# Table des matières


**TABLE DES MATIERES** .......... **1**

**CHAPITRE 1 : INTRODUCTION** .......... **3**

1. MISE EN SITUATION .......... 3
2. METHODES DE VERIFICATION DE SYSTEMES .......... 3
3. OBJECTIF .......... 4
4. STRUCTURE DU DOCUMENT .......... 5

**CHAPITRE 2 : LES AUTOMATES** .......... **6**

1. DEFINITION DE BASE .......... 6
2. SEQUENCES ET ARBORESCENCES .......... 7
3. LES AUTOMATES .......... 9
4. TRADUCTION D'AUTOMATES .......... 15

**CHAPITRE 3 : LES LOGIQUES TEMPORELLES** .......... **18**

1. PRESENTATION DES LOGIQUES TEMPORELLES .......... 18
2. EXEMPLES DE LOGIQUES .......... 20
   - *La logique LTL* .......... *20*
   - *La logique CTL* .......... *22*
   - *La logique TLA* .......... *25*
   - *La logique ACTL* .......... *28*
   - *La logique MITL* .......... *30*
   - *La logique TCTL* .......... *32*

**CHAPITRE 4 : MODEL CHECKING** .......... **35**

1. ALGORITHMES .......... 35
   - *Algorithme pour la logique* .......... *35*
   - *Algorithme pour la logique* .......... *42*
   - *Algorithme pour la logique* .......... *44*
   - *Algorithme pour la logique* .......... *46*
2. PROBLEME LIE A L'ALGORITHME DE MODEL CHECKING SYMBOLIQUE TCTL .......... 50

**CHAPITRE 5 : MODEL CHECKING SYMBOLIQUE POUR TCTL** .......... **51**

1. DEFINITION PRELIMINAIRE .......... 51
2. CARACTERISATION PAR POINT FIXE DES OPERATEURS $\forall U$ ET $\exists U$ .......... 55
   - *Prédécesseurs en un pas* .......... *55*
   - *Opérateur $\exists U$* .......... *56*
   - *Opérateur $\forall U$* .......... *57*
3. ALGORITHME DE MODEL CHECKING SYMBOLIQUE .......... 59
   - *Algorithme* .......... *59*
   - *Explication de l'algorithme* .......... *60*

4. REPRESENTATION DES CONTRAINTES ........ 60
*Borne* ........ *60*
*Matrice de bornes* ........ *61*
5. ELIMINATION DES QUANTIFICATEURS ........ 65

**CHAPITRE 6 : IMPLANTATION DE L'ALGORITHME ........ 67**

1. PARSING DE LA LOGIQUE TCTL ........ 67
2. ANALYSE ORIENTEE OBJET D'UN MODEL CHECKER SYMBOLIQUE POUR TCTL ........ 70
*Schéma* ........ *70*
*Explication* ........ *71*
3. IMPLANTATION DES OPERATEURS DE POTENTIALITE ET D'INEVITABILITE ........ 71
*Implantation de l'opérateur $\exists U$* ........ *71*
*Implantation de l'opérateur $\forall U$* ........ *72*
4. CODE ........ 73
5. EXEMPLE ........ 74
6. COMPARAISON AVEC KRONOS ........ 78

**CHAPITRE 7 : UNE LOGIQUE TEMPORISEE ORIENTEE ACTION ........ 82**

1. MOTIVATION ........ 82
2. PRESENTATION DE LA LOGIQUE ATCTL ........ 83
3. ALGORITHME DE VERIFICATION ........ 84
*La fonction $f_g$* ........ *84*
*La fonction $f_f$* ........ *86*
*Code* ........ *88*
4. SECOND ALGORITHME DE VERIFICATION ........ 89

