Pre(p,t_f) \leq m(p)$), $V'(t') = 0$ si t' est nouvellement sensibilisée dans m'.

La notation $(m,V) \xrightarrow{\theta} (m',V')$ dénote le fait que l'état (m',V') est atteignable depuis l'état (m,V) après une progression de θ unités de temps. i.e. :

- $\theta \in \mathbb{R}^+$,
- $(\bigwedge_{t \in En(m)}(V(t)+\theta) \leq tmax(t))$,
- $m' = m$,
- $\forall t' \in En(m'), V'(t') = V(t') + \theta$.

Les deux caractérisations, intervalle et horloge, de l'état du modèle TPN sont étroitement liées. Si (m,V) est un état horloge, l'état intervalle qui lui correspond est (m,Id), où : $\forall t \in En(m), Id(t) = [Max(0, tmin(t) - V(t)), tmax(t) - V(t)]$. Notons que si $tmax(t) = \infty$, alors $tmax(t) - V(t) = \infty$ pour tout $V(t)$. Ceci signifie que toutes les valeurs de l'horloge de t plus grandes ou égales à $tmin(t)$ correspondent au même intervalle de franchissement $[0, \infty]$. Par conséquent, un même état intervalle peut correspondre à une infinité d'états horloges qui ont évidemment tous le même comportement futur (i.e., ils sont *bisimilaires*). Cette remarque stipule que la caractérisation intervalle a un pouvoir d'abstraction plus grand que celui de la caractérisation horloge.

Initialement, le modèle est dans son état initial (état horloge / état intervalle). L'état du modèle évolue soit par progression du temps (les horloges augmentent / les délais diminuent) ou par franchissement de transitions. Le franchissement d'une transition est supposé instantané, mais peut mener à un autre marquage (les jetons requis disparaissent alors que ceux produits apparaissent). Nous considérerons dans ce qui suit la caractérisation horloge des états. Les définitions et les résultats présentés peuvent en général être adaptés à la caractérisation intervalle. Dans le cas contraire, des indications seront explicitement données.

2.2.2 Espace des états du modèle TPN

Le comportement du modèle TPN peut être représenté par un système de transitions temporisé. La définition suivante décrit ce système.

Définition 2.2.2 : ***Espace des états du modèle TPN***
L'espace des états du modèle TPN est défini par le système de transition temporisé $(S, \rightarrow, s_0)$ *sur l'ensemble* T *des transitions du modèle TPN, où :*
- $s_0 \in S$ *est l'état initial du modèle TPN,*
- $\rightarrow \subseteq (S \times (T \cup \mathbb{R}^+) \times S)$ *est la relation de transition définie par :*
  $(s, r, s') \in \rightarrow$ *ssi* $s \xrightarrow{r} s'$.

L'espace des états d'un modèle TPN définit sa *sémantique arborescente.* L'ensemble de tous les chemins d'exécution dans cet espace définit sa *sémantique linéaire.*

En raison de la densité du temps, le modèle *TPN* a en général un espace d'états infini et à branchement infini, même lorsqu'il est borné. Son analyse au moyen de techniques énumératives (model-checking) doit passer par des contractions (abstractions).

2.2.3 Espace des états concrets

La première abstraction de l'espace des états du modèle *TPN* consiste à abstraire les transitions temporelles. On obtient un graphe appelé *espace des états concrets*, où seuls les états atteignables par franchissement de transitions sont représentés (Penczek et Polrola, 2001; Penczek et Polrola, 2004).

Définition 2.2.3 : ***Espace des états concrets (Penczek et Polrola, 2001)***
Soit $(S, \rightarrow, s_0)$ *l'espace des états du modèle TPN. L'espace des états concrets du*

modèle TPN est le système de transition $(\Sigma, \rightsquigarrow, \sigma_0)$ *étiqueté sur* T *(l'ensemble des transitions du modèle TPN), où :*

- $\sigma_0 \in \Sigma$ *est l'état concret initial, tel que* $\sigma_0 = s_0$,
- $\rightsquigarrow \subseteq (\Sigma \times T \times \Sigma)$ *est la relation de transition définie par :*
  $\sigma \overset{t_f}{\rightsquigarrow} \sigma'$ *ssi* $\exists \sigma", \sigma \xrightarrow{\theta} \sigma" \wedge \sigma" \xrightarrow{t_f} \sigma'$ *(i.e.,* t_f *peut être franchie à partir de* σ *après une éventuelle progression du temps, menant à l'état* σ'*),*
- $\Sigma = \{\sigma | \sigma_0 \overset{*}{\rightsquigarrow} \sigma\}$.

L'espace des états concrets d'un modèle TPN définit sa *sémantique arborescente non temporisée.* L'ensemble de tous les chemins d'exécution de cet espace définit sa *sémantique linéaire non temporisée.*

En dépit de l'abstraction introduite, l'espace des états concrets reste généralement infini et à branchements infinis. D'autres abstractions sont donc nécessaires.

CHAPITRE 3

ABSTRACTION DU MODÈLE TPN

En raison de la densité du temps, un état de l'espace des états du modèle *TPN* peut avoir une infinité de successeurs. La vérification par model-checking ne peut donc pas se faire directement, et doit passer par des abstractions. L'abstraction consiste à construire un *modèle d'états abstraits* fini qui préserve les propriétés d'intérêt du modèle TPN (atteignabilité, linéaires, de branchement,...)(Penczek et Polrola, 2004). Le résultat ainsi obtenu permet de vérifier les propriétés préservées en utilisant les techniques classiques appropriées de vérification (Clarke *et al.*, 1999). Pour les propriétés d'atteignabilité par exemple, un algorithme de vérification effectue une exploration exhaustive des états du modèle abstrait à la recherche d'un état qui satisfait la propriété considérée. Pour d'autres types de propriétés (CTL, CTL^*), certains algorithmes de vérification utilisent une procédure d'étiquetage des états (Alur et Dill, 1990; Alur *et al.*, 1996), ou procèdent à un test d'équivalence de comportement entre le modèle abstrait et la spécification. On retrouve dans la littérature plusieurs techniques de construction de modèles abstraits pour le modèle TPN (Berthomieu et Menasche, 1983; Berthomieu et Vernadat, 2003; Bouchcneb et Mullins, 2003; Bucci et Vicario, 1995; Penczek et Polrola, 2004; E.Vicario, 2001; Yoneda et Ryuba, 1998) qui se distinguent généralement par la caractérisation des états adoptée (horloge ou intervalle), les conditions de finitude des modèles générés, et le type de propriétés qu'ils préservent. Avant de présenter quelques techniques de construction de modèles abstraits pour le modèle TPN, nous commencerons par donner la définition d'un modèle d'état abstrait proposée par Penczek et Polrola dans (Penczek et Polrola, 2001) et reprise dans d'autres travaux récents tels que (Penczek *et al.*, 2004) et (Berthomieu et Vernadat, 2003). Nous citerons aussi les

conditions proposées pour caractériser les propriétés préservées dans un modèle d'états abstraits.

3.1 Modèle d'états abstraits selon Penczek et Polrola (Penczek et Polrola, 2001)

Soient $\mathcal{N}$ un modèle TPN, $(\Sigma, \rightsquigarrow, \sigma_0)$ son espace d'états concrets et $\equiv$ une relation d'équivalence sur Σ qui respecte[1] les marquages. Un modèle d'états abstraits de $\mathcal{N}$ est une structure $(C, \twoheadrightarrow, \alpha_0)$, où :

- $C = \mathcal{P}_{\equiv}$,
- $\alpha_0 \in C$ est l'état abstrait initial, tel que $\sigma_0 \in \alpha_0$,
- $\twoheadrightarrow \subseteq (C \times T \times C)$ est la relation de transition abstraite qui doit au moins satisfaire la condition EE suivante :
  $EE : (\alpha \overset{t}{\twoheadrightarrow} \alpha') \Leftrightarrow (\exists \sigma \in \alpha, \exists \sigma' \in \alpha', \sigma \overset{t}{\rightsquigarrow} \sigma')$.

La condition EE garantit que des états abstraits sont reliés entre eux ssi ils contiennent des états reliés.

La relation $\twoheadrightarrow$ peut aussi satisfaire d'autres conditions supplémentaires telles que :

- $EA : (\alpha \overset{t}{\twoheadrightarrow} \alpha') \Leftrightarrow (\forall \sigma' \in \alpha', \exists \sigma \in \alpha, \sigma \overset{t}{\rightsquigarrow} \sigma')$,
- $AE : (\alpha \overset{t}{\twoheadrightarrow} \alpha') \Leftrightarrow (\forall \sigma \in \alpha, \exists \sigma' \in \alpha', \sigma \overset{t}{\rightsquigarrow} \sigma')$,
- $U : (\alpha \overset{t}{\twoheadrightarrow} \alpha') \Leftrightarrow (\forall \sigma \in noyau(\alpha), \exists \sigma' \in noyau(\alpha'), \sigma \overset{t}{\rightsquigarrow} \sigma')$, où $noyau(\alpha) \subseteq \alpha$, $\forall \alpha \in C$, et $\sigma_0 \in \alpha_0$.

La condition EA restreint la relation de transition abstraite $\twoheadrightarrow$ aux paires d'états abstraits (α, α'), telles que tous les états concrets de α' sont accessibles à partir des états concrets de α par la même transition. La condition AE restreint la relation de transition abstraite aux paires d'états abstraits (α, α'), telles que tous les états concrets de α ont des successeurs dans α' par la même transition. La condition U

[1] Si $\sigma_1 \equiv \sigma_2$ alors σ_1 et σ_2 ont un même marquage

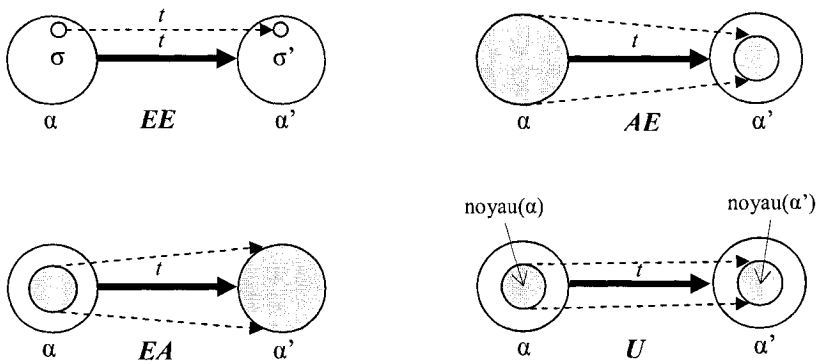


FIGURE 3.1 Conditions EE, EA, AE et U

correspond à la condition AE mais restreinte à une partie de chaque état abstrait, appelée son *noyau*. La figure 3.1 est une illustration graphique des quatre conditions. Les conditions EE, AE, EA et U sont aussi utilisées pour caractériser des états abstraits. Un état abstrait α satisfait une condition donnée, ssi la condition est satisfaite pour chaque transition $\alpha \twoheadrightarrow \alpha'$. Un espace d'états abstrait est dit *atomique* s'il satisfait la condition AE. Un état abstrait α est *atomique pour la transition* $\alpha \twoheadrightarrow \alpha'$, si la condition AE est satisfaite pour cette transition. Il est dit *atomique*, s'il est atomique pour chacune de ses transitions sortantes.

Le prochain théorème, démontré dans (Penczek et Polrola, 2001), établit une relation entre les conditions AE et EA, et les propriétés satisfaites par un modèle abstrait.

Théorème 3.1.1 *Soit $\mathcal{S} = (C, \twoheadrightarrow, \alpha_0)$ un modèle d'états abstraits d'un modèle TPN $\mathcal{N}$.*

- *Si $\mathcal{S}$ satisfait AE alors il préserve les propriétés CTL^* de $\mathcal{N}$,*
- *Si $\mathcal{S}$ satisfait EA alors il préserve les propriétés LTL de $\mathcal{N}$,*
- *Si $\mathcal{S}$ satisfait U alors il préserve les propriétés $ACTL^*$ de $\mathcal{N}$.*

Les conditions AE, EA et U apparaissent dans le théorème 3.1.1 comme des conditions suffisantes seulement. Un modèle d'états abstrait peut par exemple préserver les propriétés de branchement alors qu'il ne satisfait pas la condition AE. La même chose est vraie pour les autres conditions.

Malgré sa consistance, la définition d'un modèle d'états abstraits, proposée par Penczek et Polrola, impose des conditions trop fortes pour caractériser un espace d'états abstraits. Dans le chapitre 4, nous proposons une définition alternative, où la condition EE est relaxée et où les états abstraits ne sont plus contraints à être une partition de l'espace des états concrets du modèle TPN, mais seulement une couverture. Les conditions AE, EA et U sont elles aussi relaxées.

3.2 Opérations sur les états abstraits

Soit α un état abstrait. Nous définissons les opérations suivantes sur α :
- $succ(\alpha, t) \stackrel{def}{=} \{\sigma \mid \exists \sigma' \in \alpha,\ \sigma' \stackrel{t}{\rightsquigarrow} \sigma\}$,
- $pred(\alpha, t) \stackrel{def}{=} \{\sigma \mid \exists \sigma' \in \alpha,\ \sigma \stackrel{t}{\rightsquigarrow} \sigma'\}$,
- $pred(\alpha, t, \alpha') \stackrel{def}{=} pred(\alpha, t) \cap \alpha'$.

$succ(\alpha, t)$ est l'ensemble de tous les états concrets atteignables depuis α après le franchissement de la transition t. $pred(\alpha, t)$ est l'ensemble de tous les états concrets qui peuvent mener à des états de α après franchissement de la transition t. $pred(\alpha, t, \alpha')$ est l'ensemble de tous les états concrets de α' qui peuvent mener à des états dans α après franchissement de la transition t.

Similairement à l'espace des états du modèle TPN, un chemin d'exécution dans un modèle d'états abstraits du modèle TPN qui commence à partir de l'état abstrait α, est une séquence maximale $\zeta = \alpha_1 \stackrel{t_1}{\twoheadrightarrow} \alpha_2 \stackrel{t_2}{\twoheadrightarrow} \alpha^3 \stackrel{t_3}{\twoheadrightarrow} .....$, telle que $\alpha_1 = \alpha_0$. On dénote aussi par $\pi(\alpha)$ l'ensemble de tous les chemins d'exécution qui commencent à partir de α.

3.3 Quelques modèles d'états abstraits proposés dans la littérature

Plusieurs abstractions ont été proposées dans la littérature pour les réseaux de Petri temporels. Nous présentons, dans ce qui suit, les plus connues.

3.3.1 Graphe des classes d'états linéaire (Berthomieu et Menasche 1982)

Pour représenter l'espace d'états du modèle TPN d'une manière finie, Berthomieu et Menasche (1982) ont proposé d'abstraire le temps et de regrouper des ensembles d'états dans des *classes d'états*. Une classe d'états est une représentation symbolique d'un ensemble infini d'états qui partagent le même marquage. Dans l'abstraction qui en résulte, appelée *graphe de classes d'états linéaires* (*Linear State Class Graph* (LSCG)) (Berthomieu et Diaz, 1991; Berthomieu et Menasche, 1983; Berthomieu et Vernadat, 2003; Berthomieu et Menasche, 1982), tous les états atteignables depuis l'état initial par le franchissement de la même séquence de transitions sont agglomérés dans une classe d'états.

Soit $\omega = t_0, t_1, t_2, ...t_n$ une séquence de transitions franchissables depuis l'état initial du modèle TPN. La classe d'états qui correspond à ω (i.e., $\{\sigma \in \Sigma | \exists \sigma_1, ..., \sigma_n \in \Sigma, \sigma_0 \overset{t_0}{\rightsquigarrow} \sigma_1 \overset{t_1}{\rightsquigarrow} \sigma_2 ... \sigma_n \overset{t_n}{\rightsquigarrow} \sigma\}$) est représentée par le couple (m, F), où m est le marquage commun des états agglomérés dans la classe d'états, et F est une formule qui caractérise l'union de tous les domaines de franchissement de ces états. Chaque transition sensibilisée dans m est représentée dans F par une variable qui porte le même nom. La classe d'états initiale (m_0, F_0) coïncide avec l'état initial (i.e., m_0 est le marquage initial, et $F_0 = (\bigwedge_{t \in En(m_0)} tmin(t) \leq t \leq tmax(t))$). Soit $\alpha = (m, F)$ une classe d'états et t_f une transition. La classe α possède un successeur par t_f, dénoté $succ(\alpha, t_f)$, ssi t_f est sensibilisée dans m et peut être franchie avant toute

autre transition sensibilisée. Si tel est le cas, t_f est dite *franchissable* à partir de α. L'algorithme 1 montre comment effectuer ce test.

Algorithme 1 $isFirable(\alpha = (m, F), t_f)$

1: **si** $t_f \notin En(m)$ **alors**
2: Retourner faux
3: **fin si**
4: **Soit** $F' = F \wedge (\bigwedge_{t \in En(m) - \{t_f\}} t_f \leq t)$
5: **si** F' est consistante **alors**
6: Retourner vrai
7: **sinon**
8: Retourner faux
9: **fin si**

Les étapes 1-3 vérifient si t_f est sensibilisée dans m. L'étape 4 calcule la formule qui caractérise la partie du domaine de franchissement de α où t_f peut être franchie avant toute autre transition sensibilisée. L'étape 5 vérifie si cette partie du domaine est vide. Si t_f est franchissable à partir de α, $\alpha' = succ(\alpha, t_f)$ est calculée selon l'algorithme 2.

Algorithme 2 $succ(\alpha = (m, F), t_f)$

1: **Soit** $m'(p) = m(p) - Pre(p, t_f) + Post(p, t_f), \forall p \in P$
2: **Soit** $F' = F \wedge (\bigwedge_{t \in En(m) - \{t_f\}} t_f \leq t)$
3: Remplacer dans F' chaque variable $t \neq t_f$ par $t + t_f$
4: Éliminer t_f par substitution dans F', ainsi que toutes les variables associées aux transitions en conflit avec t_f dans m
5: **pour chaque** $t \in New(m', t_f)$ **faire**
6: Ajouter à F' la contrainte $tmin(t) \leq t \leq tmax(t)$
7: **fin pour**
8: Retourner $\alpha' = (m', F')$

Dans l'algorithme 2, l'étape 1 calcule le marquage après franchissement de t_f. L'étape 2 est similaire à l'étape 4 de l'algorithme 1. Les étapes 3 et 4 changent l'origine du temps de la classe d'état pour coïncider avec le moment où t_f est franchie. Les étapes 5-7 ajoutent les contraintes qui correspondent aux transitions nouvellement sensibilisées.

Les classes d'états sont considérées modulo une relation d'équivalence $\approx$, telle que deux classes d'états sont équivalentes ssi elles ont le même marquage et leurs domaines sont identiques (i.e., leurs formules sont équivalentes). Pour comparer deux classes d'états, chacune est mise sous forme canonique. Elles sont égales si elles ont des formes canoniques identiques (Berthomieu et Menasche, 1982). Formellement, le LSCG peut être défini comme suit :

Définition 3.3.1 : ***LSCG***

Soit $(\Sigma, \rightsquigarrow, \sigma_0)$ *l'espace des états concrets d'un modèle* $\mathcal{N}$. *Le LSCG de* $\mathcal{N}$ *est défini par la structure* $(C, \twoheadrightarrow, \alpha_0)$, *où :*

- $\alpha_0 = \{\sigma_0\} = (m_0, F_0)$ *est la classe d'états initiale,*
- $\alpha \overset{t}{\twoheadrightarrow} \alpha'$ *ssi* $\alpha' = succ(\alpha, t)$,
- $C = \{\alpha | \alpha_0 \overset{*}{\twoheadrightarrow} \alpha\}$, *où* $\overset{*}{\twoheadrightarrow}$ *est la fermeture réflexive et transitive de* $\twoheadrightarrow$.

Un algorithme de construction du LSCG est facilement déductible de sa définition. Dans (Berthomieu et Menasche, 1982) les auteurs montrent que le LSCG est fini pour tous les modèles TPN bornés. Ils montrent aussi que le LSCG préserve les propriétés d'atteignabilité et les chemins d'exécution du modèle TPN (i.e., ses propriétés linéaires) (Berthomieu et Menasche, 1982).

Limitations : Les limitations de cette technique peuvent être résumées dans les points suivants :

- La caractérisation des classes d'états utilisée ne permet pas d'identifier un à un les états agglomérés dans une même classe. De ce fait, les classes d'états ne peuvent pas être directement raffinées (partitionnées) pour restaurer les propriétés de branchement (CTL, CTL^*) (Berthomieu et Vernadat, 2003; Hadjidj et Boucheneb, 2005b; Boucheneb et Hadjidj, 2004; Boucheneb et Hadjidj, 2005; Hadjidj et Boucheneb, 2005a; Boucheneb et Hadjidj, 2006).
- Comme l'abstraction est appliquée sur l'espace des états concrets qui abstrait

les états atteignables par progression du temps, le LSCG n'est pas directement adapté pour vérifier des propriétés temporisées[2] (Hadjidj et Boucheneb, 2006b).
- La manière dont les états abstraits sont agglomérés dans un LSCG contribue à générer des LSCGs excessivement grands pour des modèles TPN relativement petits (voir tableau 7.4 dans la section 7 pour des exemples).

3.3.2 Graphe d'atteignablilité (Boucheneb et Berthelot 1993)

Dans (Boucheneb et Berthelot, 1993), les auteurs proposent une approche de construction d'un *graphe d'atteignabilité* pour le modèle TPN basé sur une notion de point de franchissement de transitions (i.e., l'instant où une transition est franchie). Les points de franchissement sont numérotés dans une séquence de franchissement à partir de l'état initial. La classe d'états générée par le $(n-1)$-ème franchissement est caractérisée par un triplet (m, TS, TE), où m est le marquage atteint. $TS : \mathbb{N} \rightarrow En(m)$ donne l'ordre de sensibilisation des transitions dans la classe, alors que la fonction $TE : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{Q}^+ \cup \{\infty\}$ donne pour chaque paire (i, j) le temps maximal ou minimal écoulé entre le i-ème et le j-ème point de franchissement au point $n-1$. La technique de construction proposée génère des graphes finis ssi le modèle TPN et borné.

Limitations : Cette technique a les mêmes limitations que la technique de construction du LSCG (voir section 3.3.1). Par contre, elle produit des abstractions plus compactes.

[2]Des propriétés où le temps est explicitement quantifié (comme dans $TCTL$ par exemple).

3.3.3 Graphe des régions géométriques (Yoneda et Reuba 1998)

Yoneda et Ryuba (1998) proposent une approche de construction d'un graphe des classes d'états dans l'objectif de le raffiner pour générer un *graphe de classes d'états atomiques* (i.e., qui préserve les propriétés CTL^*). Leur technique se limite aux modèles TPN 1-sauf[3], avec des intervalles de franchissement statiques bornés. Dans le cas de modèles TPN, avec intervalles de franchissement non bornés, l'approche peut générer des graphes infinis. La construction se base sur la caractérisation *horloge* des états. Chaque classe d'états est un triplet (m, F, ρ), où F est une formule (conjonction de contraintes atomiques), m est le marquage obtenu par franchissement de la séquence de transitions ρ, conformément aux contraintes temporelles spécifiées par F. Les variables qui apparaissent dans F sont associées aux transitions de la séquence ρ, à raison d'une variable par occurrence. Chaque variable compte le temps absolu écoulé depuis l'instant zéro, jusqu'au moment où la transition est franchie. Dans ρ, t_i^j dénote la j-éme occurrence de la transition t_i. F décrit donc l'histoire des états agglomérés dans une classe.

La classe initiale est $\alpha_0 = (m_0, \emptyset, \epsilon)$, où ϵ représente la séquence vide. Le franchissement de $t_i \in En(m)$ depuis la classe $\alpha = (m, F, \rho)$ est décrit en terme des parents des transitions sensibilisées dans m. Pour uniformiser le concept "parent", une transition fictive est ajoutée, et sert de parent aux transitions sensibilisées dans le marquage initial. Si t_i apparaît $k-1$ fois dans la séquence ρ, elle est franchissable depuis la classe α ssi la formule $(F \wedge \{parent(t_i^k, \alpha) + tmin(t_i) \leq parent(t_j^l, \alpha) + tmax(t_j)$ où $t_j \in En(m)$ et t_j apparaît $l-1$ fois dans $\rho)\}$) est consistante. Dans la formule, $parent(t, \alpha)$ dénote la variable qui correspond à la plus récente apparition du parent de t dans ρ. Le test revient donc à vérifier que t_i peut être franchie plus tôt que n'importe quelle autre transition sensibilisée. Le franchissement de

[3]Dans tous les marquages accessibles, le marquage de chaque places ne dépasse pas un jeton.

la transition t_i mène à la classe $\alpha' = (m', F', \rho')$, où $m' = m[t_i >$, $\rho' = \rho t_i$ et $F' = F \wedge min(t_i) \leq t_i^k - parent(t_i^k, \alpha) \leq max(t_i)$, ce qui revient à ajouter les contraintes qui doivent être satisfaites pour assurer le franchissement de t_i dans α. Deux classes sont équivalentes si elles ont le même marquage, les transitions sensibilisées ont les mêmes parents, et ces parents peuvent être franchis aux mêmes instants absolus. Toutes les classes construites selon cette approche satisfont la condition EA. Le graphe des régions géométriques préserve donc les propriétés linéaires.

Limitations : Cette technique se limite aux modèles TPN avec des intervalles de franchissement statiques bornés. Pour des modèles TPN avec des intervalles de franchissement non bornés, la technique peut générer des graphes infinis, même si le modèle est borné. Malgré que les graphes des régions géométriques peuvent être raffinés pour générer des modèles d'états abstraits qui préservent les propriétés de branchement, ils sont généralement beaucoup plus larges et plus lents à calculer que les LSCGs, principalement à cause de la définition complexe des états abstraits utilisée. Au même titre que les LSCGs, les graphes des régions géométriques ne sont pas directement adaptés pour vérifier des propriétés temporisées.

3.3.4 Graphe des classes d'états atomiques (Yoneda et Ryuba 1998)

Pour construire un graphe des classes d'états atomiques, Yoneda et Ryuba (1998) proposent de *raffiner*[4] les classes du graphe des régions géométriques jusqu'à ce qu'elles deviennent toutes atomiques. Si une classe $\alpha = (m, F, \rho)$ n'est pas atomique, alors il existe une contrainte f telle que les formules $F \wedge f$ et $F \wedge \neg f$ sont toutes les deux consistantes, et que la satisfaction de f est nécessaire pour que les états concrets de α aient des successeurs. La classe α est alors divisée en deux sous

[4]Le mot *raffiner* est utilisé dans se contexte pour signifier *partitionner judicieusement.*

classes $\alpha_1 = (m, F \wedge f, \rho)$ et $\alpha_2 = (m, F \wedge \neg f, \rho)$. Leurs descendants respectifs sont obtenus des descendants de α en ajoutant respectivement f et $\neg f$ à leurs formules. Le résultat final est un espace de classes d'états qui satisfait les conditions AE et EA, et préserve les propriétés CTL^* du modèle TPN.

