N° d'ordre : 06/2008-M/IN

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE
"HOUARI BOUMEDIENE"
FACULTÉ D'ÉLECTRONIQUE ET D'INFORMATIQUE

MEMOIRE

Présenté pour l'obtention du diplôme de MAGISTER

EN : INFORMATIQUE

Spécialité : Informatique Mobile

Par : OUAZAR FATIHA

SUJET

Vérification distribuée des systèmes temps réel

Soutenu le 07/07/2008, devant le jury composé de :

Mr- N. BADACHE,	**Professeur,**	**USTHB,**	**Président**
Mr- M.C. BOUKALA,	**Chargé de cours,**	**USTHB,**	**Directeur de Mémoire**
Mme- M. BOUKALA,	**Professeur,**	**USTHB**	**Examinatrice**
Mme- S. MOUSSAOUI,	**Docteur,**	**USTHB,**	**Examinatrice**
Mr- Y. HAMMAL,	**Chargé de cours,**	**USTHB,**	**Invité**

Table des matières

Résumé

Les systèmes temps réel sont des systèmes dotés d'un comportement qui est contraint par le temps, un tel système doit réagir correctement avec son environnement non seulement au regards des informations échangées mais également aux instants auxquels ces interactions se réalisent. En effet la vérification de tels systèmes nécéssite un modèle temporel formel pour la modélisation tel que les réseaux de Petri temporels (Time Petri Net,TPN), une logique temporelle temporisée pour exprimer les propriétés à vérifier sur ce modèle et un modèle checker pour decider si la propriété est vérifiée sur le système ou non. Cependant la vérification par model-checking est souvent confrontée au problème de l'explosion combinatoire due à la taille du système qui est exponentielle en nombre d'état et en nombre d'horloges du système.

Pour pallier à ce problème, plusieurs approches peuvent être suivies, telles que la représentation des systèmes par des structures de données plus compactes (BDD), les symétries,la distribution... Dans ce travail nous nous interéssons à la technique de vérification à la volée du model-checking basée sur une platforme distribuée, en utilisant la méthode des zones sur les réseaux de Petri temporels pour la génération de l'espace d'état. Ceci permet d'augmenter la mémoire principale de stockage et de distribuer la charge de calcul sur une grappe d'ordinateurs.

Mots clés : réseaux de Petri temporel, model-checking, model-checking à la volé, vérification, vérification distribuée, zones.

Introduction générale

Les systèmes temps réel ont pris une grande importance dans la vie de la societé moderne, ils sont utilisés pour effectuer des tâches et fonctions critiques **[Gsh02]** (centrale nucléaire, l'aéronautique au travers des systèmes de pilotage embarqués: avions,satellites...). Ces systèmes se distinguent des autres systèmes informatiques par le fait qu'ils sont soumis à des contraintes temporelles. Il est donc impératif d'avoir des outils et techniques de modélisation et de vérification qui puissent garder un haut degré de performance et de correction pour un système temps réel. Il ne faut pas perdre de vue que dans un tel système le temps représente un facteur important. Par conséquent il doit être pris en considération d'une manière explicite durant la phase de modélisation et de vérification.
Il existe plusieurs outils et techniques de modélisation des systèmes informatiques (files d'attente, chaines de Markov,...), dont plusieurs ont pu prouver leurs puissance et performances dans la modélisation et la vérification des systèmes.
Parmi les outils courament utilisés pour modéliser les systèmes, les réseaux de Petri (RdP) occupent une place priviligiée. Le formalisme des RdP a été introduit dans les années soixante par C.A.Petri **[Pet62].** Les réseaux de Petri sont un outil trés utilisé car ils ont montré une grande puissance d'expression et de calcul pour la modélisation et la vérification de nombreux et différents types de systèmes, et ils fournissent des méthodes d'analyse avec lesquelles on peut vérifier certaines propriétés de systèmes (bornitude, vivacité,...).

À l'origine, les RdPs ne prenaient pas en considération le caractère temporel des systèmes. Ils se contentaient d'effectuer une modélisation et une vérification du comportement logique d'un système hors son contexte temporel. Afin de pallier à cela et pour pouvoir profiter de la puissance des RdPs, des extensions temporelles des RdPs ont été proposées. Le but principal de telles extensions est de pouvoir effectuer une modélisation et une vérification du système sur le plan logique du séquencement des évènements, et surtout sur le plan temporel primordial dans le cadre de la représentation des systèmes temps-réel. Parmi ces extensions, nous trouvons les réseaux de Petri temporels (Time Petri Nets TPN).

L'idée fondamentale des réseaux de Petri temporels est d'associer des intervalles de temps [a,b] à chaque transition. Une transition ne peut être tirée que si elle est sensibilisée de façon continue au moins "a" unités de temps. Si une transition est sensibilisée continuellement pendant "b" unités de temps , elle doit être tirée immédiatement car elle a atteint son délai maximum de sensibilisation. De plus,la durée de tir d'une transition est toujours nulle **[Boy01].**

La large utilisation des réseaux de Petri temporels a été concrétisée par la conception de plusieurs outils qui les implémentent. On retrouve par exemple les outils ***ROMÉO [GRR05], TINA*** **[Had01]**, etc.

En 1990 Alur et Dill ont notamment proposé le modèle des automates temporisés, pour décrire le comportement des systèmes intégrant des contraintes quantitatives sur le temps.

Les automates temporisés contiennent des horloges qui évoluent de manière continue avec le temps et qui mesurent ainsi les délais séparant les différentes actions du système modélisé.

L'utilisation des automates temporisés comme modèle de description des systèmes temps-réel a été concrétisé par la conception de plusieurs outils efficaces

de model-checking tels que ***UPPAAL*** **[Lar97], [Pet00],** ***KRONOS*** **[Yov97] et** ***CMC*** **[Lar98]**. Des applications temps-réel industrielles ont été spécifiées et vérifiées avec succés en utilisant ces outils.

Les propriétés générales telles que la bornitude, la vivacité, le non blocage ... renseignent le modélisateur sur le comportement général du système, celles-ci doivent être complétées par l'analyse des propriétés spécifiques du système modélisé.

Plusieurs axes de recherche ont été développés visant à mettre en place des outils permettant une description formelle des propriétés spécifiques des systèmes informatiques. La logique temporelle initiée par EMERSON & AL **[Eme96],** est un outil formel qui procure une syntaxe sûre, précise et sans ambiguïté des propriétés qualitatives et quantitatives.

La logique temporelle se distingue selon deux axes:

— Soit on présente l'ensemble des évènements comme un arbre où les successeurs d'un état sont obtenus par les instances d'évènements possibles en cet état; on parle alors de logique temporelle arborescente (CTL Computation Tree Logic) **[Cla86]**.
— Soit on considère l'ensemble des éxécutions comme des séquences distinctes; on parle alors de logique temporelle linéaire (LTL Linear Temporal Logic) **[Man91]**.

Pour pouvoir exprimer des propriétés spécifiques des systèmes temps-réel c-à-d des propriétés quantitatives, des extensions temporelles des différentes logiques temporelles ont été proposées, telle que la logique TCTL (Timed CTL) à laquelle nous nous interéssons.

Aprés la description des propriétés spécifiques, viens l'étape vérification de ces propriétés sur le système.

Généralement la vérification formelle des propriétés spécifiques des systèmes nécessite les quatre éléments suivants **[Yov93]**:

1. Un outil de description (modélisation) de systèmes, tels que les RdP.
2. Un langage de description des propriétés spécifiques tel qu'une logique temporelle.
3. Une relation de satisfaction qui définit formellement la comparaison entre le programme à vérifier et sa spécification.
4. Un algorithme de décision pour conclure si la propriété est vérifiée sur le système ou non.

Cette étape consiste à vérifier si celui-ci satisfait ses spécifications, c'est-à-dire de s'assurer que les propriétés qu'il possède correspondent aux propriétés attendues.

Parmi les techniques de vérification: les approches par model-checking, se sont largement développées ces dernières années**.** Cela suppose que le comportement du système à vérifier soit modélisé et que la propriété de correction attendue est énoncée sous la forme d'une formule de logique temporelle,ensuite on peut utiliser un model-checker afin de vérifier si le modèle satisfait ou non la propriété.

Néanmoins, la vérification et la validation des systèmes temps réel souffrent de l'explosion combinatoire non seulement à la taille du système, mais aussi au nombre d'horloges.

Plusieurs approches peuvent être suivies pour pallier à ce problème, parmi ces approches, les techniques de réduction telles que la vérification à la volée, qui vérifie les propriétés en même temps que l'exploration, la vérification modulaire **[Pet05]**, la représentation symbolique telle que les diagrammes de decision, nous citons les approches distribuées, aussi celles qui évitent de présenter tous les états du système en prenant en compte certaines propriétés du modèle telles que la symétrie **[Jor99]**.

Nous nous intéressons à la vérification des systèmes temps réel en utilisant des modèles temporisés (RdP temporels et automates temporisés) pour la modélisation de ce type de systèmes, et d'effectuer une vérification sur une plate-forme distribuée pour éviter le problème de l'explosion combinatoire.

Nous utilisons une technique basée sur l'idée de faire coopérer plusieurs stations (site) ayant chacune son propre processeur et sa propre mémoire. Cette technique permet d'effectuer des traitements parallèles et d'exploiter les espaces mémoire de l'ensemble des machines.

La technique offre deux principaux avantages:

- pouvoir augmenter la taille des modèles étudiés, en partitionnant l'espace d'états entre les mémoires des différentes stations.

- minimiser le temps global de la procédure de vérification en distribuant les algorithmes de génération et de vérification.

Dans ce contexte, nous allons étudier les deux étapes principales de la vérification:

- La première étape consiste à générer de façon parallèle et distribuée l'espace d'états du système à vérifier.

- La seconde étape consiste à vérifier à la volée par des algorithmes distribués les propriétés spécifiques du système modélisé exprimées en logique temporelle TCTL.

Nous nous intéressons surtout à la vérification de la sous classe de propriétés TCTL vérifiable à la volée sur les réseaux de Petri temporels sur une plate forme distribuée.

Plan de ce mémoire:

Le premier chapitre portera sur un modèle qui permet de décrire les systèmes temps-réel: les automates temporisés.

Dans le deuxième chapitre seront définis les principaux concepts des RdPs temporels qui sont utilisés pour la modélisation des systèmes temps-réel.

Le chapitre 3 portera sur la méthode de vérification: le model-checking

Nous proposons un algorithme d'exploration d'espace d'états de manière distribuée basé sur la notion des zones proposée dans **[GRR03][Gar05]**, ainsi qu'une vérification à la volée d'une classe de propriétés exprimées en logique temporelle TCTL dans le chapitre 4.

Nous exposons les résultats obtenus dans le chapitre 5.

Nous terminerons par une conclusion générale dans laquelle nous présentons le bilan et les perspectives de ce travail.

Chapitre 1

Les automates temporisés

1.1. Introduction

Les automates temporisés, définis par alur et dill en 94**[Alu94]**, sont des modèles qui permettent de définir le comportement d'un système temporisé. Un tel système contient des contraintes temporelles qui sont définies grâce à des horloges réelles ($\in \mathbb{R}$), celles-ci permettent de forcer une action du système à être exécutée dans un intervalle de temps fini ou non.

Dans ce chapitre nous présentons le modèle des automates temporisés, les différentes notions de base et quelques variantes.

1.2. Les automates temporisés

Un automate temporisé est un système à états finis possèdant un ensemble fini d'horloges évoluant au même rythme (de façon synchrone). Les noeuds du graphe sont appelés les localités, et les arcs sont appelés les transitions.

1.2.1. Définition [Bou05]

Les automates temporisés ont été introduits par R. Alur et D.Dill dans les années 90. Il s'agit d'automates classiques munis d'horloges qui évoluent de manière continue et synchrone avec le temps. Chaque transition contient une garde (sur la valeur des horloges) décrivant quand la transition peut être exécutée et un ensemble d'horloges qui doivent être remises à zéro lors du franchissement de la transition. Chaque état de contrôle contient un invariant (une contrainte sur les horloges) qui peut restreindre le temps d'attente dans l'état et donc de forcer l'exécution d'une transition d'action. Le domaine de temps peut être l'ensemble des entiers naturels ($\mathbb{N}$), l'ensemble des rationnels positifs ($\mathbb{Q}^+$) ou bien même l'ensemble des réels positifs ($\mathbb{R}^+$).

Notation:

Soit X l'ensemble d'horloges à valeur dans $\mathbb{R}_{\geq 0}$. Une valuation v pour X est une fonction (v: $X \rightarrow \mathbb{R}_{\geq 0}$) qui associe à chaque horloge x sa valeur v(x). On note $\mathbb{R}^{\mathbb{X}}{}_{\geq 0}$ l'ensemble des valuations pour X. étant donné un réel d $\in \mathbb{R}_{\geq 0}$, on note v+d la valuation qui associe à l'horloge x la valeur v(x) +d. si Y est un sous ensemble de X, [Y $\leftarrow$ 0] v représente la valuation v' définie par: v'(x)=0 pour tout x$\in$Y et v'(x)=v(x) pour x $\in$X / Y.

On note C(X) l'ensemble des contraintes d'horloges (gardes) sur X, c-à -d: l'ensemble des combinaisons booléennes de contraintes atomique de la forme x $\sim$ c avec x$\in$ X et, $\sim \in$ {<, >, =, $\leq$, $\geq$ } et c $\in \mathbb{N}$. Lorsque une valuation satisfait une contrainte g, on note v $\models$ g.

1.2.2. Définition formelle [Lar05]

Un automate temporisé est un 7-uplets (Q, q_0, X, Σ, $\rightarrow$, Inv, l) avec:

— Q est un ensemble fini d'états de contrôle (ou de localités).
— $q_0 \in$Q est l'état initial.

— Σ est un alphabet d'actions.
— X est un ensemble fini d'horloges (à valeurs réelles positives).
— $\rightarrow_A \subseteq Q \times C(X) \times \Sigma \times 2^X \times Q$ est un ensemble fini de transitions;

e=<q, g, a, r, q'> $\in \rightarrow_A$ représente une transition de q vers q', g est la garde (contrainte) associée à e, a $\in \Sigma$ est l'étiquette de la transition, et r est l'ensemble d'horloges devant être remises à zéro.
- Inv : Q→C (X) associe un invariant à chaque état de contrôle.
- I : Q $\rightarrow 2^{AP}$ étiquette les états de contrôle par les propositions atomiques vérifiées.

Exemple:

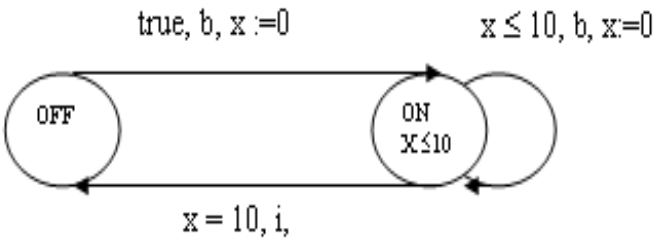


Fig 3: automate temporisé

L'exemple de la figure 3 représente un automate temporisé. OFF et ON sont les deux localités de l'automate et x est une horloge, $x \leq 10$ (dans l'état ON) est un invariant, $x \leq 10$ est une garde. La transition de OFF à ON réinitialise l'horloge x à 0. Cet automate décrit une minuterie: on part de la localité initiale OFF, à tout moment on peut appuyer sur le bouton (action b) et passer dans l'état ON avec l'horloge x remise à zéro. Le temps peut s'écouler tant que la valeur de l'horloge x

est inférieure à 10 (à tout moment une pression sur le bouton permet de remettre x à zéro) et lorsque x=10, la transition i ramène l'automate dans l'état OFF.