**CHAPITRE 8 : APPLICATION ........ 92**

*Le système* ........ *92*
*Résultat* ........ *92*

**CHAPITRE 9 : CONCLUSION ........ 97**

**BIBLIOGRAPHIE ........ 100**

**ANNEXE : CODE DU MODEL CHECKER TCTL ET DU PRECOMPILATEUR ATCTL**

# Chapitre 1

# Introduction

## 1. Mise en situation

Il est souvent souhaitable, voire indispensable, de s'assurer qu'un programme est correct, c'est à dire qu'il ne cycle pas, n'ait pas de deadlock,…et qu'il fasse bien ce qu'il est sensé faire. Pour s'en assurer, on peut effectuer une simulation avec certaines valeurs d'entrée mais, le domaine de ces valeurs d'entrées étant souvent infini, il est impossible de faire une vérification sur l'ensemble de ces valeurs d'entrée.
Le problème de la vérification d'un algorithme ou d'un système sans pouvoir énumérer tous les cas possibles se pose donc.
Ces algorithmes ou systèmes peuvent souvent être vu comme orientés vers

- Le traitement de données (programme qui effectue des calculs)
- L'interaction avec l'environnement.

Ces derniers sont appelés programmes ou systèmes réactifs : ils subissent des stimuli de la part de leur environnement et doivent y répondre directement.
Dans ce qui suit, nous traiterons plutôt de tels systèmes.

## 2. Méthodes de vérification de systèmes

Pour prouver formellement un système ou un algorithme, ou prouver qu'il possède certaines propriétés, il est nécessaire que ce système ou cet algorithme soit exprimé dans un langage ayant une sémantique formelle. Plusieurs types de langages de spécifications peuvent être utilisés pour cela. On ne considère pas ici les langages de programmation donnant une "description" trop concrète du programme ou système. Nous considérons dans ce travail que le langage utilisé permet de spécifier les évolutions possibles du système en utilisant une sémantique

- opérationnelle, c'est à dire décrivant en particulier les transitions possibles du système à tout moment.
- d'entrelacement, qui suppose que deux événements ne peuvent avoir lieu en même temps.

Des exemples de tels langages sont le langage à commandes gardées de Dijkstra [RO97], pour les algorithmes de traitement, ou CCS [Mil80] [Mil89] et LOTOS [Léo97] pour les systèmes réactifs.
A partir d'une spécification, nous pouvons alors définir un graphe décrivant les états et évolutions possibles du système spécifié. Les sommets de ce graphe représentent les différents états dans lesquels peut se trouver le système et les arcs les transitions pouvant être effectuées par le système à partir des différents états. La sémantique du système est entièrement contenue dans le graphe. Le graphe peut être fini ou infini.
Pour résoudre le problème de vérification d'algorithmes ou de systèmes, deux grandes catégories de méthodes de vérification sont généralement utilisées.
La première, développée initialement par Floyd, Dijkstra et Hoare plus spécifiquement pour les algorithmes de traitement de données, consiste à définir et utiliser un système de preuve de programme pour inférer la preuve du bon fonctionnement du programme. L'utilisation de ce genre de techniques implique la définition d'invariants de boucles qu'aucune méthode systématique ne permet généralement de définir. Une implantation informatique fournissant automatiquement une preuve de programme est donc en règle générale impossible.
Nous nous intéresserons dans ce mémoire à une deuxième catégorie de méthodes de vérification : le Model Checking, utilisé généralement pour les systèmes réactifs. Le Model Checking permet de vérifier que le comportement d'un système satisfait certaines propriétés.
Nous étudions plus particulièrement les spécifications de systèmes temporisés. Le modèle du temps sera celui d'une valeur évoluant de façon continue (temps dense) de la même façon partout dans le système (temps absolu)[1]. Etant donné la propriété du temps, le graphe donnant l'ensemble des évolutions possibles d'un système est en général infini et ne peut pas être manipulé tel quel par des outils informatiques.
Une représentation généralement finie des systèmes temporisés est donnée par les automates temporisés. Des vérifications de propriétés pour de tels systèmes peuvent alors être réalisées.
Nous supposerons ici que ces propriétés sont spécifiées à l'aide de logiques temporelles.