Limitations : En plus des limitations du graphe des régions géométriques (Yoneda et Ryuba, 1998) (voir section 3.3.3), la technique de raffinement utilisée pour générer un graphe des classes d'états atomiques restaure, en plus de la condition AE, la condition EA. La condition EA n'est cependant pas nécessaire. Sa restauration contribue par ailleurs à accentuer le problème d'explosion d'états.

3.3.5 Graphe des classes d'états pseudo-atomiques (Penczek et Polrola 2001)

Dans (Penczek et Polrola, 2001), la construction du graphe des classes d'états atomiques (Yoneda et Ryuba, 1998) a été modifiée pour générer un *graphe des classes d'états pseudo-atomiques* qui préserve les propriétés $ACTL^*$ (la partie universelle de CTL^*). Au lieu de préserver la condition AE, ce graphe préserve la condition U qui est plus faible. Sa construction utilise la même technique de raffinement que celle utilisée pour construire le graphe des classes d'états atomiques, à la différence, qu'au lieu de partitionner les classes non atomiques, seuls leurs noyaux sont raffinés.

Limitations : Cette technique souffre des mêmes limitations que la technique de génération du graphe des classes d'états atomiques de Yoneda et Ryuba (1998) (voir section 3.3.4).

3.3.6 Graphe des classes d'états forts (Berthomieu et Vernadat 2003)

Dans (Berthomieu et Vernadat, 2003), les auteurs présentent une approche pour construire un graphe de classes d'états qui préserve les propriétés linéaires du modèle TPN, appelé *graphe des classe d'états forts* (*Strong State Class Graph* (SSCG)). L'intérêt de la construction de ce graphe n'est pas de vérifier des propriétés linéaires, mais plutôt de le raffiner en un graphe atomique qui préserve les propriétés de branchement (CTL^*). La technique génère un graphe fini pour tout modèle TPN borné. Dans un certain sens, le SSCG est simplement le LSCG où les classes d'états sont caractérisées de telle sorte que le raffinement devienne possible. Chaque classe d'états du SSCG est définie par un couple (m, F), où F est une formule définie par un ensemble d'inégalités sur des variables qui correspondent aux transitions sensibilisées dans m. À chaque transition t_i est associée une variable de même nom qui donne le temps écoulé depuis sa dernière sensibilisation. La classe initiale est définie par $\alpha_0 = (m_0, \{t_i = 0, \forall t_i \in En(m_0)\})$. Le franchissement d'une transition t_f depuis une classe, ainsi que l'ensemble des inéquations de la classe successeur, sont définis en terme du temps écoulé depuis que les transitions sont sensibilisées, en utilisant des variables supplémentaires temporaires qui dénotent les temps de franchissement possibles de t_f. Deux classes sont équivalentes si elles ont les mêmes marquages, et leurs formules ont des domaines identiques. Pour forcer la terminaison de la construction, dans le cas d'un modèle TPN borné avec des intervalles de franchissement non bornées, les auteurs définissent une opération sur les classes d'états, appelée *relaxation*, afin de les normaliser. Toutes les classes du SSCG satisfont la condition EA.

Limitations : Le SSCG a les mêmes limitations du LSCG (voir section 3.3.1), cependant, il est possible de le raffiner pour restaurer les propriétés de branchements. En terme de taille, le SSCG est généralement plus grand que le LSCG, d'une part à cause de la nouvelle caractérisation des classes d'états utilisée, mais aussi à cause

de l'opération de relaxation qui provoque une assez importante fragmentation des classes. Par conséquent le LSCG est plus approprié que le SSCG pour vérifier des propriétés linéaires.

3.3.7 Graphe des classes d'états atomiques (Berthomieu et Vernadat 2003)

Pour obtenir un modèle abstrait qui préserve les propriétés CTL^*, dans (Berthomieu et Vernadat, 2003) les auteurs proposent de raffiner le SSCG en vue de restaurer la condition AE. Les classes d'états sont partitionnées de proche en proche jusqu'à ce qu'elles deviennent toutes atomiques. Le graphe qui en résulte est appelé *Graphe des classes d'états atomiques* (*Atomic State Class Graph* (ASCG)). À la différence de l'approche proposée dans (Yoneda et Ryuba, 1998), les auteurs ne restaurent pas la condition EA, ce qui leur permet de construire des modèles plus petits, plus rapidement.

Limitations : Comme la construction du ASCG doit passer par celle du SSCG, cette technique hérite limitations du SSCG (voir section 3.3.6). De plus, en raison de la redondance excessive des états dans le SSCG, la convergence de la procédure de raffinement est considérablement ralentie.

CHAPITRE 4

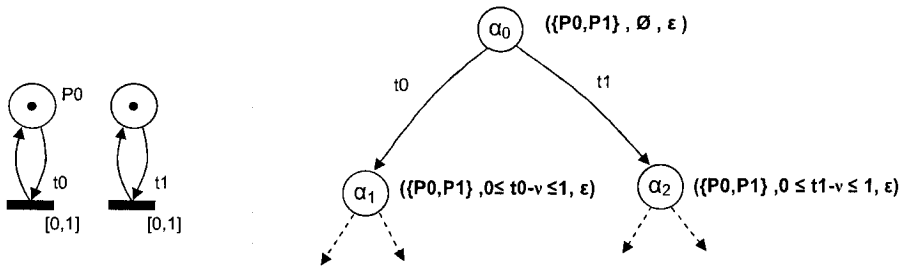
NOTRE CARACTÉRISATION DU MODÈLE D'ÉTATS ABSTRAITS

Dans ce chapitre nous établissons un ensemble de conditions pour caractériser un modèle d'états abstraits et les propriétés qu'il préserve. Nous montrons aussi que nos conditions sont beaucoup plus faibles et plus correctes que celles proposées dans (Penczek et Polrola, 2001).

4.1 Notion d'espace d'états abstraits dans la littérature

Dans (Penczek et Polrola, 2001), les auteurs ont introduit un ensemble de conditions pour caractériser un espace d'états abstraits et les propriétés qu'il préserve. Dans leur proposition, la condition EE sert à caractériser un espace d'états abstraits d'une manière générale. EA, AE et U, sont des conditions suffisantes pour préserver respectivement les propriétés LTL, CTL^* et $ACTL^*$ (voir section 3.1 pour plus de détails). La condition EE est introduite dans le contexte d'une relation équivalence $\equiv$ sur l'espace des états concrets d'un modèle TPN $\mathcal{N}$. La relation $\equiv$ doit respecter les marquages, i.e., si les états concrets (m, V) et (m', V') sont tels que $(m, V) \equiv (m', V')$, alors $m = m'$ (la caractérisation d'états considérée étant celle des horloges).

D'après (Penczek et Polrola, 2001), le graphe des régions géométriques de Yoneda et Ryuba (voir section 3.3.3 pour la définition) introduit dans (Yoneda et Ryuba, 1998) satisfait la condition EE . Cependant, cette affirmation est contredite par le modèle TPN de la figure 4.1(a) qui fournit un contre exemple. Dans la figure 4.1(b), les états abstraits α_0, α_1 et α_2 contiennent tous l'état initial σ_0, tout en étant distincts les uns des autres. Cette observation indique clairement qu'on ne se retrouve pas



a) TPN

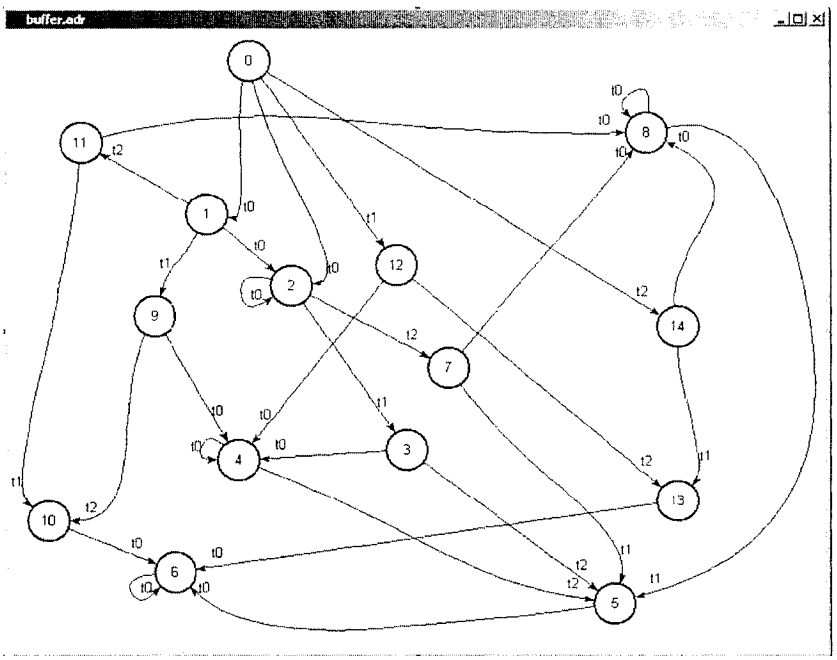
b) Partie initiale de l'espace des régions d'états géométriques

FIGURE 4.1 TPN dont l'espace des états abstraits décrit dans (Penczek et Polrola, 2001) ne vérifie pas EE

dans le contexte d'une relation d'équivalence[1]. De plus, $\sigma_0 \overset{t_1}{\rightsquigarrow} \sigma_0$ est une transition valide dans l'espace des états concrets, alors que $\alpha_1 \overset{t_1}{\twoheadrightarrow} \alpha_0$ ne l'est pas dans le graphe des régions géométriques. La condition EE n'est donc pas satisfaite.

La condition EE a été reprise dans (Berthomieu et Vernadat, 2003), où les auteurs n'exigent plus que les états abstraits soient des classes d'équivalences, mais plutôt, qu'ils constituent une couverture[2] de l'espace des états concrets. Malgré cette correction, la condition EE n'est toujours pas satisfaite par les deux modèles abstraits proposés dans (Berthomieu et Vernadat, 2003), à savoir, le *SSCG* et le *ASCG* (voir section 3.3.6 et section 3.3.7 pour les définitions). Un contre exemple peut facilement être déduit de la figure 4.2 (page 45) qui montre le SSCG (voir section 3.3.6 pour une description) du modèle TPN de la figure 2.1, construit avec l'outil TINA (Berthomieu *et al.*, 2004) qui implémente l'approche proposée dans (Berthomieu et Vernadat, 2003). La boucle sur la classe *class 6* indique que l'unique état de cette classe transite par t_0 vers lui même. Même si cet état appartient aussi à la classe *class 5*, il n y a pas de transition *class* 6 $\overset{t_0}{\twoheadrightarrow}$ *class* 5, ce qui contredit la condition EE.

[1]Les classes d'équivalences induites par une relation d'équivalence sont deux à deux disjointes.
[2]Une couverture d'un ensemble Σ est une collection d'ensembles dont l'union contient Σ.



class 0
P0 P1 P2
0 <= t0 <= 0
0 <= t1 <= 0
0 <= t2 <= 0

class 1
P0 P1 P2
0 <= t0 <= 0
1 <= t1 < 2
1 <= t2 < 2
t1 - t2 <= 0
t2 - t1 <= 0

class 2
P0 P1 P2
0 <= t0 <= 0
2 <= t1
2 <= t2

class 3
P0 P2
0 <= t0 <= 2
2 <= t2

class 4
P0 P2
0 <= t0 <= 0
2 <= t2

class 9
P0 P2
0 < t0 <= 2
2 <= t2

class 12
P0 P2
2 <= t0 <= 2
2 <= t2

class 5
P0
0 <= t0 <= 2

class 6
P0
0 <= t0 <= 0

class 10
P0
0 < t0 <= 2

class 13
P0
2 <= t0 <= 2

class 7
P0 P1
0 <= t0 <= 2
2 <= t1

class 8
P0 P1
0 <= t0 <= 0
2 <= t1

class 11
P0 P1
0 < t0 <= 2
2 <= t1

class 14
P0 P1
2 <= t0 <= 2
2 <= t1

FIGURE 4.2 SSCG du modèle TPN de la figure2.1

Dans (Penczek *et al.*, 2004), la condition apparaît sous la forme : $EE : (\alpha \overset{t_f}{\twoheadrightarrow} \alpha') \Rightarrow (\exists \sigma \in \alpha, \exists \sigma' \in \alpha', \sigma \overset{t_f}{\rightsquigarrow} \sigma')$. Sous cette forme, la condition est incomplète. En effet, elle implique qu'un espace d'états abstraits, après suppression de certaines

transitions entre ses états abstraits, reste toujours un espace d'états abstraits. Par conséquent, elle permet aussi d'affirmer que l'état abstrait initial constitue, à lui seul, un espace d'états abstraits valide, ce qui est évidemment faux.

Notons que la validité des espaces d'états abstraits proposés dans (Yoneda et Ryuba, 1998) et (Penczek et Polrola, 2001) n'est cependant pas remise en cause. La condition EE, telle qu'elle est introduite dans (Penczek et Polrola, 2001), est trop forte pour caractériser correctement un espace d'états abstraits. Les corrections qui apparaissent dans (Berthomieu et Vernadat, 2003;Penczek *et al.*, 2004) sont par contre plutôt incomplètes.

4.2 Notre définition d'espace d'états abstraits

Nous proposons dans ce qui suit de redéfinir la notion de modèle d'états abstraits à base d'une condition moins forte que la condition EE, notée EE_r.

Définition 4.2.1 : ***Espace d'états abstraits***

Soit $(\Sigma, \rightsquigarrow, \sigma_0)$ *l'espace des états concrets d'un modèle TPN* $\mathcal{N}$. *Un espace d'états abstraits de* $\mathcal{N}$ *est un tuple* $(C, \twoheadrightarrow, \alpha_0)$, *où :*

- α_0 *est l'état abstrait initial, tel que* $\sigma_0 \in \alpha_0$,
- $\twoheadrightarrow \subseteq (C \times T \times C)$ *est la relation successeur qui satisfait la condition* $EE_r = SG \wedge EE'$, *où :*
  - $SG : (\sigma \overset{t_f}{\rightsquigarrow} \sigma') \Rightarrow (\forall \alpha \in C \text{ t.q. } \sigma \in \alpha, \exists \alpha' \in C, (\sigma' \in \alpha' \wedge \alpha \overset{t_f}{\twoheadrightarrow} \alpha'))$,
  - $EE' : (\alpha \overset{t_f}{\twoheadrightarrow} \alpha') \Rightarrow (\exists \sigma \in \alpha, \exists \sigma' \in \alpha', \sigma \overset{t_f}{\rightsquigarrow} \sigma')$.

Soient $\mathcal{N}$ un modèle TPN, $\rho = \sigma_1 \rightsquigarrow \sigma_2 \rightsquigarrow \sigma_3 \rightsquigarrow .....$ un chemin d'exécution dans son espace des états concrets, et $\rho_\alpha = \alpha_1 \twoheadrightarrow \alpha_2 \twoheadrightarrow \alpha_3 \twoheadrightarrow .....$ un chemin d'exécution dans un espace d'états abstraits de $\mathcal{N}$. On dit que ρ est *inscriptible* dans ρ_α ssi

$\sigma_i \in \alpha_i, \forall i \geq 1$. Dans ce cas, on dit aussi ρ_α *couvre* ρ.

La sous condition SG (*Sequence Guarantee* (SG)) a pour conséquence que chaque chemin d'exécution de l'espace des états concrets est *inscriptible* dans un chemin d'exécution d'un espace d'états abstraits. La sous condition EE' impose de ne pas connecter deux classes qui n'ont pas d'états reliés. La condition EE_r, combinée au fait $\sigma_0 \in \alpha_0$, assure qu'un espace d'états abstraits couvre tous les états concrets du modèle, i.e., $\forall \sigma \in \Sigma, \exists \alpha \in C \ t.q. \ \sigma \in \alpha$.

Nous propose aussi de relaxer les conditions AE, EA et U, où l'équivalence est remplacée par une simple implication. Les formes relaxées sont :

- $EA_r : (\alpha \overset{t_f}{\twoheadrightarrow} \alpha') \Rightarrow (\forall \sigma' \in \alpha', \exists \sigma \in \alpha, \sigma \overset{t_f}{\rightsquigarrow} \sigma')$,
- $AE_r : (\alpha \overset{t_f}{\twoheadrightarrow} \alpha') \Rightarrow (\forall \sigma \in \alpha, \exists \sigma' \in \alpha', \sigma \overset{t_f}{\rightsquigarrow} \sigma')$,
- $U_r : (\alpha \overset{t_f}{\twoheadrightarrow} \alpha') \Rightarrow (\forall \sigma \in noyau(\alpha), \exists \sigma' \in noyau(\alpha'), \sigma \overset{t_f}{\rightsquigarrow} \sigma')$, où $noyau(\alpha) \subseteq \alpha \vee noyau(\alpha) \neq \emptyset$, $\forall \alpha \in C$, et $\sigma_0 \in noyau(\alpha_0)$.

Le prochain théorème établit que ces conditions sont suffisantes pour préserver respectivement les propriétés : CTL^*, LTL et $ACTL^*$.

Théorème 4.2.1 *(Hadjidj et Boucheneb, 2005b)*
Soient $\mathcal{N}$ un modèle TPN, *$\mathcal{S} = (\Sigma, \rightsquigarrow, \sigma_0)$ son espace des états concrets, et $\mathcal{C} = (C, \twoheadrightarrow, \alpha_0)$ un espace d'états abstraits de $\mathcal{N}$ (qui satisfait donc la condition EE_r).*

- *Si $\mathcal{C}$ satisfait la condition AE_r alors il préserve les propriétés CTL^* de $\mathcal{N}$,*
- *Si $\mathcal{C}$ satisfait la condition EA_r et $\alpha_0 = \{\sigma_0\}$ alors il préserve les propriétés LTL de $\mathcal{N}$,*
- *Si $\mathcal{C}$ satisfait la condition U_r alors il préserve les propriétés $ACTL^*$ de $\mathcal{N}$.*

Preuve: AE_r : Sachant qu'en absence de transitions silencieuses[3](Tarasyuk, 1998), des états bisimilaires vérifient les mêmes propriétés CTL^* (Penczek et Polrola,

[3]Des activités internes du systèmes non visibles à un observateur externe.

2004), il suffit de montrer que si $\mathcal{C}$ satisfait AE_r, il est alors bisimilaire à $\mathcal{S}$. Pour cela, il suffit de trouver une relation de bisimulation qui contient (α_0, σ_0).

Soit $\approx$ la relation définie par : $\forall(\alpha, \sigma) \in (C \times \Sigma)$, $(\alpha, \sigma) \in \approx$ ssi $\sigma \in \alpha$. Les trois propriétés suivantes sont vraies pour $\approx$:

1. $(\alpha_0, \sigma_0) \in \approx$,
2. $\forall(\alpha, \sigma) \in \approx$, $(\alpha \overset{t_f}{\twoheadrightarrow} \alpha') \Rightarrow \exists\sigma', (\sigma \overset{t_f}{\rightsquigarrow} \sigma') \wedge (\alpha', \sigma') \in \approx$,
3. $\forall(\alpha, \sigma) \in \approx$, $(\sigma \overset{t_f}{\rightsquigarrow} \sigma') \Rightarrow \exists\alpha', (\alpha \overset{t_f}{\twoheadrightarrow} \alpha') \wedge (\alpha', \sigma') \in \approx$.

La propriété (1) est immédiate de la définition de α_0. La propriété (2) est une conséquence directe de la condition AE_r. Finalement, la propriété (3) est vraie parce que $\mathcal{C}$ satisfait la condition SG. De ces propriétés, on conclut que $\approx$ est une bisimulation.

EA_r : On doit monter que $\mathcal{S}$ et $\mathcal{C}$ possèdent des chemins d'exécution qui correspondent, i.e., (a) tout chemin d'exécution de $\mathcal{S}$ est inscriptible dans un chemin d'exécution $\mathcal{C}$, et (b) tout chemin d'exécution de $\mathcal{C}$ couvre un chemin d'exécution de $\mathcal{S}$. Par définition, la condition SG garantit (a). Il reste à prouver que (b) est aussi vraie. Soit $\alpha_0 \overset{t_0}{\twoheadrightarrow} \alpha_1 .... \alpha_{n-1} \overset{t_{n-1}}{\twoheadrightarrow} \alpha_n$ un chemin d'exécution de longueur n dans l'espace des états abstraits. De la transition $\alpha_{n-1} \overset{t_{n-1}}{\twoheadrightarrow} \alpha_n$ et la condition EA_r on peut déduire que $\forall\sigma_n \in \alpha_n, \exists\sigma_{n-1} \in \alpha_{n-1},\ t.q.\ \sigma_{n-1} \overset{t_{n-1}}{\rightsquigarrow} \sigma_n$. De la même façon, on peut déduire que $\exists\sigma_{n-2} \in \alpha_{n-2},\ t.q.\ \sigma_{n-2} \overset{t_{n-2}}{\rightsquigarrow} \sigma_{n-1}$. De proche en proche on peut retrouver un chemin d'exécution dans le graphe des états concrets qui est inscriptible dans le chemin des états abstrait de départ. Comme $\alpha_0 = \{\sigma_0\}$, l'état initial de ce chemin est forcément $\{\sigma_0\}$ (i.e., le chemin retrouvé est un chemin d'exécution de $\mathcal{S}$).

U_r : Sachant que deux systèmes équivalents par simulation satisfont les mêmes propriétés $ACTL^*$ (Penczek et Polrola, 2001;Browne *et al.*, 1988;Grumberg et Long, 1991), il suffit de monter que $\mathcal{S}$ et $\mathcal{C}$ sont équivalents par simulation. Soient les deux

relations $\sim$ et $\sim'$ définies comme suit :

- $\forall(\sigma, \alpha) \in (\Sigma \times C)$, t.q. $(\sigma, \alpha) \in \sim$ ssi $\sigma \in \alpha$,
- $\forall(\alpha, \sigma) \in (C \times \Sigma)$, t.q. $(\alpha, \sigma) \in \sim'$ ssi $\sigma \in noyau(\alpha)$.

De la condition SG et du fait que $(\sigma_0, \alpha_0) \in \sim$, on peut facilement prouver que $\sim$ est une relation de simulation de $\mathcal{C}$ à $\mathcal{S}$ (i.e., $\mathcal{C}$ simule $\mathcal{S}$). Quant à la relation $\sim'$, on $(\alpha_0, \sigma_0) \in \sim'$. De plus, si $(\alpha, \sigma) \in \sim'$ et $\alpha \overset{t_f}{\twoheadrightarrow} \alpha'$, la condition U_r nous garantit que $\sigma \overset{t_f}{\rightsquigarrow} \sigma'$, avec $\sigma' \in noyau(\alpha')$. Donc $(\alpha', \sigma') \in \sim'$. D'où $\sim'$ est une relation de simulation de $\mathcal{S}$ à $\mathcal{C}$. ■

CHAPITRE 5

LE GRAPHE DES ZONES D'ÉTATS CONCRETS

Dans ce chapitre, nous proposons de construire un espace d'états abstrait pour le modèle TPN basé sur la caractérisation horloge des états, similairement à ce qui est proposé dans (Yoneda et Ryuba, 1998), mais avec une caractérisation des états abstraits beaucoup plus simple. Comme la caractérisation horloge des états permet d'identifier un à un les états agglomérés dans un état abstrait, notre abstraction peut être raffinée pour restaurer les propriétés de branchement. Nous appelons cette abstraction *graphe des zones d'états concrets* (*Concrete State Zone Graph* (CSZG)). Les états abstraits sont appelés *zones d'états.*

5.1 Graphe des zones d'états concrets (Boucheneb et Hadjidj, 2004)

Dans un premier temps nous donnons une définition du CSZG valable pour les réseaux de Petri temporels à intervalles de temps bornés. Nous compléterons ensuite cette définition pour le cas des réseaux de Petri temporels à intervalles de temps non bornés.

Définition 5.1.1 : ***CSZG (incomplète)***

Le CSZG est l'espace des états abstraits $(Z, \twoheadrightarrow, \alpha_0)$*, tel que :*

- $\alpha_0 = \{\sigma_0\}$,
- $\alpha \overset{t}{\twoheadrightarrow} \alpha'$ *ssi* $\alpha' = succ(\alpha, t)$,
- $Z = \{\alpha | \alpha_0 \overset{*}{\twoheadrightarrow} \alpha\}$, *où* $\overset{*}{\twoheadrightarrow}$ *est la fermeture réflexive et transitive de* $\twoheadrightarrow$.

Pour construire le CSZG d'un modèle TPN, nous caractérisons une zone d'états α par le couple (m, F), où m est le marquage commun aux états agglomérés dans α, et F une formule qui caractérise le domaine d'horloges de tous les états de α. L'horloge de chaque transition sensibilisée dans m est représentée dans F par une variable de même nom. Le domaine de F, dénoté $Dom(F)$, est l'ensemble $\{V|(m, V) \in \alpha\}$. La zone d'états initiale du CSZG est le couple $\alpha_0 = (m_0, F_0)$, où m_0 est le marquage initial et $F_0 = (\bigwedge_{t \in En(m_0)} t = 0)$. Soient $\alpha = (m, F)$ une zone d'états et t_f une transition. α a un successeur par t_f, dénoté $succ(\alpha, t_f)$, ssi t_f est sensibilisée dans m et peut être franchie avant toute autre transition sensibilisée. L'algorithme 3 montre comment effectuer ce test. Les étapes 1-3 vérifient si t_f est sensibilisée dans m.

Algorithme 3 $isFirable(\alpha = (m, F), t_f)$

1: **si** $t_f \notin En(m)$ **alors**
2: Retourner faux
3: **fin si**
4: **Soit** $F' = F \wedge (\theta \geq 0) \wedge (t_f + \theta \geq tmin(t_f)) \wedge (\bigwedge_{t \in En(m)}(t + \theta \leq tmax(t)))$
5: **si** F' est consistante **alors**
6: Retourner vrai
7: **sinon**
8: Retourner faux
9: **fin si**

L'étape 4 calcule la formule qui caractérise la partie du domaine de franchissement de α d'où t_f peut être franchie avant toute autre transition sensibilisée. L'étape 5 vérifie si cette partie est vide. Si t_f est franchissable à partir de α, $\alpha' = succ(\alpha, t_f)$ est calculée selon l'algorithme 4.

Dans l'algorithme 4, l'étape 1 calcule le marquage après franchissement de t_f. Les étapes 2-5 incrémentent chaque horloge de θ unités de temps, pour coïncider avec le moment où t_f est franchie. Les étapes 6-8 ajoutent les contraintes qui correspondent aux transitions nouvellement sensibilisées. Le CSZG est généré progressivement en calculant les successeurs de la zone d'états initiale et ceux de chaque zone d'états nouvellement calculée, jusqu'à ce qu'aucune nouvelle zone d'états ne soit générée.