Un état ou une configuration d'un automate temporisé est une paire (q, v) $\in Q \times \mathbb{R}^{X+}$ où q représente l'état de contrôle courant et v une valuation pour les horloges **[Bou06].**

La sémantique d'un automate temporisé est définie sous la forme d'un système de transition temporisé où les transitions correspondent soit à l'écoulement du temps, soit à l'exécution d'une action.

1.2.3. Sémantique des automates temporisés [Lar05]

La sémantique d'un automate temporisé A= (Q, q_0, X, Σ, $\rightarrow$, Inv, l) est définie par le système de transition temporisé (STT) T_A =(S, s_{init}, $\rightarrow$, l):

— $S = Q \times \mathbb{R}^{\mathbb{X}+}$,
— $s_{init}=(q_0, v_0)$ avec $v_0(x) = 0$, $\forall x \in X$,
— $\rightarrow$ correspond à deux types de transitions:

Les transitions d'actions: $(q,v) \rightarrow^a (q',v')$ ssi $\exists q \rightarrow^{g,a,r} q' \in \rightarrow_A A$ et $v \models g$, $v'=v[r \leftarrow 0]$ et $v' \models Inv(q')$.

Les transitions de temps: soit $t \in \mathbb{R}^+$. $(q,v) \rightarrow^t (q, v+t)$ ssi $v+t' \models Inv(q)$ pour tout $0 \leq t' \leq t$.

— I(q, v) = I(q) pour tout q.

Informellement le système part de la configuration initiale (état de contrôle q_0 et toutes les horloges à zéro), puis effectue deux types de transitions:
- En laissant le temps s'écouler d'une durée d dans $\mathbb{R}^+$: la valeur de toutes les horloges est alors incrémentée d'une durée d, menant à une valuation d'horloge v+d. on passe alors d'une valuation (s,v) à (s, v+d). On parle alors de transition de délai.

- En activant une transition du système. Les horloges devant être réinitialisées sur la transition prennent une valeur nulle, les valeurs des autres horloges restent inchangées. Il s'agit d'une transition discrète.

Exemple:

Dans l'automate de la figure 3, (ON, x = 5) est un état de l'automate temporisé.
- transition de délai: (ON, x = 5) $\rightarrow^{d}$(ON, x = 7) avec d=2.
- transition discrete: (ON, x=10) $\rightarrow^{i}$(OFF, x=10). La transition (ON, i, x=10, OFF) est exécutée.

Pour la modélisation des systèmes temps réel, le modèle des automates temporisés constitue un modèle approprié. Mais l'espace d'états d'un AT est infini car le temps est modélisé par une variable dense. Pour certaines études, il peut être trés pratique de transformer un AT en un automate à états finis afin d'obtenir une représentation finie de son espace d'états **[Kho06]**. L'avantage de cette approche est que l'on peut alors utiliser des méthodes d'études applicables aux automates à états fini. La transformation de l'automate temporisé en un automate de région AR répond à cette nécessité.

1.3. Le graphe des régions

La présence des valuations d'horloges produit un nombre infini de configurations pour un automate temporisé. Il est donc nécessaire d'utiliser des techniques particulières pour vérifier des propriétés classiques comme l'accessibilité ou plus généralement des propriétés décrites avec des logiques temporelles **[Lar05]**.
La technique de graphe des régions, proposée par Alur et Dill pour analyser le comportement des automates temporisés est très utilisée. Le problème de l'accessibilité d'un état de contrôle est donc le suivant : Étant donné un AT A et un état de contrôle q_F , est-ce que q_F est atteignable depuis la configuration initiale s_{init}? Pour

cela, on remplace ce problème d'accessibilité sur le système de transitions infini AT par un problème d'accessibilité sur un graphe fini (l'automate des régions) RA. Les états de RA correspondent à des ensembles de configurations de A vérifiant les mêmes propriétés d'accessibilité. Plus précisément ses états sont des éléments de Q×$(\mathbb{R}_+{}^X)_{/\sim}$où $\sim$ est une équivalence sur les valuations d'horloges telle que (1) $(\mathbb{R}_+{}^X)_{/\sim}$ est fini et (2) pour tout u et v avec u $\sim$ v, on a : (q, u) et (q, v) permettent d'atteindre les mêmes états de contrôle. Notons que pour avoir les mêmes états accessibles, il suffit de pouvoir exécuter les mêmes transitions d'action (donc de satisfaire les mêmes gardes), d'arriver dans des configurations équivalentes (aprés les remises à zéro) et de pouvoir simuler les mêmes transitions de temps.
L'équivalence $\sim$ dépend de la constante maximale notée M apparaissant dans les gardes et les invariants de l'automate considéré.

La construction repose sur une relation d'équivalence d'index fini définie sur l'ensemble des configurations **[Jou00]**, et telle qu'à partir de deux configurations équivalentes, les mêmes comportements soient possibles.

1.3.1. Définition [Bou05]

Deux configurations (q, v) et (q', v') sont équivalentes si q=q' et si v $\equiv_M$v' où M est la constante maximale apparaissant dans l'automate et v$\equiv_M$v' lorsque pour toute horloge x,
1) v(x) > M $\Leftrightarrow$ v'(x) > M.
2) si v(x) $\leq$ M, alors $\llcorner v(x) \lrcorner$ = $\llcorner v'(x) \lrcorner$ et ({v(x)}=0 $\Leftrightarrow$ {v'(x)}=0) , ($\llcorner \alpha \lrcorner$: la partie entière de α et $\{\alpha\}$ la partie fractionnaire).
et pour toute les paires d'horloges (x, y),
3) si v(x) $\leq$ M et v(y) $\leq$ M, alors {v(x)} $\leq$ {v(y)} $\Leftrightarrow$ {v'(x)} $\leq$ {v'(y)}.
Intuitivement, les deux premières conditions expriment que deux valuations équivalentes satisfont les mêmes contraintes de l'automate, la dernière condition exprime qu'à partir de deux configurations équivalentes, l'écoulement du temps permettra aux horloges d'atteindre de nouvelles valeurs entière dans le même

ordre. L'équivalence $\equiv_M$ est l'équivalence des régions, une classe d'équivalence étant alors appelée une région.

1.3.2. Définition

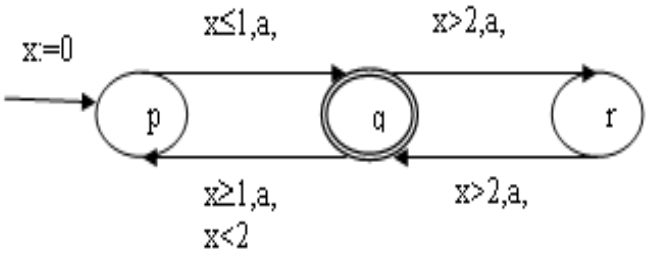
Soit X un ensemble fini d'horloges, C$\subseteq$ C (X) un ensemble de contraintes et R un ensemble fini de régions. Alors R est compatible avec C si on a :
$\forall \Psi \in$ C, $\forall \alpha \in$ R, ($\alpha \models \Psi$) ou ($\alpha \models \neg\Psi$).

La notion de compatibilité entre un ensemble de régions et des contraintes sur les horloges permet de s'assurer que les régions ne regroupent que des valuations qui ont des comportements similaires face aux contraintes d'horloges que l'on se donne.

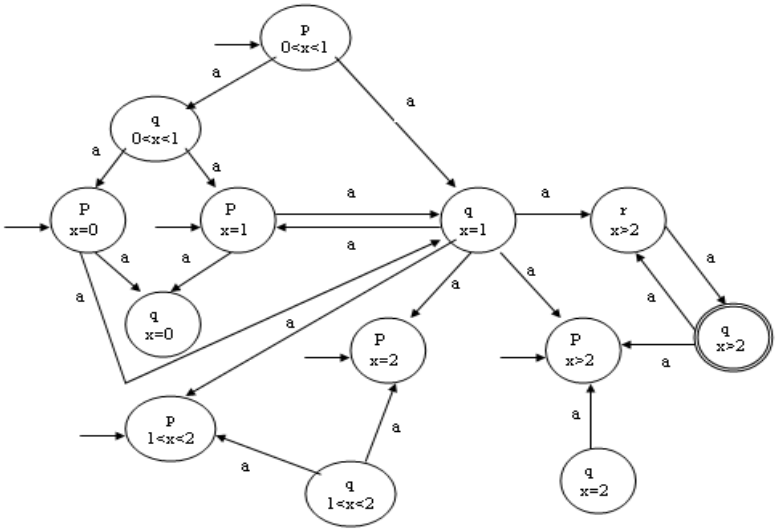
À partir de l'automate temporisé initial et de cette relation d'équivalence, on construit un automate fini de la façon suivante:
Les états de l'automate sont les paires (q, R) où q est un état de l'automate temporisé et R une région comme construite ci-dessus; les transitions sont (q, R) $\longrightarrow^a$ (q', R') s'il existe une transition q$\xrightarrow{g,a,Y:=0}$ q' dans l'automate A, une valuation v $\in$R et t $\geq$ 0 tels que v+t $\models$ Inv(q), v+t $\models$ g, [Y$\leftarrow$ 0](v+ t) $\models$ Inv(q') et [Y$\leftarrow$ 0](v+ t)$\in$ R'.

L'automate fini résultant R_A est appelé l'automate des régions associé à l'automate temporisé initial. La propriété fondamentale de cet automate fini est qu'il reconnaît exactement l'ensemble des mots a1,a2,... tels qu'il existe un mot temporisé (a1, t1), (a2, t2),... reconnu par l'automate initial.

Exemple [Jou00]:



Automate temporisé



Automate des régions

Remarque: la complexité du graphe des régions est de l'ordre de $|X|!.\ 2^{2|X|}.\prod_{y\in X}(c_y+1)$ où:
Xest le nombre d'horloges.

1.4. Extensions des automates temporisés

Pour représenter plus facilement des systèmes, des extensions des automates temporisés ont été considérées.

1.4.1. Contraintes diagonales

Dans le modèle des automates temporisés que nous avons présenté, les contraintes sur les horloges qui sont autorisées sont très simples et permettent uniquement de comparer la valeur d'une horloge à une constante. Un autre type de contraintes était utilisé, les contraintes dites diagonales, qui permettent de faire des tests de type x - y $\sim$ c où x et y sont des horloges, $\sim$ est un signe de comparaison et c une constante entière. Le modèle des automates temporisés utilisant ce type de contraintes a les propriétés suivantes:
- l'ajout de contraintes diagonales n'augmente pas l'expressivité des automates temporisés.
- L'ajout des contraintes diagonales apporte de la concision **[Bou05]**.
La décidabilité de l'accessibilité de cette extension est aussi basée sur la construction d'un ensemble des régions comme vu précédemment .

1.4.2. Les contraintes additives [Bou05]

D'autres types de contraintes peuvent être ajoutées aux automates temporisés, nous allons considérer les contraintes dites additives à savoir celles de la forme

x+y$\sim$c, où x et y sont des horloges, $\sim$ est un signe de comparaison et c un entier positif. Cette extension des automates temporisés permet d'exprimer des langages qui ne sont pas reconnus par des automates temporisés classiques **[Bér00]**.
Ce modèle d'automates temporisés avec contraintes additives satisfait les propriétés suivantes **[Bér00]**:
- L'accessibilité dans les automates temporisés avec contraintes additives utilisant **deux horloges** est un problème décidable.
- L'accessibilité dans les automates temporisés avec contraintes additives utilisant **quatre horloges** est un problème indécidable.

La décidabilité du modèle utilisant deux horloges provient d'une extension de la construction de l'automate des régions, mais à partir de quatre horloges, le modèle devient indécidable. En ce qui concerne les automates temporisés avec contraintes additives munis de trois horloges , la décidabilité de l'accessibilité est un problème ouvert, mais il a été montré qu'il n'est pas possible de leur associer un automate des régions fini **[Rob04]**.

1.4.3. Les mises à jour

Dans le modèle original, les seules opérations autorisées sur les horloges sont les remises à zéro. Il est naturel de considérer des opérations un peu plus compliquées sur les horloges. Une mise à jour est une opération de la forme **[Bou02]** x :$\sim$ c ou x :$\sim$ y +c où x et y sont des horloges, $\sim$ est un signe de comparaison et c est un entier. Par exemple, la mise à jour x :$\leq$ c signifie que l'on affecte à x, de manière non déterministe, une valeur inférieure ou égale à la constante c; la mise à jour x := y - 1 indique que l'on affecte à x la valeur de y décrémentée de 1. les remises à zéro classiques correspondent aux mises à jour x :=0 .
L'accessibilité dans les automates temporisés [Bou04]:
- avec mises à jour de la forme x := c est décidable.
- avec auto-incrémentation et sans contrainte diagonale est décidable (auto-incrémentation: mises à jour de la forme x:=x+1).

- avec auto-incrémentation et contrainte diagonale est indécidable.
- avec auto-décrémentation est indécidable (auto-decrémentation: mises à jour de la forme: x:=x-1).

1.4.4. Les automates hybrides linéaires

Les automates hybrides linéaires étendent les automates temporisés par:
- Des contraintes linéaires générales (par exemple: 3x1 + 4x2 - 2x3 <78),
- Des mises à jour plus riches que les simples remises à zéro: on peut associer des fonctions affines sur les variables X à chaque transition.
- Des pentes différentes pour les variables selon les états: au lieu d'avoir des horloges, on dispose de variables dynamiques.

En fait, chacune de ces extensions prise séparément rend déjà l'ensemble des problèmes de vérification indécidables **[Hen98]**. Nous l'avons vu pour les contraintes additives et les mises à jour étendues. C'est aussi le cas pour les variables avec des pentes différentes: l'accessibilité est indécidable dans les automates où une variable peut avoir deux pentes différentes dans le système **[Ras05]**.

1.5. Conclusion

Nous avons présenté dans ce chapitre, les notions de base des automates temporisés, qui sont des automates à états fini munis d'horloges qui évoluent de manière continue et synchrone avec le temps.

Le modèle des automates temporisés est convenable pour modéliser un système temps réel, mais il n'est pas convenable pour l'étudier et pour vérifier les propriétés spécifiques de ce système.

En effet, les méthodes d'étude de tels systèmes sont en général basées sur l'analyse de leurs espaces d'états et par conséquent, exigent que cet espace d'états soit fini. Or l'espace d'états d'un automate temporisé est infini car tout état de l'automate temporisé est défini par (q, v) où q est une localité (état) de l'automate et v une valuation pour les horloges. Comme les horloges prennent leurs valeurs dans $\mathbb{R}^+$, le nombre de valeurs prises par chaque horloge est infini. Ainsi l'espace d'états d'un automate temporisé est infini.

La transformation de l'automate temporisé en un automate à états fini engendre une explosion combinatoire de l'espace d'états, mais la transformation est nécessaire pour réaliser l'étude du système temps réel. Une méthode de transformation de l'automate temporisé en automate à états finis est la transformation en automate des régions.