## 3. Objectif

L'objectif de ce travail est l'implantation d'un algorithme de Model Checking. Cet algorithme doit permettre de vérifier des propriétés sur des systèmes temporisés. Plusieurs objectifs secondaires peuvent être déterminés pour atteindre cet objectif principal. Tout d'abord, il faut comprendre et se familiariser avec les idées, concepts, méthodes, etc. qu'on retrouve dans le cadre du Model Checking. Pour cela, il faut se familiariser avec des logiques temporelles classiques, et avec les modèles classiques utilisés pour représenter les systèmes dont on veut vérifier les propriétés, comprendre et assimiler des algorithmes de Model Checking classiques utilisant les logiques temporelles et les modèles avec lesquels on s'est familiarisé.
La deuxième étape secondaire est de définir l'algorithme qui sera finalement implanté et effectuer cette implantation. Il ne sera pas demandé ici que l'implantation soit optimisée, ait de bonnes performances, et ait une interface conviviale, etc. Le code devant servir de base pour le développement d'un outil plus complexe, la clarté du code, sa modularité et la bonne documentation de celui ci sont les propriétés principales que devra avoir l'implantation recherchée.
Il est demandé que, si possible, les propriétés soient exprimées en terme d'actions effectuées dans le système et non en fonction des états et leurs propriétés. Pour cela, une logique

[1] Certains modèles de temps décrivent le temps comme une valeur entière évoluant par incrément d'une unité à chaque évolution de temps. Par ailleurs, un système décrivant des processus concurrents peut modéliser le temps comme étant absolu (le temps évolue simultanément pour tous les processus) ou relatif (chaque processus à "son" temps évoluant à son propre rythme).

temporisée orientée action doit être utilisée. Un objectif de ce travail est donc d'étudier les méthodes existantes de Model Checkings orientés actions et, si nécessaire, de créer un nouveau formalisme ainsi qu'un nouvel algorithme de Model Checking utilisant ce formalisme.

## 4. Structure du document

Dans ce qui suit, nous parlons tout d'abord du Model Checking en général avant de nous intéresser à un algorithme de Model Checking utilisant une logique temporisée bien particulière et à l'implantation qui en a été faite. Nous exposons pour finir une extension de cette logique.

- Dans le chapitre 2, nous parlons de structures utilisées pour modéliser les comportements de systèmes et dans le chapitre 3 des logiques temporelles.
- Dans le chapitre 4, nous parlons des algorithmes de vérification utilisant des logiques temporelles et des automates présentés auparavant.
- Dans le chapitre 5, nous présentons l'algorithme de Model Checking symbolique qui a été implanté dans le cadre de ce mémoire. Cet algorithme utilise la logique TCTL et est basé sur la manipulation de contraintes sur les horloges. En plus, de la présentation de l'algorithme, nous présentons dans ce chapitre une méthode efficace pour implanter ces contraintes.
- Dans le chapitre 6, nous détaillons quelques aspects de l'implantation qui a été faite de l'algorithme de vérification symbolique présenté au chapitre 5.
- Dans le chapitre 7, nous présentons une extension de TCTL et deux algorithmes de Model Checking, dont un qui a été implanté en utilisant l'implantation de l'algorithme TCTL présenté au chapitre 5. Cela a permis de satisfaire le souhait de pouvoir spécifier les propriétés à vérifier en terme d'actions effectuées dans le système.
- Dans le chapitre 8, nous présentons la vérification de quelques propriétés effectuées à l'aide du programme implanté dans le cadre de ce travail.
- Nous terminons au chapitre 9 par une conclusion sur ce qui a été fait dans ce travail et présentons quelques perspectives de continuation.

# Chapitre 2

# Les automates

Un algorithme ou un système informatique peut être vu comme et, représenté par, un automate se trouvant dans un état et effectuant des transitions pour passer dans d'autres états. Dans ce chapitre, nous présentons divers concepts utilisés par la suite en essayant d'harmoniser au mieux diverses notions présentées dans des documents de références différents de manières différentes

## 1. Définition de base

Dans ce qui suit, nous supposons qu'il existe deux ensembles d'éléments distincts :

**1)** L'ensemble $P$ des propositions qui seront généralement associées aux états.

**2)** L'ensemble $A$ des actions qui seront généralement associées aux transitions entre états.

Nous distinguons l'ensemble $\mathtt{L}$ des actions visibles de l'action invisible $\tau$ ($\notin \mathtt{L}$). Intuitivement, l'action $\tau$ permet de modéliser un changement d'état du système non marqué par une action visible par l'environnement.