Algorithme 4 $succ(\alpha = (m, F), t_f)$

1: **Soit** $m'(p) = m(p) - Pre(p, t_f) + Post(p, t_f), \forall p \in P$
2: **Soit** $F' = F \wedge (\theta \geq 0)$
3: Remplacer dans F' chaque variable $t \neq t_f$ par $(t - \theta)$ {Cette substitution augmente les horloges de toutes les transitions sensibilisées par exactement θ unités de temps}
4: Ajouter à F' les contraintes : $(tmin(t_f) \leq t_f)$ et $(\bigwedge_{t \in En(m)} t \leq tmax(t))$,
5: Éliminer de F' t_f par substitution, ainsi que toutes les variables associées aux transitions en conflit avec t_f dans m
6: **pour chaque** $t \in New(m', t_f)$ **faire**
7: Ajouter à F' la contrainte $t = 0$
8: **fin pour**
9: Retourner (m', F')

On peut déduire de l'algorithme 4 que la formule F de chaque zone d'états (m, F) est une conjonction de contraintes atomiques de la forme $(t - t' \prec c)$ ou $(t \prec c)$, où $c \in \mathbb{Q} \cup \{\infty, -\infty\}$, $\prec \in \{<, \leq\}$ et $t, t' \in T$ (voir section 1 pour la définition), tel que c est le résultat d'une combinaison finie, par sommation et différence, des délais statiques de franchissement (minimaux et maximaux) de transitions. F est donc la formule associée à un polyèdre convexe, i.e., une *zone* (voir section 1.6 pour la définition). Pour tester si deux zones d'états sont identiques, il faut vérifier si elles ont le même marquage, et que leurs formules ont des formes canoniques identiques.

Lemme 5.1.1 *Soit m un marquage atteignable d'un modèle TPN avec des intervalles de franchissement bornés, tel que $\forall t \in En(m), tmax(t) \neq \infty$. Le nombre de zones d'états qui ont le marquage m est fini.*

Preuve: Pour une zone d'états (m, F) avec un tel marquage, toutes les constantes qui apparaissent dans F sont bornées. En effet :

- La constante c de chaque contrainte de la forme $(t \prec c)$ ne peut pas être plus grande en valeur absolue que $tmax(t)$ (i.e., $|c| \leq tmax(t)$) (La valeur de l'horloge associée à transition t est toujours positive, et ne peut pas dépasser $tmax(t)$, d'après la sémantique du modèle TPN),

- La constante c de chaque contrainte atomique de la forme $(t - t' \prec c)$ ne peut pas être plus grande en valeur absolue que le maximum entre $tmax(t)$ et $tmax(t')$ (i.e., $|c| <= max(tmax(t), tmax(t'))$),
- la constante c de chaque contrainte atomique résulte d'une combinaison finie par sommation et différence des délais minimaux et maximaux statiques de franchissement de transitions,
- Pour toute transition t, $tmin(t), tmax(t) \in \mathbb{Q}^+$.

Il s'ensuit que chaque constante c de chaque contrainte atomique peut seulement prendre un nombre fini de valeurs (i.e., le nombre de contraintes est fini), et donc nous déduisons que le nombre de zones d'états possibles pour le marquage m est fini.

■

Du lemme 5.1.1 précédent, le CSZG est fini pour tous les modèles TPN bornés avec des intervalles statiques de franchissement bornés. Pour ces modèles, le nombre de marquages ainsi que le nombre de zones d'états qui partagent un même marquage sont aussi finis.

5.1.1 Réduction de la complexité de calcul des zones d'états du CSZG

Pour représenter et calculer les zones d'états, nous utilisons les matrices de bornes (DBM) (Behrmann *et al.*, 2002; S.Yovine, 1993) (voir Section 1.7). Les DBMs ont l'avantage d'implémenter efficacement toutes les opérations nécessaires au calcul du CSZG, avec une complexité qui ne dépasse pas généralement $O(n^2)$, (n étant l'ordre de la DBM, i.e., le nombre de transitions sensibilisées dans le marquage de la zone d'états). Cependant, cette efficacité est assurée seulement si chaque DBM manipulée est mise sous sa forme canonique (voir Section 1.7). L'opération de canonisation d'une DBM a une complexité de l'ordre de $O(n^3)$. Comme les opérations de calcul

impliquées dans la construction du CSZG ne gênèrent pas forcément des DBMs sous formes canoniques, la canonisation est une opération très sollicitée. C'est donc l'opération la plus coûteuse dans le calcul du CSZG.

Nous établissons dans les Propositions 5.1.1 et 5.1.2 qui suivent une implémentation du test de franchissement d'une transition en $O(n)$, et du calcul de la zone d'états successeur, directement dans sa forme canonique, en $O(n^2)$. Notons que dans ce qui suit, toute constante c qui apparaît dans une opération qui implique des bornes, mais qui n'est pas elle même une borne, sera considérée implicitement comme étant la borne $(c, \leq)$.

Proposition 5.1.1 *Soient (m, F) une zone d'états, (m, B) sa forme canonique et t_f une transition.*
t_f est franchissable à partir de (m, B) ssi :
- *$t_f \in En(m)$, et*
- *$tmin(t_f) \leq Min_{t \in En(m)}(tmax(t) + B(t_f, t))$.*

Preuve: Intuitivement, la proposition stipule qu'une transition t_f est franchissable ssi son horloge peut atteindre *tmin(t_f)* avant que toute autre transition sensibilisée dans m n'atteigne son *tmax*. Pour la preuve formelle nous utilisons la notion de *graphe de contraintes* (voir section 1.7.2). Rappelons en premier deux résultats sur les graphes de contraintes. Soit F une formule (un ensemble de contraintes) représentée par un graphe de contraintes.

1. F est consistante ssi le graphe des contraintes de F n'a aucun cycle négatif,
2. Si le graphe de contraintes de F n'a aucun cycle négatif, le poids du plus court chemin allant du noeud y au noeud x est égal à $Sup(x - y, F)$.

Comme la formule de chaque zone d'états atteignable est consistante, son graphe de contraintes n'a aucun cycle négatif. Soient $\alpha = (m, F)$ une zone d'états atteignable, B la DBM associée à F dans sa forme canonique, et G son graphe de contraintes.

En utilisant la définition de B et les résultats précédents, le poids du plus court chemin, dans G, allant d'un noeud y à un noeud x est égal à $B(x,y)$. D'après l'algorithme 3, on déduit que la transition t_f est franchissable depuis α ssi, t_f est sensibilisée pour le marquage m, et que la formule $F \wedge (\bigwedge_{t \in En(m)} t - t_f \leq tmax(t) - tmin(t_f))$ est consistante. En d'autres termes, le graphe de contraintes de F complété avec l'ensemble des arcs $\{(t_f, t, tmax(t) - tmin(t_f)) | t \in En(m)\}$[1] n'a aucun cycle négatif. Du moment qu'avant d'ajouter ces arcs le graphe ne contenait pas de cycles négatifs, le graphe complété n'aura pas de cycles négatifs ssi aucun cycle qui passe par l'un des arcs ajoutés n'est négatif (voir figure 5.1, les arcs ajoutés sont en pointillés). Le poids du plus petit cycle passant par un arc ajouté $(t_f, t, tmax(t) - tmin(t_f))$ est $tmax(t) - tmin(t_f) + B(t_f, t)$. Ce poids doit être non négatif, i.e., $\forall t \in En(m), tmax(t) - tmin(t_f) + B(t_f, t) \geq 0$. ■

Proposition 5.1.2 *Soient (m, F) une zone d'états, (m, B) sa forme canonique, et t_f une transition. Si t_f est franchissable depuis (m, B), son franchissement mène à la zone d'états (m', B') calculée comme suit :*

- $\forall p \in P, m'(p) = m(p) - Pre(p, t_f) + Post(p, t_f)$,
- $\forall t \in En(m')$,
  - $B'(t,t) = 0$,
  - *Si t est nouvellement sensibilisée :* $B'(t,o) = B'(o,t) = 0$,
  - *Si t n'est pas nouvellement sensibilisée :*

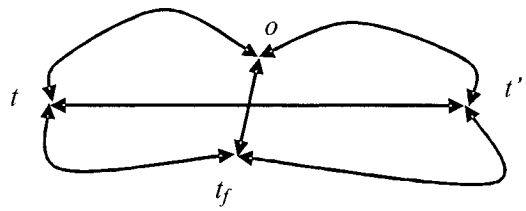
    $B'(t,o) = Min_{t' \in En(m)}(tmax(t') + B(t,t'))$, *et*

    $B'(o,t) = Min(B(o,t), B(t_f,t) - tmin(t_f))$,
- $\forall (t,t') \in (En(m'))^2$, *avec* $t \neq t'$,
  - *Si t et t' sont nouvellement sensibilisées :* $B'(t,t') = B'(t',t) = 0$,
  - *Si t est nouvellement sensibilisée et t' ne l'est pas :* $B'(t,t') = B'(o,t')$ *et* $B'(t',t) = B'(t',o)$,

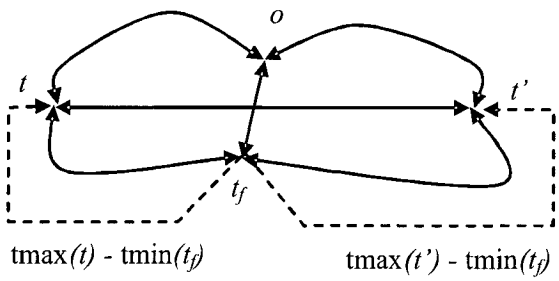
[1] Un arc d'origine x, de destination y et de poids p est dénoté par le triplet (x,y,p).

- *Si t et t' ne sont pas nouvellement sensibilisées :*

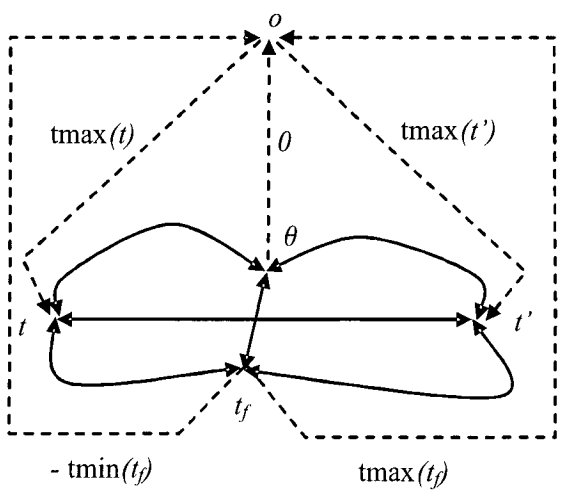
  $B'(t,t') = Min(B(t,t'), B'(t,o) + B'(o,t'))$.



a) Graphe de contraintes de F (le poids de chaque arc (y,x) est B(x,y))



b) Graphe de contraintes de $F \wedge (\wedge_{t\ En(M)} t - t_f \leq \text{tmax}(t) - \text{tmin}(t_f))$



c) Graphe de contraintes après renommage de o en θ dans F, et l'ajout des arcs qui correspondent à: $\theta \geq 0 \wedge (\text{tmin}(t_f) \leq t_f) \wedge (\wedge_{t\ En(m)} t \leq \text{tmax}(t))$

FIGURE 5.1 Les arcs ajoutés dans le graphe des contraintes de F

Preuve: Supposons que la transition t_f est franchissable depuis la zone d'états $\alpha = (m, B)$. De l'algorithme 4, on déduit que la matrice B' de la zone atteignable de α par le franchissement de t_f, peut être calculée en utilisant le *graphe des contraintes* correspondant à B comme suit :

1 Renommer le noeud o en θ, puis ajouter un noeud o et l'arc $(\theta, o, 0)$. Cette opération correspond à l'étape 2 de l'algorithme 4.

2 Ajouter l'arc $(t_f, o, -tmin(t_f))$ et tous les arcs de l'ensemble $\{(o, t', tmax(t'))|t' \in En(m)\}$.
Ceci correspond à l'ajout des contraintes $(tmin(t_f) \leq t_f)$ et $(\bigwedge_{t \in En(m)} t \leq tmax(t))$ à l'étape 4 de l'algorithme 4.

3 Renommer le noeud t_f et tous les noeuds associés aux transitions en conflit avec t_f pour m, pour éviter d'avoir plusieurs noeuds avec un même nom.

4 Pour chaque transition t nouvellement sensibilisée dans m', ajouter un nouveau noeud t et les deux arcs $(t, o, 0)$ et $(o, t, 0)$. Ceci correspond à l'ajout de la contrainte $t = 0$ à l'étape 7 de l'algorithme 4.

Pour chaque couple (x, y) de $(En(m') \cup \{o\})^2$, $B'(x, y)$ est le poids du plus court chemin du noeud y au noeud x, dans le graphe des contraintes complété (voir figure 5.1(c), les arcs rajoutés sont en pointillés) :

- Si t n'est pas nouvellement sensibilisée, le plus court chemin du noeud o au noeud t est le plus court chemin parmi ceux qui passent par un arc $(o, t', tmax(t'))$, avec $t' \in En(m)$. Autrement, sa valeur est 0,
- Si t n'est pas nouvellement sensibilisée, le plus court chemin du noeud t au noeud o est le plus court chemin parmi ceux qui passent par $(\theta, o, 0)$ ou bien $(t_f, o, -tmin(t_f))$. Autrement, sa valeur est 0. Notons que θ correspond au noeud o dans le graphe des contraintes de F,
- Si t et t' ne sont pas nouvellement sensibilisées, le plus court chemin du noeud t' au noeud t est le plus court chemin parmi ceux qui passent par le noeud o et ceux qui ne passent pas par le noeud o. Notons que tous les arcs ajoutés sont

soient rentrants ou sortants au noeud o,

- Si t et t' sont nouvellement sensibilisées, $t - t' = 0$,
- Si t est nouvellement sensibilisée et t' ne l'est pas, $t - t' = -t'$ et $t' - t = t'$.

■

5.2 Normalisation des zones d'états

Pour les modèles TPN bornés avec des intervalles de franchissement statiques non bornés, le nombre de zones d'états qui partagent le même marquage n'est pas forcément fini, car les horloges dans ce cas ne sont pas bornées. À titre d'exemple, le modèle *TPN* de la figure 2.1 (page 25) possède seulement quatre marquages atteignables, mais son CSZG est infini. En effet, si les transitions t_1 et t_2 ne sont pas franchies [2], la transition t_0 peut être franchie un nombre infini de fois, générant à chaque franchissement une zone d'états différente. La zone d'états atteinte après le k-ème franchissement est $(p_0 + p_1 + p_2, t_0 = 0 \wedge t_1 = t_2 \wedge k \leq t_1 \leq 2k)$. Nous proposons dans ce qui suit une solution à ce problème d'infinitude qui découle du constat suivant : Pour une transition t dont le domaine de franchissement statique $Is(t)$ n'est pas borné, si la valeur de son horloge est plus grande ou égale $tmin(t)$ (la borne inférieure de son domaine de franchissement statique), sa valeur exacte n'est plus significative. La transition devient alors franchissable à n'importe quel moment et n'influence le franchissement d'aucune autre transition. Nous formalisons cette propriété dans la définition qui suit.

Définition 5.2.1 : ***Équivalence par horloges***

Soit $(S, \rightarrow, s_0)$ *l'espace des états d'un modèle TPN. Deux états horloges* $s, s' \in S$, *tels que* $s = (m, V)$ *et* $s' = (m', V')$, *sont dits* équivalents par horloges, *noté* $s \doteq s$,

[2] Le franchissement de ces transitions peut être retardé infiniment parce que leurs délais maximaux de franchissements sont infinis.

ssi :

- $m = m'$,
- $\forall t \in En(m)$ *t.q.* $tmax(t) \neq \infty$, $V(t) = V'(t)$,
- $\forall t \in En(m)$ *t.q.* $tmax(t) = \infty$, $V(t) = V'(t) \ \vee \ (V(t) \geq tmin(t) \wedge V'(t) \geq tmin(t))$.

Le théorème suivant établit que les états équivalents par horloges sont bisimilaires.

Théorème 5.2.1 *La relation $\doteq$ est une relation de bisimulation.*

Preuve: Il est facile de voir que si $s \doteq s'$ alors s et s' correspondent tous les deux au même état intervalle. Il est s'en suit alors que s et s' sont bisimilaires. ■

Nous étendons la relation d'équivalence par horloges $\doteq$ aux zones d'états comme suit :

Soient α et α' deux zones d'états. $\alpha \doteq \alpha'$ ssi :

- $m = m'$,
- $\forall s = (m, V) \in \alpha, \exists s' = (m', V') \in \alpha'$ *t.q.* $V' \doteq V$,
- $\forall s' = (m', V') \in \alpha', \exists s = (m, V) \in \alpha$ *t.q.* $V \doteq V'$.

Notons aussi que si deux zones états sont équivalentes par horloges, elles sont aussi bisimilaires. La solution au problème d'infinitude consiste donc à définir une opération de normalisation sur les zones d'états pour borner les constantes qui apparaissaient dans leurs formules, tout en préservant la bisimilarité. La définition suivante formalise ce concept de normalisation (dit aussi *k-normalisation* ou *k-approximation*)

Définition 5.2.2 : *k-normalisation*

On appelle k-normalisation, *où* $k \in \mathbb{Q}^+$, *toute opération qui, appliquée à une zone d'états* $\alpha = (m, F)$, *génère une zone d'états* $\alpha' = (m', F')$, *telle que :*

- $\alpha \doteq \alpha'$,
- *Pour toute constante* $c \neq \pm\infty$ *qui apparaît dans une contrainte atomique de* F', $c \leq k$.

Plusieurs implémentations de la k-normalisation sont proposées dans la littérature pour le modèle des automates temporisés (Pettersson, 1999) et les réseaux de Petri temporisés (Gardey *et al.*, 2003; Rokicki, 1993). L'objectif étant de forcer la terminaison de la construction de modèles d'états abstraits dans certains cas spécifiques, où la construction normale n'est pas garantie de terminer. Dans (Gardey *et al.*, 2003) les auteurs construisent un espace d'états abstrait pour le modèle TPN, appelé *Zone Based Graph* (ZBG), en utilisant une opération de normalisation appelée *k-approximation*, similaire à celle utilisée pour les automates temporisés (Pettersson, 1999). Une zone d'états est normalisée en ignorant [3] dans sa formule toutes les contraintes atomiques de la forme $t \leq c$ ou $t - t' \leq c$ (resp, $t < c$ ou $t - t' < c$) où la constante c est plus grande (resp, plus grande ou égale) que la plus grande constante finie qui apparaît dans les intervalles de franchissement statiques du modèle TPN. Une contrainte de la forme $t \geq c$ (resp, $t > c$) où la constante c est plus grande (resp, plus grande ou égale) que $tmin(t)$ est remplacée avec $t > tmin(t)$. L'opération de *relaxation*, proposée dans (Berthomieu et Vernadat, 2003) pour forcer la terminaison dans le cas du SSCG (voir section 3.3.6 pour une description), peut être aussi interprétée comme une opération de normalisation si les classes d'états[4] sont considérées comme des agglomérations d'états horloges. De cette perspective, la relaxation complète une classe d'états avec tous les états concrets équivalents par horloges à ses propres états. Comme cette opération peut rendre le domaine de la classe non convexe, la classe est divisée en plusieurs sous classes pour préserver la convexité. Nous proposons dans ce qui suit une implémentation de la

[3] Ignorer une contrainte revient à remplacer sa constante par ∞.

[4] Les classes d'états sont l'équivalent de zones d'états dans notre approche.

k-normalisation plus efficace que celles proposée (Berthomieu et Vernadat, 2003) et (Gardey *et al.*, 2003) (voir section 7, tableau 7.1). L'idée est de compléter une zone d'états avec des états bisimilaires à ses propres états, tout en maintenant son domaine convexe. Ainsi, il ne sera pas nécessaire de la diviser pour restaurer la convexité. Nous appelons cette implémentation "$norm_k$".

Soient $\mathcal{N} = (P, T, Pre, Post, m_0, Is)$ un modèle TPN, $T_\infty = \{t \in T \mid tmax(t) = \infty\}$ l'ensemble de ses transitions qui ont des intervalles statiques de franchissement non bornés, k la plus grande constante finie qui apparaît dans les intervalles statiques de franchissement de $\mathcal{N}$, et $\alpha = (m, F)$ une zone d'états de $\mathcal{N}$ sous forme canonique. $norm_k(\alpha)$ est calculée selon l'algorithme 5.

Algorithme 5 $norm_k(\alpha = (m, F))$

1: **pour chaque** $t \in T_\infty \cap En(m)$ **faire**
2: **si** la contrainte $t \geq c$ (resp, $t > c$) est telle que $c \geq tmin(t)$ **alors**
3: Éliminer la variable t de F
4: Ajouter la contrainte $t \geq tmin(t)$
5: **sinon si** la contrainte $t \leq c$ (resp, $t < c$) est telle que $c \geq tmin(t)$ (resp, $c > tmin(t)$) **alors**
6: Enlever la contrainte $t \leq c$ (resp, $t < c$)
7: Enlever de F toute contrainte atomique $t - t' \leq c$" (ou $t - t' < c$"), telle que $t' \geq c'$ (ou $t' > c'$) est une contrainte atomique et c" $+ c' \geq tmin(t)$
8: **fin si**
9: **fin pour**

Proposition 5.2.1 *$norm_k$ est une k-normalisation.*

Preuve: Notons en premier que $norm_k$ enlève toutes les contraintes atomiques qui ont des constantes plus grandes que la plus grande constante qui apparaît dans les intervalles statiques de franchissement. Il reste à montrer que l'opération $norm_k(\alpha)$ génère une zone d'états équivalente par horloge à α. Dans l'algorithme 5, le premier cas (i.e., les étapes 2-4) étend les bornes de la transition t, ce qui donne une formule F' telle que $Dom(F) \subseteq Dom(F')$. Puisque ceci n'affecte pas les bornes d'autres

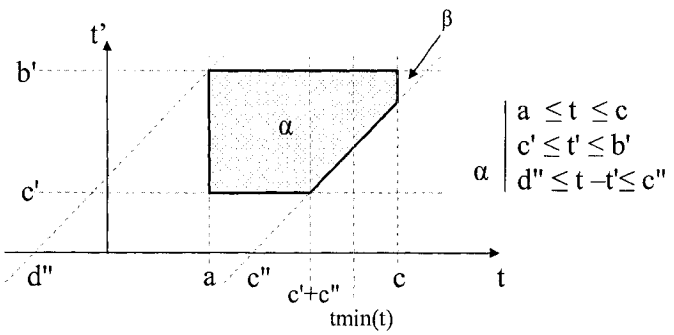


FIGURE 5.2 Le domaine d'une zone d'états

transitions, il en résulte que les projections de $Dom(F)$ et $Dom(F')$ sur les variables $En(m) - \{t\}$ sont identiques. Avec le fait que $t \geq tmin(t)$ dans F et F', nous pouvons conclure que pour tout $V \in Dom(F)$, il existe $V' \in Dom(F')$ telle que $(m, V) \doteq (m, V')$, et vice versa. Ceci veut dire que $\alpha \doteq (m, F')$.

La preuve du deuxième cas (i.e., les étapes de 5-8) est plus complexe mais possède une explication géométrique assez simple. La figure 5.2 montre le domaine d'une zone d'états α projeté sur deux transition (t et t'). Pour α, la contrainte $t \leq c$ est telle que $c \geq tmin(t)$. Si $c' + c" < tmin(t)$ (comme le montre la figure 5.2), le fait d'enlever la contrainte $t \leq c$ peut rajouter des états avec des valuations dans la zone indiquée par β. Cependant, tous ces états sont équivalents par horloge à des états qui sont déjà dans α. Dans le cas où $c' + c" \geq tmin(t)$ (i.e., la ligne verticale $t = tmin(t)$ est à gauche de la ligne $t = c' + c"$), le fait d'enlever les conditions $t \leq c$ et $t - t' \leq c"$ peut étendre la zone α à droite jusqu'à l'infini. Mais ceci va aussi avoir comme effet de rajouter à α des états équivalents par horloges à des états qui sont déjà dans α.

■

Maintenant, avec la normalisation, la définition du CSZG peut être complétée pour traiter tous les modèles TPN bornés.

Définition 5.2.3 : *CSZG*

Le CSZG est l'espace d'états abstraits $(Z, \twoheadrightarrow, \alpha_0)$, *tel que :*

- $\alpha_0 = norm_k(\{\sigma_0\})$,
- $\alpha \overset{t}{\twoheadrightarrow} \alpha'$ *ssi* $\alpha' = norm_k(succ(\alpha, t))$,
- $Z = \{\alpha | \alpha_0 \overset{*}{\twoheadrightarrow} \alpha\}$, *où* $\overset{*}{\twoheadrightarrow}$ *est la fermeture réflexive et transitive de* $\twoheadrightarrow$.

La normalisation des zones d'états calculées par l'algorithme 4 nous garantit que le CSZG généré sera fini ssi le modèle TPN est borné, même s'il possède des transitions avec des intervalles de franchissement statiques non bornées. Cependant, la condition EA_r ne sera pas forcément satisfaite, bien que le CSZG généré préserve les propriétés linéaires du modèle TPN.

Lemme 5.2.1 *Soit* m *un marquage atteignable d'un modèle TPN tel que* $\forall t \in En(m)$, $tmax(t) \neq \infty$. *Le nombre de zones d'états qui ont le marquage* m *est fini.*

Preuve: Notons que l'opération de normalisation n'a aucun effet sur une zones d'états (m, F) dont les transitions sensibilisées ont tous des intervalles de franchissement statiques bornés. Le lemme 5.1.1 nous assure que le nombre des zones d'états de marquage m est toujours fini. Dans le cas contraire, l'opération de normalisation borne toute constante qui apparaît dans la formule F avec la plus grande constante qui apparaît dans les intervalles de franchissement statiques du modèle TPN, ou la replace avec ∞. Avec une preuve similaire à celle du lemme 5.1.1 on peut déduire que le nombre de formules atomiques qui peuvent apparaître dans F est fini, ce qui nous assure que le nombre de zones de marquage m est aussi fini. ■

Le théorème suivant établit quelques propriétés du CSZG.

Théorème 5.2.2

(i) Le CSZG est un espace d'états abstraits,

(ii) Le CSZG est fini ssi le modèle TPN est borné,

(iii) Le CSZG préserve les propriétés linéaires du modèle TPN.

(iv) Le CSZG préserve les propriétés d'atteignabilité du modèle TPN.

Preuve:

(i) De la définition de l'opération $norm_k$, il est possible de voir que pour toute zone d'états α, $\alpha \subseteq norm_k(\alpha)$. De la définition du CSZG, il est évident que $\sigma_0 \in \alpha_0$ et par induction $\forall \alpha \in Z, \alpha \cap \Sigma \neq \emptyset$. Donc la condition EE' est vraie. Pour la condition SG, soient $\sigma \overset{t}{\rightsquigarrow} \sigma'$ une transition telle que $\sigma \in \Sigma$, α une zone d'états telle que $\sigma \in \alpha$ et $\alpha' = norm_k(succ(\alpha, t))$. De la définition des l'opérations $succ$ et $norm_k$, il est évident que $\sigma' \in \alpha'$.