La transformation de l'automate temporisé en automate des régions est difficilement applicable aux systèmes temps réel complexes car elle induit une explosion de l'espace des états .

Dans le prochain chapitre, nous présentons un autre modèle pour la modélisation des systèmes temps réel, il s'agit des réseaux de Petri temporels.

Chapitre 2

Les réseaux de Petri temporels

2.1. Introduction

À la fin des années 80, plusieurs techniques ont été proposées pour décrire des systèmes de plus en plus complexes d'une manière complète et non ambigue, ces techniques reposent en général sur des modèles puissants qui permettent d'assurer une vérification à priori de ces systèmes.

L'objet de ce chapitre est de donner les différents formalismes servant à décrire le fonctionnement des systèmes, dans lesquels le temps apparait comme un paramètre quantifiable et continu.

Système temps réel [Loh02]

Un système temps réel est un système informatique doté d'un comportement qui est contraint par le temps, à la différence de systèmes non temporisés, un système temps réel doit interagir correctement avec son environnement non seule-

ment au regard des informations échangées, mais également au regard des instants auxquels ces interactions se réalisent.

Plusieurs modèls ont été proposés pour spécifier et vérifier de tels systèmes. Parmi ceux-ci, deux ont été développées à partir des réseaux de Petri: il s'agit d'une part des réseaux temporisés (ou Timed Petri Nets) **[Ram74]**, et d'autre part des réseaux temporels (ou Time Petri Nets) **[Mer74]**.

Les réseaux temporisés sont obtenus à partir des réseaux de Petri en associant une durée de tir à chaque transition. De plus, les transitions doivent être tirées dès qu'elles sont sensibilisées. Ces réseaux, et divers modèles analogues, sont essentiellement utilisés pour l'analyse des performances.

2.2. Les RdP temporels

Les réseaux de Petri temporel sont utilisés pour modéliser les systèmes où le temps joue un rôle critique. On les obtient en associant, à chacune des transitions, un intervalle de temps [a,b]. Ce type de réseaux exprime nativement la notion de Délai. En explicitant les débuts et fins d'actions, ils peuvent exprimer la notion de Durée.

2.2.1. Définition

Un réseau de Petri temporel est défini comme étant un réseau de Petri ordinaire où on associe à chaque transition deux dates : min et max ($0 \leq \min \leq$ max, max

est éventuellement infini) **[Ber 01].** L'intervalle ainsi obtenu, [min, max], est relatif à la date θ où la transition t a été sensibilisée. Dans ce cas, la transition t ne peut être franchie ou tirée avant la date min+θ, et elle doit être tirée au plus tard à la date max+θ, sauf bien sûr si elle est désensibilisée par une autre transition. La durée de tir d'une transition est supposée nulle **[Bbd 03].**

2.2.2. Définition formelle

Un réseau de Petri temporel (TPN) est un tuple (P,T,Pré,Post,M_0,IS)
où (P,T,Pré,Post,M_0) est un réseau de Petri ordinaire.
IS : T$\rightarrow$ $Q^+\times$ ($Q^+\cup\infty$) est une fonction d'intervalle statique qui associe à chaque transition t du réseau un intervalle à bornes rationnelles IS (t)= [α,β], avec $0 \leq \alpha \leq \beta$ (β peut être infini).
La fonction IS associe à chaque transition du réseau un intervalle à bornes rationnelles IS(t). La plus petite borne de cet intervalle est appelée date de tir au plus tôt de t (notée SMin (t)). La plus grande borne est appelée date de tir au plus tard de t (notée SMax(t)).

Dans un réseau temporel, franchir une transition t sensibilisée n'est permis que dans l'intervalle de tir qui lui est associé. Cet intervalle est relatif à la date de sensibilisation de la transition. Initialement, et si t est sensibilisée par le marquage initial, l'intervalle de tir de t correspond à son intervalle statique IS (t). Mais avec l'évolution du réseau, l'intervalle de temps associé à cette transition va également évoluer : il est décalé, vers l'origine des temps, d'une quantité égale à la durée écoulée depuis la sensibilisation de la transition. Cet intervalle « dynamique » est exprimé avec une application **I**. Elle fait correspondre à chaque transition un nouvel intervalle de temps **I**(t), dans lequel elle peut être tirée. Les bornes de cet intervalle sont désignées comme étant les dates au plus tôt DMin (t), et au plus tard DMax (t) de franchissement de t **[Bbd 03]**.

Exemple1:
La figure suivante représente un réseau de Petri temporel :

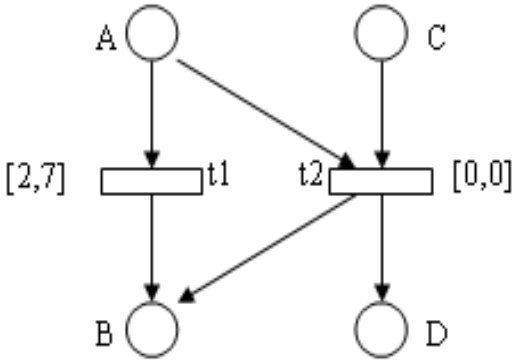


Figure1 : réseau temporel

Dans cet exemple lorsque un jeton arrive dans la place A, il sensibilise la transition t1, qui pourra être franchie dans un délai compris entre 2 et 7 unités de temps; cependant si un jeton arrive dans la place C avant que 2 unités de temps ne se soient écoulées depuis la sensibilisation de t1, alors c'est t2 qui va être franchie.

2.2.3. Sémantique des RdP temporels

La sémantique des TPN peut être donnée en terme de systèmes de transitions temporisés (TTS) **[Lar95]** qui sont des systèmes de transitions classiques avec deux types d'étiquettes: des étiquettes discrètes pour les évènements et les étiquettes réelles positives pour l'écoulement du temps.

La sémantique d'un TPN T= (P, T, Pré, Post, M_0, **IS**) est un système de transitions temporisé S_T=(Q, q_0,$\rightarrow$) avec:
- Q = $\mathbb{N}^P \times(\mathbb{R}_{\geq 0})^n$,avec n =|T |; v $\in(\mathbb{R}_{\geq 0})^n$ est une valuation telle que la valeur v_i représente le temps écoulé depuis le dernier instant où la transition t_i a été sensibilisée.
- q_0 =(M_0, 0).
- $\rightarrow\in$Q$\times$ (T $\cup\mathbb{R}_{\geq 0}$) $\times$ Q est la relation de transition composée des transitions discrètes et continues suivantes:

- la relation de transition discrète est définie comme suit:

$$\forall t_i \in T,(M,v)\rightarrow^{t_i}(M',v') \text{ssi} \begin{cases} M \geq Pre(\bullet, t_i) \wedge M' = M - Pre(\bullet, t) + Post(\bullet, t) \\ Dmin(t_i) \leq v_i \leq Dmax(t_i) \\ v'_k = \begin{cases} 0 & si \uparrow enabled(t_k, M, t_i) \\ v_k & sinon \end{cases} \end{cases}$$

- La relation de transition continue est définie comme suit:

$$\forall d \in \mathbb{R}_{\geq 0},(M,v)\rightarrow^{e(d)}(M,v') \text{ssi} \begin{cases} \forall k \in [1..n], (M \geq Pre(\bullet, t_k) \Rightarrow v'_k \leq Dmax(t_k) \\ v' = v + d \end{cases}$$

$\uparrow$enabled (t_k, M, t_i) est vrai si t_k est sensibilisée par le tir de la transition t_i à partir du marquage M, et faux dans le cas contraire. Une transition t_k est nouvellement sensibilisée par le tir d'une transition t_i à partir du marquage M si " elle n'est pas sensibilisée par M- Pre (p, t_i) et elle est sensibilisée par M'= M- Pre(p, t_i)+ Post(p, t_i)" **[Ber91]**. De plus la transition t_k est nouvellement sensibilisée par son propre tir. Formellement:
$\uparrow$enabled (t_k, M, t_i) = (M- Pre($\bullet$, t_i)+ Post($\bullet$, t_i) $\geq t_k$) $\wedge$ (M- Pre ($\bullet$, t_i)$< t_k$)$\vee$ (t_k=t_i).

Une exécution d'un TPN *T* est un chemin dans S_T à partir de q_0. L'ensemble des exécutions de *T* est notée [*T*]. Un marquage M est accessible dans *T* ssi il existe une exécution $(M_0, 0) \Rightarrow^{\sigma} (M, v)$ dans [*T*]. L'ensemble des marquages accessibles de *T* est noté reach (*T*). Si l'ensemble reach (*T*) est fini, le réseau *T* est alors borné.

2.2.4. Définition des états d'un TPN et règles de tir [Bbd 03]

L'état d'un réseau temporel est un couple E = (M,I), où M est un marquage du réseau et I est la fonction intervalle de tir qui :
- Associe, à chaque transition t sensibilisée, son intervalle de tir.
- Elle assigne l'intervalle vide pour les transitions non sensibilisées.

L'état initial est défini par $E_0 = (M_0, I_0)$ où M_0 est le marquage initial. I_0 est la fonction qui associe à chaque transition t sensibilisée par le marquage initial son intervalle de tir statique **IS** (t).

La définition des états permet de déterminer les conditions de franchissement d'une transition t. t est franchissable dans un état E (M,I) à une date relative θ si :
1) t est sensibilisée par le marquage M (condition standard des réseaux de Petri ordinaires: $M \geq Pre(t)$).
2) θ appartient à l'intervalle de tir de t, I (t) = [DMin,DMax].
3) Aucune autre transition n'est tirable avant t. En d'autres termes:
$\forall k \in T, M \geq pre(k) \Rightarrow \theta \leq DMax(k)$ **[Ber 01]**.

La première condition est celle autorisant le tir dans les RdPs ordinaires, la deuxième et troisième conditions résultent de l'obligation de tirer les transitions dans leurs intervalles de tir.

Le tir d'une transition t, à une date θ depuis l'état E = (M,I), conduit vers un état E' = (M',I') déterminé comme suit :
1) M' = M - Pré (t) + Post (t) (comme dans les réseaux de Petri ordinaires).
2) Le nouvel intervalle de tir I'(k), pour toute transition t_k est défini par :

a) Si t_k n'est pas sensibilisée par M' alors I' (t_k) = ϕ.
b) Si t_k est sensibilisée par M et n'est pas en conflit avec t alors :
I'(t_k) = [Max (0, DMin (t) - θ), DMax (t) -θ], si DMax(t_k) est fini.
I' (t_k) = [Max (0, DMin (t) - θ), ∞ [, sinon.
c)I' (t_k) = IS (t_k) sinon.

Autrement dit, les transitions non sensibilisées par le nouveau marquage reçoivent des intervalles vides. Les transitions nouvellement sensibilisées par M' (i.e. non sensibilisées par M-pré (t) et sensibilisées par M') reçoivent leurs intervalles de tir statiques. Les autres transitions (i .e celles qui étaient sensibilisées par M et n'ont pas été désensibilisées par le tir de t) voient leurs intervalles de tir décalés, vers l'origine du temps, de la valeur θ.
Deux transitions t et t' sont en conflit pour un marquage M si toutes deux sont sensibilisées par M, mais, pour au moins une place p, M(p) < Pre(p,t) + Pre(p,t').

2.2.5. Echéanciers et graphe d'accessibilité [Bbd03]

Les règles de franchissement données ci-dessus définissent une relation d'accessibilité sur l'ensemble des états d'un réseau temporel. On définit, comme pour les réseaux de Petri ordinaires, les séquences de transitions successivement tirables. Pour chaque séquence, on peut déterminer un échéancier qui lui est associé. Un échéancier est un couple (s, u) constitué d'une séquence de tir s et d'une séquence de dates de tir u. Un échéancier est dit réalisable, depuis un état E, si et seulement si les transitions de la séquence s sont successivement tirables depuis l'état E, aux dates correspondantes dans la séquence u. On peut ainsi déterminer le fonctionnement d'un réseau temporel par l'ensemble des états accessibles, depuis son état initial (i .e . en déterminant l'ensemble des échéanciers réalisables depuis E_0) **[Ber 01]**.

Représenter le fonctionnement d'un réseau de Petri temporel par le graphe d'accessibilité de ses états (comme le fonctionnement d'un RdP simple qui est

représenté par son GMA) est en général impossible: le temps étant continu et les transitions pouvant être tirées à tout instant dans leur intervalle de tir. Par conséquent, un état peut avoir une infinité de successeurs déterminés par les règles de tir données précédement.

2.2.6. Domaine de tir

Avant de définir les classes d'états d'un réseau, donnons d'abord une définition plus compacte pour les états, Un état peut être vu comme étant un couple (M,D) où M est un marquage et D un ensemble de vecteurs appelé Domaine de tir. Chaque vecteur possède une composante par transition sensibilisée. La i ème projection du domaine est l'intervalle associé à la i ème transition sensibilisée [Ber 01]. Ces domaines peuvent exprimer un ensemble de solutions des inéquations dont les variables sont associées à chacune des transitions sensibilisées **[Bme 83]**.
Exemple 2:

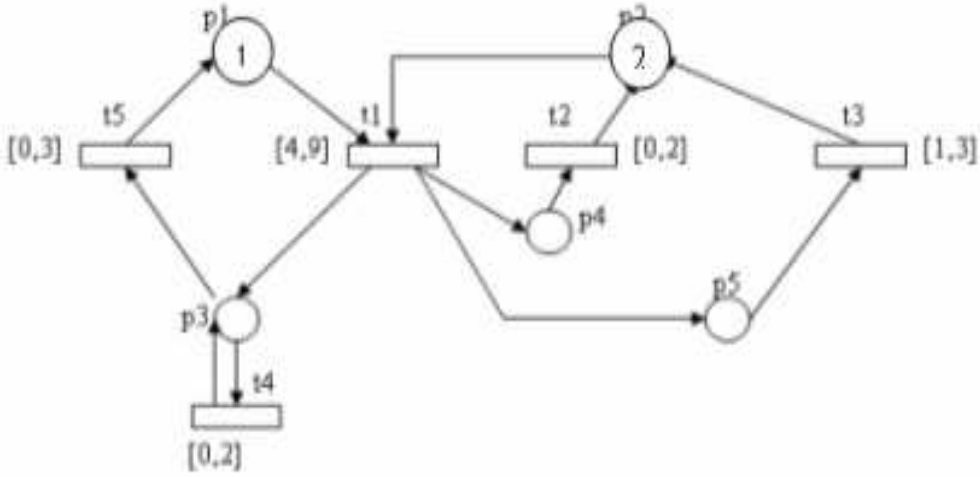


figure2 réseau temporel

Pour l'exemple de la figure 2, l'état initial est donné par : (M_0,D_0) avec :
- M_0 = p1(1), p2(2) .