$\mathtt{A} =_{\text{déf}} \mathtt{L} \cup \{\tau\}$.

Une horloge est une variable spéciale dont la valeur augmente automatiquement avec le passage du temps. Nous supposons dans ce qui suit que le domaine d'une horloge est $R^+$.

Nous dénoterons dans ce qui suit

- p désigne une proposition faisant partie de $\mathtt{P}$.
- a, b désignent des actions faisant partie de $\mathtt{L}$.
- H comme étant un ensemble d'horloges.
- x, y comme étant des horloges de H.

## 2. Séquences et arborescences

Une trace ou séquence d'états ($s_0$, $s_1$, $s_2$,…) est une séquence finie ou infinie d'états labellés par un ensemble de propositions. La figure 1.a montre une trace où le premier état est labellé par p, le second par p et q, le troisième de nouveau par p,…

On trouve également des séquences similaires, appelée ici séquence d'actions, dans lesquelles on ne retrouve pas des propositions mais des actions. Une séquence d'actions est une séquence finie ou infinie ($s_0$, $a_0$, $s_1$) ($s_1$, $a_1$, $s_2$) … où $s_i$ est un état et $a_i \in A$.

La figure 1.b montre une séquence d'actions où l'action a est d'abord effectuée, puis les actions b, c,…

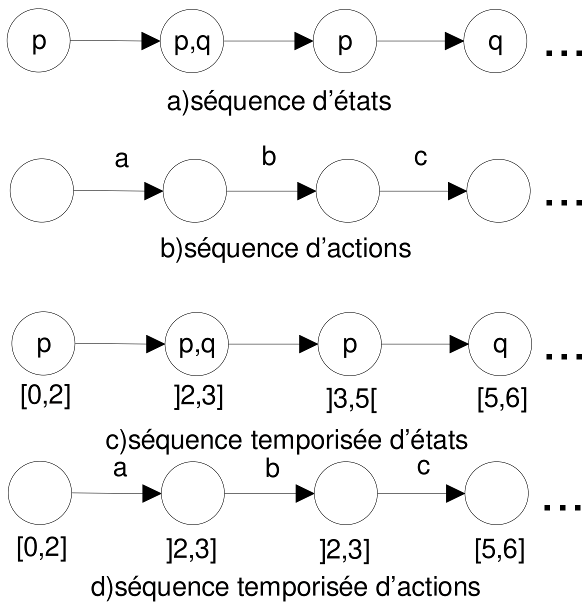



Figure 1

On peut rajouter à ces séquences une notion quantitative de temps durant lequel on reste dans les différents états, en associant à chaque état un intervalle. On obtient ainsi une séquence temporisée d'états et une séquence temporisée d'actions. Dans ce qui suit, nous supposons que le temps est isomorphe à $R^+$.

Un intervalle est un couple | a, b | où $a, b \in 3^+$ et $| \in \{[, ]\}$. Pour un intervalle I, la fonction Min(I) donne la borne inférieure de I. De même, la fonction Max(I) donne la borne supérieure de l'intervalle I.