(ii) Si le modèle TPN est borné, le nombre de ses marquages accessibles est fini. Comme le lemme 5.2.1 nous assure que le nombre zones d'états d'un marquage donné est fini, on déduit que le CSZG est fini.

(iii) Par construction, on peut prouver que n'importe quel chemin d'exécution dans l'espace des états concrets est inscriptible dans un chemin d'exécution du CSZG, et chaque chemin d'exécution du CSZG couvre un chemin d'exécution de l'espace des états concrets du modèle TPN. Par conséquent, chaque marquage accessible dans le CSZG l'est aussi dans l'espace des états concrets du modèle TPN et vice-versa.

■

5.3 Contraction du CSZG par inclusion

Nous proposons dans ce qui suit de générer un modèle d'états abstraits plus compact que le CSZG en agglomérant les zones d'états de ce dernier par inclusion. Le graphe résultant, que nous notons IC-CSZG (*Inclusion Contracted CSZG*) (Boucheneb et Hadjidj, 2005), peut être obtenu avec une simple modification de l'algorithme de construction du CSZG. Durant cette construction, lorsqu'une nouvelle zone d'états

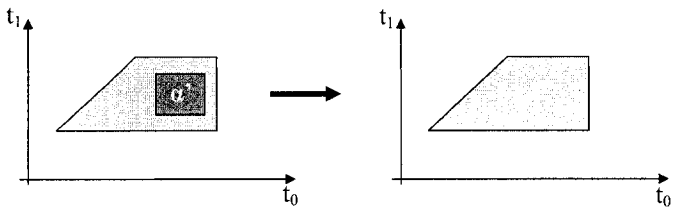


FIGURE 5.3 Contraction par inclusion

$\alpha = (m, F)$ est générée, si une zone d'états déjà calculée $\alpha' = (m', F')$ est telle que $m = m'$ et $Dom(F) \subseteq Dom(F') \vee Dom(F') \subseteq Dom(F)$, la plus grande entre α et α' remplace la plus petite (voir figure 5.3). Le test d'inclusion est effectué en $O(n^2)$ (S.Yovine, 1993) comme suit : Soient $\alpha = (m, F)$ et $\alpha' = (m', F')$ deux zones d'états avec B et B' leurs matrices de bornes respectives sous formes canoniques. La zone d'états α est incluse dans α', ssi :

- $m = m'$,
- $\forall (x, y) \in (En(m) \cup \{o\})^2$, $(B(x, y) \leq B'(x, y))$ (le symbole o représente la valeur zéro).

Durant le processus de construction, les petites zones d'états seront remplacées par les zones d'états qui les incluent. Si la zone d'états initiale est remplacée par une autre zone, cette dernière devient la zone d'états initiale. Le graphe résultant est en général une fraction de la taille du CSZG, et calculé en une fraction du temps aussi (voir section 7, tableau 7.2 pour des résultats expérimentaux).

Théorème 5.3.1

(i) Le IC-CSZG est un espace d'états abstraits,

(ii) Le IC-CSZG est fini ssi le modèle TPN est borné,

(iii) Le IC-CSZG préserve les propriétés d'atteignabilité du modèle TPN.

Preuve:

(i) Notons que le IC-CSZG peut être obtenu du CSZG en regroupant de proche

en proche chaque zone d'états dans celle qui l'inclut (si elle existe). À chaque regroupement, la condition EE_r reste évidemment valide, et le restera donc aussi lorsque tous les regroupements possibles sont effectuées (i.e., à l'obtention du IC-CSZG). Notons que selon cette procédure, les marquages accessibles dans le IC-CSZG restent identiques à ceux du CSZG.

(ii) $\Rightarrow$: Comme les marquages du IC-CSZG et ceux du CSZG sont identiques, si le IC-CSZG est fini, le modèle TPN est borné.
$\Leftarrow$: Si le modèle TPN est borné, son CSZG est fini. Le IC-CSZG, qui n'est qu'une contraction du CSZG, est aussi fini.

(iii) Découle du fait que les marquages accessibles du IC-CSZG sont identiques à ceux du CSZG.

■

À titre d'illustration, soit le modèle TPN de la figure 2.1. Son CSZG montré dans la figure 5.4 contient 12 noeuds et 24 arcs. La figure 5.5 montre le IC-CSZG correspondant. L'abstraction par inclusion a permis d'obtenir un graphe avec 6 noeuds et 14 arcs, ce qui représente une réduction en taille de près de 50%. Pour des réductions plus significatives (plus que 100 plus petit en taille et plus 100 fois plus rapide à calculer), voir section 7, tableau 7.2.

5.4 Contraction du CSZG par combinaison convexe

L'objectif est toujours de construire un espace d'états abstraits le plus compact possible. L'idée dans cette contraction est de regrouper dans un même noeud, des zones d'états dont l'union des domaines est convexe[5] (voir figure 5.6).

[5]Les zones d'états regroupées doivent avoir le même marquage.

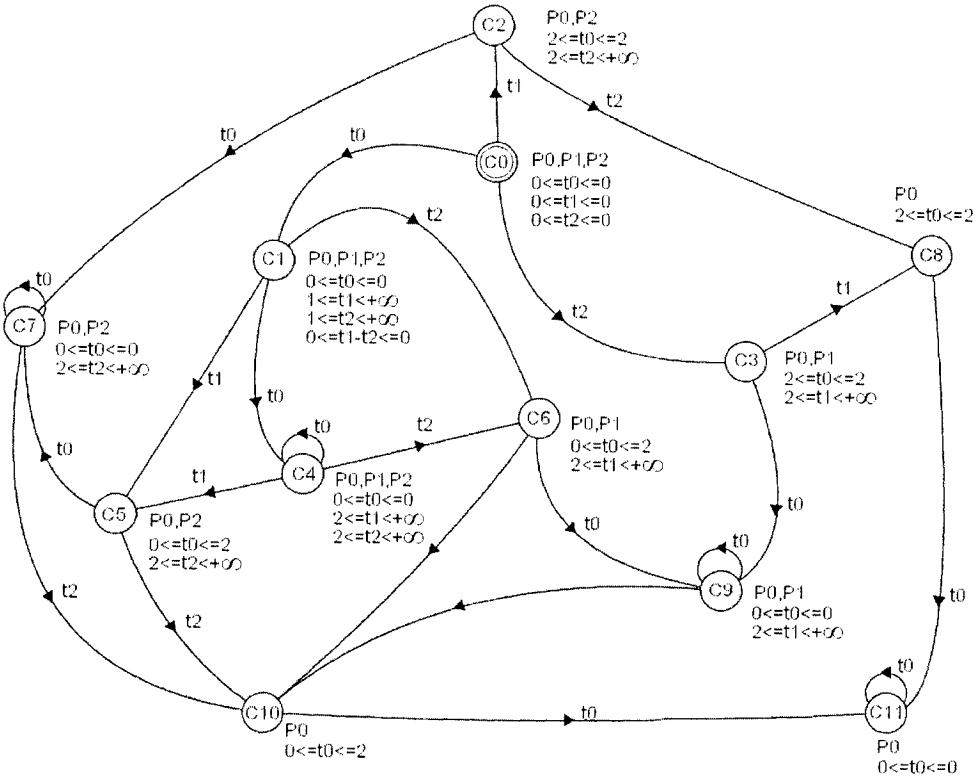


FIGURE 5.4 Le CSZG du modèle de la figure 2.1

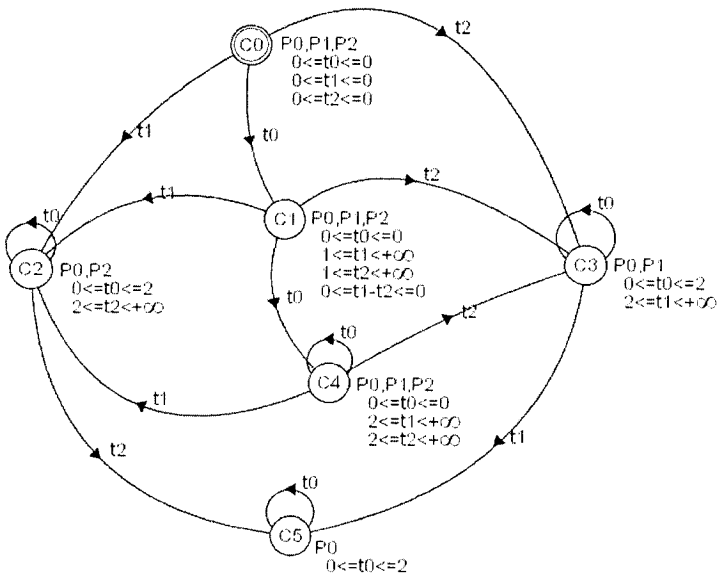


FIGURE 5.5 Le IC-CSZG du modèle de la figure 2.1

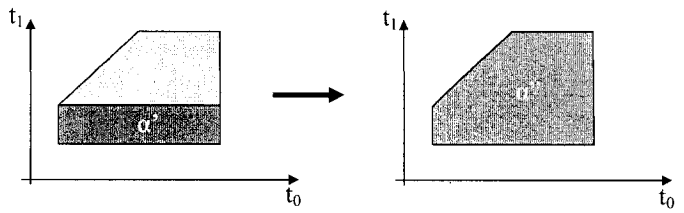


FIGURE 5.6 Contraction par combinaison convexe

Le graphe résultant est noté CC-CSZG (*Convex grouping Contracted CSZG*) (Hadjidj et Boucheneb, 2005a). Avant d'expliquer comment effectuer la contraction, nous allons définir ce qu'est une *couverture convexe* de zones d'états.

Définition 5.4.1 : ***couverture convexe de zones d'états***

Soient $\alpha_i = (m, F_i)$ $(i = \overline{1,n})$ n *zones d'états, et soit* $\mathcal{F} = \bigvee_{i=\overline{1,n}} F_i$.

La couverture convexe *des formules* F_i $(i = \overline{1,n})$ *est la formule* $\breve{F}_{(F1,..,Fn)}$ *définie par :* $\bigwedge_{(x,y)\in(En(m)\cup\{o\})^2} x - y \prec_{\mathcal{F}}^{x-y} Sup(x-y, \mathcal{F})$, *où* $\prec_{\mathcal{F}}^{x-y}$ *est soit* $\leq$ *ou* $<$, *selon que* $x - y$ *atteigne ou non son supremum dans le domaine de* $\mathcal{F}$.

La couverture convexe *des zones d'états* α_i $(i = \overline{1,n})$ *est la zone d'états* $\breve{\alpha} = (m, \breve{F}_{(F1,..,Fn)})$.

Proposition 5.4.1 *Soient* $\alpha_i = (m, F_i)$ $(i = \overline{1,n})$ n *zones d'états, et* $\breve{F}_{(F1,..,Fn)}$ *la* couverture convexe *des* F_i $(i = \overline{1,n})$. *Les propriétés suivantes sont vraies :*

(i) L'union des domaines des $F_i (i = \overline{1,n})$ *est incluse dans le domaine de* $\breve{F}_{(F1,..,Fn)}$,

(ii) Si l'union des domaines des F_i $(i = \overline{1,n})$ *est convexe alors* $\breve{F}_{(F1,..,Fn)}$ *est sa forme canonique.*

Preuve:

(i) Des faits suivants :

- $\forall i = \overline{1,n}$, $\forall (x,y) \in (En(m) \cup \{o\})^2$,

$Sup(x-y,F_i) \leq Max_{j=\overline{1,n}}(Sup(x-y,F_j))$,

- $Sup(x-y,\mathcal{F}) = Max_{j=\overline{1,n}}(Sup(x-y,F_j))$, avec $\mathcal{F} = \bigvee_{i=\overline{1,n}} F_i$.

on peut conclure que le domaine de chaque formule F_i est inclus dans le domaine de $\breve{F}_{(F1,..,Fn)}$. Donc, le domaine de l'union (i.e., celui de $\mathcal{F}$) l'est aussi.

(ii) Si l'union des domaines de F_i $(i = \overline{1,n})$ est convexe, elle peut être représentée par la formule $\bigwedge_{(x,y)\in(En(m)\cup\{o\})^2} x - y \prec_{\mathcal{F}}^{x-y} Sup(x-y, \bigvee_{i=\overline{1,n}} F_i)$, qui n'est rien d'autre que $\breve{F}_{(F1,..,Fn)}$. ■

Proposition 5.4.2 *Soient C et A_i $(i = \overline{1,n})$* n+1 *ensembles tels que $A_i \subseteq C$ $(i = \overline{1,n})$, alors : $(\bigcup_{i=\overline{1,n}} A_i = C)$ ssi $(C - A_1 - ... - A_{n-1}) \subseteq A_n$.*

Preuve: Observons que si A, B, et C sont trois ensembles tels que $A, B \subseteq C$, alors $C = A \cup B$ ssi $(C - A) \subseteq B$. La preuve de la proposition est une conséquence directe de l'observation précédente, en remplaçant A par $(\bigcup_{j=\overline{1,n-1}} A_j)$ et B par A_n. ■

Soient $\alpha_i = (m, F_i)$ $(i = \overline{1,n})$ n zones d'états. Des propositions 5.4.1 et 5.4.2, ces zones d'états peuvent être regroupées en une seule zone d'états convexe ssi la couverture convexe de leurs domaines $\breve{F}_{(F1,..,Fn)}$ satisfait :

$$(Dom(\breve{F}_{(F1,..,Fn)}) - Dom(F_1) - .. - Dom(F_{n-1})) \subseteq Dom(F_n).$$

Pour vérifier cette condition, nous devons savoir comment caractériser le domaine résultant de la différence entre deux domaines convexes. Ce domaine n'est pas nécessairement convexe, néanmoins, il peut être partitionné en un nombre fini de domaines convexes. L'algorithme 6 suivant calcule cette partition. Il prend comme argument deux formules, F et F', qui caractérisent deux domaines convexes et retourne une partition de $Dom(F) - Dom(F')$.

Le test de la convexité de l'union de n domaines F_i $(i = \overline{1,n})$ consiste donc à calculer

Algorithme 6 *complment*(F, F')

1: Soient $X = F$ et $Part = \emptyset$ {$Part$ contiendra une partition de $Dom(F) - Dom(F')$ à la fin de l'exécution de l'algorithme}
2: **pour chaque** contrainte atomique f de F' **faire**
3: **si** $(X \wedge \neg f)$ est consistante **alors**
4: $Part := Part \cup \{X \wedge \neg f\}$
5: $X := (X \wedge f)$
6: **fin si**
7: **fin pour**
8: Retourner $Part$

leur couverture convexe $\breve{F}_{(F1,..,Fn)}$, la différence entre le domaine de $\breve{F}_{(F1,..,Fn)}$ et tous les domaines des F_i $(i = \overline{1, n-1})$, et finalement à vérifier si cette différence est incluse dans le domaine de F_n.

Notons que les zones d'états qui ne se combinent pas deux à deux peuvent se combiner trois à trois ou plus. La figure 5.7 illustre quelques situations impliquant le regroupement convexe de zones d'états qui ont seulement deux transitions sensibilisées. Dans le cas **a)**, les zones d'états α et α' sont combinées dans la zone d'états α". Le cas **b)** montre deux zones d'états dont l'union n'est pas convexe. Le cas **c)** illustre une situation où trois zones d'états α, α' et α" ne peuvent pas être regroupées lorsqu'elles sont considérées deux à deux, mais se regroupent parfaitement en α" si elles sont prises ensemble. Les cas **d)** et **e)** montrent d'autres situations où le regroupement deux à deux n'est pas possible mais devient possible pour d'autres regroupements.

Pour atteindre un maximum de contractions, nous devons tester tous les regroupements possibles des zones d'états ayant le même marquage. Cette opération est malheureusement assez coûteuse en temps de calcul. Les résultats expérimentaux ont cependant montré qu'en se limitant à des regroupement deux à deux, nous obtenons des contractions très satisfaisantes, en des temps très acceptables. De plus, lorsque deux zones d'états sont telles que l'une est incluse dans l'autre, leur regrou-

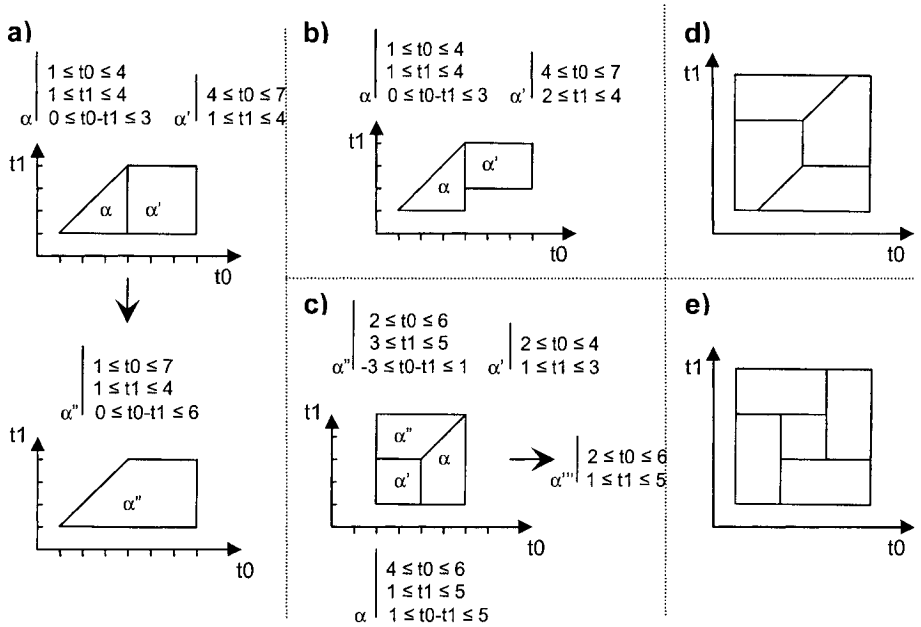


FIGURE 5.7 Regroupement convexe de zones d'états

pement convexe est la zone d'états la plus grande. De cette façon, avant de tester le regroupement convexe, nous vérifions l'inclusion.

Théorème 5.4.1

(i) Le CC-CSZG est un espace d'états abstraits,

(ii) Le CC-CSZG est fini ssi le modèle TPN est borné,

(iii) Le CC-CSZG préserve les propriétés d'atteignabilité du modèle TPN.

Preuve:

(i) Le CC-CSZG est un espace d'états abstrait s'il vérifie les propriétés suivantes :

1. La zone d'états initiale contient l'état initial du modèle TPN,
2. La condition EE_r est satisfaite.

(1) est garanti par l'algorithme de construction et par le fait que les regroupements convexes de n zones d'états n'est rien d'autre que leur union (pas de perte ni de création d'états).

Pour vérifier (2), considérons en premier la sous condition EE' de EE_r. Soit $\mathcal{Z}^i = (Z^i, \twoheadrightarrow^i, \alpha_0^i)$ le graphe des zones d'états partiel calculé après application de la règle de franchissement i fois. Après chaque application de la règle, l'opération de regroupement convexe est appliquée pour effectuer des agglomérations si c'est possible. Si on suppose que la condition EE' est valide pour $\mathcal{Z}^i$ (i.e., $\forall(\alpha, t_f, \alpha') \in Z^i \times T \times Z^i$, $(\alpha \overset{t_f}{\twoheadrightarrow}{}^i \alpha') \Rightarrow (\exists\sigma \in \alpha, \exists\sigma' \in \alpha', \sigma \overset{t_f}{\rightsquigarrow} \sigma')$), il est facile de montrer que cette condition restera valide après un regroupement convexe. Par induction on peut montrer que cette condition est vraie pour $\mathcal{Z}^i, \forall i \geq 0$, et de cette façon, lorsque la construction termine.

Pour la sous condition SG de EE_r (i.e., $(\sigma \overset{t_f}{\rightsquigarrow} \sigma') \Rightarrow (\forall\alpha \in Z \ t.q. \ \sigma \in \alpha, \exists\alpha' \in Z$, $(\ \sigma' \in \alpha' \wedge \alpha \overset{t_f}{\rightsquigarrow} \alpha'))$, elle sera vraie à la fin de la construction. Dans le graphe résultant, si une zone d'états α contient l'état σ et $\sigma \overset{t_f}{\rightsquigarrow} \sigma'$, on déduit que t_f peut être franchie depuis α et l'opération *succ* a certainement calculé une zone d'états α' successeur de α par t_f qui contient σ'. Notons que même si α' a été regroupée avec d'autres zones d'états, la condition SG reste vraie.

(ii) $\Rightarrow$: Si le CC-CSZG est fini, le nombre de marquages atteignables est fini aussi. De ce fait, le modèle TPN est borné.

$\Leftarrow$: On peut montrer par induction que chaque zone d'états dans le CC-CSZG est le regroupement (union) de certaines zones d'états dans le CSZG. De cette façon, si le modèle TPN est borné, son CSZG sera fini. Donc, son CC-CSZG le sera aussi.

(iii) Découle du fait que l'agglomération par regroupement convexe ne provoque ni création ni perte d'états concrets. Si un état concret est accessible dans le CSZG, il le restera aussi dans le CC-CSZG. Donc les marquages accessibles dans le CSZG sont préservés dans le CC-CSZG, de même que les propriétés d'atteignabilité. ■

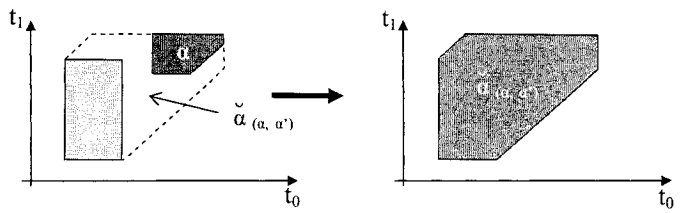


FIGURE 5.8 Contraction par couverture convexe

5.5 Contraction du CSZG par couverture convexe

La couverture convexe définie dans la section 5.4 peut être utilisée pour construire une contraction du CSZG très compacte, où toutes les zones d'états ont des marquages différents (i.e., une seule zone d'états par marquage). Durant la construction, chaque zone d'états nouvellement calculée est combinée par couverture convexe avec la zone d'états de même marquage (si elle existe). Si la zone d'états qui en résulte est différente de celle qui existe déjà, tous ses successeurs sont recalculés. Le graphe résultant est noté CH-CSZG (*Convex Hull contracted CSZG*). Comme la couverture convexe est une sur approximation de l'union, les états concrets accessibles dans le CH-CSZG ne le sont pas forcément tous dans le CSZG (voir figure 5.8).

Théorème 5.5.1

(i) Le CH-CSZG est un espace d'états abstraits,

(ii) Si le CH-CSZG est fini alors le modèle TPN est borné.

Preuve:

(i) La preuve est similaire à celle du théorème 5.4.1.

(ii) Comme le CH-CSZG est un espace d'états abstrait, il couvre tous les états accessibles du modèle TPN (voir Section 4.2). Donc il couvre aussi tous ses marquages accessibles. Si le CH-CSZG est fini, le nombre de marquages du modèle TPN est

fini aussi. Donc le modèle TPN est borné.

■

D'une manière générale le CH-CSZG, et se calcule très rapidement comparativement au CSZG. Il constitue de ce fait un moyen efficace d'avoir une borne supérieure sur les marquages du modèle TPN (le nombre de ses noeuds). Comme la bornitude et l'atteignabilité des marquages sont des propriétés non décidables pour le modèle TPN (Berthomieu et Diaz, 1991;Berthomieu et Menasche, 1983), le théorème 5.5.1 définit des conditions suffisantes pour vérifier ces propriétés, i.e. :

- Si le CH-CSZG est borné, le modèle TPN l'est aussi,
- Si un marquage n'est pas atteignable dans le CH-CSZG, il ne l'est pas pour le modèle TPN.

De plus, le CH-CSZG permet de calculer rapidement une borne supérieure des marquages du modèle TPN.

5.6 Contraction du LSCG par inclusion, combinaison convexe et couverture convexe

Au même titre que le CSZG, le LSCG (Berthomieu et Menasche, 1983) peut lui aussi être contracté par inclusion, par combinaison convexe et par couverture convexe. Les modèles abstraits résultants sont notés respectivement IC-LSCG (*Inclusion Contracted LSCG*), CC-LSCG (*Convex grouping Contracted LSCG*) et CH-LSCG (*Convex Hull contracted LSCG*) (Hadjidj et Boucheneb, 2006a). Comme les classes d'états dans le LSCG et les zones d'états dans le CSZG sont caractérisées mathématiquement de la même manière (i.e., un marquage m est une formule F qui définit une zone sur $En(m)$), les mêmes approches de calcul des trois contractions du CSZG restent valables pour celles du LSCG.

Théorème 5.6.1

(i) Le IC-LSCG, le CC-LSCG et le CH-LSCG sont des espaces d'états abstraits,

(ii) Le IC-LSCG et le CC-LSCG sont finis ssi le modèle TPN est borné,

(iii) Le IC-LSCG et le CC-LSCG préservent les propriétés d'accessibilité du modèle TPN,

(iv) Si leCH-LSCG est borné alors le modèle TPN est borné aussi.

Preuve: Les preuves sont identiques à celles données pour le IC-CSZG, le CC-CSZG et le CH-CSZG. ■

5.7 Le graphe des zones d'états concrets atomiques

Un *graphe des zones d'états concrets atomiques*, noté A-CSZG (*Atomic CSZG*), est un graphe de zones d'états qui préserve les propriétés de branchements du modèle TPN. Ces propriétés sont celles qu'on peut exprimer avec des logiques temporelles telles que CTL et CTL^*. Sachant qu'en absence de transitions silencieuses[6], la bismilarité préserve les propriétés de branchements (Browne *et al.*, 1988; Nicola et Vandrager, 1995), l'objectif est donc de construire un graphe de zones d'états bisimilaire à l'espace des états concrets du modèle TPN (i.e., qui satisfait la condition AE_r). Une technique consiste à utiliser un algorithme de raffinement destiné, à priori, à résoudre le problème de la *partition minimale induite par une relation* (Paige et Tarjan, 1987;Tarasyuk, 1998). À partir d'une partition d'un espace d'états concrets, les classes[7] d'états sont partitionnées de proche en proche, jusqu'à l'obtention d'une partition pré-stable (i.e., préserve la condition AE_r). La technique génère la plus petite partition plus fine que celle d'origine et bisimilaire à l'espace des états concrets. Cette même technique appliquée à un espace d'états abstraits, qui est une

[6]Activités internes d'un système non visibles pour un observateur externe.