- $D_0 = 4 \leq t1 \leq 9$
Le tir de t_1 à une date relative $\theta 1$ comprise entre 4 et 9 mène à l'état E_1 (M_1,D_1) :
- M_1 = p3(1), p4(1), p5(1).

$$\text{- } D_1 = \left\{ \begin{array}{c} 0 \leq t2 \leq 2 \\ 1 \leq t3 \leq 3 \\ 0 \leq t4 \leq 2 \\ 0 \leq t5 \leq 3 \end{array} \right\}$$

La date $\theta 1$ n'apparait pas dans l'expression du système qui définit D_1 puisque t_1 était la seule transition sensibilisée par M_0. Le tir de t_2 depuis E_1à une date $\theta 2$ comprise entre 0 et 2 mène à l'état E_2 (M_2,D_2) avec :
- M_2 = p2, p3, p5.

$$\text{- } D_2 = \left\{ \begin{array}{c} Max(0, 1 - \theta 2) \leq t3 \leq 3 - \theta 2 \\ 0 \leq t4 \leq 2 - \theta 2 \\ 0 \leq t5 \leq 3 - \theta 2 \end{array} \right\}$$

Le paramètre $\theta 2$ apparait dans le système D_2 comme une constante. Puisque le temps est continu, $\theta 2$ peut prendre toute valeur réelle entre 0 et 2, l'état E_1 admet donc une infinité de successeurs par t_2 **[Bbd 03]**.

2.3. Méthode d'analyse énumérative par les classes d'états

Plutôt que de considérer l'état atteint à partir de l'état initial, obtenu en tirant un échéancier particulier (s,u), considérons l'ensemble des états que l'on peut atteindre en tirant des échéanciers ayant tous comme support commun une séquence s. Ces états ainsi regroupés définissent une Classe d'états **[Ber01]**.

2.3.1. Définition

Une classe d'état est constituée par l'ensemble des états accessibles depuis l'état initial, en tirant tous les échéanciers réalisables (s,u) ayant comme support la séquence s. Une classe d'état est caractérisée par cette séquence de tir s, et par le marquage M accessible par cette séquence à partir du marquage initial, on la note C = (M,D), où :
- M est le marquage accessible depuis M_0 par la séquence s (commun à tous les états de la classe).
- D est le domaine de tir de la classe, défini par l'union des intervalles de tir des états constituant la classe.

Remarque

La classe C_0 est constituée d'un seul état E_0 (M_0,D_0) .
Le domaine d'une classe peut être exprimé comme étant l'ensemble des solutions d'un système d'inéquations linéaires, $A \times t^* \leq b$: A est une matrice de coefficients, t* et b des vecteurs et t^*_i est une variable correspondant à la ième transition sensibilisée par M **[Bme 83]**.

En pratique, pour calculer l'ensemble des classes d'états, on utilise une méthode récursive permettant d'obtenir la classe correspondant à la séquence s.t (s_1, s_2,..., s_n, t), à partir de la classe ayant comme support la séquence s, à condition que t soit sensibilisée par le marquage de cette dernière classe. Ceci nous permet d'obtenir l'ensemble des classes à partir de la classe initiale grâce aux règles de transitions entre états définies dans ce qui suit.

2.3.2. Transitions entre classes [Bbd 03]

Une transition t est tirable depuis une classe d'états C (M, D), si et seulement si les conditions suivantes sont satisfaites :
1) t est sensibilisée par le marquage M.

2) Le domaine D contient un vecteur dans lequel la composante relative à la transition t possède une valeur inférieure ou égale aux composantes, relatives aux autres transitions sensibilisées par M.
La première condition est celle des réseaux de Petri ordinaires. La deuxième condition exprime le fait qu'il doit exister un vecteur dans le domaine D où : l'intervalle de la composante correspondante à t lui permet d'être tirée la première, tout en respectant les dates au plus tard des autres transitions sensibilisées. L'expression de cet intervalle est complexe dans ce cas, il peut y avoir des relations entre les différentes dates de tir des transitions **[Bme 83]**.Sous forme d'inéquations, D étant l'ensemble des solutions du système $A \times t^* \leq b$, la condition 2 est satisfaite pour la transition t_i si et seulement si le système suivant admet une solution en t*:
1) $A \times t^* \leq b$.
2) $t^*_i \leq t^*_j$ pour toute variable $t^*_j \neq t^*_i$.
Le calcul de la nouvelle classe C' (M', D') accessible depuis C (M, D), en tirant une transition t_i se fait comme suit :
1) M' = M - pré (t_i) + Post (t_i).
2) Le domaine D' est déterminé en quatre étapes **[Kha 97]**:
a) Ajouter au système $A \times t^* \leq b$ les conditions de tir 2 exprimant que t_i doit être tirée avant les autres transitions sensibilisées, c'est-à -dire $t_i \leq t_j \forall\ t_j \neq t_i$.
b) les variables associées aux transitions en conflit avec t (t non comprise) pour le marquage M sont éliminées du système. Ces transitions sont celles distinctes de t, sensibilisées par M, et non sensibilisées par le marquage M - Pre(t).
c) dans ce système réduit, la transition tirée correspondant à la variable ti. On remplace dans ce système chaque variable t_j avec $t_j \neq t_i$ par $t_j + t_i$, puis on élimine t_i.
d) Compléter le dernier système obtenu en ajoutant des variables pour chaque transition nouvellement sensibilisée, et leur associer leurs domaines de tir statiques.

Remarque:

Les éliminations effectuées dans les étapes (b) et (c) préservent les contraintes

temporelles induites sur les variables restantes. Pour cette opération, on peut utiliser la méthode d'élimination classique de Fourier-Motzkin **[Gar05]**.

2.3.3. Egalité entre les états

Deux classes C (M,D) et C' (M',D') sont définies comme égales si leurs marquages et leurs domaines de tir sont identiques. Comparer les deux domaines revient à comparer les ensembles des solutions des deux systèmes d'inégalités. Cette opération est coûteuse dans le cas général. Toutefois, elle peut être effectuée efficacement dans le cas décrit par le lemme suivant :

Lemme:

Les domaines de tir, des classes d'états d'un réseau de Petri temporel, peuvent être exprimés comme l'ensemble des solutions du système d'inéquations ayant pour forme générale :

$a_i \leq t^*_i \leq b_i$.

$t^*_j - t^*_k \leq c_{jk}$. Pour tous j et k, $j \neq k$.

t^*_i est la variable associée à la i ème transition sensibilisée par le marquage M, a_i, b_i et c_{jk} sont des constantes (b_i et c_{jk} peuvent être non bornées). Ces formes canoniques d'inéquations peuvent être calculées en un temps pôlynomial. Ainsi, on peut faire la comparaison entre deux domaines en comparant simplement leurs formes canoniques **[Bme 83]**.

2.3.4. Graphes de classes

La relation d'accessibilité sur les classes d'états permet de construire un graphe : les sommets représentent les classes d'états. Un arc étiqueté par t existe entre deux classe C et C' si et seulement si t est une transition tirable depuis C, et le tir de t dans C conduit à la classe C' **[Bbd 03]**. D'après la définition des classes d'états,

il est clair que n'importe quelle séquence tirable depuis l'état initial correspond à un chemin dans le graphe des classes. En outre, l'existence d'un chemin étiqueté s, de la classe initiale à une classe C, implique qu'un échéancier de séquence s est réalisable depuis l'état initial.

2.3.4.1. Calcul du graphe des classes:

Le graphe des classes est calculé par application successive de la fonction de calcul des successeurs à partir de la classe d'état initiale C_0= (M_0, D_0) avec D_0= $\{\alpha_k \leq \theta_k \ \leq \beta_k \ / \ t_k \in \text{enabled}(M_0)\}$. La convergence est assurée en mémorisant les classes d'états générées et en arrêtant l'analyse des successeurs d'une classe si celle-ci a déjà été précédemment analysée.

Algorithm 1 Algorithme de calcul du graphe des classes d'états:

Visited $\leftarrow \emptyset$
Waiting $\leftarrow \{(M_0, D_0)\}$
  Tant que Waiting $\neq \emptyset$ faire
   (M, D) $\leftarrow$ pop(Waiting)
   Pour tout t $\in$ firable(M, D) faire
    si next(M, D, t)$\notin$ Visited $\cup$ Waiting alors
     Waiting $\leftarrow$ {next(M, D, t)}
    Fin si
   Fin pour
   Visited $\leftarrow$ Visited $\cup$ {(M, D)}
  Fin
Fin algorithme

2.3.5. Graphe de marquage et graphe des classes

Dans un graphe de classes, il peut y avoir des classes avec le même marquage , mais avec des domaines de temps différents. Ce qui signifie que l'information temporelle affecte généralement le comportement du réseau.

2.4. Analyse et exploitation du graphe des classes d'états

2.4.1. Analyse du comportement des systèmes temporels

Dans la spécification d'un système, il peut y avoir plusieurs manières d'intervention du temps. Lorsque le système est représenté par un TPN, l'analyse de ces propriétés revient à analyser les propriétés de l'ensemble des classes et/ou de l'ensemble des échéanciers du réseau. Parmi ces propriétés, on a :
1) Les invariants sur l'ensemble des marquages accessibles du réseau (contraintes d'exclusion mutuelle, absence de blocage, etc.) vérifiables sur le graphe des classes d'états.
2) Propriétés de l'ensemble des séquences de tir (vivacité, terminaison, etc.)vérifiables sur le graphe des classes d'états.
3) Des contraintes de nature temporelle (bornes d'accessibilité à un marquage d'un autre, contraintes de synchronisation, etc.) déterminées en spécifiant des échéanciers particuliers (le graphe de classes ne permet pas l'extraction « directe » d'échéanciers) **[Bbd 03]**.

2.4.2. Accessibilité d'un marquage

Notons A (M_0) (A pour accessible) l'ensemble des marquages d'un RdPT accessibles depuis le marquage initial. Le problème de l'accessibilité est déterminé par

l'appartenance d'un marquage donné à $A(M_0)$ ou non. Ce problème est indécidable pour les réseaux temporels **[Bbd 03]**.

2.4.3. Propriété de bornitude

De la même manière que pour l'accessibilité, on peut prouver que la bornitude d'un graphe de classes est un problème indécidable pour les RdPTs.
La bornitude pour un réseau de Petri est équivalente à la propriété de finitude de l'ensemble $A(M_0)$. Conséquence immédiate, la finitude de l'ensemble des classes d'états d'un réseau de Petri temporelle est indécidable.
Maintenant supposons qu'un réseau temporel est borné, peut on déduire que le nombre de ses classes d'états est fini ?

Théorème1 [Ber01] :
Un réseau temporel est borné s'il n'admet pas de paire de classes d'états C=(M, D) et C'=(M', D') telles que:
- C' est accessible depuis C
- $M' \geq M$
- $D' = D$
- $\forall$ p. M' (p) >M (p)$\Rightarrow$ M' (p) $\geq \max_{(t \in T)}$\{Pre (p, t)\}.

Théorème2 [Bme83]:
Le nombre de classes du système est fini $\Leftrightarrow$ le réseau est borné.

Ce théorème montre que toute condition nécessaire (Resp. suffisante) de bornitude fournira une condition nécessaire (Resp. suffisante) pour la finitude du nombre de classes.

2.4.4. Propriétés de l'ensemble des marquage (ou séquences de tir)

Lorsque la bornitude du graphe des classes est vérifiée, on peut vérifier les propriétés concernant les marquages et séquences de tir (telle que l'absence de blocage, quasi-vivacité, invariants et autres propriétés) en réalisant un examen exhaustif du graphe des classes d'états **[Bbd 03]**.

2.4.5. Analyse temporelle et existence d'échéanciers

Généralement, les propriétés temporelles (telles que l'accessibilité à un marquage dans un intervalle de temps déterminé, encadrement temporel de la durée d'une séquence de tir,etc.) ne peuvent être vérifiées par examen du graphe des classes d'états. Mais il est possible d'associer un intervalle de tir à un arc du graphe (correspondant à l'intervalle de tir d'une transition). Cela dit, on ne peut déduire l'ensemble des échéanciers relatif à une séquence de cette manière, à cause de l'interdépendance des dates relatives des intervalles de l'échéancier **[Bbd 03]**.

Démontrer l'existence d'un échéancier réalisable, pour une séquence s, revient à construire un système d'inéquations, dont chaque solution q détermine un échéancier réalisable (s,q). Une technique qui permet de réaliser cette construction consiste à renommer la variable relative à la transition tirée t_i par q_i au lieu de la supprimer (étape 3 de la transition entre classes). q_i apparait ainsi dans les classes suivantes comme paramètre. A la dernière classe de la séquence, on obtient autant de paramètres q_1, q_2,...,q_n que de transitions de la séquence. Toute solution en q du système fournit un échéancier réalisable.
On peut ainsi montrer l'existence ou non d'un échéancier particulier.

2.5. Extensions des réseaux de Petri temporels

Nous avons jusqu'ici présenté le modèle réseau de Petri temporel présenté par Merlin, mais il existe dans la littérature d'autres modèles étendus de réseaux temporels dont nous allons citer les plus connus**[Bbd 03]**

2.5.1. Réseaux de Petri à arcs temporels "Timed-arc Petri nets "

Ils ont été introduits en 1983 par B. Walter. Ce sont des réseaux temporels identiques à ceux de Merlin, sauf que les intervalles de temps sont posés sur les arcs d'incidence aux transitions. Une transition ne peut être tirée que si, pour chaque place en entrée, il existe un jeton dont l'âge est bien dans l'intervalle relatif à l'arc reliant cette place à la transition.

2.5.2. Réseaux de Petri p-temporels [Loh02]

Ce sont des réseaux de Petri dont la définition est identique à celle des réseaux de Merlin (W.Khansas), la seule différence est que les intervalles temporels sont associés aux places, un jeton dans une place annoté d'un intervalle [a, b] ne peut pas quitter cette place avant d'y avoir passé au moins a unités de temps, et doit la quitter avant b unités de temps; si un jeton n'arrive pas à quitter une place avant b unités de temps, il devient un jeton mort et ne peut plus participer à la validation de transitions .

2.5.3. Réseaux de Petri statiquement temporisés

Ils sont proposés par A. Cerone et A. Maggilio-Schettini. Ce modèle est très général car on y affecte les intervalles simultanément sur les places, les arcs et les

transitions. De plus on y trouve deux sémantiques de tir : sémantique forte et sémantique faible.
La première est celle des réseaux de Merlin forçant le tir d'une transition arrivant à son délai le plus tard (une transition est forcée lorsque elle atteint la borne maximum de son intervalle de tir). Quant à la deuxième, on ne force jamais le tir d'une transition (un jeton peut rester indéfiniment dans une place **[Loh02]**.

2.6. Conclusion

Nous avons présenté les extensions temporelles des réseaux de Petri offrant la possibilité de modéliser les systèmes à contraintes temps réel et de modéliser un temps opératoire compris dans un intervalle. Nous avons également présenté une technique d'analyse des RdPTs (la technique des graphes de classes d'états) qui est trés utilisée dans les travaux de recherche. Toutefois, cette méthode présente des limites intrinsèques à ne pas négliger :
- Même si le nombre de classes d'états est borné, il peut être trés élevé, et par conséquent difficile à manipuler. (Problème de l'explosion combinatoire du nombre d'états).
De plus pour vérifier des propriétés temporelles avec le graphe des classes, la méthode la plus courante consiste à avoir recours à des observateurs. Cela présente deux inconvénients principaux: la taille du système est augmentée pour chaque observateur ajouté, ce qui le rend plus sujet au phénomène d'explosion combinatoire. Et chaque propriété à vérifier requiert un observateur et donc un calcul de graphe des classes supplémentaire.