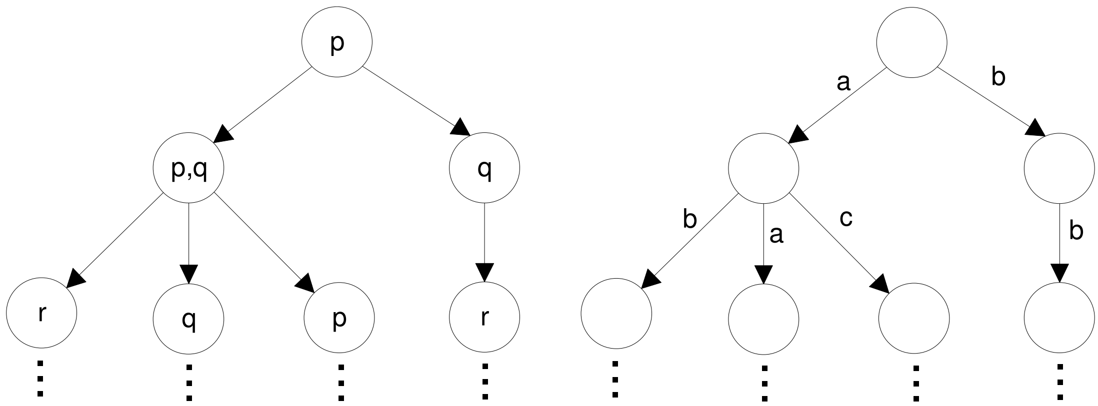
L'intervalle I – t est l'intervalle contenant l'ensemble des valeurs t' tel que $\exists t'' \in I$ avec $t' = t'' - t$ et $t' \geq 0$.

Une séquence d'intervalles est une suite finie ou infinie d'intervalles I = $I_0$, $I_1$, $I_2$,… qui partitionne $R^+$, c'est à dire telle que

- pour tout intervalle $I_i$ de I avec $i > 0$, $I_{i-1}$ et $I_i$ sont adjacents.
- toute valeur $t \in R^+$ appartient à un et un seul intervalle $I_i$.

Une séquence temporisée d'états est un couple (σ, I) constitué d'une trace σ et d'une séquence d'intervalles I. Le $i^{ème}$ intervalle de la séquence d'intervalle est associé au $i^{ème}$ état de la trace. Notons qu'il y a autant d'intervalles dans I qu'il y a d'états dans σ. La figure 1.c montre une séquence temporisée d'états dans laquelle on reste dans l'état initial, labellé par p, durant l'intervalle [0,2]. Ensuite, on passe dans l'état suivant, labellé par p et q, et on y reste durant l'intervalle ]2,3],…

De même, une séquence temporisée d'actions est un couple (σ, I) constitué d'une séquence d'actions σ et d'une séquence d'intervalles I. Comme dans le cas d'une séquence temporisée d'états, le $i^{ème}$ intervalle de la séquence d'intervalle est associé au $i^{ème}$ état de la séquence d'actions. La figure 1.d montre une telle séquence où on reste dans l'état initial durant l'intervalle [0,2] avant d'effectuer l'action a et de se retrouver dans l'état suivant de la séquence. On reste dans cet état pendant l'intervalle ]2,3], puis l'action b est effectuée,…
Certains ensembles de séquences présentées ci-avant peuvent être représentés sous forme d'arborescence.
Un arbre de branchement est un arbre ayant des branches finies ou infinies dont la racine et les nœuds sont labellés par des propositions. Une branche d'un tel arbre est une séquence d'états. La figure 2.a montre un tel arbre où partant de la racine, labellée par p, on peut transiter vers un fils gauche ou un fils droit. Si on décide de transiter vers le fils gauche, on se retrouve dans un nœud labellé par p et q à partir duquel on a le choix entre trois fils,…
Un arbre d'actions est un arbre ayant des branches finies ou infinies dont les arcs sont labellés par une action. Une branche d'un tel arbre représente une séquence d'actions. La figure 2.b représente un tel arbre où, partant de la racine, on peut au choix effectuer une action a ou une action b. Si on effectue l'action b, on se retrouve dans un nœud où on ne peut que refaire une action b,…




a)arbre de branchement b)arbre d'actions

Figure 2

Les arbres temporisés rajoutent par rapport à ces deux types d'arbres une notion de temps quantitatif.
Un arbre temporisé est un arbre fini ou infini dans lequel un intervalle est associé à chaque nœud. Cet intervalle représente les instants pendant lesquels on peut se trouver dans le nœud. Un endroit dans un arbre de branchement se caractérise donc par un nœud de l'arbre et un instant. Lorsqu'on est dans un nœud d'un arbre temporisé, à chaque instant t inclus dans