[7]Classe au sens élément d'une partition.

couverture[8] plutôt qu'une partition, reste aussi valide, mais ne génère pas forcément une partition minimale. La préservation de la convexité des domaines des états abstraits est un autre facteur qui nous éloigne de cet objectif. Berthomieu et Vernadat, dans (Berthomieu et Vernadat, 2003), ainsi que Yoneda et Ryuba, dans (Yoneda et Ryuba, 1998), utilisent la technique de raffinement. Ils commencent par construire un modèle d'états abstraits du modèle TPN qui préserve les propriétés linéaires, puis le raffinent pour restaurer les propriétés de branchement. Cependant, les modèles raffinés sont généralement très grands, avec un degré élevé de redondance d'états[9]. Dans certains cas, ces modèles sont même plus grands que ceux qu'on obtient après raffinement. La redondance provient en partie de la manière dont les états abstraits sont agglomérés pour préserver les propriétés linéaires. Quant à la taille, elle est due en grande partie à la préservation de la convexité et à l'opération de normalisation utilisée pour forcer la terminaison. Toutes ces raisons contribuent à dégrader les performances de la procédure de raffinement, en induisant une convergence lente et une explosion importante des états durant le raffinement (voir section 7 pour les résultats expérimentaux). Dans ce qui suit, nous proposons de raffiner des modèles d'états abstraits très compacts qui ne préservent pas forcément les propriétés linéaires. En occurrence, le IC-CSZG ou le CC-CSZG, au lieu du CSZG.

Soient $(\Sigma, \rightsquigarrow, \sigma_0)$ l'*espace des états concrets* d'un modèle *TPN* et $\mathcal{Z} = (Z, \twoheadrightarrow, \alpha_0)$ un graphe de zone d'états. $\mathcal{Z}$ est dit atomique ssi il satisfait la condition AE_τ. En d'autres termes, $\mathcal{Z}$ est atomique ssi : $\forall(\alpha, t_f, \alpha') \in \twoheadrightarrow, (\alpha = Pred(\alpha', t_f, \alpha))$, où $Pred(\alpha', t_f, \alpha)$ est l'ensemble de tous les états de α qui peuvent mener par le franchissement de la transition t_f à des états dans α'. Dans le cas où la condition $\alpha = Pred(\alpha', t_f, \alpha)$ n'est pas satisfaite pour l'arc (α, t_f, α'), α est partitionnée en deux parties : $Pred(\alpha', t_f, \alpha)$ et $\alpha - Pred(\alpha', t_f, \alpha)$.

[8]Une couverture d'un ensemble Σ est une collection d'ensembles dont l'union contient Σ.
[9]Un états concret peut apparaître dans plusieurs classes d'états.

Soient $\alpha = (m, F)$ et $\alpha' = (m', F')$ deux zones d'états telles que $\alpha \overset{t_f}{\twoheadrightarrow} \alpha'$. $\alpha" = (m, F") = Pred(\alpha', t_f, \alpha)$ est calculé selon l'algorithme 7 (on supposera dans l'algorithme que $\forall p \in P, m'(p) = m(p) - Pre(p, t_f) + Post(p, t_f)$).

Algorithme 7 $Pred(\alpha' = (m', F'), t_f, \alpha = (m, F))$

1: Soit $F" = (F' \wedge \bigwedge_{t \in New(m', t_f)} t = 0)$
2: Éliminer par substitution toutes les transitions dans $New(m', t_f)$ de $F"$
3: Ajouter à $F"$ les contraintes : $(tmin(t_f) \leq t_f)$, $(\bigwedge_{t \in En(m)} t \leq tmax(t))$ et $\theta \geq 0$
4: Remplacer dans $F"$ chaque variable t avec $t+\theta$, puis éliminer θ par substitution
5: Ajouter toutes les contraintes de F à $F"$
6: Retourner $(m, F")$

Dans l'algorithme 7, sachant que le franchissement de la transition t_f initialise l'horloge de chaque transition nouvellement sensibilisée à zéro, l'étape 1 extrait de α' le sous ensemble d'états où les horloges des transitions nouvellement sensibilisées sont égales à zéro. L'étape 3 ajoute la contrainte de franchissement de t_f. L'étape 4 recule dans le temps (chaque horloge est décrémentée de θ unités de temps). Finalement, l'étape (5) ajoute toutes les contraintes de la zone d'états α.

5.7.1 Partitionnement des zones d'états non atomiques

Notons que le domaine de $\alpha - Pred(\alpha', t_f, \alpha)$ n'est pas nécessairement convexe. L'algorithme 6 est utilisé pour partitionner ce domaine en un ensemble de parties convexes. La taille de la partition qui en résulte peut atteindre le nombre de contraintes atomiques de la formule de α (i.e., $n^2 + n$, où n est le nombre des transitions sensibilisées dans α), ce qui reflète une importante fragmentation des zones d'états. Pour réduire cette fragmentation, nous contractons la partition de $\alpha - Pred(\alpha', t_f, \alpha)$ par combinaison convexe (voir section 5.4).

5.7.2 Le raffinement

L'algorithme 8 génère un A-CSCG en utilisant une procédure de raffinement classique combinée avec l'abstraction par inclusion pour accélérer la convergence. Dans

Algorithme 8 $rafine(\mathcal{Z})$

1: **tantque** $\mathcal{Z}$ n'est pas atomique **faire**
2: **si** $\alpha \in Z$ n'est pas atomique pour une transition $\alpha \overset{t}{\rightarrow\!\!\!\!\rightarrow} \alpha'$ **alors**
3: Soit $\alpha" = Pred(\alpha', t_f, \alpha)$
4: Soit $Part = \alpha - Pred(\alpha', t_f, \alpha)$
5: Grouper les zones d'états dans $Part$ par combinaison convexe
6: Soit $Part = Part \cup \{\alpha"\}$
7: Remplacer α par $Part$ dans $\mathcal{Z}$
8: Regrouper les zones d'états par inclusion
9: **fin si**
10: **fin tantque**

l'algorithme, l'étape 7 remplace la zone d'états α par sa partition $Part$. Chaque sous zone de la partition hérite tous les arcs sortants et rentrants de α. Dans le cas où α possède une boucle (α, α, t), chaque sous zone de la partition est connectée par la transition t, à toutes les sous zones (y compris elle-même). Si une sous zone est incluse dans une zone existante, les deux zones sont regroupées dans un seul noeud. Dans le cas où l'opération de division retourne un ensemble vide, l'arc (α, α', t_f) est supprimé. L'algorithme 8 termine parce qu'il raffine un graphe de zone d'états fini, et le nombre de manières de partitionner une zone d'états est fini. L'étape 8 a pour objectif de minimiser au maximum la redondance des états durant le rafinement. Comme l'algorithme raffine le IC-CSZG ou le CC-CSZG où il n y a aucune zone d'états incluse dans une autre, cette opération est considérablement simplifiée. Lorsqu'une zone d'états est partitionnée, chacune de ses sous zones ne peut inclure aucune autre zone. L'inverse n'est cependant pas vrai. De cette façon, nous devons seulement vérifier si une sous zone de la partition est incluse dans une autre zone d'états. Cette contraction appliquée durant le processus de raffinement réduit le nombre de zones d'états, et du même coup, le nombre de zones d'états à

partitionner. Comme conséquence, la procédure de raffinement converge plus rapidement et génère des graphes plus petits. Le théorème 5.7.1 établit la convergence de la technique de raffinement proposée.

Théorème 5.7.1

(i) L'A-CSZG préserve les propriétés CTL du modèle TPN,*

(ii) Le raffinement combiné avec la contraction par inclusion génère un A-CSZG fini ssi le modèle est borné.

Preuve:

(i) Comme le raffinement établit la condition AE_r a la fin de son traitement, le théorème 4.2.1 nous assure que l'A-CSZG préserve les propriétés CTL^*.

(ii) La preuve est une conséquence directe des faits suivant :

- Le IC-CSZG ou le CC-CSZG sont finis ssi le modèle TPN est borné,
- Le nombre de partitionnements possibles d'une zone d'états est toujours fini (conséquence du lemme 5.2.1 et du fait que le nombre de contraintes atomiques d'une zone d'états est fini),
- La contraction par inclusion appliquée durant la procédure de raffinement a pour effet de réduire le nombre de zones d'états.

■

5.7.3 Pourquoi préserver la convexité ?

La convexité des zones d'états est une caractéristique importante à préserver si nous voulons garder le calcul simple, mais constitue un obstacle pour atteindre une abstraction minimale. La simplicité du calcul est particulièrement garantie par l'usage des matrices de bornes (Behrmann *et al.*, 2002; Daws *et al.*, 1996). Cette

structure de données s'adapte très bien à tous les aspects de calcul impliqués dans la construction de graphes de zones d'états, leurs contractions et leurs raffinements. Elle a par ailleurs le désavantage de ne pas permettre de représenter efficacement des domaines non convexes (Les DBMs ne sont pas fermées par rapport à l'union). Dans ce sens, les CDDs (*Clock Difference Diagrams*) (Behrmann *et al.*, 2002; Behrmann *et al.*, 1999) semblent être une meilleure alternative. Les CDDs permettent de représenter d'une manière assez concise l'union de domaines convexes. Ils sont aussi fermés par rapport à l'union, l'intersection et la complémentation. Par ailleurs, ils n'ont pas de forme canonique unique. De ce fait, les CDDs sont moins appropriés que les DBMs lorsqu'il s'agit de calculer les successeurs et les prédécesseurs de zone d'états. Une description détaillée des CDDs peut être trouvée dans (Behrmann *et al.*, 1999), où les auteurs utilisent cette structure de données pour représenter les domaines de la liste PASSED, de l'algorithme d'atteignabilité implémenté dans l'outil UPPAAL (voir figure 6.2 page 97). Cependant, ils utilisent toujours des DBMs pour calculer les successeurs des états abstraits. L'utilisation des CDDs a permis aux auteurs d'obtenir un gain appréciable en terme d'usage de la mémoire (42% en moyenne), mais au prix d'une légère augmentation des temps de calcul (20% en moyenne) (Behrmann *et al.*, 2002; Larsen *et al.*, 1999). Notons que les états abstraits dans la liste PASSED sont manipulés avec seulement deux opérations de base qui sont bien supportées par les CDDs, à savoir l'union et l'inclusion. Dans une approche basée sur un algorithme de raffinement, chaque état abstrait peut subir plusieurs opérations de partitionnement. Ce raffinement implique des opérations de calcul (successeur et prédécesseurs) qui ne sont pas bien supportées par les CDDs. Les DBMs apparaissent donc comme un meilleur choix. Néanmoins les CDDs restent une structure de données très prometteuse, qui mérite plus d'investigation.

CHAPITRE 6

VÉRIFICATION DES PROPRIÉTÉS TEMPORISÉES DU MODÈLE TPN

Plusieurs approches sont proposées dans la littérature pour vérifier les propriétés non temporisées du modèle TPN (Berthomieu et Menasche, 1983; Boucheneb et Hadjidj, 2004; Boucheneb et Hadjidj, 2005; Boucheneb et Berthelot, 1993; Gardey *et al.*, 2003; Hadjidj et Boucheneb, 2005b; Lilius, 1999; Okawa et Yoneda, 1997). La plupart d'entre elles sont basées sur la *méthode des classes d'états* (Berthomieu et Menasche, 1983), où une classe d'états est une représentation symbolique d'un ensemble infini d'états qui partagent un même marquage (voir section 3.3.1). Ces approches proposent des techniques de construction d'espaces de classes d'états qui préservent l'atteignabilité (Boucheneb et Berthelot, 1993; Lilius, 1999), les propriétés linéaires (Berthomieu et Menasche, 1983), et les propriétés de branchement (Boucheneb et Hadjidj, 2004; Boucheneb et Hadjidj, 2005; Hadjidj et Boucheneb, 2005b; Okawa et Yoneda, 1997), qui sont par la suite vérifiées en utilisant des méthodes de model-checking standards (Clarke *et al.*, 1999). À notre connaissance, il n'y pas encore de techniques pour vérifier des propriétés temporisées du modèle TPN basées sur la méthode des classes d'états. Celles connues se basent soit sur la traduction en automates temporisés (Alur et Dill, 1990), soit sur la méthode des *observateurs* (Toussaint *et al.*, 1997). Les techniques basées sur la traduction définissent des procédures de conversion d'un modèle TPN en un automate temporisé (Alur et Dill, 1990) sémantiquement équivalent, afin d'utiliser les outils de model-checking existants (Cassez et Roux, 2003; Cortés *et al.*, 2002; Gu et Shin, 2002; Lime et Roux, 2003; Okawa et Yoneda, 1997). Le model-checking est par la suite effectué sur l'automate temporisé, et les résultats sont interprétés de nouveau sur le modèle

TPN d'origine. Bien qu'elles soient appropriées, ces techniques sont exposées à la complexité exponentielle en nombre d'horloges inhérente au modèle des automates temporisés. La difficulté d'interpréter les résultats d'un modèle à un autre est une autre difficulté qui entrave l'utilisation de telles techniques. La méthode des *observateurs* (Toussaint *et al.*, 1997), quant à elle, modifie le modèle TPN considéré, selon la propriété à vérifier, en lui greffant des places et des transitions qu'on qualifie d'observateur. Une propriété temporisée dans le modèle initial devient alors une propriété d'atteignabilité dans le modèle transformé, qu'on vérifie avec la méthode des classes. Malheureusement, cette technique se limite à des propriétés temporisées très particulières, et ne permet de couvrir qu'une classe assez réduite des propriétés exprimables dans la logique $TCTL$.

Nous proposons dans ce chapitre une approche de vérification des propriétés temporisées du modèle TPN basées sur la méthode des classes, et prouvons sa décidabilité pour tous les modèles TPN bornés. Notons que notre approche est assez générique. Elle pourrait être adaptée à d'autres extensions temporisées des réseaux de Petri, ainsi qu'aux automates temporisés (Alur et Dill, 1990).

Les propriétés temporisés que nous considérons sont principalement une sous classe de la logique $TCTL$ (Alur *et al.*, 1993), que nous appelons $TCTL_{TPN}$, suffisante en général pour vérifier les propriétés temporisées les plus intéressantes (atteignabilité, sécurité, vivacité ...). L'approche de model-checking que nous proposons est basée sur une technique d'exploration à la volée (Behrmann *et al.*, 2002; Bouajjani *et al.*, 1997; Kupferman *et al.*, 1996), combinée avec l'abstraction par inclusion[1] pour mieux confiner le problème d'explosion combinatoire d'états (Boucheneb et Hadjidj, 2005). Étant donné un modèle TPN et une propriété $TCTL_{TPN}$, le modèle est en premier mis en parallèle avec un autre modèle TPN spécifique, que nous appelons *Alarm-*

[1]Lorsqu'une classe d'états est explorée, il n'est pas utile d'explorer une autre classe d'états qui y est incluse.

clock. Le graphe des classes d'états linéaires (LSCG) du modèle TPN combiné est par la suite exploré à la recherche d'événements temporaux pertinents pour décider de la valeur de vérité de la propriété considérée. *Alarm-clock* possède deux transitions qui ont des priorités de franchissement spéciales. À l'exploration, ces transitions peuvent être sensibilisées pour qu'elles soient franchies à des moments bien spécifiques (le début ou la fin d'un intervalle de temps), ce qui permet de capturer des événements temporaux importants. Le calcul des classes d'états est de ce fait adapté pour prendre en considération le nouveau concept de priorité introduit dans *Alarm-clock*.

6.1 Logique temporelle temporisée $TCTL_{TPN}$

Nous définissons une logique temporelle pour laquelle nous donnons des algorithmes pour vérifier la satisfaction de ses formules dans le contexte du modèle TPN. La logique que nous considérons est principalement une sous classe de la logique temporisée $TCTL$ (Alur *et al.*, 1993), dont les propositions atomiques sont exprimées sur les marquages (Hadjidj et Boucheneb, 2006b; Hadjidj et Boucheneb, 2005c).

Soit PR l'ensemble des propositions sur les marquages du modèle TPN (i.e., $\{\wp | \wp : M \rightarrow \{true, false\}\}$, où M est l'ensemble des marquages du modèle TPN). Notre logique temporelle, que nous appelons $TCTL_{TPN}$, est définie comme suit :

$$TCTL_{TPN} ::= \exists(\wp_1 U_I \wp_2) \mid \forall(\wp_1 U_I \wp_2) \mid \wp_1 \mapsto_I \wp_2$$
$$\mid \exists\Diamond_I \wp_1 \mid \forall\Diamond_I \wp_1 \mid \exists\Box_I \wp_1 \mid \forall\Box_I \wp_1 \mid \wp_1 \rightsquigarrow_{I_r} \wp_2$$

$\wp_1$ et $\wp_2$ sont des propositions sur les marquages (i.e., $\wp_1, \wp_2 \in PR$). L'index I_r est un élément de $\mathbb{Q}^+_{[\,]}$ de la forme $[0, c]$ ou $[0, c[$, où $c \in \mathbb{Q}^+ \cup \{\infty\}$. La formule $\wp_1 \rightsquigarrow_{I_r} \wp_2$ est une abréviation de la formule $\forall\Box(\wp_1 \Rightarrow \forall\Diamond_I \wp_2)$ qui exprime la réponse bornée. La formule $\wp_1 \mapsto_I \wp_2$ exprime aussi la réponse bornée, mais avec une sémantique

légèrement différente. Informellement $\phi = \wp_1 \mapsto_I \wp_2$ est valide à un état s ssi pour tout chemin d'exécution qui commence à partir de s, si $\wp_1$ devient valide pour la première fois sur ce chemin à l'état s', alors :

1. Si $\wp_2$ n'est pas valide à l'état s' alors $\wp_2$ deviendra fatalement vraie pour la première fois à un état s", dans l'intervalle de temps I (relativement au temps d'occurrence de s'), et ϕ sera récursivement vraie à partir de l'état s",
2. Si $\wp_2$ est vraie à l'état s' alors $\downarrow I$ doit être égale à zéro, et ϕ doit être aussi vraie au prochain état qui ne satisfait pas en même temps $\wp_1$ et $\wp_2$.

Intuitivement, ceci veut dire que sur un chemin d'exécution qui commence à l'état s, si $\wp_1$ est valide pour la première fois sur ce chemin à l'état s', alors $\wp_2$ est forcément vraie pour la première fois sur ce chemin à un état s" qui se trouve dans l'intervalle I relativement à s'. De plus, ϕ doit être récursivement vraie à partir de l'état qui suit s".

Formellement :

-$\mathcal{M}, s \models \wp_1 \mapsto_I \wp_2$ ssi $\forall \rho \in \pi(s)$, tel que $time(\rho) = \infty$, si $(\exists r_1 \geq 0,\ \breve{\rho}(r_1) \models \wp_1$ et $\forall r < r_1,\ \breve{\rho}(r) \not\models \wp_1)$ alors :

1. $(\mathcal{M}, \breve{\rho}(r) \not\models \wp_2) \Rightarrow (\exists r_2, r_2 - r_1 \in I,\ \breve{\rho}(r_2) \models \wp_2,\ \forall r,\ r_1 < r < r_2,\ \breve{\rho}(r) \not\models \wp_2$ et $\mathcal{M}, \breve{\rho}(r_2) \models \wp_1 \mapsto_I \wp_2)$,
2. $(\mathcal{M}, \breve{\rho}(r) \models \wp_2) \Rightarrow (\downarrow I = 0$ et $\exists r_2 > r_1$ tel que $\mathcal{M}, \breve{\rho}(r_2) \models \neg(\wp_1 \wedge \wp_2)$ et $\forall r, r_1 \leq r < r_2,\ \mathcal{M}, \breve{\rho}(r) \models (\wp_1 \wedge \wp_2)$ et $\mathcal{M}, \breve{\rho}(r_2) \models \wp_1 \mapsto_I \wp_2)$.

Dans ce qui suit, $\wp_1 \mapsto_I \wp_2$ sera désignée sous le nom *première réponse bornée.* Le prochain théorème établit que la propriété qui exprime la réponse bornée est un cas particulier de la première réponse bornée.

Théorème 6.1.1 *$\mathcal{M}, s \models \wp_1 \mapsto_{I_r} \wp_2$ ssi $\mathcal{M}, s \models \wp_1 \rightsquigarrow_{I_r} \wp_2$.*

Preuve: Soient $\rho \in \pi(s)$ et $\breve{\rho}$ le chemin d'exécution dense qui lui correspond.

$\Rightarrow$: soit $s_{r_1} = \breve{\rho}(r_1)$ le premier état à partir de s tel que $\wp_1$ est vraie (si s_{r_1} n'existe

pas, la preuve se termine). Soit $s_{r_2} = \breve{\rho}(r_2)$, avec $r_1 \leq r_2$, le premier état en commençant à partir de s_{r_1} tel que $\wp_2$ est vraie (s_{r_2} doit exister par hypothèse). La sémantique de $\wp_1 \mapsto_{I_r} \wp_2$ assure que le temps $\theta = r_2 - r_1$ qui sépare s_{r_1} et s_{r_2} est tel que $\theta \leq b$. Il s'ensuit que $\forall r \leq r_2$, si $s_r = \breve{\rho}(r)$ satisfait $\wp_1$ alors $\exists r' \geq r$, $r' - r \leq b$ et $s_{r'}$ satisfait $\wp_2$ (dans ce cas $s_{r'}$ sera s_{r_2}). La sémantique récursive de $\wp_1 \mapsto_{I_r} \wp_2$ assure aussi que la propriété est aussi vraie pour le sous chemin qui commence à partir de s_{r_2}. En contre partie, ceci assure que $\forall r \geq 0$, si $s_r = \breve{\rho}(r)$ satisfait $\wp_1$ alors $\exists r' \geq r$, $r' - r \leq b$ et $s_{r'}$ satisfont $\wp_2$, ce qui signifie que $\wp_1 \rightsquigarrow_{I_r} \wp_2$ est vraie pour le chemin ρ.

$\Leftarrow$: Soit $s_{r_1} = \breve{\rho}(r_1)$ le premier état à partir de s tel que $\wp_1$ est vraie (si s_{r_1} n'existe pas, la preuve se termine). Soit $s_{r_2} = \breve{\rho}(r_2)$, $r_1 \leq r_2$, le premier état à partir de s_{r_1} tel que $\wp_2$ est vraie. De la sémantique de $\wp_1 \rightsquigarrow_{I_r} \wp_2$ nous déduisons que $r_2 - r_1 \leq b$. Cette propriété est aussi vraie à partir de s_2. En d'autres mots, $\wp_1 \mapsto_{I_r} \wp_2$ est vraie pour le chemin d'exécution ρ. ■

6.2 Model-checking à la volée des propriétés $TCTL_{TPN}$

Le model-checking à la volée d'un modèle TPN $\mathcal{N}$, pour une formule $TCTL_{TPN}$ ϕ, pourrait être réalisé en vérifiant la valeur de vérité de ϕ pendant la construction progressive du LSCG de $\mathcal{N}$. La construction s'arrête dès que la valeur de vérité de ϕ est établie, ce qui permet d'éviter d'explorer la totalité du LSCG dans la majorité des cas. Cette technique a été prouvée très efficace pour vérifier les automates temporisés pour une sous classe de la logique temporisée $TCTL$ similaire à $TCTL_{TPN}$ (Behrmann *et al.*, 2002). Des outils tels que UPPAAL (Behrmann *et al.*, 2002) implémentent cette technique, avec des performances meilleures que celles des techniques standards de vérification par model-checking (Larsen *et al.*, 1995). En premier lieu nous donnons un algorithme pour vérifier la première réponse bornée, puis nous montrons comment adapter cet algorithme pour vérifier le reste des

propriétés $TCTL_{TPN}$.

6.2.1 Model-checking de la première réponse bornée

Soient $\mathcal{N}$ un modèle TPN et $\phi = \wp_1 \mapsto_I \wp_2$ la propriété de la première réponse bornée telle que $I = [a, b]$. Le model-checking de ϕ sur $\mathcal{N}$ peut être réalisé en examinant chaque chemin d'exécution du LSCG de $\mathcal{N}$, jusqu'à ce que la valeur de vérité de ϕ soit établie. Le LSCG est construit progressivement, en profondeur, pendant que $\wp_1$ est vérifiée. Si $\wp_1$ est satisfaite dans une classe d'états α, $\wp_2$ est recherchée sur chaque chemin d'exécution qui commence à partir de α (i.e., $\forall \rho \in \pi(\alpha)$). Pour chaque chemin d'exécution $\rho \in \pi(\alpha)$, $\wp_2$ doit être satisfaite pour la première fois dans une classe d'états α' telle que le temps qui sépare α et α' est dans l'intervalle de temps I. Si c'est le cas, la vérification de ϕ est reprise de nouveau à partir de α', et ainsi de suite, jusqu'à ce que toutes les classes d'états soient explorées. Sinon, l'exploration est arrêtée, avec ϕ déclarée invalide.

Deux situations importantes doivent être résolues avec cette technique. D'un côté, comment mesurer le temps qui sépare le moment où $\wp_1$ est satisfaite et celui où $\wp_2$ est satisfaite. D'un autre côté, comment traiter le cas des chemins infinis qui résultent des cycles. Pour traiter ces deux situations, nous proposons de mettre le modèle TPN $\mathcal{N}$ en parallèle avec le modèle TPN de la figure 6.1, que nous appelons *Alarm-clock* (Hadjidj et Boucheneb, 2006b;Hadjidj et Boucheneb, 2005c). Le modèle TPN qui en résulte sera dénoté $\mathcal{N}||Alarm$, et sera utilisé à la place de $\mathcal{N}$ pour vérifier ϕ. La vérification de ϕ procède maintenant de la manière suivante : Durant la génération du LSCG de $\mathcal{N}||Alarm$, si $\wp_1$ est satisfaite dans la classe d'états $\alpha = (m, F)$, la transition t_a est sensibilisée dans α pour capturer l'évènement qui correspond au début de l'intervalle I. t_a est sensibilisée en changeant le marquage m tel que la place p_a contienne un jeton, et en remplaçant F par $F \wedge t_a = a$

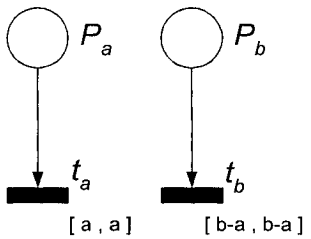


FIGURE 6.1 Le modèle TPN Alarm-clock

(ces deux actions correspondent à mettre un jeton dans la place p_a). Le processus de génération continue à la recherche d'un marquage qui satisfait $\wp_2$. Si $\wp_2$ est satisfaite avant que t_a ne soit franchie, ϕ est déclarée invalide et l'exploration est arrêtée. Lorsque t_a est franchie, (ce qui signifie que l'intervalle de temps I est atteint, et qu'il faut commencer à chercher $\wp_2$), t_b est sensibilisée dans la classe d'états $\alpha' = (m', F')$ qui en résulte, pour capturer l'événement qui correspond à la fin de l'intervalle I. t_b est sensibilisée dans α' en marquant la place p_b, et en remplaçant F' par $F' \wedge t_b = b - a$. Si t_b est franchie durant l'exploration, ϕ est déclarée invalide et l'exploration est arrêtée. Si avant le franchissement de t_b, $\wp_2$ est satisfaite dans une classe d'états $\alpha" = (m", F")$, la transition t_b est désactivée dans $\alpha"$. t_b est désactivée dans $\alpha"$ en enlevant le jeton de la place p_b, et en éliminant la variable t_b de $F"$. Après la modification de $\alpha"$, ϕ est de nouveau testée à partir de $\alpha"$. Notons qu'avec cette démarche, le fait de connaître une classe d'états et la transition qui la générée, est suffisant pour déterminer l'action à entreprendre[2]. Par conséquent la stratégie d'exploration du LSCG (en profondeur, en largeur...) n'est pas importante, ce qui en revanche résout le problème des cycles et des chemins infinis pour tous les modèles TPN bornés[3]. Soient $\alpha = (m, F)$ une classe d'états et t_f la transition qui l'a générée. Les différents cas qui peuvent se produire durant l'exploration sont donnés dans ce qui suit (voir section 6.2.4 pour un exemple d'illustration) :

[2]Pour des raisons d'uniformité, nous supposons l'existence d'une transition fictive t_ϵ comme la transition qui a généré la classe d'états initiale.