Dans le prochain chapitre, nous présentons une autre méthode de génération d'éspace d'états fini, nous allons profiter de l'expressivité des réseaux de Petri temporels pour la modélisation des systèmes temps réel, nous allons également

présenter une logique temporelle temporisée convenable qui sert à exprimer les propriétés des systèmes temps réel.

Chapitre 3

Le model-checking

3.1. Introduction

La vérification formelle des systèmes a connu un succès incontestable . Au cours de ces dernières années ces techniques de vérification ont été étendues afin de prendre en compte des notions quantitatives permettant en particulier de spécifier le délai qui sépare différentes actions du système.

Dans ce cadre le modèle des réseaux de Petri temporels **[Mer74]** et le modèle des automates temporisés ont été bien étudiés, et plusieurs outils de vérification ont été développés pour ces modèles. L'approche par model-checking suppose de modéliser le comportement du système à vérifier et d'énoncer la propriété attendue sous la forme d'une formule de logique temporelle. Ensuite utiliser un model checker pour vérifier si la propriété est satisfaite par le modèle ou non.

Les techniques de vérification par model-checking reposent sur le calcul et l'exploration de l'espace d'états du modèle. Dans le cas des systèmes temporisés en temps dense, cet espace d'états est infini et des techniques d'abstraction doivent être utilisées afin d'obtenir une représentation finie de l'espace d'états. Les automates temporisés et les réseaux de Petri temporels constituent deux mod-

èles trés appropriés à la modélisation de systèmes temps réel. Nous nous intéressons plus particulièrement aux réseaux de Petri temporels. Ils constituent un modèle permettant d'exprimer facilement les comportements de systèmes parallèles et distribués (synchronisation,...). De plus comparativement aux automates temporisés, les techniques de model-checking et en particulier le model-checking de propriétés temporelles quantitatives ont été peu étudiées.

Dans ce chapitre, nous présentons une autre méthode de calcul de l'espace d'états d'un réseau de Petri temporel (TPN) qui est la technique du graphe des zones proposée dans **[GRR03]**. Nous proposons de distribuer cet espace d'états (zones) sur plusieurs sites pour réduire le problème de l'explosion combinatoire de l'espace d'états dans le cas de systèmes temps réel complexes.

3.2. La méthode des classes d'états

La méthode d'analyse présentée dans le chapitre 1(Les TPN) permet pour les réseaux de Petri temporels une analyse d'accessibilité semblable à celle permise pour les réseaux de Petri par la technique du graphe des marquages. Cette technique a été utilisée dans de nombreux travaux universitaires ou industriels, et a été intégrée à plusieurs outils d'analyse de systèmes.

Les limites intrinsèques de la méthode ne doivent toutefois pas être perdues de vue. Une première limite est qu'il ne peut être énoncé de condition nécessaire et suffisante pour la propriété de bornitude pour les réseaux temporels, une seconde limite est que le nombre de classes d'états d'un réseau de Petri temporel peut être trés grand ce qui pose le problème de l'explosion combinatoire du nombre d'états.

3.3. La méthode des zones

Le graphe des classes donne l'ensemble des marquages du réseau et préserve son langage non temporisé. C'est-à-dire qu'il fournit l'ordre d'apparition des évènements (tir de transition). Il permet donc de vérifier des propriétés de type logique temporelle linéaire. Cependant, nous pouvons remarquer que si la classe C' est le successeur par la transition t de la classe C, cela n'implique pas forcement que tous les états de C ont un successeur par t de C' mais uniquement qu'il existe au moins un état de C qui possède un successeur par t dans C'. C'est pourquoi nous ne pouvons pas vérifier des propriétés de type logique temporelle arborescente (CTL) sur le graphe des classes. De plus le graphe des classes ne fournit pas d'informations quantitatives sur les instants auxquels les tirs des transitions se sont produits.

Le graphe des classes ne permet pas dans l'état d'envisager la vérification de propriétés temporelles quantitatives, nous nous sommes intéressés à une autre technique d'exploration de l'espace d'états: la méthode des zones proposée dans **[GRR03]**.

3.3.1. Définition d'une zone

Soit C un ensemble d'horloges. Une zone est un ensemble convexe de valuations d'horloges, représentant un ensemble de contraintes de la forme x_i - $x_j \leq c_{ij}$, $x_i \leq c_{i0}$, $x_i \geq c_{0j}$. Avec c_{ij}, c_{i0}, $c_{0j} \in \mathbb{Z}$.

De même que pour les classes d'états de Berthomieu , une zone peut être encodée par une DBM. Une zone est une forme particulière de polyèdre qui peut être représentée par une DBM efficacement (Difference Bounded Matrice), en utilisant une horloge supplémentaire x_0 qui est toujours égal à 0. Cette représentation

(DBM) permet une représentation plus facile et des algorithmes moins complexes par rapport à un polyèdre général.

Si Z est une zone, nous notons z_{ij} la contrainte sur la différence de l'horloge x_i-x_j c-à-d. Z représente la conjonction de contraintes atomiques: $\forall$ x_i, $x_j \in C$, (x_i-$x_j \leq z_{ij}$).

3.3.2. Définition d'une DBM

Une DBM est une structure de données qui permet de représenter efficacement les zones.

Pour avoir une forme unifiée de contraintes d'horloges, on introduit une horloge de référence x_0 avec la valeur 0 (zéro). Alors $C_0 = C \cup \{x_0\}$.

Pour construire le DBM qui représente une zone D, on commence par énumérer toutes les horloges qui appartiennent à C_0 à partir de 0,...,n et l'indexe de l'horloge ajoutée est x_0 . Chaque horloge est indiquée par une ligne de la matrice (DBM), la ligne est utilisée pour sauvegarder la borne inférieure de la différence entre une horloge et les autres horloges, ainsi la colonne correspondante est utilisée pour les bornes supérieures. Les éléments de la matrice sont calculés selon trois étapes :

— Pour toute contrainte de D: x_i-$x_j \leq n$, D_{ij}=(n, $\leq$).
— Pour toute différence d'horloges x_i-x_j qui n'est pas bornée dans D , $D_{ij} = \infty$, tel que ∞ est une valeur spéciale qui indique qu'il n'y a pas de borne.
— Ajouter les contraintes implicites telle que toutes les horloges sont positives, c-à-d: x_0 - $x_i \leq 0$, et telle que la différence entre une horloge et elle-même est toujours 0, c-à-d: x_i - $x_i \leq 0$.

Exemple:

On considère la zone D = x - x_0< 20 ∧ y - x_0 <20 ∧ y - x ≤ 10 ∧ x - y ≤ -10 ∧x_0 - z <5.
Pour construire la matrice qui représente D, on numérote les horloges : x_0, x, y, z. la matrice résultante est la suivante:

$$\text{M(D)=}\begin{pmatrix} (0,\leq) & (0,\leq) & (0,\leq) & (5,<) \\ (20,<) & (0,\leq) & (-10,\leq) & \infty \\ (20,\leq) & (10,\leq) & (0,\leq) & \infty \\ \infty & \infty & \infty & (0,\leq) \end{pmatrix}$$

3.3.3. Etats symboliques

De la même manière que dans le graphe des classes, on donne une définition d'un regroupement d'états de la sémantique d'un réseau de Petri temporel.

3.3.3.1. Définition d'un état symbolique

Un état symbolique **Z** d'un réseau de Petri temporel ***N*** est un couple **Z** =(M,z) où M est un marquage et z un polyèdre convexe de $\mathbb{R}^{+|enabled(t)|}$ appelé zone.
Une zone est caractérisée par un ensemble de contraintes de la forme:

$$\begin{cases} -z_{ij} \leq x_i - x_j \leq z_{ij} \\ -a_i \leq x_i \leq b_i \end{cases} \qquad \forall t_i,t_j \in \text{enabled(M)}$$

t_i,t_j∈enabled(M): t_i,$t_j$sont sensibilisées par le marquage M.

Les variables x_i représentent les valuations des horloges associées aux transitions du réseau de Petri temporel ***N***.

À la différence d'une classe d'état, un état symbolique représente les valuations réelles des horloges du réseau de Petri temporel. L'idée sous jacente est de pouvoir utiliser les informations temporelles contenues dans la zone pour déterminer les séquences de transitions possibles en fonction de la date d'entrée dans l'état symbolique.

Une transition est dite franchissable, dans l'état symbolique (M,z), s'il existe une valuation v $\in$z telle que la transition est franchissable. L'état successeur est alors calculé en appliquant la sémantique des réseaux de Petri temporels.

Remarque:
De la même manière que pour les domaines de tir d'une classe d'états, les contraintes définissant la zone sont compatibles avec la définition des DBM. Il est ainsi envisageable d'implémenter de manière efficace des calculs sur les états symboliques d'un réseau de Petri temporel.

On a étendu les deux types de transitions de la sémantique d'un réseau de Petri temporel (transitions discrètes et transitions continues) aux états symboliques.

3.3.3.2. Successeur discret d'un état symbolique

Soit **Z** = (M, Z) un état symbolique et tf une transition franchissable. L'état symbolique successeur de **Z** par franchissement d'une transition t_f , noté $post_{t_f}$(**Z**), est défini par:

$$post_{tf}(\mathbf{Z})=\begin{cases} M - Pre(\bullet, t) + Post(t, \bullet) \\ Z \cap \{x_f \geq \alpha_f\})[x_r \leftarrow 0] \end{cases}$$

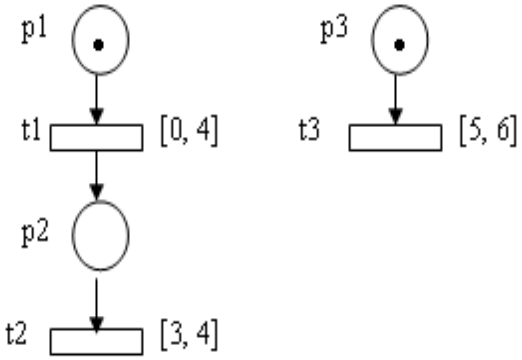
où x_r={$x_i \in$X $|\uparrow$enabled(t_i,M, t_f)}.

ainsi post$_{t_f}$(***Z***) *contient tous les états de la sémantique accessibles à partir de* ***Z*** *en franchissant la transition* t_f.

3.3.3.3. Successeur temporel d'un état symbolique

Soit Z = (M, Z) un état symbolique. L'état symbolique obtenu par écoulement de temps, noté Post (***Z***), est défini par:
Post (***Z***) =(M, Z $\cap$ { $x_i \leq \beta_i$ | $t_i \in$ enabled(M)}).
Post (***Z***) *contient tous les états de la sémantique accessibles à partir de Z en laissant le temps s'écouler.*

Exemple:



L'état symbolique initial de ce réseau de Petri est $\boldsymbol{Z}_0$=(M_0,x_1= x_3=0). Selon la sémantique, il est possible d'écouler du temps dans ce marquage initial tant que les horloges des transitions n'atteignent pas leurs dates de tir au plus tard.

Par conséquent, le successeur temporel de $\boldsymbol{Z}_0$ est $\boldsymbol{Z}_1$ = post ($\boldsymbol{Z}_0$)= (M0, x1 = x3

$\in$[0, 4]). Pour $\boldsymbol{Z}_1$, seule la transition t_1 est franchissable, le successeur discret de Z_1par t_1 est alors l'état symbolique $\boldsymbol{Z}_2$ = ((0, 1, 1), { $x_2 = 0 \wedge 0 \leq x_3 \leq 4 \wedge 0 \leq x_3 - x_2 \leq 4$}).

3.3.4. Le graphe des zones

En utilisant les états symboliques, on peut calculer les états atteignables du système en appliquant l'algorithme suivant **[Gar05]**:

Algorithm 2 Algorithme de calcul du graphe des zones:

$\boldsymbol{Z}_0 = (M, Z_0)$
Waiting $\leftarrow \{\boldsymbol{Z}_0\}$
Passed $\leftarrow \emptyset$
Tant que Waiting $\neq \emptyset$ faire
  Z = pop(Waiting) // prendre le premier élément de la pile
  Si **Z** $\notin$Passed alors
    Pour tout t $\in$ enabled (**Z**) faire
      Z′ = post ($post_t$(**Z**)) // successeur temporel
      Waiting $\leftarrow$ Waiting $\cup$ { **Z**′}
    Fin pour
    Passed$\leftarrow$ Passed $\cup$ { **Z**}
  Fin si
Fin Algorithme

Théorème:

L'algorithme de calcul du graphe des zones converge pour les réseaux de Petri temporels bornés dont les bornes des dates de tir sont des rationnels positifs.

La preuve de ce théorème est dans **[Gar05]**.

Exemple:

En considérant le réseau de Petri temporel de l'exemple précédent. L'application de l'algorithme de calcul des zones génère le graphe de la figure 5.

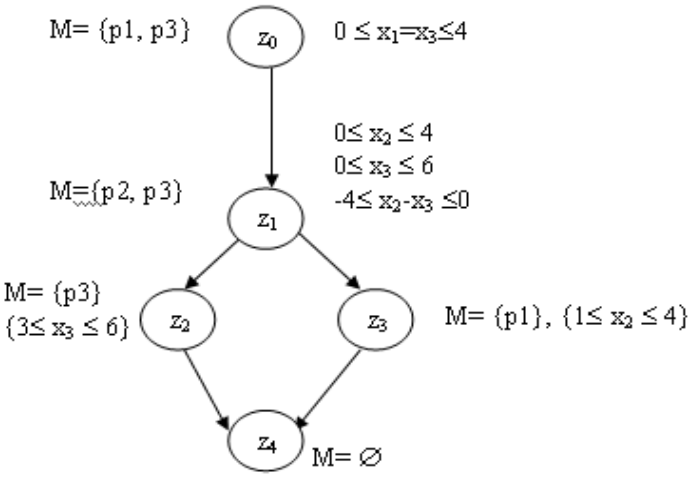


Fig5: exemple de graphe des zones

3.4. Technique de vérification par model-checking

Le but de la vérification est d'assurer qu'un système satisfait un certain nombre de propriétés. Pour cela, il est nécessaire de modéliser formellement le comportement de ce système : Rappelons que le système est modélisé par un RdP temporel, le graphe des zones décrit l'évolution de ce système. Une fois le système décrit, et les propriétés souhaitées spécifiées en logique temporelle quantitatives (TCTL) l'algorithme dit model-checking permet de répondre automatiquement à la question :«est-ce que le système satisfait les propriétés souhaitées ?».

3.4.1. Model-Checking à la volée

La vérification à la volée consiste à vérifier les propriétés attendues du système durant la génération de l'espace d'états. Cette technique n'explore qu'une partie

de l'espace d'états ce qui permet de réduire le problème de l'explosion combinatoire.

Pour pouvoir utiliser un model checker il faut modéliser formellement le système et spécifier les propriétés à vérifier à l'aide d'une logique temporelle adéquate. La logique temporelle temporisée qui permet d'exprimer les propriétés quantitatives d'un système temps réel est la logique temporelle temporisée TCTL.

3.5. Définition de la logique temporelle

La logique temporelle est une logique modale introduite par Pnueli **[Pnu77]**, afin de décrire et de spécifier au moyen d'expressions axiomatiques, des propriétés qualitatives et quantitatives des systèmes.

Une logique temporelle est un langage permettant de vérifier certaines propriétés spécifiques des systèmes, comme la propriété d'exclusion mutuelle.