l'intervalle I associé à ce nœud, il est possible de soit faire une transition vers un nœud fils, la borne inférieure de l'intervalle associé à ce fils devant être t, soit de vieillir jusqu'à un instant t' toujours inclus dans I. On peut également avoir des nœuds sans fils où à chaque instant, on ne peut que vieillir. Chaque parcours dans de tels arbres est soit une séquence temporisée de branchement, soit une séquence temporisée d'actions suivant que ce soient les nœuds de l'arbre qui sont labellés par des ensembles de propositions ou les transitions par des actions.

Notons qu'il existe également des séquences et des arbres, temporisé ou non, où on retrouve simultanément des propositions et des actions.

## 3. Les automates

Ayant défini $\mathcal{A}$ et $\mathcal{P}$,
Un automate A peut être défini comme un 3-uple (S, $\delta$, $S_0$ ) où
S est l'ensemble des états.
$\delta$ est la fonction de transition entre états.
$S_0$ est l'ensemble des états initiaux (certaines définitions réduisent cet ensemble à un seul état).

Un automate peut définir un langage L(A) ou encore l'ensemble des comportements possibles d'un système. Suivant ce que l'automate modélise, on peut définir $\delta$ de diverses façons :
- $\delta : S \times S$ : définit les changements d'états possibles.
- $\delta : S \times \mathcal{A} \rightarrow S$ : définit un automate déterministe qui change d'état en avalant un symbole de $\mathcal{A}$.
- $\delta: S \times \mathcal{A} \rightarrow 2^S$ : détermine un automate non déterministe qui change d'état en avalant un symbole de $\mathcal{A}$.

Ayant $\delta : S \times \mathcal{A} \rightarrow S$ ou $S \times \mathcal{A} \rightarrow 2^S$, on parle de système à transitions labellé (LTS pour labelled transition system).

De même, on peut rajouter divers éléments :
- $\mu : S \rightarrow 2^{\mathcal{P}}$ : une fonction qui associe à chaque état un ensemble de propositions.
- $F \subseteq S$ : un ensemble d'états d'acceptation. Cet ensemble sert à définir le langage L(A) associé à l'automate.

Ainsi, pour définir un langage de mots finis, on peut définir un LTS A avec L(A) défini classiquement comme donné par [ASU89] c'est à dire l'ensemble des strings avalés par A et aboutissant à un état de F.

Pour définir un langage de mots infinis ($\omega$-mots), on peut définir L(A) comme l'ensemble des strings avalés en empruntant un chemin dans A partant d'un état de $S_0$ et traversant infiniment souvent un sommet de F.
Avec cette définition nous parlons d'automate de Büchi (et d'acceptation de Büchi)
Ainsi l'automate de Büchi représenté à la figure 3 définit le langage $(a + b)^*ac^\omega$.

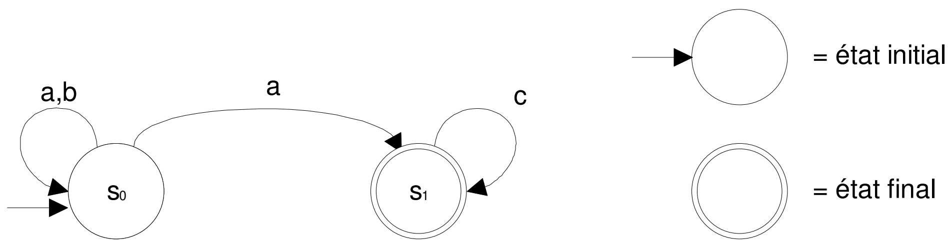



Figure 3:exemple d'automate de Büchi

Si le but est de décrire et d'analyser un système, on peut définir une structure de Kripke comme un 4-uple $A = (S, \delta, s_0, \mu)$
avec S un ensemble d'états finis
$\delta \subseteq S \times S$
$\mu : S \rightarrow 2^P$
$s_0$ est l'état initial.
Partant de $s_0$, seuls les chemins $s_0, s_1, s_2, \ldots$ infinis sont acceptés dans une structure de Kripke. Ceci peut être plus formellement spécifié en rajoutant aux structures de Kripke un ensemble d'états d'acceptation F et en introduisant l'acceptation de Büchi, avec F contenant toujours tous les états de S.
Un exemple de structure de Kripke est représenté par la figure 4, où les propositions associées à chaque état s sont notées à coté du nœud correspondant. Ces propositions peuvent représenter, par exemple, les propriétés des états de la structure de Kripke.