[3]Le LSCG est fini pour tous les modèles TPN bornés.

1- Le cas où $t_a, t_b \notin En(m)$ et $t_f \notin \{t_a, t_b\}$ correspond à la situation où on cherche $\wp_1$,
   - Dans le cas où $\wp_1$ est satisfaite dans α alors que $\wp_2$ ne l'est pas, t_a est sensibilisée dans α,
   - Dans le cas où $\wp_1$ et $\wp_2$ sont toutes les deux satisfaites dans α, avec $a \neq 0$, l'exploration est arrêtée et ϕ est déclarée invalide,

2- Le cas où $t_a \in En(m)$ correspond à la situation où $\wp_1$ a été satisfaite avant, et où il faut s'assurer que $\wp_2$ n'est pas satisfaite, à moins que $a = 0$. Si $\wp_2$ est satisfaite dans α avec $a > 0$, l'exploration est arrêtée et ϕ est déclarée invalide,

3- Le cas où $t_f = t_a$ correspond à la situation où l'intervalle de temps I vient d'être atteint, et où il faut commencer à chercher $\wp_2$. Si $\wp_2$ est satisfaite dans α, on se retrouve dans une situation où on cherche $\wp_1$(i.e., (1)), autrement, t_b est sensibilisée dans α,

4- Le cas où $t_b \in En(m)$ correspond à une situation où on est en train de chercher $\wp_2$. Si $\wp_2$ est satisfaite dans α, alors on désensibilise t_b pour tomber dans une situation où on cherche $\wp_1$ (i.e., (1)),

5- Le cas où $t_f = t_b$ correspond à une situation où l'intervalle I est dépassé alors qu'on cherchait à vérifier $\wp_2$. Dans ce cas, on arrête l'exploration et on déclare ϕ invalide.

Notons qu'une certaine prudence est nécessaire lors de la manipulation des transitions t_a et t_b. Si la transition t_a peut être franchie au même instant qu'une autre transition t, et t est franchie avant t_a, φ pourrait être déclarée incorrectement fausse si la classe d'états qui en résulte satisfait $\wp_2$. Une situation similaire pourrait survenir pour la transition t_b si elle est franchie avant une transition t qui peut être franchie à un le même instant. Pour traiter ces deux situations, on associe une haute priorité de franchissement à la transition t_a, de telle sorte qu'elle soit franchie avant toute autre transition qui pourrait être franchie au même instant. À la différence, on

associe une priorité de franchissement basse à t_b, de telle sorte qu'elle soit franchie après toute autre transition qui pourrait être franchie au même instant. Pour traiter ce concept de priorité, on doit compléter la manière de décider si une transition est franchissable ou pas (i.e., opération $isfirable$ décrite par l'algorithme 1), et la manière dont un successeur d'une classe d'état $\alpha = (m, F)$, par une transition t_f, est calculé (i.e., opération $succ$ décrite par l'algorithme 2).

Algorithme 9 $isFirable_{AC}(\alpha = (m, F), t_f)$

1: **si** $t_f \notin En(m)$ **alors**
2: Retourner faux
3: **fin si**
4: **si** $t_a \in En(m) \wedge t_f \neq t_a$ **alors**
5: Soit $F' = F \wedge (\bigwedge_{t \in En(m)-\{t_f,t_a\}} t_f \leq t) \wedge t_f < t_a$
6: **sinon si** $t_b \in En(m) \wedge t_f = t_b$ **alors**
7: Soit $F' = F \wedge (\bigwedge_{t \in En(m)-\{t_f\}} t_f < t)$
8: **sinon**
9: Soit $F' = F \wedge (\bigwedge_{t \in En(m)-\{t_f\}} t_f \leq t)$
10: **fin si**
11: **si** F' est consistante **alors**
12: Retourner vrai
13: **sinon**
14: Retourner faux
15: **fin si**

$isFirable_{AC}(\alpha, t_f)$ remplace $isFirable(\alpha, t_f)$ pour vérifier si une transition est franchissable ou non. Ce qui change est la manière dont la formule F' est calculée. Dans le cas où t_a est sensibilisée et qu'on veut franchir une autre transition t_f (étape 4), on doit faire attention à ce que t_f soit franchie dans un temps qui précède celui du franchissement de t_a. Dans le cas où t_b est sensibilisée, et c'est elle la transition qu'on veut franchir (étape 6), on doit faire attention à ce que t_b est la seule transition qui peut être franchie. Les cas qui restent sont traités exactement comme avant. $succ_{AC}(\alpha, t_f)$ remplace aussi $succ(\alpha, t_f)$ pour la génération des classes d'états durant l'exploration. Ce qui change c'est aussi la manière dont la formule F' est calculée.

Algorithme 10 $succ_{AC}(\alpha = (m, F), t_f)$

1: Soit $m'(p) = m(p) - Pre(p, t_f) + Post(p, t_f), \forall p \in P$
2: **si** $t_f \notin En(m)$ **alors**
3: Retourner faux
4: **fin si**
5: **si** $t_a \in En(m) \wedge t_f \neq t_a$ **alors**
6: Soit $F' = F \wedge (\bigwedge_{t \in En(m) - \{t_f, t_a\}} t_f \leq t) \wedge t_f < t_a$
7: **sinon si** $t_b \in En(m) \wedge t_f = t_b$ **alors**
8: Soit $F' = F \wedge (\bigwedge_{t \in En(m) - \{t_f\}} t_f < t)$
9: **sinon**
10: Soit $F' = F \wedge (\bigwedge_{t \in En(m) - \{t_f\}} t_f \leq t)$
11: **fin si**
12: Remplacer dans F' chaque variable t avec $t + t_f$
13: Éliminer par dans F', t_f et toutes les variables associées avec des transitions en conflit avec t_f pour m
14: **pour chaque** $t \in New(m', t_f)$ **faire**
15: Ajouter à F' la contrainte : $tmin(t) \leq t \leq tmax(t)$
16: **fin pour**
17: Retourner (m', F')

6.2.2 Les algorithmes de model-checking

Le model-checking à la volée d'une formule $TCTL_{TPN}$ ϕ est basé sur la technique d'exploration implantée par l'Algorithme 11.

L'algorithme utilise deux listes : WAIT et COMPUTED, pour gérer les classes d'états, et appelle une fonction de satisfaction polymorphique $checkStateClass_\phi$ pour vérifier la validité de la formule ϕ. COMPUTED contient toutes les classes d'états calculées durant l'exploration, alors que WAIT contient seules les classes d'états qui ne sont pas encore explorées. Par conséquent, WAIT est une sous liste de COMPUTED[4]. L'algorithme 11 génère les classes d'états par franchissement de transitions. La classe d'états initiale est supposée résulter du franchissement d'une transition fictive t_ϵ. À chaque fois qu'une classe d'états α est générée par franchisse-

[4]d'un point de vue implémentation, la liste WAIT est une liste de références à des classes d'états dans la liste COMPUTED.

Algorithme 11 *modelChek*(ϕ)

1: Soit continue=true {variable globale }
2: Soit valid=true {variable globale }
3: Soit COMPUTED= $\emptyset$
4: Soit α_0= (m_0, F_0)
5: Soit α'_0=$checkStateClass_\phi(\alpha_0, t_\epsilon)$
6: Soit WAIT=$\{\alpha'_0\}$
7: **tantque** continue **faire**
8: Retirer $\alpha = (m, F)$ de WAIT
9: **pour chaque** $t \in En(m)$ tel que $isFirable_{AC}(\alpha, t)$ **faire**
10: α' :=$succ_{AC}(\alpha, t)$
11: α" :=$checkStateClass_\phi(\alpha', t)$
12: **si** $\alpha" \neq \emptyset \wedge \nexists \alpha_p \in COMPUTED$ tel que $\alpha" \subseteq \alpha_p$ **alors**
13: **pour chaque** $\alpha_p \in$ COMPUTED tel que $\alpha_p \subseteq \alpha$" **faire**
14: Enlever α_p de COMPUTED et de WAIT
15: **fin pour**
16: Ajouter α" à COMPUTED et à WAIT
17: **fin si**
18: **fin pour**
19: **fin tantque**
20: Retourner valid

ment d'une transition t, α et t sont communiquées à $checkStateClass_\phi$ pour prendre des décisions et effectuer les actions nécessaires. En général, $checkStateClass_\phi$ sensibilise ou désensibilise les transitions t_a et t_b dans α. Elle met aussi à jour deux variables booléennes globales : *continue* et *valid*, pour guider le processus d'exploration. Finalement, elle retourne soit α après modification ou $\emptyset$. $\emptyset$ est retournée si α n'a plus besoin d'être explorée (i.e., ignorée). L'exploration continue tant que *continue* est *vraie*. *valid* est utilisée pour enregistrer l'état de vérité de ϕ. Après que $checkStateClass_\phi$ est appelée, la classe d'états α' retournée est insérée dans la liste WAIT si elle n'est pas incluse dans une classe d'états déjà calculée (i.e., $\nexists \alpha \in$ COMPUTED telle que $\alpha' \subseteq \alpha$). Sinon, α' est ignorée. Si α' est insérée dans la liste WAIT, toutes les classes d'états de la liste COMPUTED qui sont incluses dans α' sont supprimées des deux listes : COMPUTED et WAIT. Cette stratégie qui atténue considérablement le problème d'explosion des états, est justifiée par

le fait que si $\alpha \subseteq \alpha'$ alors $\pi(\alpha) \subseteq \pi(\alpha')$. Ainsi, au lieu d'explorer α et α', seule α' est explorée. L'opération $checkStateClass_\phi$ prend comme paramètres une classe d'états et la transition qui l'a générée.

Trois implémentations différentes de $checkStateClass_\phi$ sont nécessaires pour couvrir les trois formes principales d'une formule $TCTL_{TPN}$, i.e., $\wp_1 \mapsto_I \wp_2$, $\forall(\wp_1 U_I \wp_2)$ et $\exists(\wp_1 U_I \wp_2)$, avec $I = [a, b]$ (la borne b peut être soit finie ou infinie). La première implémentation (algorithme 12, page 93) correspond à $\phi = \wp_1 \mapsto_I \wp_2$. Ces étapes correspondent exactement à celles décrites dans la section 6.2.1. La seconde implémentation (algorithme 13, page 94) correspond à la formule $\phi = \forall(\wp_1 U_I \wp_2)$. Elle vérifie si la proposition $\wp_1$ est vraie dans la classe initiale (étapes 1-7) et toutes ses classes d'états successeurs (étapes 8-12), jusqu'à ce que t_a soit franchie. À partir de l'instant où t_a est franchie, cette implémentation sensibilise la transition t_b (étapes 13-23), puis vérifie continuellement la satisfaction de $\wp_1$ ou $\wp_2$ (étapes 24-32), jusqu'à ce que $\wp_2$ devienne vraie, où que t_b soit franchie (étapes 33-36). Si $\wp_b$ devient vraie dans une classe d'états α, α n'est plus explorée (étapes 22 et 26). La dernière implémentation de $checkStateClass_\phi$ (algorithme 14, page 14) correspond à la formule $\phi = \exists(\wp_1 U_I \wp_2)$. Cette implémentation, comparée à la précédente, initialise la variable globale *valid* à faux aussitôt que l'état initial est atteint (étape 2), et arrête l'exploration d'une classe d'états α si celle ci ne satisfait pas la sémantique de ϕ (étapes 10, 15, 29, 33). Elle arrête aussi l'exploration dès qu'un chemin d'exécution satisfaisant est trouvé (étapes 20-21 et 24-26). Le théorème suivant établit la décidabilité de notre approche de model-checking pour tous les TPNs bornés.

Théorème 6.2.1 *Le model-checking de $TCTL_{TPN}$ est décidable pour tous les modèles TPN bornés.*

Preuve: Si un modèle TPN $\mathcal{N}$ est borné, il a un nombre fini de marquages accessibles. La composition parallèle de $\mathcal{N}$ en parallèle avec le modèle $Alarm-clock$,

qui est lui aussi borné (il a seulement trois marquages $\{(p_a = 0, p_b = 0), (p_a = 1, p_b = 0), (p_a = 0, p_b = 1)\}$), résulte aussi en un modèle TPN borné. Comme le nombre de classes d'états possibles qui partagent le même marquage est toujours fini (Berthomieu et Menasche, 1983), le graphe des classes d'états de $\mathcal{N}||Alarm$ est aussi fini. En d'autres mots, l'algorithme de model-checking de $TCTL_{TPN}$ termine toujours pour les modèles TPN bornés (au pire des cas, toutes les classes d'états accessibles sont explorées). ■

Algorithme 12 $checkStateClass_{\wp_1 \mapsto_I \wp_2}(\alpha = (m, F), t)$

1: **si** $t_a, t_b \notin En(m) \wedge t \notin \{t_a, t_b\}$ **alors**
2: **si** $\wp_1(m) = vrai$ **alors**
3: **si** $\wp_2(m) = faux$ **alors**
4: sensibiliser t_a dans α
5: **sinon**
6: **si** $a > 0$ **alors**
7: valid=faux
8: continue=faux
9: **fin si**
10: **fin si**
11: **fin si**
12: **sinon si** $t_a \in En(m)$ **alors**
13: **si** $\wp_2(m) = vrai)$ **alors**
14: valid=faux
15: continue=faux
16: **fin si**
17: **sinon si** $t = t_a$ **alors**
18: **si** $\wp_2(m) = false)$ **alors**
19: sensibiliser t_b dans α
20: **fin si**
21: **sinon si** $t_b \in En(m)$ **alors**
22: **si** $\wp_2(m) = true$ **alors**
23: désensibiliser t_b dans α
24: **fin si**
25: **sinon si** $t = t_b$ **alors**
26: valid=faux
27: continue=faux
28: **fin si**
29: Retourner α

Algorithme 13 $checkStateClass_{\forall(\wp_1 U_I \wp_2)}(\alpha, t)$

1: **si** $t = t_\epsilon$ **alors**
2: **si** $\wp_1(m) = vrai$ **alors**
3: sensibiliser t_a dans α
4: **sinon**
5: valid=faux
6: continue=faux
7: **fin si**
8: **sinon si** $t_a \in En(m)$ **alors**
9: **si** $\wp_1(m) = faux$ **alors**
10: valid=faux
11: continue=faux
12: **fin si**
13: **sinon si** $t = t_a$ **alors**
14: **si** $\wp_2(m) = faux$ **alors**
15: **si** $\wp_1(m) = faux$ **alors**
16: valid=faux
17: continue=faux
18: **sinon**
19: sensibiliser t_b dans α
20: **fin si**
21: **sinon**
22: Retourner $\emptyset$
23: **fin si**
24: **sinon si** $t_b \in En(m)$ **alors**
25: **si** $\wp_2(m) = vrai$ **alors**
26: Retourner $\emptyset$
27: **sinon**
28: **si** $\wp_1(m) = faux$ **alors**
29: valid=faux
30: continue=faux
31: **fin si**
32: **fin si**
33: **sinon si** $t = t_b$ **alors**
34: valid=faux
35: continue=faux
36: **fin si**
37: Retourner α

Algorithme 14 $checkStateClass_{\exists(\wp_1 U_I \wp_2)}(\alpha, t)$

1: **si** $t = t_\epsilon$ **alors**
2: valid=faux
3: **si** $\wp_1(m) = vrai$ **alors**
4: sensibiliser t_a dans α
5: **sinon**
6: continue=faux
7: **fin si**
8: **sinon si** $t_a \in En(m)$ **alors**
9: **si** $\wp_1(m) = faux$ **alors**
10: Retourner $\emptyset$
11: **fin si**
12: **sinon si** $t = t_a$ **alors**
13: **si** $\wp_2(m) = faux$ **alors**
14: **si** $\wp_1(m) = faux$ **alors**
15: Retourner $\emptyset$
16: **sinon**
17: sensibiliser t_b dans α
18: **fin si**
19: **sinon**
20: valid=vrai
21: continue=faux
22: **fin si**
23: **sinon si** $t_b \in En(m)$ **alors**
24: **si** $\wp_2(m) = true$ **alors**
25: valid=vrai
26: continue=faux
27: **sinon**
28: **si** $\wp_1(m) = faux$ **alors**
29: Retourner $\emptyset$
30: **fin si**
31: **fin si**
32: **sinon si** $t = t_b$ **alors**
33: Retourner $\emptyset$
34: **fin si**
35: Retourner α

6.2.3 Comparaison avec l'algorithme d'atteignabilité de UPPAAL

Comparé à l'algorithme d'atteignabilité implémenté dans l'outil UPPAAL (Behrmann *et al.*, 2002) (voir figure 6.2), notre algorithme d'exploration (i.e., algorithme 11) présente quelques améliorations. Dans UPPAAL l'évolution d'un état symbolique (l'équivalent d'une classe d'états dans notre approche) suit le cheminement suivant : L'état symbolique est en premier lieu créé, inséré dans la liste WAIT, retiré de WAIT, testé pour la satisfaction de la propriété considérée, et finalement inséré dans PASSED s'il n'est pas déjà inclus dans un autre état symbolique. En suivant ces étapes, on peut voir qu'un état symbolique peut être inséré dans la liste WAIT et testé, alors qu'il pouvait être ignoré depuis sa création. En effet, un état symbolique nouvellement calculé pourrait être ignoré s'il est déjà inclus dans un état symbolique anciennement calculé (i.e., un état symbolique de la liste WAIT ou de la liste PASSED). Dans le cas où il n'est pas inclus, tous les états symboliques anciennement calculés qui y sont inclus pourraient être éliminés des deux listes : WAIT et PASSED. Dans notre algorithme d'exploration, nous effectuons ces optimisations, ce qui permet d'avoir un meilleur contrôle sur le problème d'explosion détats. Nous utilisons une seule liste (i.e., la liste COMPUTED) pour sauvegarder toutes les classes d'états telles que aucune n'est incluse dans l'autre. La liste WAIT, contient seulement les classes d'états de COMPUTED qui ne sont pas encore explorées.

6.2.4 Exemple d'illustration

Pour illustrer notre approche de vérification, nous considérons le modèle TPN *cyclic* de la figure 6.3. La propriété $TCTL_{TPN}$ que nous vérifions est $\phi = \wp_1 \rightsquigarrow_{[0,3]} \wp_2$, avec $\wp_1(m) = (m(p_0) = 0)$ et $\wp_2(m) = (m(p_1) = 1)$. Pour des raisons de simplicité, nous avons considéré une modèle TPN cyclique avec un seul chemin d'exécution,

```
PASSED:= {}
WAITING:= {(l̄0, D0)}
repeat
    begin
      get (l̄, D) from WAITING
      if (l̄, D) ⊨ β then return "YES"
      else if D ⊈ D' for all (l̄, D') ∈ PASSED then
          begin
            add (l̄, D) to PASSED
            SUCC:={(l̄s, Ds) : (l̄, D) ⇝ (ls, Ds) ∧ Ds ≠ ∅}
            for all (l̄s', Ds') in SUCC do
                put (l̄s', Ds') to WAITING
          end
    end
until WAITING={}
return "NO"
```

FIGURE 6.2 L'algorithme d'atteignabilité d'UPPAAL

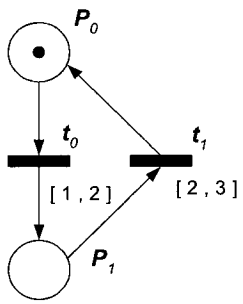


FIGURE 6.3 Modèle TPN *cyclic*

pour lequel la propriété ϕ est trivialement valide.

Le processus de vérification de ϕ commence par construire le modèle TPN $cyclic||Alarm$, tel que $a = 0$ and $b = 3$, puis se déroule selon les étapes suivantes (afin de simplifier les notations, un marquage sera dénoté par seulement ses places qui sont marquées) :

1. Calculer la classe d'états initiale de $cyclic||Alarm$ (étape 4, algorithme 11, page 91),

   résultat : $\alpha_0 = ((p_0 = 1), 1 \leq t_0 \leq 2)$.

2. Vérifier si $\wp_1$ est valide dans α_0, (étape 2, algorithme 12, page 93)
   résultat : $\wp_1$ n'est pas valide dans α_0.
3. Franchir t_0 à partir de α_0 et mettre le résultat dans α_1 (étape 10, algorithme 11, page 91),
   résultat : $\alpha_1 = ((p_1 = 1), 2 \leq t_1 \leq 3)$.
4. Vérifier si $\wp_1$ est valide dans α_1 (étape 2, algorithme 12, page 93)
   résultat : $\wp_1$ est valide dans α_1.
5. Sensibiliser t_a dans α_1 (étape 4, algorithme 12, page 93),
   résultat : α_1 devient $((p_1 = 1, p_a = 1), 2 \leq t_1 \leq 3 \wedge t_a = 0)$.
6. Franchir t_a à partir de α_1 et mettre le résultat dans α_2 (étape 10, algorithme 11, page 91)
   résultat : $\alpha_2 = ((p_1 = 1), 2 \leq t_1 \leq 3)$.
7. vérifier si $\wp_2$ est satisfaite dans α_2 (étape 18, algorithme 12, page 93),
   résultat : $\wp_2$ n'est pas satisfaite dans α_2.
8. Sensibiliser t_b dans α_2 (étape 19, algorithme 12, page 93),
   résultat : α_2 devient $((p_1 = 1, p_b = 1), 2 \leq t_1 \leq 3 \wedge t_b = 3)$.
9. franchir t_1 à partir de α_2 et mettre le résultat dans α_3 (étape 4, algorithme 11, page 91),
   résultat : $\alpha_3 = ((p_0 = 1, p_b = 1), 1 \leq t_0 \leq 2 \wedge 0 \leq t_b \leq 1)$.
10. Vérifier si $\wp_2$ est satisfaite dans α_3 (étape 22, algorithme 12, page 93),
    résultat : $\wp_2$ est satisfaite dans α_3.
11. Désactiver t_b dans α_3 (étape 23, algorithme 12, page 93),
    résultat : α_3 devient $((p_0 = 1), 1 \leq t_0 \leq 2)$.
12. Déclarer ϕ valide puisque α_3 a déjà été explorée ($\alpha_3 = \alpha_0$) (étape 20, algorithme 11, page 91).

CHAPITRE 7

RÉSULTATS EXPÉRIMENTAUX

Nous avons implémenté et testé nos approches de construction de modèles abstraits et de vérification pour le modèle TPN dans notre outil *RT-Studio* (décrit brièvement dans la section 7.1). Tous les résultats présentés dans ce chapitre sont obtenus sur un PC Pentium-4, 3 Gigahertz avec *2 Gigabytes* de *RAM*. Nous commençons par donner une brève description de notre outil, puis nous présentons nos résultats que nous comparons avec ceux d'outils tels que TINA (Berthomieu *et al.*, 2004) et UPPAAL (Behrmann *et al.*, 2002). Les évaluations expérimentales sont présentées selon l'ordre d'introduction de nos résultats dans cette thèse. Dans une première partie, nous évaluons nos différentes approches d'abstraction du modèle TPN introduites dans le chapitre 5. Nous commençons par donner une évaluation de notre implémentation de la k-normalisation (introduite dans la section 5.2), suivie d'une évaluation du graphe des zones d'états concrets (CSZG) et de ses abstractions (introduits dans la section 5.1), puis celle des différentes abstractions du LSCG (introduites dans la section 5.6), et enfin celle de graphe des zones d'états concrets atomique (A-CSZG) (introduit dans la section 5.7). Dans une deuxième partie nous évaluons notre approche de vérification des propriétés temporisées pour le modèle TPN, développée dans le chapitre 6.

7.1 L'outil RT-Studio

RT-Studio (Real Time Studio) est un environnement logiciel composé d'une interface graphique écrite en JAVA (voir figure 7.1), et d'un noyau de calcul écrit en C++. L'outil permet dans sa version actuelle d'éditer et analyser des réseaux de

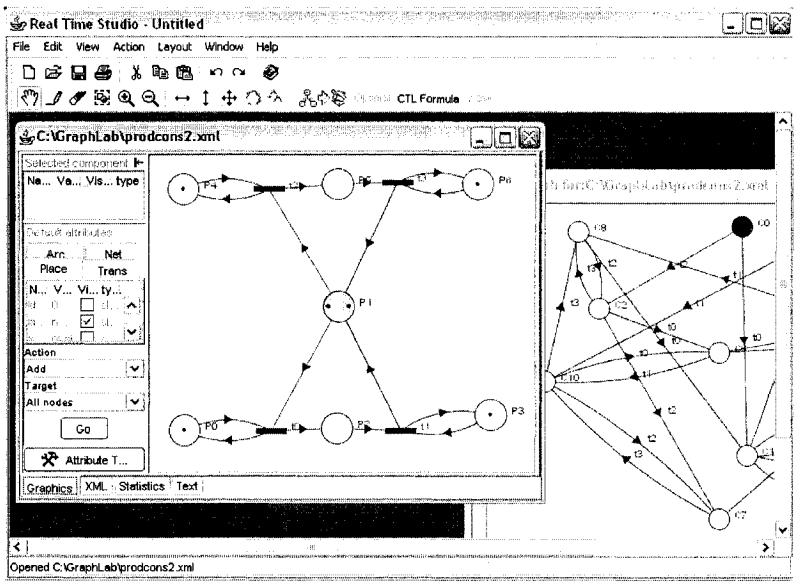


FIGURE 7.1 Notre outil "RT-Studio"

Petri simples et des réseaux de Petri temporels (Merlin et Farber, 1976). En plus des fonctions classiques d'édition, RT-Studio propose la construction des différents modèles d'états abstraits que nous avons proposés dans cette thèse (i.e., CSZG, IC-CSZG, CC-CSZG, CH-CSZG, A-CSZG, IC-LSCG, CC-LSCG et CH-LSCG) et ceux proposées dans (Berthomieu et Vernadat, 2003) (i.e., LSCG, SSCG et ASCG). Il intègre aussi un model-checker CTL, un model-checker pour notre logique $TCTL_{TPN}$ pour vérifier des propriétés temporisées, et un minimiseur par bisimulation[1] (Fisler et Vardi, 1999). Notre outil supporte principalement deux formats de fichier : un format texte (celui de l'outil TINA (Berthomieu *et al.*, 2004)), et un format XML "maison".