La syntaxe de ce langage est basée essentiellement sur une grammaire procurant une spécification sûre, précise et sans ambiguité des propriétés temporelles.

Elle est composée de la logique propositionnelle (formée à l'aide de propositions c à d: des énoncés susceptibles d'être vrais ou faux) à laquelle sont ajoutés des opérateurs temporels linéaires qui sont essentiellement la potentialité F (éventuellement dans le futur), l'invariance G (toujours dans le futur) la précédence U (jusqu'à dans le futur {Until}), et les quantificateurs A (pour tout chemin) et E (pour certains chemins).

3.5.1. Les classes de logique temporelle [Had03]

Le parallélisme (et/ou le non déterminisme) implique l'existence de différentes exécutions possibles pour un même système.
Pour prendre en compte l'ensemble des possibilités de ces exécutions on peut :

- examiner les propriétés sur des exécutions arborescentes, on représente alors l'ensemble des exécutions sous forme d'arbre, où les successeurs d'un état sont obtenus par les instances d'évènements possibles à cet état; on parle alors de ***la logique temporelle arborescente.***
- examiner les propriétés sur des exécutions linéaires, on considère alors l'ensemble des exécutions comme des séquences distinctes, on parle alors de ***la logique temporelle linéaire.***

3.5.1.1. La logique CTL:

La logique CTL est une logique temporelle arborescente, introduite par Clarke et Emerson. Elle est destinée à la spécification des systèmes concurrents à états finis.
Elle permet d'exprimer les propriétés usuelles des systèmes telles que la sûreté (fiabilité) et la vivacité.

Syntaxe de CTL [Eme96] :

CTL est formée des règles S1, S2, S3 et P0.
S1 : chaque proposition atomique P est une formule d'état.
S2 : si f et g sont des formules d'état alors $(f \wedge g)$ et $(\neg f)$ sont des formules d'état.
S3 : si f est une formule de chemin alors $E\,f$ et $A\,f$ sont des formules d'état.
P0: si f et g sont des formules d'états alors Xf et fUg sont des formules de chemin.

La logique CTL se focalise sur la notion d'état. En effet, on peut décrire la syntaxe sans définir les formules de chemin à l'aide des quatre opérateurs:
AX*f* : pour tout état successeur de l'état considéré, *f* est vérifiée
EX*f* : il existe un état successeur de l'état considéré pour lequel *f* est vérifiée.
Af U*g* : pour toute séquence issue de l'état considéré, *f* est vérifiée jusqu'à ce que *g* le soit.
Ef U*g* : il existe une séquence issue de l'état considéré telle que *f* est vérifiée jusqu'à ce que *g* le soit.

6.1.2- Abréviation de CTL [Zou98] :

Soit f une formule d'état, la sémantique de CTL est définie comme suit :
La formule ***EF****f* dit qu'il existe une exécution pour laquelle f est nécessairement vraie. Cette propriété est connue sous le nom de la potentialité (figure 3.b).
La formule ***AF****f* signifie que, pour toute exécution possible, on atteindra un état qui vérifie f (f est nécessairement vraie).C'est la propriété d'inévitabilité (figure 3.a).
La formule ***EG****f* spécifie qu'il existe une exécution pour laquelle f reste toujours vraie. C'est la propriété de quasi-invariance (figure 3.d).
La formule ***AG****f* signifie que pour toute exécution possible, la propriété f est vraie sur tous les états de cette exécution (f reste toujours vraie). C'est la propriété d'invariance (figure 3.c).

La combinaison de ces opérateurs permet d'exprimer des propriétés plus ou moins complexes.
Exemple 3 [Had03]:

Soit la proposition pi désignant l'évènement "le processus i demande la section critique" et qi l'évènement "le processus i entre effectivement en section critique".
La formule ***AG*** (pi⇒ ***AF***qi) permet de spécifier la propriété de vivacité "tout processus demandant la section critique l'obtiendra nécessairement).

Exemple 4 [Had03]:

La formule ***AG*** (req ⇒AreqUserv) exprime qu'à partir de tout état qui contient une requête, dans toute séquence la requête sera présente jusqu'à ce qu'elle soit servie.

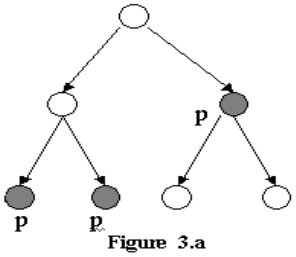


Figure 3.a
Inévitabilité AFp

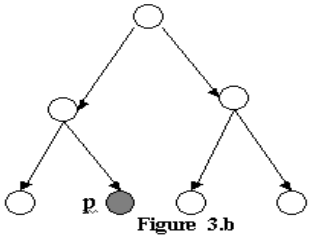


Figure 3.b
potentialité EFp

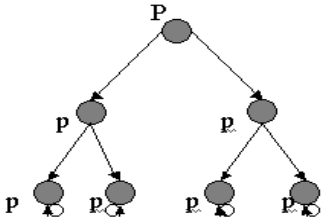


Figure 3.c
Invariance AGp

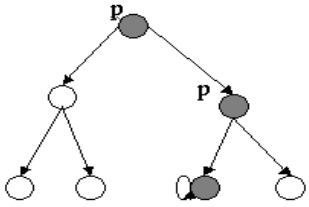


Figure 3.d
quasi-invariance EGp

3.6. CTL et la temporisation (TCTL)

La logique CTL ne permet de décrire que des propriétés qualitatives. Une façon d'introduire le temps dans cette logique est de borner la portée des opérateurs temporels. TCTL (Timed Computation Tree Logique) est une logique temporisée, extension de CTL, qui permet d'exprimer des propriétés sur des systèmes temps réels dits temporisés c.à.d des systèmes dont le comportement est conditionné par le passage du temps.

Nous nous intéressons qu'à cette logique pour vérifier les systèmes temps réel.

3.6.1. Syntaxe de TCTL

Soit AP un ensemble de propositions atomiques; les formules de TCTL sont définies par les règles suivantes :
S1 : chaque proposition atomique P est une formule de TCTL.
S2 : si f et g sont des formules de TCTL alors (f$\vee$g) et ($\neg$f) sont des formules de TCTL .
S3 : si f est une formule de chemin alors E f U$\sim$c g et A fU$\sim$c g sont des formules de TCTL, avec $\sim\in${<,<=, =,>,>=} et c$\in \mathbb{N}$. c peut être un intervalle de temps .

3.6.2. Sémantique de TCTL

Pour une séquence d'exécution (chemin) π = (s_0 ,s_1 ,...) du système, on note π(t) l'état du système à l'instant t sur cette séquence π et suc(s) l'ensemble des chemins issus de s.
La relation de satisfaction des formules de TCTL est définie par :
s$\models$ p $\Leftrightarrow$ p$\in$L(s)
s$\models$ f$\vee$g $\Leftrightarrow$s$\models$ f ou s$\models$ g
s$\models$ $\neg$ f $\Leftrightarrow\neg$ s$\models$ f
s$\models$ **E** f **U**$\sim$c g $\Leftrightarrow\exists$ π $\in$ suc(s); $\exists$ t$\sim$c tel que π(t) $\models$ g et $\forall$t'(0$\leq$ t'<t), π(t')$\models$ f
s$\models$ **A** f**U**$\sim$c g$\Leftrightarrow$ $\forall$ $\pi\in$suc(s); $\exists$ t$\sim$c tel que π(t) $\models$ g et $\forall$t'(0<=t'<t), π(t') $\models$ f
On définit également quelques abréviations classiques:
-**EF**$\sim$c f pour **E**(true **U**$\sim$c f), signifie qu'il existe une exécution pour laquelle f devient vraie avant l'instant c (potentialité bornée).
-**AF**$\sim$c f pour **A**(true **U**$\sim$c f);
-**EG**$\sim$c f pour $\neg$**AF**$\sim$c $\neg$f , signifie qu'il existe une exécution pour laquelle f reste vraie jusqu'à l'instant c (quasi_invariance borné).
-**AG**$\sim$c f pour $\neg$**EF**$\sim$c$\neg$f ;

On peut également utiliser des intervalles de temps pour la spécification des propriétés.

Exemple:
AG (pb⇒**AG** $_{(\leq 5)}$alarme) permet d'énoncer que « si un problème arrive, l'alarme se déclenchera immédiatement et durera au moins 5 unités de temps» .

3.7. Conclusion

Dans ce chapitre, nous avons présenté une autre méthode de génération de l'espace d'états d'un réseau de Petri temporel, autre que la méthode des classes d'états c'est la méthode des zones.

Nous avons également présenté une logique temporelle TCTL qui permet de spécifier les propriétés quantitatives des systèmes temps réel.

Dans le chapitre suivant, nous allons proposer un algorithme de distribution de l'espace d'états d'un TPN généré par la méthode des zones, sur un ensemble de stations afin d'éviter le problème de l'explosion combinatoire de la vérification par model checking; ainsi que des algorithmes de vérification à la volée des propriétés exprimées en TCTL.

Chapitre 4

Génération distribuée de l'espace d'états

4.1. Introduction

Les techniques de vérification par model-checking reposent sur le calcul et l'exploration de l'espace d'états du modèle. Dans le cas des systèmes temporisés en temps dense, cet espace d'états est infini et des techniques d'abstraction doivent être utilisées afin d'obtenir une représentation finie de l'espace d'états.

Le problème de la vérification par model-checking est l'explosion combinatoire de l'espace d'états, afin de pallier à ce problème plusieurs techniques peuvent être utilisées. Nous nous intéressons à l'approche distribuée qui permet de faire coopérer plusieurs machines pour générer l'espace d'états et réaliser la vérification; dans ce cas, l'espace d'états et la charge de calcul sont répartis entre les différentes machines.

Nous nous sommes interéssés à repartir les différents états d'un système temps réel et d'assurer la vérification par la méthode du model-checking à la volée d'une classe de propriétés exprimées en logique temporelle TCTL.

Dans ce chapitre nous nous intéressons à la génération distribuée de l'espace

d'états en nous basant sur l'algorithme des zones et à la vérification des propriétés exprimées en TCTL sur les réseaux de Petri temporels.

4.2. Génération distribuée de l'espace d'états

Plusieurs travaux ont développé des outils de génération distribuée de l'espace d'états d'un modèle (RdP, AT,...) **[Bou07] [Bhv01]** sur plusieurs stations de travail (machines); partitionner cet ensemble d'états sur différentes machines peut se faire en utilisant une fonction de hachage H: V $\rightarrow$ S tel que:
V est l'ensemble de tous les états du modèle.
S = {0,.., N-1} est l'ensemble des identifiants de machines (stations).
L'approche que nous avons utilisé pour la distribution est inspirée de **[Bhv01]**.

Étant donné un réseau de Petri temporel qui modélise un système temps réel. La première étape à faire dans le cadre d'une vérification par le model-checking est la construction de l'espace d'états à partir de ce modèle.

L'approche de construction consiste à explorer les différents états que l'on peut atteindre à partir des états initiaux. Cette étape est appelée génération d'espace d'états.Afin de construire rapidement l'espace d'états du système à vérifier, nous distribuons la tâche d'exploration sur plusieurs machines (un réseau de stations). Dans ce cas, chaque machine est responsable pour la génération d'une partie de l'ensemble des états du système.

La génération distribuée est alors effectuée par plusieurs machines. Dans une telle opération, chaque machine génère une partie de l'espace d'états en calculant les successeurs des états qu'elle détient. Une fonction de hachage est utilisée pour distribuer l'ensemble des états entre les différentes machines. Une machine peut

envoyer alors, selon la fonction d'attribution (de hachage), un état à une autre machine. L'opération est initiée par la machine qui détient l'état initial.

On distingue alors deux types d'arcs: arcs locaux qui relient les états de la même station; et les arcs qui relient deux états qui appartiennent à deux différentes stations, ce dernier type permet la transmission de messages entre stations.

4.2.1. Algorithme de distribution

Nous proposons un algorithme inspiré de **[Bhv01]**. Cet algorithme permet une génération distribuée de l'espace d'états.
On considère un ensemble de N machines: w_0, w_1,..., w_{N-1} et un système représenté par un réseau de Petri temporel *N*. Soit S_0 son état initial tel que S_0= (M_0, Z_0) avec M_0 est le marquage initial et Z_0 la zone initiale, et une fonction succ de succession permettant de calculer pour un état quelconque l'ensemble de ses états successeurs.

Chaque machine wi explore un ensemble d'états Si. Elle construit alors une partie {S_i, T_i} du graphe des zones. L'ensemble Si est calculé à partir d'une fonction de hachage qui permet d'attribuer les états aux différentes machines.

Chaque machine wi stocke dans sa mémoire locale les informations concernant les états et les transitions de sa partie {S_i, T_i}.

Les états visités et explorés, par une machine wi, sont stockés dans des ensembles disjoints : waiting (états visités non encore explorés) et passed (états explorés: on a déjà généré leurs états successeurs).

Une machine wi peut envoyer et recevoir des messages par invocation des primitives de communication. Il est à noter qu'il y a deux principaux types de

messages. Le premier type sert à échanger les états et les transitions (et les zones) (messages de données). L'autre sert pour la terminaison (message de contrôle).

Le calcul est initié par une machine initiatrice. Cette machine doit avoir un index qui correspond à h (S_0). Elle explore alors l'état initial S_0= (M_0, Z_0).

L'algorithme est composé d'une boucle principale, qui assure les différentes actions permettant de calculer tous les états atteignables. Ce même algorithme est exécuté par toutes les machines.

L'algorithme distribué d'exploration est le suivant :

Algorithm 3 Algorithme de la machine i

```
Début
passed:= ∅
termi:=faux
  Si h(S0) = i alors
    waiting:= S0
  Fsi
 Tant que ¬termi faire
  Si waiting ≠ ∅ alors
    Prendre (S) de waiting
    waiting:= waiting \ S /// S= (M, Z)
    passed := passed ∪ {S}
    Pour tout S'∈ succ(S) faire
      Si h(S') = i alors
        passed := passed ∪ {S'}
      Sinon envoyer S' à   h(S')
      Fsi
    FinPour
  Sinon
    termi:= vrai
  Fsi
 Fait
Fin algorithme
```

4.2.1.1. Gestion de la terminaison:

Le principe utilisé pour la détection de la terminaison est le suivant:
La vérification à la volée consiste à effectuer la vérification au moment de la génération des configurations et cette méthode permet d'arrêter l'analyse dès que la valeur de vérité de la formule est connue. Donc si une machine détecte la valeur de vérité de la formule, elle envoie un message de type END_CHECK au coordinateur, ce dernier diffuse un message d'arrêt à toutes les machines pour arrêter le calcul.

4.3. Vérification d'une partie de TPN-TCTL à la volée

Une vérification à la volée explore l'espace d'états d'un modèle en même temps que la propriété est vérifiée. À la différence des méthodes générant la totalité de l'espace d'états, cette méthode permet d'arrêter l'analyse dès que la valeur de vérité de la propriété est connue. Cette méthode a montré son efficacité pour l'analyse de systèmes temps réel, son efficacité a été prouvée sur les automates temporisés avec des outils tel qu'UPPAAL. Bien que la classe de propriétés vérifiables soit moindre que pour les autres méthodes de vérification, **[Gar05]** des études récentes ont montré que dans la plupart des cas, elles étaient suffisantes pour l'analyse des systèmes.