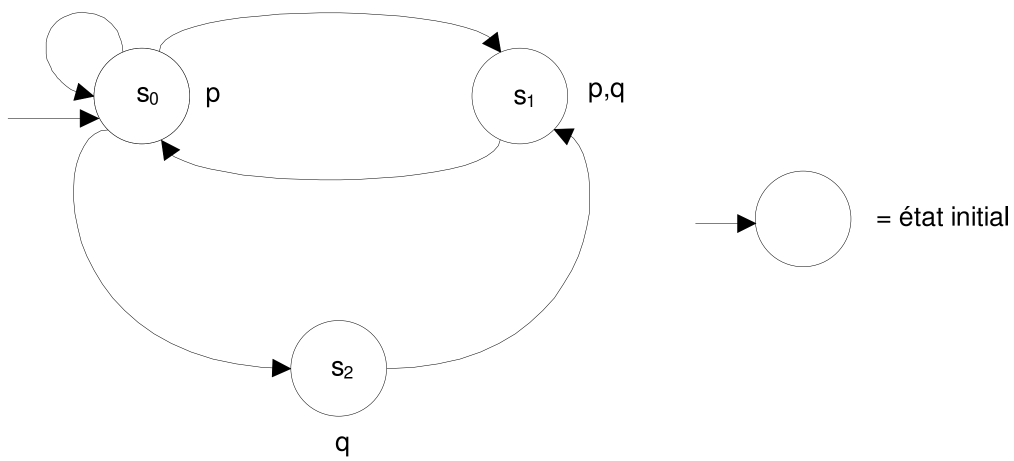



Figure 4:exemple de structure de Kripke

Un automate temporisé est un automate étendu en ajoutant des horloges et des contraintes sur ces dernières. Ces horloges permettent d'introduire une notion quantitative de temps. Cela permet de représenter des systèmes temporisés sous forme d'automate.
Nous allons présenter ici deux types d'automates temporisés, utilisés dans les chapitres suivant. Le premier, que nous appellerons structure temporisée, a été défini dans [Alu91]. Le

second, légèrement différent, que nous appellerons graphe temporisé, a été défini dans [Yov93].
Pour ces deux automates temporisés, on définit une valuation $\nu$ comme étant une fonction qui associe une valeur à chaque horloge d'un automate temporisé. $\nu(x)$ donne la valeur donnée à x par $\nu$.
La valuation $\nu + t$ est la valuation $\nu$' tel que pour tout x, $\nu'(x) = \nu(x) + t$.
La valuation $\nu\ [x \to 0]$, est la valuation $\nu$' telle que $\nu'(x) = 0$ et pour tout $y \neq x$, $\nu'(y) = \nu(y)$.
On peut généraliser cet opérateur pour un ensemble d'horloges H'. Dans ce cas, la valuation $\nu\ [H' \to 0]$ est la valuation $\nu$' telle que pour tout $x \in H'$, $\nu'(x) = 0$ et pour tout $x \notin H'$, $\nu'(x) = \nu(x)$.