[1]Tous les états bisimilaires d'une abstraction du modèle TPN, qui ont les mêmes marquages, sont regroupés dans un seul noeud.

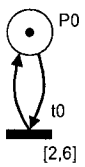


FIGURE 7.2 Modèle TPN simple

7.2 Évaluation des différentes abstractions pour le modèle TPN

Pour couvrir une large gamme de comportements, induisant *concurrence*, *synchronisation* et *séquentiallité*, nous considérons dans nos tests des modèles TPN obtenus en combinant, d'une manière appropriée, des modèles TPN plus simples. La composition parallèle de n copies du modèle simple de la figure 7.2, que nous dénotons par $S(n)$, est un cas de composition parallèle pure. La composition parallèle de *n-1* copies du modèle de la figure 7.3.b avec une copie du modèle de la figure 7.3.a, en fusionnant toutes les places nommées p_1 dans une seule place, que nous dénotons par $P(n)$, est un cas de composition parallèle combinée avec un partage de ressources (synchronisation). Pour des modèles TPN plus significatifs, nous avons repris l'exemple donné dans (Berthomieu et Vernadat, 2003), qui est montré dans la figure 7.3. Cette figure montre une modélisation modulaire du modèle classique du passage à niveau. Le modèle de n trains traversant le passage à niveau, que nous dénotons par $T(n)$, est obtenu par la composition synchronisée du modèle du contrôleur avec son paramètre m[2] initialisé à n, le modèle de la barrière, et n copies du modèle du train.

[2] m représente un nombre de jetons.

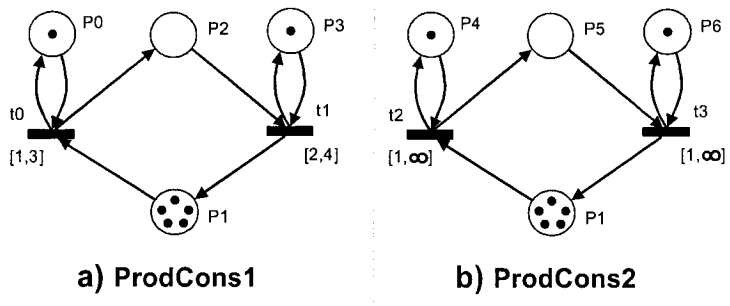


FIGURE 7.3 Modèle Producteur/consommateur

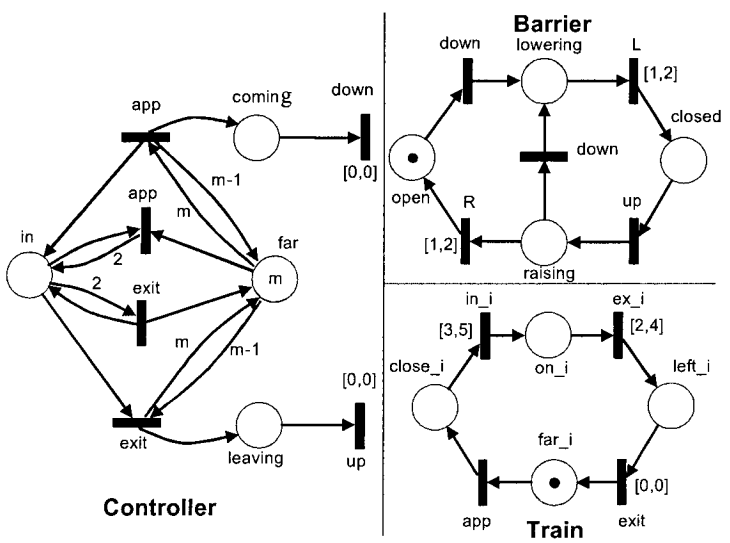


FIGURE 7.4 Modèle TPN du passage à niveau

TABLEAU 7.1 Évaluation de notre implémentation de la k-normalisation

TPN	$norm_k$	$Relax$	$\theta^{Relax}_{norm_k}$	$k-approx$	$\theta^{k-approx}_{norm_k}$
P(2)	1773/6131	7963/42566	6.39	12280/42280	6.90
cpu(s)	0.06	0.51	8.50	0.49	8.16
P(3)	16600/82834	122191/1111887	12.41	?	-
cpu(s)	1.73	20.04	11.58		-
P(4)	68451 / 429014	659377 / 7987583	17.38	?	-
cpu(s)	12.15	183.87	15.13		-
- P(5)	196707 / 1447269	?	-	?	-
cpu(s)	51.04		-		-

7.2.0.1 Notre implémentation de la k-normalisation

Le tableau 7.1 présente une évaluation de notre implémentation de la k-normalisation $norm_k$ (voir section5.2), et compare nos résultats avec ceux de deux autres implémentations, à savoir, la *relaxation* proposée dans (Berthomieu et Vernadat, 2003) pour construire le SSCG et la *k-approximation* proposée dans (Gardey *et al.*, 2003) pour construire le ZBG. Chaque ligne dans le tableau 7.1 donne les résultats obtenus pour le modèle dont le nom est mentionné dans la première colonne. Les colonnes deux, trois et cinq donnent respectivement les tailles (noeuds/arcs) des graphes obtenus pour : $norm_k$, *relaxation* et *k-approximation*. Les colonnes quatre et six donnent les rapports entre les résultats obtenus respectivement pour la *relaxation* et la *k-approximation* et ceux obtenus pour $norm_k$. Un point d'interrogation dans le tableau désigne une situation où le calcul n'a pas abouti après une heure, ou que l'exécution a échouée par manque d'espace mémoire. Notons que pour ce test, nous avons explicitement sélectionné des modèles TPN avec des intervalles de franchissement non bornés. Pour des modèles TPN avec des intervalles de franchissement bornés, la k-normalisation n'a aucun effet.

D'une manière générale, les résultats présentés dans le tableau 7.1 montrent un net écart de performances entre l'utilisation de $norm_k$ par rapport à celui de la

relaxation et la *k-approximation*. Cet écart s'explique pour la *relaxation*, par le recours à la fragmentation des classes d'états afin de préserver la convexité (Gardey *et al.*, 2003). Pour la *k-approximation*, l'utilisation exclusive de la plus grande constante qui apparaît dans le modèle TPN pour borner les contraintes atomiques, induit en général un nombre très grand de zones d'états par marquage[3].

7.2.0.2 Le graphe des zones d'états concrets (CSZG) et ses abstractions

Le tableau 7.2 résume les résultats obtenus pour le SSCG (obtenus avec l'outil Tina) et ceux obtenus pour le CSZG et ses contractions (par inclusion, combinaison convexe, et couverture convexe). Le tableau 7.3 donne les taux d'améliorations de nos résultats par rapport à ceux obtenus pour le SSCG. Elle reprend les résultats de le tableau 7.2 sous forme de rapports des résultats obtenus pour le SSCG (première colonne de le tableau 7.2) et ceux obtenus pour les différentes abstractions. Les deux dernières lignes de le tableau donnent respectivement : le maximum des taux et la moyenne des taux pour chacune des colonnes.

D'une manière générale, les contractions du CSZG sont une fraction de la taille du SSCG correspondant, et calculées en une fraction de temps aussi. Notons que les rapports présentés au tableau 7.3 augmentent généralement avec les tailles des modèles considérés. Comme les contractions par inclusion et combinaison convexe préservent les propriétés d'atteignabilité, elles constituent forcément une meilleure alternative pour vérifier de telles propriétés.

[3]Ce nombre augmente exponentiellement avec la valeur de la constante utilisée.

TABLEAU 7.2 Le SSCG et le CSZG avec ses contractions (par inclusion, par combinaison convexe et par couverture convexe)

TPN	SSCG	CSZG	IC-CSZG	CC-CSZG	CH-CSZG
S(3)	211/594	211/594	6/18	3/9	1/3
cpu(s)	0	0	0	0	0
S(4)	4489/17232	4489/17232	24/107	23/141	1/4
cpu(s)	0.14	0.14	0.01	0.07	0
S(5)	123061/598800	123061/598800	120/695	24/173	1/5
cpu(s)	6.48	6.28	0.37	4.77	0
P(2)	7963/42566	1773/6131	159/611	98/383	21/60
cpu(s)	0.51	0.06	0.01	0.04	0
P(3)	122191/1111887	16600/82834	1261/6991	818/4621	56/210
cpu(s)	20.04	1.73	0.27	1.54	0.01
P(4)	659377/7987583	68451/429014	5502/37999	3569/25467	126/560
cpu(s)	183.87	12.15	3.21	31.88	0.04
P(5)	?	196707/1447269	16701/133280	10517/88210	252/1260
cpu(s)		51.04	19.32	280.25	0.10
T(1)	11/14	11/14	10/13	10/13	10/13
cpu(s)	0.00	0.00	0.00	0.00	0
T(2)	141/254	141/254	41/82	37/74	30/61
cpu(s)	0.01	0.00	0.00	0.00	0
T(3)	5051/13919	5051/13919	232/677	125/351	94/271
cpu(s)	0.20	0.16	0.01	0.01	0
T(4)	351271/1193376	351271/1193376	1807/7091	626/2472	318/1221
cpu(s)	22.88	21.51	0.30	0.43	0.02
T(5)	?	?	18052/89292	8294/45387	1150/5587
cpu(s)			6.57	34.00	0.13

TABLEAU 7.3 Taux d'amélioration de nos abstractions par rapport au SSCG

TPN	SSCG	CSZG	IC-CSZG	CC-CSZG	CH-CSZG
S(3)	1	1.00	33.54	67.08	201.25
cpu	1		-	-	-
S(4)	1	1.00	165.81	132.45	4344.20
cpu	1	1.00	14.00	2.00	> 140.00
S(5)	1	1.00	885.71	3664.27	120310.17
cpu	1	1.03	17.51	1.36	> 6480.00
P(2)	1	6.39	65.62	105.04	623.81
cpu	1	8.50	51.00	12.75	> 510.00
P(3)	1	12.41	149.55	226.89	4639.39
cpu	1	11.58	74.22	13.01	2004.00
P(4)	1	17.38	198.78	297.80	12604.90
cpu	1	15.13	57.28	5.77	4596.75
P(5)	-	-	-	-	-
cpu	-	-	-	-	-
T(1)	1	1.00	1.09	1.09	1.09
cpu	1		-	-	-
T(2)	1	1.00	3.21	3.56	4.34
cpu	1	10.00	10.00	10.00	10.00
T(3)	1	1.00	20.86	39.85	51.97
cpu	1	1.25	20.00	20	200.00
T(4)	1	1.00	173.59	498.59	1003.67
cpu	1	1.06	76.26	53.21	1144.00
T(5)	-	-	-	-	-
cpu	-	-	-	-	-
MAX	1	17.38	885.72	3664.27	120310.17
	1	15.13	76.27	53.21	> 6480.00
MOY	1	4.32	169.78	503.66	14378.48
	1	6.20	40.04	14.76	> 1885.59

7.2.0.3 Les abstractions du graphe de classes d'états linéaires (LSCG)

D'une manière similaire aux tableaux 7.2 et 7.3, les tableaux 7.4 et 7.5 présentent les résultats obtenus pour le LSCG et ses contractions (par inclusion, combinaison convexe, et couverture convexe). D'une manière générale, on retrouve des schémas de performances semblables à ceux obtenus pour les différentes contractions du CSZG, avec la remarque que les contractions du LSCG sont plus petites et plus rapide à calculer. Cependant, ces contraction ne sont pas raffinables, à la différence de celles du CSZG.

TABLEAU 7.4 Le LSCG et ses contractions

TPN	LSCG	IC-LSCG	CC-LSCG	CH-LSCG
S(3)	79 / 210	6 / 21	3 / 9	1 / 3
cpu(s)	0	0	0	0
S(4)	837 / 3032	24 / 132	4 / 16	1 / 4
cpu(s)	0.04	0	0	0
S(5)	10951 / 50570	120 / 850	5 / 25	1 / 5
cpu(s)	0.67	0.05	0.15	0
P(2)	748 / 2460	41 / 137	36 / 122	21 / 60
cpu(s)	0.04	0	0	0
P(3)	4604 / 21891	121 / 581	105 / 492	56 / 210
cpu(s)	0.40	0.01	0.01	0
P(4)	14086 / 83375	275 / 1631	240 / 1365	126 / 560
cpu(s)	1.83	0.03	0.03	0.01
P(5)	31657 / 217423	514 / 3604	485 / 3105	252 / 1260
cpu(s)	5.67	0.07	0.21	0.03
T(1)	11 / 14	10 / 13	10 / 13	10 / 13
cpu(s)	0	0	0	0
T(2)	123 / 218	37 / 74	37 / 74	30 / 61
cpu(s)	0	0	0	0
T(3)	3101 / 7754	172 / 494	121 / 339	94 / 271
cpu(s)	0.13	0.01	0.02	0
T(4)	134501 / 436896	1175 / 4599	419 / 1565	318 / 1221
cpu(s)	8.84	0.16	0.20	0.01
T(5)	?	10972 / 55682	2348 / 14145	1150 / 5587
cpu(s)	> 3600.00	2.04	43.00	0.09

TABLEAU 7.5 Taux d'amélioration des contractions du LSCG

TPN	LSCG	IC-LSCG	CC-LSCG	CH-LSCG
S(3)	1	10.70	24.08	72.25
cpu	-	-	-	-
S(4)	1	24.80	193.45	773.80
cpu	1	> 40.00	> 40.00	> 40.00
S(5)	1	63.42	2050.70	10253.50
cpu	1	13.40	4.47	> 670.00
P(2)	1	18.02	20.30	39.60
cpu	1	> 40.00	> 40.00	> 40.00
P(3)	1	37.74	44.38	99.61
cpu	1	40.00	40.00	> 400.00
P(4)	1	51.13	60.72	142.07
cpu	1	61.00	61.00	183.00
P(5)	1	60.49	69.38	164.74
cpu	1	81.00	27.00	189.00
T(1)	1	1.09	1.09	1.09
cpu	-	-	-	-
T(2)	1	3.07	3.07	3.75
cpu	-	-	-	-
T(3)	1	16.30	23.60	29.74
cpu	1	13.00	6.50	> 130.00
T(4)	1	98.96	288.00	371.28
cpu	1	55.25	44.20	884.00
T(5)	-	-	-	-
cpu	-	> 1988.95	> 105.88	40000.00
MAX	1	98.96	2050.70	10253.50
	1	81.00	61.00	> 40000.00
MOY	1	35.07	252.62	1086.49
	1	> 258.32	> 41.00	> 4726.22

TABLEAU 7.6 Le ASCG et le A-CSZG

TPN	ASCG	A-CSZG	θ^{ASCG}_{A-LSCG}	Optl
S(3)	181 / 1509	181 / 1509	1.00	181 / 1509
cpu(s)	0.19	0.05	3.80	0.20
S(4)	3157 / 36380	3157 / 36380	1.00	3157 / 36380
cpu(s)	35.16	2.97	11.84	12.06
S(5)	?	68341 / 1008235	-	?
cpu(s)	> 3600.00	247.596	> 14.53	?
P(2)	2661 / 28263	2411 / 26138	1.08	2334 / 25046
cpu(s)	8.00	1.01	7.92	94.00
P(3)	35484 / 535101	30828 / 480987	1.11	28319 / 430875
cpu(s)	415.29	35.62	11.66	3500.00
P(4)	?	151384 / 2887295	-	?
cpu(s)	> 3600.00	358.06	> 10.05	
P(5)	?	472940 / 10407836	-	?
cpu(s)	> 3600.00	1993.17	> 1.81	
T(1)	12 / 16	11 / 15	1.07	11 / 15
cpu(s)	0	0	-	0
T(2)	196 / 859	188 / 814	1.05	185 / 786
cpu(s)	0.04	0.01	4.00	0.05
T(3)	6983 / 50044	6918 / 49025	1.02	6905 / 48749
cpu(s)	5.00	1.49	5.18	63.00
T(4)	?	356940 / 3447624	-	?
cpu(s)	> 3600.00	288.21	> 12.49	

7.2.0.4 Le graphe des zones d'états concrets atomique A-CSZG

Le tableau 7.6 compare les résultats obtenus pour le ASCG en raffinant le SSCG, avec ceux obtenus pour le A-CSZG en raffinant le CC-CSZG. La colonne quatre donne les taux d'améliorations de nos résultats par rapport à ceux TINA (i.e., les rapports entre les résultats obtenus pour le ASCG et le A-CSZG). La dernière colonne du tableau donne les tailles des graphes atomiques optimaux suivies des temps de minimisation (sans les temps de raffinement).

D'après les résultats obtenus, on peut noter que les tailles des ASCGs et celles des A-CSZGs sont assez semblables, et assez proches de l'optimal. Cependant, les temps de calculs sont assez différents. On note aussi que la construction du ASCG a échoué pour plusieurs modèles TPN, alors que celle du A-CSZG a réussi. Cette différence de

comportement, que les tableaux n'expliquent pas, est due au schéma de calcul suivi par la procédure de raffinement. Dépendamment du modèle d'états abstrait qui est raffiné (le SSCG ou le CC-CSZG), ce schéma est différent. La figure 7.5 montre des statistiques sur le processus de raffinement du SSCG et du CC-CSZG, pour obtenir respectivement le ASCG et le A-CSZG. Selon ces statistiques, le schéma de raffinement varie considérablement dépendamment du modèle d'états abstraits raffiné. Lorsque c'est le CC-CSZG qui est raffiné, le raffinement suit un schéma presque linéaire[4]. Lorsque c'est le SSCG qui est raffiné, la taille du graphe en cours de raffinement commence à augmenter jusqu'à une taille maximale, puis se met à diminuer jusqu'à ce que le ASCG est obtenu. La taille maximale atteinte durant le raffinement dépend du modèle TPN en cours d'analyse, mais aussi de la taille finale du ASCG. Nous avons constaté qu'en général, plus le ASCG est grand, plus grand est le rapport entre la taille maximale atteinte durant le raffinement et la taille finale du ASCG. Pour certains modèles, ce rapport est tel que la taille maximale croit hors de contrôle, provoquant une forte explosion d'états (voir figure 7.5 S(5)). La même remarque est aussi valable pour les temps de raffinement.

Dans notre procédure de raffinement, deux critères font en sorte que le schéma de raffinement du CC-CSZG en A-CSZG soit linéaire et converge rapidement. D'un côté, le CC-CSZG est assez compact, et ne présente pas une grande redondance d'états comparativement au SSCG. D'un autre côté, notre procédure de raffinement utilise l'abstraction par inclusion, ce qui permet de réduire la redondance des états encore davantage durant le processus de raffinement (voir section 5.7.2 pour plus de détails). Avec la réduction de la redondance des états, la possibilité d'avoir un nombre important de zones d'états, de même marquage et distinctes, est considérablement réduite. Ce qui explique, en contre partie, les résultats obtenus.

[4]La taille du graphe croit d'une manière linéaire durant sa construction

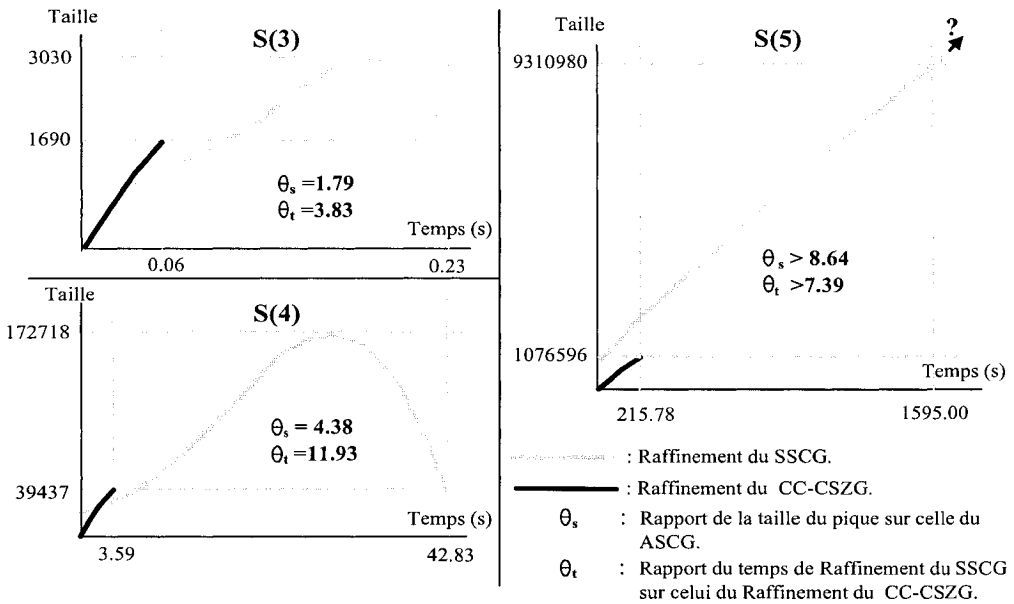


FIGURE 7.5 Schéma de raffinement du SSCG et du CC-CSZG

7.3 Évaluation de l'approche de vérification des propriétés temporisées du modèle TPN

Dans cette section nous rapportons les résultats obtenus pour le modèle du passage à niveau de la figure 7.3. Nous comparons aussi nos résultats avec ceux obtenus avec l'outil UPPAAL 3.5.7. La figure 7.6 montre le modèle automate temporisé du passage à niveau construit au moyen de l'outil UPPAAL. Le modèle du contrôleur a été modélisé comme étant la composition parallèle de trois automates, pour maintenir une sémantique équivalente à celle de la version TPN.

Les propriétés considérées sont :

- La barrière n'est jamais ouverte lorsqu'un train traverse le passage :

$$\phi_1 = \forall\Box\neg(open \wedge \bigvee_{1\leq i\leq n} on\_i)$$

- Si un train approche, la barrière se ferme dans moins de 2 unités de temps :

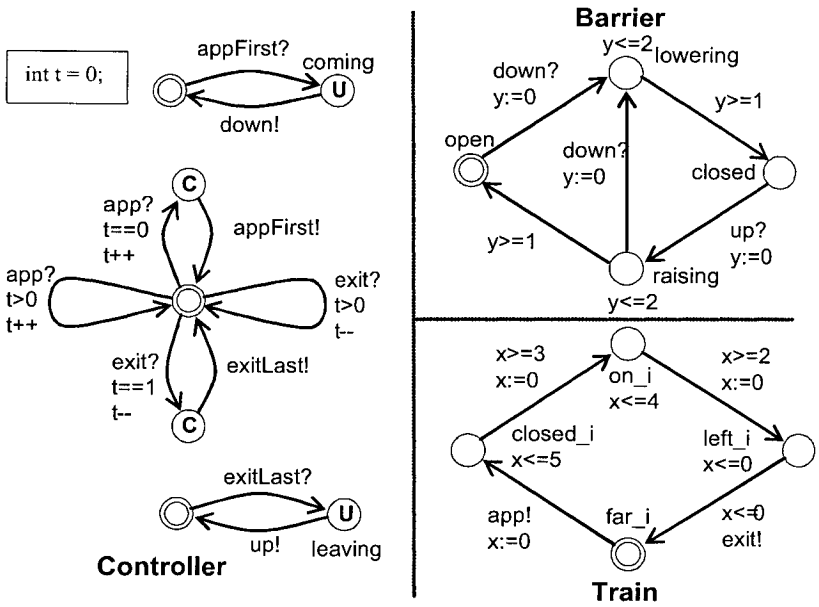


FIGURE 7.6 Le modèle TPN du passage à niveau version TA

$$\phi_2 = coming \rightsquigarrow_{[0,2]} closed$$

- Le modèle du passage à niveau ne bloque jamais :

$$\phi_3 = \forall\Box(\exists t \in En(m))$$

Les propriétés correspondantes pour UPPAAL sont :

- ϕ_1^u = A[] !($barrier$.open && ($train_1.on\_i$||..||$train_n.on\_i$)),
- ϕ_2^u = $ctrlUp$.coming=>($barrier$.closed && barrier.y<=2),
- ϕ_3^u = A[] not deadlock.

Ou $barrier$, $train_i$, $ctrlUp$ sont respectivement des instances de Barrière, *Train* et la partie supérieure du contrôleur de la figure 7.6. Notons que nous avons considéré seulement des propriétés qui nécessitent une exploration exhaustive du graphe des classes du modèle TPN. Pour alléger l'écriture des propriétés, les propositions atomiques coïncident avec les places (i.e., une proposition atomique est vraie ssi la place correspondante est marquée avec au moins un jeton).

Le tableau 7.7 rapporte les résultats obtenus pour le model-checking, avec notre approche, de la propriété sélectionnée. Chaque résultat est donné en terme de *nombre*

TABLEAU 7.7 Propriétés $TCTL_{TPN}$ vérifiées avec notre approche

TPN	ϕ_1	ϕ_2	ϕ_3
2 trains	38 / 116	42 / 91	38 / 116
cpu(s)	0	0	0
3 trains	173 / 790	182 / 646	173 / 790
cpu(s)	0	0.01	0.01
4 trains	1176 / 7162	1194 / 6073	1176 / 7162
cpu(s)	0.12	0.10	0.12
5 trains	10973 / 81370	11008 / 71152	10973 / 81370
cpu(s)	2.37	2.04	2.30
6 trains	128116 / 1103250	128184 / 986939	128116 / 1103250
cpu(s)	110.81	100.92	111.18

de classes d'états sauvegardées / nombre de classes d'états explorées (i.e., la taille finale de la liste COMPUTED (PASSED pour UPPAAL) et le nombre total des classes d'états explorées), suivie du temps d'exploration. Notons que toutes les propriétés sélectionnées ont été trouvées valides.

Le tableau 7.8 rapporte les résultats obtenus en utilisant l'outil UPPAAL pour vérifier les mêmes propriétés. Le tableau 7.9 compare les résultats de le tableau 7.8 avec ceux du tableau 7.7 (i.e, les rapports des résultats d'UPPAAL sur ceux de RT-Studio). Pour les propriétés testées, nos temps de vérification sont en moyenne 13.85 fois plus petits que ceux d'UPPAAL, et la vérification requière 1.48 fois mois d'espace mémoire.

Le tableau 7.10 donne les taux d'améliorations de nos résultats par rapport à ceux d'UPPAAL pour la propriété $\phi(b) = \mathit{coming} \leadsto_{[0,b]} \mathit{closed}$, où b prend des valeurs croissantes. La colonne quatre de ce tableau donne les rapports des résultats d'UPPAAL sur ceux de RT-Studio. Des résultats obtenus, nous pouvons voir que notre approche n'est pas sensible à l'ordre de grandeur de la valeur de b, alors qu'elle l'est dans le cas d'UPPAAL. La raison de cette différence est en rapport avec la caractérisation des états utilisée. Dans notre cas, si la propriété est déjà valide, le fait d'augmenter la valeur du paramètre b n'a aucun effet sur le processus de vérification.