Nous proposons dans cette partie une solution basée sur la méthode du calcul distribué de l'espace d'états (zones) (présentée ci dessous), l'idée générale pour la vérification à la volée de propriétés TCTL est : utiliser le graphe des zones pour calculer de manière distribuée l'espace d'états du système et vérifier la propriété au fur et à mesure du calcul (technique de vérification à la volée).

Dans cette partie, on présente un sous ensemble de propriétés TPN-TCTL pour lesquelles nous donnons des algorithmes à la volée pour vérifier leur validité.

4.3.1. Définition

TPN-TCTL est la sous classe des propriétés de logique temporelle TCTL définie par **[Gar05]**:

TPN-TCTL::= $\exists\varphi \mathrm{U}_I \psi$/ $\forall\varphi \mathrm{U}_I \psi$/ $\exists \mathrm{F}_I \varphi$ / $\forall \mathrm{F}_I\ \varphi$ / $\exists \mathrm{G}_I \varphi$ / $\forall\varphi \mathrm{G}_I$.
où φ, $\psi \in$ GMEC I$\in$ {[a,b], [a,b), (a,b], (a,b), avec a $\in\mathbb{N}$, b$\in \mathbb{N}\cup\{\infty\}$.

Définition

Soit un TPN avec n places P={ p_1, p_2, . . . , p_n}. Une GMEC (Contrainte d'Exclusion Mutuelle Généralisée) est définie inductivement par:

GMEC::= $(\sum_{i=1}^{n} a_i * M(p_i)) \bowtie$ c / $\varphi \vee \psi / \varphi \wedge \psi / \varphi \Rightarrow \psi$
où $a_i \in \mathbb{Z}$, M est un mot clé $p_i \in$P, $\bowtie\in\{\leq, \geq, <, >, =\}$, c$\in\ \mathbb{N}$ et $\varphi, \psi\ \in$GMEC. Les opérateurs ($\vee, \wedge, \Rightarrow$) ont la signification habituelle.

La signification de M(p_i)$\bowtie$ c est que le marquage courant de la place p_i est en relation $\bowtie$ avec c.

Par la suite, nous présentons les algorithmes qui permettent de vérifier les propriétés TCTL en parallèle avec la génération distribuée de l'espace d'états.

La génération distribuée de l'espace d'états permet de répartir les états entre les différentes machines qui permet donc de pallier au problème de l'explosion combinatoire de l'espace d'états dans le cas de systèmes temps réel de grande taille. Ce qui implique la distribution des algorithmes de vérification.

La vérification distribuée consiste à répartir les tâches de vérification entre plusieurs machines c-à-d faire coopérer plusieurs machines pour exécuter les algorithmes de vérification.

Dans ce qui suit, nous présentons les algorithmes de vérification d'un sous ensemble de propriétés quantitatives exprimées en logique temporelle TCTL. De même, nous nous intéressons à une vérification à la volée c-à-d : vérifier au moment de la génération distribuée ce qui permet d'arrêter la génération dès que la valeur de vérité de la formule est connue.

4.4. Les algorithmes

4.4.1. Le model-checking de $\exists\varphi\mathbf{U}_I\psi$

Le but de l'algorithme est de détecter une exécution (s_0,v_0),...,(s_n,v_n) telle que pour tous les états (s_i,v_i) $\models\varphi$ et (s_n,v_n) $\models\psi$ dans l'intervalle de temps spécifié I.

Algorithm 4 *check EU(s, φ, ψ, I)*

Visited := $\emptyset$
s_0 = post(M_0, **0**) /// **0** = Z_0
retourner *check* EU_{aux}*(s, φ, ψ, I)*

Algorithm 5 *check EU_{aux}(s, φ, ψ, I)*

(M, Z):= s
si M $\models\psi\wedge$ Z $\cap$ I $\neq \emptyset$ alors
retourner true
diffuser (terminaison) /propriété vérifiée alors arrêt de la génération/
sinon si M $\neg \models \varphi$ alors
retourner false
diffuser(terminaison) /propriété non vérifiée /
sinon
Z':= Z $\cap$ [0, Imax)
Si Z' = $\emptyset$ alors
retourner false
diffuser(terminaison) /propriété non vérifiée /
Sinon
Visited := visited $\cup$ {s}
Pour tout t $\in$ firable (M, Z') faire
s' := post(post$_t$(M, Z')) /successeur temporel du successeur discret/
si h(s')=i alors
si s'$\not\subseteq$ visited alors
si check EU_{aux}(s', φ, ψ, I) alors
retourner true
diffuser(terminaison) /propriété vérifiée /
fsi
fsi
sinon
envoyer(check EU_{aux}(s', φ, ψ, I) à h(s')
fsi
Fin pour
retourner false
envoyer au coordonateur proprieté non verifiée // il n'ya pas de successeurs qui verifient ψ
Fsi
Fsi
Falgorithme

4.4.2. Le model-checking de $\exists \mathbf{G}_I \varphi$

Algorithm 6 check ∃G(s, φ, I)

Visited := ∅
s_0 = post(M_0, **0**) /// **0** = Z_0
retourner *check* EG_{aux}*(s, φ, I)*

Algorithm 7 *check* EG_{aux}*(s, φ, I)*

(M, Z):= s
si M $\neg \models \varphi$ alors
 retourner false
 diffuser (terminaison) //propriété non vérifiée //
sinon
 Visited := visited ∪ {s}
 Si I⊆ Z ∩ I alors
  retourner true
  diffuser (terminaison) //propriété vérifiée //
 sinon // chercher les successeurs s'ils vérifient φ
  Pour tout t ∈ firable (s') faire
   s' := post(post$_t$(s)) / successeur temporel du successeur discret/
   si s'$\not\subseteq$ visited alors
    si h(s')=i alors
     si M si check EGaux(s',φ,I) alors
      retourner true
      diffuser (terminaison) //propriété vérifiée //
      Fsi
     Sinon envoyer(EGaux(s', φ, I) à h(s')
     Fsi
   fsi
  Fpour
 retourner false //pas de successeurs qui vérifient la propriete//
 envoyer au coordonateur proprieté non verifiée // il n'ya pas de successeurs qui verifient φ
 Fsi
Fsi
Falgorithme

4.4.3. Le model-checking de $\forall\varphi\mathbf{U}_I\psi$

Soit Z = (M, Z) un état symbolique pour lequel on veut tester la propriété $\forall\varphi\mathbf{U}_I\psi$.

Intuitivement, on peut itérativement décider de sa valeur de vérité de la façon suivante :

— S'il existe des évaluations de z plus grande que Imax, alors la propriété est fausse.
— Si M $\models \psi$ et que toutes les évaluations de l'horloge z de Z sont dans I alors la propriété est vraie pour cet état.
— Si M $\neg \models \varphi$ alors la propriété est fausse.
— on calcule les successeurs temporels de Z .
— on supprime tous les états tels que ψ est vraie et z∈I et on note (M, Z') ce nouvel état symbolique. Ces états sont supprimés car ils vérifient tous la propriété : on a seulement besoin de tester les états qui n'ont pas encore validés ψ pour des valuations z <I_{min} .
— on calcule les successeurs par le franchissement de transitions.

Algorithm 8 check **AU(s,φ,ψ, I)**

Visited := ∅
prefix:= ∅
s_0 = post(M_0, **0**) /// **0** = Z_0/successeur temporel/
retourner *check* AU_{aux}*(s, φ,ψ I)*

Algorithm 9 check **AU(s,φ,ψ, I)**

Z' := Z
(M, Z):= s
Z = $Z_{(0,I_{min})} \cup Z_{(I_{min},I_{max})} \cup Z_{(I_{max},inf)}$
si M $\models \psi$ alors
si $Z_{(I_{max},inf)} \neq \emptyset$ alors
retourner false
diffuser (terminaison) //propriété non vérifiée /
Fsi
Z':= Z $\cap$ [0, I_{max})
sinon si M $\neg \models \varphi$ alors
retourner false
diffuser (terminaison) //propriété non vérifiée //
sinon
Z' := timeNext(Z) // successeur temporel
Visited := visited $\cup$ {s}
Pour tout t $\in$ firable (M, Z') faire
Prefix:= prefix $\cup$ s
s' := discretNext(M, Z',t) //successeur discret
si h(s')=i alors
si s'$\in$ prefix alors
retourner false
diffuser (terminaison) //propriété non vérifiée //
Fsi
Si s'$\notin$ visited alors
si not check AUaux(s', φ, ψ, I) alors
retourner false
diffuser (terminaison) //propriété non vérifiée //
Fsi
Fsi
prefix := prefix \ s
Sinon envoyer (check AUaux(s', φ, ψ, I) à h(s')
Fin pour
retourner true
diffuser (terminaison) //propriété vérifiée //
Fsi
Falgorithme

Nous avons présenté ici les algorithmes de vérification des propriétés $\exists\varphi\mathbf{U}_I\psi$,

$\exists \mathbf{G}_I \varphi$, et $\forall \varphi \mathbf{U}_I \psi$. Les autres propriétés peuvent être déduites en utilisant les définitions suivantes:

4.4.4. Le model-checking de $\exists \mathbf{F}_I \varphi$

Par définition $\exists \mathbf{F}_I \varphi \equiv$ True$\mathrm{U}_I \varphi$.

4.4.5. Le model-checking de $\forall\ \mathbf{G}_I \varphi$

Par définition $\forall\ \mathbf{G}_I \varphi \equiv \neg \exists \mathrm{F}_I \varphi$.

4.4.6. Le model-checking de $\forall \mathbf{F}_I \varphi$

Par définition $\forall \mathrm{G}_I \varphi \equiv \forall$true $\mathrm{U}_I \varphi \equiv \neg \exists G_I \neg \varphi$

4.5. Conclusion

Dans ce chapitre, nous avons proposé un algorithme de génération distribuée de l'espace d'états pour remédier au problème de l'explosion combinatoire de l'espace d'états. La génération distribuée peut se faire en utilisant un ensemble de machines reliées en réseau.

Nous avons également proposé des algorithmes pour la réalisation de la vérification distribuée et à la volée (vérification au moment de la génération distribuée de l'espace d'états). Essentiellement nous avons présenté des algorithmes pour la vérification d'un sous ensemble de propriétés quantitatives exprimées en logique temporelle TCTL sur les réseaux de Petri temporels bornés. Bien que ce sous ensemble de la classe de propriétés vérifiables soit moindre que pour les autres méthodes (non à la volée) de vérification, **[Gar05]** des études récentes ont montré que dans la plupart des cas, elles étaient suffisantes pour l'analyse des systèmes.

Dans le chapitre suivant nous présentons les résultats de ces différents algorithmes que nous avons implémenté.

Chapitre 5

Applications et resultats

5.1. Introduction

Afin d'analyser les performances et la complexité des algorithmes proposés dans le chapitre précédent, nous avons procédé à l'implémentation de ces derniers en utilisant le language C++. Ce dernier offre un mécanisme simple et efficace, permettant la communication et les envois des messages entre les processus, qui est le principe des *sockets*.
Dans ce chapitre, nous allons discuter cette implémentation en présentant les resultats des tests concernant les différents algorithmes distribués.

5.2. Génération distribuée du graphe des zones

Soit un réseau de Petri temporel TPN. Les configurations (états) successeurs possibles pour une configuration (M,z) peuvent être calculées à partir de la sémantique des réseaux de Petri temporels (transitions discrètes et transitions continues). L'application de l'algorithme présenté dans le chapitre 4 permet de générer d'une façon distribuée l'ensemble des configurations (graphe des zones) d'un

réseau de Petri temporel. Cependant l'attribution des configurations aux machines se fait à l'aide d'une fonction de hachage comme sera présenté dans le paragraphe suivant.

5.3. Fonction d'attribution des configurations aux machines

Nous considérons une fonction de répartition h qui associe à chaque marquage d'une configuration (composé du marquage et de la zone) un entier compris entre 0 et N-1, N est le nombre de machine. Nous calculons la fonction de hachage de la façon suivante:

— Une fonction coef qui génère de façon aléatoire des entiers.
— On calcule pour chaque marquage un numéro $1 \leq n \leq Nbre$de procéssus en fonction de la fonction aléatoire qui n'est pas unique.

5.4. Tests et résultats

Nous avons testé les algorithmes sur 3 machines ayant 512 mo de RAM chacune. Les algorithmes implémentés et testés sont les suivants:

— L'algorithme distribué de génération de l'espace d'états
— Les algorithmes distribués du model-checking de propriétés TCTL vérifiables à la volée.

5.4.1. Test de la génération distribuée de configurations (espace d'etats en utilisant les zones)

Concernant la génération de l'espace d'états, nous avons considéré les résultats suivants:

1. La répartition des états se fait selon la fonction de hashage, le nombre de configurations dans chaque machine est calculé. Le tableau suivant montre les résultats.
2. Les performances de l'algorithme: l'avantage de la distribution est d'éviter le problème de l'explosion combinatoire de l'espace d'états, et comme nous avons utilisé le principe de vérification à la volée qui permet d'arrêter la génération dès que la valeur de vérité de la formule est connue, en plus sur une platforme distribuée donc le temps d'exécution est réduit énormément et le problème de l'explosion combinatoire est complètement évité.

5.4.2. Résultats de la vérification distribuée des formules TCTL

Le principe de vérification utilisé est la vérification à la volée, donc dès que la valeur de vérité de la formule TCTL introduite est connue, la génération est arrêtée par tous les processus. Le test de la vérité des formules TCTL à la volée et dans le cas distribué montre des bonnes performances. Les machines coopèrent pour vérifier la formule. Une fois que l'une des machines termine la vérification elle envoie au coordinateur le résultat et ce dernier diffuse un message d'arrêt à toutes les machines qui arrêtent le calcul. Les résultats sont présentés comme dans l'exemple suivant:
Les algorithmes ont été appliqués aux problème des philosophes qui est un problème très connu en informatique. Un philosophe peut avoir les états suivants:

— penser pendant une période de temps (intervalle de temps associé à cet évènement)
— manger pendant une période de temps déterminée afin de permettre aux autres de manger.
— une fonction de hachage non aléatoire a été utilisée pour le problème des philosophes afin d'avoir une distribution équilibrée.