TABLEAU 7.8 Propriétés $TCTL_{TPN}$ vérifiées avec l'outil UPPAAL

TPN	ϕ_1^u	ϕ_2^u	ϕ_3^u
2 trains	36 / 89	54 / 117	38 / 101
cpu(s)	0.01	0.02	0.02
3 trains	176 / 601	344 / 1306	194 / 826
cpu(s)	0.02	0.40	0.30
4 trains	1372 / 5859	3084 / 18243	1456 / 8947
cpu(s)	0.100	0.51	0.49
5 trains	14758 / 74047	35218 / 277612	14408 / 115182
cpu(s)	3.06	36.08	13.72
6 trains	192022 / 1106973	?	173332 / 1692129
cpu(s)	267.88	> 3600.00	806.20

TABLEAU 7.9 Taux d'amélioration par rapport à l'outil UPPAAL

TPN	$\theta_{\phi_1}^{\phi_1^u}$	$\theta_{\phi_2}^{\phi_2^u}$	$\theta_{\phi_3}^{\phi_3^u}$
2 trains	0.81	1.29	0.90
cpu	-	-	-
3 trains	0.81	1.99	1.06
cpu	-	40.00	30.00
4 trains	0.87	2.93	1.25
cpu	0.83	5.10	4.08
5 trains	0.96	3.81	1.40
cpu	1.29	17.69	5.97
6 trains	1.05	-	1.51
cpu	2.42	> 35.67	7.25

TABLEAU 7.10 Taux d'amélioration par rapport à l'outil UPPAAL pour la propriété $\phi(b) = \mathit{Coming} \rightsquigarrow_{[0,b]} \mathit{closed}$, avec des valeurs croissantes de b.

TPN	Ours	UPPAAL	θ_{Ours}^{UPPAAL}
$\phi(2)$	1194 / 6073	3084 / 18243	2.93
cpu(s)	0.10	0.51	5.10
$\phi(10)$	1194 / 6073	16301 / 65792	11.30
cpu(s)	0.10	4.30	43.00
$\phi(20)$	1194 / 6073	32422 / 131420	22.55
cpu(s)	0.10	13.60	136.00
$\phi(30)$	1194 / 6073	48950 / 197212	33.87
cpu(s)	0.10	29.83	298.30
$\phi(50)$	1194 / 6073	84686 / 341231	58.61
cpu(s)	0.10	88.43	884.30
$\phi(100)$	1194 / 6073	173658 / 695112	119.55
cpu(s)	0.10	348.48	3484.80

Pour UPPAAL, les états sont caractérisés au moyen d'horloges. Cette caractérisation nécessite une opération de normalisation pour forcer la terminaison (Bengtsson et Yi, 2003). La taille du *graphe des états symboliques* (l'équivalent du LSCG) à explorer augmente considérablement avec la valeur de la plus grande constante avec laquelle les horloges sont comparées (Bengtsson et Yi, 2003; Alur et Dill, 1990; Behrmann *et al.*, 2002). À titre d'exemple, la dernière ligne du tableau 7.10 montre que pour b=100 nos résultats sont 3485 fois plus rapides à calculer que ceux d'UPPAAL, et nécessitent 120 fois moins d'espace mémoire.

CONCLUSION

Bilan

Dans cette thèse, nous nous sommes intéressés au modèle des réseaux de Petri temporels (modèle TPN) comme formalisme de modélisation, pour lequel nous avons proposé un ensemble d'approches pour la vérification de ses propriétés temporisées et non temporisées. La technique de vérification retenue étant celle dite par *model-checking* (Clarke *et al.*, 1999).

Pour les propriétés non temporisées, nos contributions sont plutôt des propositions qui visent à atténuer le problème de l'explosion des états. Nous avons commencé par définir un contexte formel pour caractériser un modèle d'états abstrait vis-à-vis des propriétés qu'il préserve, puis nous avons proposé un ensemble de modèles d'états abstraits pour le modèle TPN qui préservent ses propriétés d'atteignabilité, linéaires et de branchement. Ces propriétés peuvent par la suite être vérifiées efficacement avec des méthodes classiques de model-checking (Clarke *et al.*, 1999). Nous avons par ailleurs montré que notre contexte formel est plus approprié que celui proposé dans la littérature (Berthomieu et Vernadat, 2003;Cortés *et al.*, 2002;Penczek et Polrola, 2001) pour caractériser les abstractions connues du modèle TPN (Berthomieu et Menasche, 1983; Boucheneb et Hadjidj, 2004; Boucheneb et Hadjidj, 2005; Boucheneb et Berthelot, 1993; Gardey *et al.*, 2003; Hadjidj et Boucheneb, 2005b; Lilius, 1999; Okawa et Yoneda, 1997). La première de nos abstractions, le CSZG (*Concrete State Zone Graph*), préserve les propriétés linéaires du modèle TPN, et peut être raffiné pour restaurer ses propriétés de branchement. Pour forcer la terminaison de la construction du CSZG pour des modèles TPN bornés avec des intervalles de franchissement non bornés, nous avons proposé une implémentation efficace de la k-normalisation (Hadjidj et Boucheneb, 2005a) que nous avons appelée $norm_k$

(Hadjidj et Boucheneb, 2005a). Il est alors devenu possible de construire des abstractions pour certains modèles TPN, qu'on ne peut construire raisonnablement[5] avec les approches existantes(Berthomieu et Vernadat, 2003; Gardey *et al.*, 2003). Pour vérifier les propriétés d'atteignabilité, nous avons proposé deux abstractions pour contracter le CSZG (Hadjidj et Boucheneb, 2005a) et le LSCG (Berthomieu et Menasche, 1983). La première, par inclusion (Boucheneb et Hadjidj, 2006), regroupe toutes les zones d'états (classes d'états) de même marquage, telle que l'une est incluse dans l'autre. La deuxième, par combinaisons convexes (Hadjidj et Boucheneb, 2005a), regroupe des zones d'états (classes d'états) de même marquage, telle que leur union est convexe. Les taux de contractions que procurent ces deux abstractions peuvent facilement dépasser 3000 en tailles et 300 en temps de calcul. Les taux semblent aussi augmenter avec les tailles des modèles considérés. Pour les propriétés de branchement, nous avons proposé un algorithme de raffinement qui utilise l'abstraction par inclusion pour accélérer la convergence (Boucheneb et Hadjidj, 2005). Une zone d'états qui n'est pas atomique est partitionnée en plusieurs zones, qui sont par la suite combinées par inclusion, avec le reste des zones du graphe, afin de réduire la redondance des états. De plus, nous raffinons une contraction du CSZG (par inclusion ou par combinaisons convexes) plutôt que le CSZG lui-même afin de mieux confiner le problème de l'explosion des états. Ces améliorations nous ont permis de construire des modèles d'états abstraits atomiques jusqu'à 10 fois plus rapidement, avec 10 fois moins de ressources mémoires.

Pour les propriétés temporisées du modèle TPN, nous avons proposé une logique temporelle et donné sa sémantique formelle. Pour cette logique, nous avons proposé une méthode de vérification à la volée, basée sur la méthode des classes (Berthomieu et Menasche, 1983), et prouvé sa décidabilité pour tous les modèles TPN bornés (Hadjidj et Boucheneb, 2006b). Pour évaluer notre approche de vérification, nous

[5]Dans des temps et avec des ressources raisonnables.

l'avons comparé avec l'algorithme d'atteignabilité implémenté dans l'outil UPPAAL (Behrmann *et al.*, 2002). D'une manière générale, nous avons réussi à vérifier des modèles TPN, en 7 fois moins de temps que l'outil UPPAAL et avec moins d'espace mémoire. Pour certaines propriétés, les réductions en temps dépassent les 3400, avec plus que 110 fois moins de ressources mémoire. Enfin, pour valider notre travail d'un point de vue pratique, nous avons développé un outil que nous avons appelé *RT-studio*, dans lequel nous avons intégré toutes nos approches.

Perspectives

Dans la continuité de notre travail nous envisageons les points suivant :

- Étendre nos approches, de construction de modèles d'états abstraits et de vérification, au modèle TPN augmenté de variables entières "à la UPPAAL" (Behrmann *et al.*, 2002). Cette extension est nécessaire pour simplifier la modélisation de l'aspect données d'un système, en plus de son aspect contrôle et temporel. Cependant le défi réside dans l'élaboration d'une représentation symbolique pour faire cohabiter des informations temporelles et discrètes, ainsi que des techniques d'abstraction appropriées. Dans ce sens, les progrès réalisés dans la représentation symbolique et le calcul d'espaces d'états de très grandes tailles, pour les réseaux de Petri non temporisés, seraient à considérer (Molinaro *et al.*, 2002; Ciardo, 2004). De même, les techniques d'ordre partiel sont très appropriées dans ce contexte (Godefroid et Wolper, 1994; Valmari, 1992).
- Développer une approche de vérification des propriétés LTL du modèle TPN basée sur les modèles d'états abstraits proposés. Les contractions du LSCG et CSZG par inclusion et combinaison convexe ont la propriété de préserver les propriétés d'atteignabilité du modèle TPN, en plus de contenir toute l'information nécessaire pour retrouver exactement tous les chemins d'exécution possibles. Sur la base de cette information nous développerons une approche de reconstitution des

chemins d'exécution valides, que nous exploiterons pour élaborer une technique de vérification de propriétés linéaires.

- Adapter nos approches aux automates temporisés, et étudier dans quelle mesure des structures de donnée telles que les CDDs (Behrmann *et al.*, 2002; Behrmann *et al.*, 1999) peuvent être exploitées pour améliorer les performances de nos approches.
- Comme le problème de l'explosion d'états reste toujours l'handicap majeur des techniques de vérification par model-checking, nous envisageons aussi étudier les techniques de vérification modulaires qui permettent de transformer la vérification d'un système complexe en preuves sur ses sous-systèmes. La vérification d'un sous-système peut être réalisée soit par énumération (model-checking), soit par déduction à partir de la structure du modèle.

RÉFÉRENCES

AALST, W. V. D. (1993). Interval timed coloured Petri nets and their analysis. *In Proc. of ICATPN'93.* Springer-Verlag, vol. 961 of LNCS, 452–472.

ABDULLA, P. A. ET NYLÉN, A. (2001). Timed Petri nets and BQOs. *In Proc. of ICATPN'01.* Springer-Verlag, vol. 2075 of LNCS, 53–70.

ALUR, R., COURCOUBETIS, C. ET DILL, D. (1993). Model checking in dense real-time. *Information and Computation*, 104(1), 2–34.

ALUR, R., COURCOUBETIS, C., DILL, D., HALBWACHS, N. ET WONG-TOI, H. (1992). An implementation of three algorithms for timing verification based on automata emptiness. *In Proc. of RTSS'92.* IEEE Comp. Soc. Press, 157–166.

ALUR, R. ET DILL, D. (1990). Automata for modelling real-time systems. *In Proc. Of ICALP'90.* Springer-Verlag, vol. 443 of LNCS, 322–335.

ALUR, R., HENZINGER, T. ET HO, P. (1996). Automatic symbolic verification of embedded systems. *IEEE Trans. on Software Eng.*, 22(3), 181–201.

ANDRÉ, C. (1989). Synchronized elementary net systems, advances in Petri nets. *Grzegorz Rosenberg editor.* Springer-verlag, vol. 424 of LNCS.

BANKS, J. (2000). Simulation fundamentals : simulation fundamentals. *In Proc. of the 32nd conference on Winter simulation WSC'00.* Society for Computer Simulation International, San Diego, CA, USA, 9–16.

BEHRMANN, G., BENGTSSON, J., DAVID, A., LARSEN, K. G., PETTERSSON, P. ET YI, W. (2002). UPPAAL implementation secrets. *In Proc. LNCS.* Springer-Verlag, vol. 2469, 3–22.

BEHRMANN, G., LARSEN, K., PEARSON, J., WEISE, C. ET YI, W. (1999). Efficient timed reachability analysis using clock difference diagrams. *In Proc. of CAV'99*. Springer-Verlag, vol. 1633 of LNCS, 341–353.

BENGTSSON, J. ET YI, W. (2003). On clock difference constraints and termination in reachability analysis in timed automata. *In Proc. of ICFEM'03*. Springer-Verlag, vol. 2885 of LNCS, 491–503.

BERTHOMIEU, B. ET DIAZ, M. (1991). Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Software Eng*, 17(3), 259–273.

BERTHOMIEU, B. ET MENASCHE, M. (1982). A state enumeration approach for analyzing time Petri nets. *Applications and Theory of Petri Nets*, 27–56.

BERTHOMIEU, B. ET MENASCHE, M. (1983). An enumerative approach for analyzing time Petri nets. *In Proc. of the IFIP 9th World Computer Congress*. IFIP, North Holland, vol. 9 of Information Processing, 41–46.

BERTHOMIEU, B., RIBET, P.-O. ET 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.

BERTHOMIEU, B. ET VERNADAT, F. (2003). State class constructions for branching analysis of time Petri nets. *In Proc. of TACAS'03*. Springer-Verlag, vol. 2619 of LNCS, 442–457.

BOBBIO, A. ET HORVATH, A. (2001). Model checking time Petri nets using nusmv. *In Proc. of the 5th Int. Workshop on Performability Modeling of Computer and Communication Systems (PMCCS5)*. 100–104.

BOUAJJANI, A., TRIPAKIS, S. ET YOVINE, S. (1997). On-the-fly symbolic model checking for real-time systems. *In Proc. of RTSS'97.* 232–243.

BOUCHENEB, H. ET BERTHELOT, G. (1993). Towards a simplified building of time Petri nets reachability graph. *In Proc. of the 5th Int. Workshop on Petri Nets and Performance Models.* 46–55.

BOUCHENEB, H. ET HADJIDJ, R. (2004). Towards optimal CTL* model checking of time Petri nets. *In Proc. of the International Workshop on Discrete Event Systems (WODES'04).* Reims-France, 469–474.

BOUCHENEB, H. ET HADJIDJ, R. (2005). Using inclusion abstraction to construct atomic state class graphs for time Petri nets. *International Journal of Embedded Systems*, 5.

BOUCHENEB, H. ET HADJIDJ, R. (2006). CTL* model checking for time Petri nets. *Theoretical Computer Science*, 353(1-3), 208–227.

BOUCHENEB, H. ET MULLINS, J. (2003). Analyse de réseaux de Petri temporels. calculs des classes en $o(n^2)$ et des temps de chemin en $(m \times n)$. *Technique et Science Informatiques*, 22(4), 435–459.

BROWNE, M. C., CLARKE, E. M. ET GRÜMBERG, O. (1988). Characterizing finite kripke structures in propositional temporal logics. *Theoretical Computer Science*, 59, 115–131.

BUCCI, G. ET VICARIO, E. (1995). Compositional validation of time-critical systems using communicating time Petri nets. *IEEE transactions on software engineering*, 21(12), 969–992.

CASSEZ, F. ET ROUX, O. H. (2003). Traduction structurelle des réseaux de

Petri temporels vers le automates temporisés. *4ieme Colloque Francophone sur la Modélisation des Systémes Réactifs (MSR'03)*. Hermes Science.

CIARDO, G. (2004). Reachability set generation for Petri nets : Can brute force be smart ? *In Proc. of the 25th International Conference on Application and Theory of Petri nets, (ICATPN 2004)*. Springer-Verlag, Bologna, Italy, vol. 3099 of LNCS, 17–34.

CLARKE, E. ET EMERSON, E. (1981). Design and synthesis of synchronization skeletons using osium branching-time temporal logic. *In Proc. of Workshop on Logic of Programs*. Springer-Verlag, vol. 131 of LNCS, 52–71.

CLARKE, E. M., FILKORN, T. ET JHA, S. (1993). Exploiting symmetry in temporal logic model checking. *In Proc. of CAV'93*. Lecture Notes in Computer Science, vol. 697, 450–462.

CLARKE, E. M., GRUMBERG, O. ET LONG, D. E. (1994). Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16, 1512–1542.

CLARKE, E. M., GRUMBERG, O. ET PELED, D. (1999). Model checking. *Model Checking*. MIT Press, Cambridge, MA.

COOLAHAN, J. ET ROUSSOPOULOS, N. (1983). Timing requirements for time-driven systems using augmented Petri nets. *IEEE Trans. on Software Eng*, SE-9(5), 603–616.

CORTÉS, L. A., ELES, P. ET PENG, Z. (2002). Verification of real-time embedded systems using Petri net models and timed automata. *In Proc. of the 8th Int. Conf. on Real-Time Computing Systems and Applications (RTCSA'02)*. 191–199.

DAWS, C., OLIVERO, A., TRIPAKIS, S. ET YOVINE, S. (1996). The tool kronos. *In Proc. of Hybrid Systems III, Verification and Control.* Springer-verlag, vol. 1066 of LNCS, 208–219.

DEMBINSKI, P., PENCZEK, W. ET POLROLA, A. (2002). Verification of timed automata based on similarity. *Fundamenta Informaticae*, 51(1-2), 59–89.

DIAZ, M. ET SENAC, P. (1994). Time stream Petri nets a model for timed multimedia information. *In Proc. of the 15th International Conference on Application and Theory of Petri Nets.* Springer-verlag, Zaragoza (Spain), vol. 815 of LNCS, 219–238.

DILL, D. (1989). Timing assumptions and verification of finite state concurrent systems. *In Proc. of Automatic Verification Methods for Finite-State Systems.* Springer-Verlag, vol. 407 of LNCS, 197 – 212.

EMERSON, E. A. ET SISTLA, A. P. (1996). Symmetry and model checking. *Formal Methods in System Design*, 9, 105–131.

E.VICARIO (2001). Static analysis and dynamic steering of time dependent systems. *IEEE Transactions on Software Engineering*, 27(8), 728–748.

FISLER, K. ET VARDI, M. Y. (1999). Bisimulation and model checking. 1703 of LNCS, 338–341.

FLOYD, R. W. (1962). Acm algorithm 97 : Shortest path. *Communications of the ACM*, 5(6), 345.

GARDEY, G., ROUX, O. H. ET ROUX, O. F. (2003). Using zone graph method for computing the state space of a time Petri net. *In Proc. of FORMATS'03.* Springer-Verlag, vol. 2791 of LNCS, 246–259.

GODEFROID, P. ET WOLPER, P. (1994). A partial approach to model checking. *Information and Computation*, 110(2), 305–326.

GRUMBERG, O. ET LONG, D. (1991). Model checking and modular verification. vol. 527 of LNCS, 250–265.

GU, Z. ET SHIN, K. (2002). Analysis of event-driven real-time systems with time Petri nets. *In Proc. of DIPES'02.* Kluwer, vol. 219 of IFIP, 31–40.

HAAR, S., KAISER, L., SIMONOT-LION, F. ET TOUSSAINT, J. (2000). On equivalence between timed state machines and time Petri nets. *Technical Report RR-4049.* INRIA. France.

HADJIDJ, R. ET BOUCHENEB, H. (2005a). Improving state class constructions for CTL* model checking of time Petri nets. *International Journal on Software Tools for Technology Transfer (accepted).*

HADJIDJ, R. ET BOUCHENEB, H. (2005b). Much compact time Petri net state class spaces useful to restore CTL* properties. *In Proc. of the Fifth International Conference on Application of Concurrency to System Design (ACSD'05).* IEEE Computer Society Press, 224–233.

HADJIDJ, R. ET BOUCHENEB, H. (2005c). On the fly TCTL model checking for time Petri nets using the state class method. *ACSD'06 (accepted).*

HADJIDJ, R. ET BOUCHENEB, H. (2006a). Efficient reachability analysis for time Petri nets. *ACM Transactions on Software Engineering and Methodology (submited).*

HADJIDJ, R. ET BOUCHENEB, H. (2006b). On the fly TCTL model checking for time Petri nets. *Theoretical Computer Science (submited).*

HANISCH, H.-M. (1993). Analysis of place/transition nets with timed arcs and its application to batch process control. *In Proc. of ICATPN'93.* Springer-Verlag, vol. 691 of LNCS, 282–299.

HULGAARD, H. ET BURNS, S. M. (1995). Efficient timing analysis of a class of Petri nets. *In Proc. of CAV'95.* Springer-Verlag, vol. 939 of LNCS, 923–936.

KUPFERMAN, O., HENZINGER, T. A. ET VARDI, M. Y. (1996). A space-efficient on-the-fly algorithm for real-time model checking. *In Proc. of CONCUR'96.* Springer-Verlag, vol. 1119 of LNCS, 514–529.

LARSEN, K. G., PETTERSSON, P. ET YI, W. (1995). Model-checking for real-time systems. *In Proc. of Fundamentals of Computation Theory.* vol. 965 of LNCS, 62–88.

LARSEN, K. G., WEISE, C., YI, W. ET PEARSON, J. (1999). Clock difference diagrams. *Nordic Journal of Computing*, 26(3), 271–298.

LILIUS, J. (1999). Efficient state space search for time Petri nets. *In Proc. of MFCS Workshop on Concurrency, Brno'98.* Elsevier Science Publishers, vol. 18 of ENTCS, 113–133.

LIME, D. ET ROUX, O. H. (2003). State class timed automaton of a time Petri net. *In Proc. of the 10th Int. Workshop on Petri Nets and Performance Models (PNPM'03).* IEEE Comp. Soc. Press, 124–133.

MERLIN, P. ET FARBER, D. J. (1976). Recoverability of communication protocols - implication of a theoretical study. *IEEE Trans. on Communications*, 24(9), 1036–1043.

MOLINARO, P., DELFIEU, D. ET ROUX, O. (2002). Improving the calculus of

the marking graph of Petri nets with BDD-like structures. *In Proc. of IEEE international conference on systems, man and cybernetics (SMC'02)*. IEEE Computer Society Press, vol. 1, 43–48.

MOSKEWICZ, M., MADIGAN, C., ZHAO, Y., ZHANG, L. ET S.MALIK (2001). Chaff : Engineering an efficient sat solver. *In Proc. of the 38th Design Automation Conference (DAC'01)*. 530–535.

NICOLA, R. D. ET VANDRAGER, F. (1995). Three logics for branching bisimulation. *Journal of the ACM*, 42(2), 458–487.

OHTA, A. ET TSUI, K. (2000). Turing machine equivalence of time asymmetric choice nets. *IEICE Trans. on Fundamentals in Electronics, Communications and Computers*, E83-A(11), 2278–2281.

OKAWA, Y. ET YONEDA, T. (1997). Symbolic CTL model checking of time Petri nets. *Electronics and Communications in Japan*, 80(4), 11–20.

PAIGE, R. ET TARJAN, R. (1987). Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6), 973–989.

PENCZEK, W. ET POLROLA, A. (2001). Abstractions and partial order reductions for checking branching properties of time Petri nets. *In Proc. of ICATPN'01*. Springer-Verlag, vol. 2075 of LNCS, 323–342.

PENCZEK, W. ET POLROLA, A. (2004). Specification and model checking of temporal properties in time Petri nets and timed automata. *In Proc. of of ICATPN'04*. 37–76.

PENCZEK, W., POLROLA, A., WOZNA, B. ET ZBRZEZNY, A. (2004). Bounded model checking for reachability testing in time Petri nets. *Proc. of CSP'04*.

124–135.

PENCZEK, W., WOZNA, B. ET ZBRZEZNY, A. (2002). Towards bounded model checking for the universal fragment of TCTL. *In Proc. of FTRTFT'02.* Springer-Verlag, vol. 2469 of LNCS, 265–288.

PETTERSSON, P. (1999). Modelling and verification of real-time systems using timed automata : Theory and practice. *Ph.D. Theses.* Uppsala University.

POLROLA, A. ET PENCZEK, W. (2004). Minimization algorithms for time Petri nets. *Fundamenta Informaticae*, 60, 307–331.

POLROLA, A., PENCZEK, W. ET SZRETER, M. (2003). Reachability analysis for timed automata using partitioning algorithms. *Fundamenta Informaticae*, 55(2), 203–221.

QUEILLE, J. ET SIFAKIS, J. (1981). Specification and verification of concurrent systems in CESAR. *In Proc. of the Fifth International Symposium in Programming.* Springer-Verlag, vol. 137, 337–351.

RAMCHANDANI, C. (1974). Analysis of asynchronous concurrent systems by timed Petri nets. *Technical Report MAC-TR-120.* Massachusets Institute of Technology.

ROKICKI, T. G. (1993). Representing and modeling digital circuits. *Ph.D. Theses.* Stanford University.

RUSU, V., MARCHAND, H. ET JERON, T. (2005). Automatic verification and conformance testing for validating safety properties of reactive systems. *In Proc. of the International Symposium of Formal Methods Europe (FM 2005 : Formal Methods).* Newcastle, UK, vol. 3582 of LNCS, 189 – 204.

SIFAKIS, J. (1977). Use of petri nets for performance evaluation. *In Proc. of the Third International Symposium IFIP W.G. 7.3., Measuring, modelling and evaluating computer systems (Bonn-Bad Godesberg, 1977).* Elsevier Science Publishers, Amsterdam, 75–93.

SIFAKIS, J. ET YOVINE, S. (1996). Compositional specification of timed systems. *In Proc. of STACS'96.* Springer-Verlag, vol. 1046 of LNCS, 347–359.

S.YOVINE (1993). Methodes et outils pour la verification symbolique de systemes temporises. *Thèse de Doctorat.* Institut Nationale Polytechnique de Grenoble,France.

TARASYUK, I. V. (1998). Tau-equivalences and refinement. *In Proc. of International Refinement Workshop and Formal Methods Pacific - 98 (IRW/FMP'98).* 110–128.

TOUSSAINT, J., SIMONOT-LION, F. ET THOMESSE, J. P. (1997). Time constraint verifications methods based time Petri nets. *In Proc. of the 6th Workshop on Future Trends in Distributed Computing Systems (FTDCS'97).* Tunis, Tunisia, 262–267.

TRIPAKIS, S. ET YOVINE, S. (2001). Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1), 25–68.

TSAI, J., YANG, S. ET CHANG, Y. (1995). Timing constraint Petri nets and their application to schedulability analysis of real-time system specifications. *IEEE Trans. On Software Eng*, 21(1), 32–49.

VALMARI, A. (1992). A stubborn attack on state explosion. *Formal Methods in System Design*, 1(4), 297–322.

VIRBITSKAITE, I. B. ET POKOZY, E. A. (1999). A partial order method for the verification of time Petri nets. *In Proc. of Fundamental of Computation Theory.* Springer-Verlag, vol. 1684 of LNCS, 547–558.

WALTER, B. (1983). Timed Petri nets for modelling and analysing protocols with real-time characteristics. *In Proc. of the 3rd IFIP Workshop on Protocol Specification, Testing and Verification.* North Holland, 149–159.

YONEDA, T. ET RYUBA, H. (1998). CTL model checking of time Petri nets using geometric regions. *IEICE Trans. Inf. and Syst.*, 3, 1–10.