Le tableau suivant montre les premiers résultats, en analysant les résultats du tableau on déduit que l'aspect temporel affecte le résultat de la vérification:

— Nbre de philo: représente le nombre de philosophes
— Nbre de conf: représente le nombre d'états symboliques
— Résultat: représente la valeur de vérité de la formule (propriété)
— Temps: représente le temps de calcul CPU
— NB proc: représente le nombre de procéssus utilisé pour la génération et la vérification

Formule TCTL	*Nbre de philo*	*nbre de conf*	*Résultat*	*Temps(s)*	*NB proc*
$EtrueU_{[0\,5]}p_7$	*6*	*12*	*formule vérifiée*	*<0.01*	*3*
$Ep_0U_{[0\,2]}p_{70}$	*6*	*7*	*formule verifiée*	*<0.01*	*3*
$Ap_0U_{[0\,2]}p_{70}$	*6*	*1*	*formule non verifiée*	*<0.01*	*3*
$Ep_0U_{[0\,2]}p_{70}$	*8*	*81*	*formule vérifiée*	*<0.06*	*3*
$AtrueU_{[0\,8]}p_{22}$	*8*	*15*	*formule vérifiée*	*<0.01*	*3*
$AtrueU_{[0\,8]}p_{22}$	*10*	*22*	*formule vérifiée*	*<0.02*	*3*
$Ap1U_{[0\,8]}p_{29}$	*10*	*1*	*formule vérifiée*	*<0.00*	*3*
$Ep_0U_{[0\,2]}p_{10}$	*10*	*79*	*formule vérifiée*	*<0.04*	*3*
$Ep_0U_{[4\,5]}p_{13}$	*10*	*21*	*formule non verifiée*	*<0.01*	*3*
$Ep_0U_{[0\,2]}p_{13}$	*10*	*67*	*formule vérifiée*	*<0.04*	*3*
$Ep_0U_{[0\,2]}p_{13}$	*10*	*55*	*formule vérifiée*	*<0.08*	*1*
$Ep_0U_{[0\,5]}p_{58}$	*20*	*1012*	*formule vérifiée*	*0.46*	*3*
$Ep_0U_{[0\,5]}p_{19}$	*20*	*246*	*formule vérifiée*	*0.14*	*1*
$Ep_0U_{[0\,5]}p_{13}$	*20*	*1324*	*formule vérifiée*	*0.62*	*3*
$Ep_0U_{[0\,5]}p_{43}$	*20*	*3756*	*formule vérifiée*	*5.80*	*1*
$Ep_0U_{[0\,5]}p_{43}$	*20*	*147*	*formule vérifiée*	*0.09*	*3*
$Ep_0U_{[0\,5]}p_{49}$	*20*	*7201*	*formule vérifiée*	*12.89*	*1*
$Ep_0U_{[0\,5]}p_{49}$	*20*	*427*	*formule vérifiée*	*0.17*	*3*
$Ep_0U_{[0\,5]}p_{13}$	*25*	*11527*	*formule vérifiée*	*11.08*	*3*
$Ep_0U_{[0\,5]}p_{58}$	*25*	*206*	*formule vérifiée*	*0.19*	*3*
$Ep_0U_{[0\,5]}p_{58}$	*25*	*8711*	*formule vérifiée*	*14.57*	*1*
$Ep_0U_{[0\,5]}p_{10}$	*25*	*14431*	*formule vérifiée*	*26.32*	*2*

5.5. Conclusion

Ce chapitre a été consacré à la présentation des résultats de test de la vérification à la volée des propriétés TCTL sur une platforme distribuée en utilisant les réseaux de Petri temporels comme formalisme de modélisation et la méthode des zones pour la génération de l'éspace d'états (l'ensemble de configurations). Les résultats des tests montrent que la vérification à la volée et distribuée est un moyen très efficace pour pallier au problème de l'explosion combinatoire de l'espace d'états, en augmentant la capacité de stockage et un gain en rapidité de vérification (calcul).

Conclusion générale

Les systèmes temps réel se distinguent des autres systèmes informatiques par le fait qu'ils soient soumis à des contraintes temporelles. Il est donc impératif d'avoir des outils et techniques de modélisation et de vérification qui puissent garder un haut degré de performance et de correction pour ce type de système. Par conséquent le temps dans un tel système doit être pris en considération d'une manière explicite durant la phase de modélisation et de vérification. Il existe plusieurs outils et techniques de modélisation des systèmes informatiques (files d'attente, chaines de Markov,...), dont plusieurs ont pu prouver leurs puissance et performances dans la modélisation et la vérification des systèmes. Parmi les outils courament utilisés pour modéliser les systèmes, les réseaux de Petri (RdP) occupent une place priviligiée. Les réseaux de Petri sont un outil trés utilisé car ils ont montré une grande puissance d'expression et de calcul pour la modélisation et la vérification de nombreux et différents types de systèmes, et ils fournissent des méthodes d'analyse avec lesquelles on peut vérifier certaines propriétés de systèmes (bornitude, vivacité,...).

Cependant, quand il s'agit d'un système temps-réel les RdPs atteignent trés vite leurs limites. Ceci est dû au fait que les RdPs ne prennent pas en considération le caractère temporel des systèmes. Ils se contentent d'effectuer une modélisation et une vérification du comportement logique d'un système hors son contexte temporel. Afin de palier à ce problème, et pour pouvoir profiter de la puissance

des RdPs, des extensions temporelles des RdPs ont été proposées. Le but principal de telles extensions est de pouvoir éffectuer une modélisation et une vérification du système sur le plan logique du séquencement des évènement, et surtout sur le plan temporel primordial dans le cadre de la représentation des systèmes temps-réel.

Les propriétés générales telles que la bornitude, la vivacité, le non blocage ... reseignent le modélisateur sur le comportement général du système, celles-ci doivent être complétées par l'analyse des propriétés spécifiques du système modélisé.
Plusieurs axes de recherche ont été développés visant à mettre en place des outils permettant une description formelle des propriétés spécifiques des systèmes. La logique temporelle est un outil formel qui procure une syntaxe sûre, précise et sans ambiguïté des propriétés qualitatives et quantitatives. Cependant pour pouvoir exprimer des propriétés spécifiques des systèmes temps-réel c-à-d des propriétés quantitatives, des extensions temporelles des différentes logiques temporelles ont été proposées, telle que la logique TCTL (Timed CTL).

Aprés la description des propriétés spécifiques, viendra l'étape vérification de ces propriétés sur le système.Parmi les techniques de vérification: les approches par model-checking, "vérification automatique" se sont largement développées ces dernières années**.** Cela suppose de modéliser le comportement du système à vérifier et d'énoncer la propriété de correction attendue sous la forme d'une formule de logique temporelle. En suite on peut utiliser un model-checker afin de vérifier si le modèle satisfait ou non la propriété.

Néanmoins, la vérification et la validation des systèmes temps réel souffrent de l'éxplosion combinatoire non seulement à la taille du système, mais aussi au nombre d'horloges.

Pour pallier ce problème, une nouvelle approche a été introduite qui consiste à éffectuer une vérification à la volée des proprietés TCTL (vérifiable à la volée)

sur un ensemble de procéssus interconnectés et communiquant en échangeant des messages. Ce qui permet d'éffectuer une vérification à la volée sur une platforme distribuée.

Dans ce cadre, nous avons proposé un algorithme de génération distribuée de l'espace d'états d'un système temps réel modelisé par un RdPT en utilisant la méthode des zones pour la génération de l'ésoace d'états, nous avons proposé également des algorithmes de vérification à la volée et distribuée d'une sous classe de propriétés TCTL vérifiables à la volée.

Afin de valoriser notre travail, nous avons implémenté ces algorithmes pour prouver par les tests les avantages d'une vérification à la volée sur une platforme distribuée. Les tests montrent les avantages que peut apporter une vérification à la volée et distribuée par rapport aux tailles des systèmes qu'on peut vérifier ou par rapport au temps de calcul.

Comme perspectives il serait interessant de:

— Compléter notre vérificateur par l'implémentation d'un outil distribué de vérification à la volée en utilisant la méthode des zones sur les RdPT.
— Integrer notre vérificateur à l'outil Roméo.
— Enrichir notre vérificateur pour les RdPT non sauf en considérant la multisensibilisation.
— Enrichir notre travail pour le cas des RdPT contenant l'infini comme date de tir au plus tard des transitions en proposant une approximation adéquate.
— Proposer une vérification distribuée non à la volée pour permettre la vérification de toutes les propriétés TCTL et fair TCTL.
— La définition des algorithmes de répartition à partir de l'analyse temporelle des différents états.

Bibliographie

[Alu94a] :ALUR R., DILL D., « A Theory of Timed Automata », Theoretical Computer Science, vol. 126, n"°2, p. 183-235, 1994.

[Bbd03] :B.Berthomieu, M.Boyer, M.Diaz «les réseaux de Petri :chapitre5 'les réseaux de Petri temporels'» Hermes, 2003

[Ber00] :BÉRARD B., DUFOURD C., « Timed Automata and Additive Clock Constraints »,Information Processing Letters, vol. 75, n"°1-2, p. 1-7, 2000

[Ber01] :B. Berthomieu «La méthode des classes d'états pour l'analyse des réseaux temporels», In modélisation des systèmes réactifs (MSR'01), pages 275-290, Toulouse, France, octobre 2001. Hermes.

[Bhv01] :G. Behrmann, T. Hune, F. Vaandrager « Distributing Timed Model Checking», by esprit Project 26270, verification of Hybrid Systems (VHS), 2001.

[Bme83] :B.Berthomieu, M.Menasche «an enumerative approch for analysing time Petri nets" IFIP congress series, vol 9,p 41-46,1983

[Bou02] :P. Bouyer «Modèles et Algorithmes pour la Vérification des Systèmes Temporisés» phd thesis,Laboratoire Specification et Verification. ENS Cachan,France, apr 2002.

[Bou04a] :P. Bouyer« Forward Analysis of Updatable Timed Automata », Formal Methods in System Design, vol. 24, n"°3, p. 281-320, 2004.

[Bou05] :Patricia Bouyer, «On conciseness of extensions of Timed Automata», journal of automata , languages and combinatorics, 2005.

[Bou07] :M.C Boukala, L. Petrucci «Toward distributed verification of Petri nets properties» VECo'S07, 1st international workshop on verification and evaluation of computer and communication systems May 2007

[Boy01] :M. Boyer « Contribution à la modélisation des systèmes à temps contraint et application au multimédia», Thèse de doctorat, Université Toulouse 3 Paul abatier, 2001.

[BRG05] :H. Boucheneb, H. Roux & G. Gardey « TCTL model checking of Time Petri Nets » 2005.

[Cla86] :E.M. Clarke, A. Emerson & A.P. Sistla « Automatic vérification of finite-state concurrent systems using temporal logic specifications ». CM transaction on programming languages and systems : p.244-263, April 1986.

[Eme96] :E.A.Emerson,A.P.Sistla «Symetrie and Model-Checking »,In formal Methods and System Design 9,pp105-031, 1996

[Gar05] :G. Gardey, «contribution à la vérification et au contrôle des systemes temps réel. Application aux réseaux de Petri temporels et aux automates temporisés», thèse de doctorat École centrale de Nantes et l'université de Nantes, Décembre 2005.

[GRR03] :G. Gardey, O.H. Roux and O.F. Roux ,«using zone graph method for computing the state space of a time Petri net». In In formal modeling and analysis of timed systems,(FORMATS'03), volume LNCS 2791. Springer-Verlag, September 2003.

[Gsh02] :A.Gu, K.G.Shin «Analysis of event-driven real time systems with time Petri nets, A translation approch»,IFIP conference proceedings, vol,219.Proceedings of the IFIP 17th world computer congress-Tc10 stream on distributed and parallel Embedded systems :design and analysis of distributed embedded systems,pages 31-40, 2002

[Had01] :S. Haddad, F. Vernadat « Méthodes d'analyse des réseaux de Petri in Les réseaux de Petri. Modèles fondamentaux». Hermes Science, traité IC2 Information-Command-Communication, ISBN é-7462-0250-6, 2001.

[Had03] :S. Haddad, F.Vernadat «vérification et mise en oeuvre des réseaux de Petri». Vérification des propriétés spécifiques Paris,Hermes science publication, 2003

[Hen98] :HENZINGER T. A., KOPKE P. W., PURI A., VARAIYA P., « What's Decidable about Hybrid Automata ? », Journal of Computer and System Sciences, vol. 57, n"°1, p. 94-124, 1998.

[Jor99] :J.B. Jorgensen, L.M. Krinsten.«Computer aided verification of Lamportas fast mutual extension algorithme using Colored Petri Nets and occurence graphs with symetries». IEEE transaction on parallel and distributed systems ;1999.

[Jou00] :jean Pierre.Jouannaud : «cours vérification des systèmes réactifs temps réel» 2000.

[Kha97] :W. Khansa : « Réseaux de Petri p-temporels : contribution à l'étude des systèmes à évènements discrets » Thèse de doctorat, Université de Savoie, Annecy, France 1997.

[Kho06] :Ahmed Khoumsi, Mustapha Nour elfath «Méthode de transformation d'Automates temporisés avec invariants de localités».6e conférence de Modélisation simulation. Maroc. avril 2006.

[Lar98] :F. Laroussine, K.G.Larsen «CMC :A tool for compositional model-checking of real-time systems». Proc IFIP Joint Int. conf Formal description techniques & protocol specification, Testing and verification (Forte-PSTV'98), kluwer Academic publishers, 1998, P.439-456

[Lar05] :F. Laroussine, «Model-checking temporisé : Algorithmes efficaces et complexité». Theoretical computer science 2005

[Lar95] :K.G. Larsen.,P. Y1 W. Pettersson «Model-checking for real-time systems», Proc 10th International Conference on Fundamentals of Computation Theory (FCT'05),vol.965 de lecture Notes in computer science, springer,p. 62-88,1995.

[Lar97] :K.G. Larsen.,P. Y1 W. Pettersson «Uppal in a Nutshell», journal of software tools for technology transfer, vol 1, n˚1-2, p. 134-152, 1997.

[Loh02] :C. LOHR, «contribution à la conception de systèmes temps réel en s'appuyant sur la technique de description formelle RT-LOTOS». Thèse de doctorat, institut national polytechnique de Toulouse. France 2002.

[Man91] :Z. Manna & A. Pnueli « the temporal logic of reactive and concurrent systems » Springer Verlag, 1991.

[Mer74] :P. Merlin «A study of the recoverability of computer systems» PHD. Thesis, univ of California Irvine, 1974

[Pet05] :L. Petrucci, C.Lakos.«Distributed and modular state space exploration for timed Petri nets», LIPN, CNRS UMR 7030, Université Paris VIII, 2005.

[Pet00] :P.K.Pettersson, G.Larsen «UPPAL 2k» Bulletin of the european association for theoretical computer science, vol 70, 2000 P 40-44

[Pet62] :C.A.Petri «kommunikation mit automaten» phd thesis, university of bonn 1962

[Ram74] :C. Ramchandani.«Analysis of asynchronous concurrent systems by timed Petri nets». PhD thesis. Massachusets Institute of technology. Cambridge, MA, 1974. Project MAC report MAC-TR-120.

[Ras05] :RASKIN J.-F., « An Introduction to Hybrid Automata », Chapitre Handbook of Net-worked and Embedded Control Systems, p. 491-518, Springer, 2005.

[Rob04] :ROBIN A., «Aux frontières de la décidabilité...», Master's thesis, DEA Algorithmique, Paris, 2004.

[Sch01] :P. Schnoebelen, B. Bérard, M.Bidoit, F.Laroussinie,A. Petit «Systems and software verification-model-checking techniques and tools», Springer, 2001.

[Yov93] :S. Yovine «Méthodes et outils pour la vérification symbolique des systèmes temporisés». thèse de doctorat, INP de Grenoble, France, 1993

[Yov97] :S. Yovine,«Kronos : A verification tool for real-time systems», journal of software tools for technology transfer, vol 1, n°1-2, p, 123-133, 1997

[Zou98] :M. Zouaoui «définition d'un langage de type logique temporelle pour la spécification et l'évaluation des performances» thèse de doctorat d'état, univ Pierre et Marrie Curie, Paris, 1998