Pour une structure ou un graphe temporisé, un état est un couple $<s, \nu>$ où s est un sommet de l'automate et $\nu$ est une valuation. Un état caractérise donc un "endroit" du système temporisé à un instant donné.
Si $<s, \nu>$ est un état, on définit l'état $<s, \nu> + t \equiv <s, \nu + t>$

**Structure temporisée [Alu91]**
Dans les structures temporisées, l'ensemble des contraintes possibles sur les horloges de H est défini par la grammaire :
$\varphi \to x \leq c \mid c \leq x \mid \neg\varphi \mid \varphi \wedge \varphi$
où $c \in \angle$.
Certaines contraintes telles que $x = c$, $x < c$, true, $\varphi \vee \varphi$ peuvent être définie comme des abréviations. (elle sont équivalentes à des contraintes permises par la grammaire)

Une structure temporisée est un automate ayant des propriétés temporelles ; elle est définie par un 6-uples $(S, s_{init}, \mu, H, \Delta, \delta)$, où
- S est l'ensemble fini des sommets de la structure.
- $s_{init} \in S$ est l'état initial.
- $\mu$ est une fonction qui associe à chaque sommet de S un ensemble de propositions $\subseteq P$.
- H est l'ensemble des horloges utilisées par la structure.
- $\Delta$ associe à chaque sommet de S une contrainte sur les horloges, appelée invariant du sommet.
- $\delta \subseteq S \times S \times 2^H$ est un ensemble de transitions. Une transition entre états s et $s' \in S$ est labellée par un ensemble d'horloges réinitialisées.

L'ensemble des états d'une structure temporisée est l'ensemble des états $<s, \nu>$ où les valeurs données aux horloges par $\nu$ satisfait $\Delta(s)$. L'état initial est l'état $<s_{init}, \nu_0>$ où $\nu_0$ est la valuation donnant la valeur 0 à chaque horloge de H (on suppose que $\nu_0$ satisfait $\Delta(s_0)$). A chaque instant t, la structure temporisée se trouve dans un état $<s, \nu>$. Toutes les horloges vieillissent à la même vitesse. Partant de $<s, \nu>$, si le système vieillit de t, les valeurs des horloges seront données par $\nu + t$ après vieillissement. Si on reste dans un sommet s pendant l'intervalle de temps (t, t'), les valeurs des horloges doivent satisfaire l'invariant de s durant tout cet intervalle. Si en t' la structure temporisée effectue une transition vers un sommet s', il faut que les valeurs des horloges satisfassent l'invariant de s'. L'état dans lequel on se trouve en t' dépend de l'intervalle (t, t') durant lequel on est resté en s. Si cet intervalle est ouvert à droite, alors on se trouve déjà en s', sinon on se trouve encore en s. Lors d'une transition, les horloges associées à cette transition sont réinitialisées à zéro. La manière dont on initialise les horloges dépend également du fait que l'intervalle est ouvert ou fermé à droite. Si l'intervalle est ouvert à droite, le changement de sommet est d'abord effectué, puis l'initialisation des horloges. Par contre, l'initialisation des horloges est faite avant le changement de sommet dans le cas où l'intervalle est fermé à droite. Il est à noter que, dans ce cas là, les horloges remises à zéro ne valent plus zéro lorsqu'on arrive dans l'état d'arrivée.

Si on a l'exécution

(s, [0, 3])(s', ]3, 4[)(s'', [4, 7])…

A l'instant 3, on se trouve toujours dans le sommet s et toutes les horloges ont la valeur 3. Comme l'intervalle [0, 3] est fermé à droite, on effectue d'abord la mise à zéro des horloges puis le changement de sommet. Au moment où on arrive en s', la valeur des horloges initialisées lors de la transition n'est plus zéro, mais juste plus grande. Par contre, lors de la transition à l'instant 4, le changement de sommet est effectué avant la remise à zéro des horloges. Lorsqu'on arrive dans s'', ces horloges valent bien dans ce cas zéro.

Nous voyons que dans ce modèle, une transition ($\in$ $\delta$) d'une structure temporisée implique non seulement un changement d'états mais également un vieillissement. On ne peut donc pas au même moment se trouver dans deux états différents.

Dans un système réel, des événements survenant dans ce système provoquent des changements d'états du système. Ces événements ne sont en général pas instantanés et prennent un temps t. Bien que les transitions ne sont pas instantanées, ce temps t pris lors des changements d'états n'est pas directement spécifiable au niveau des transitions d'une structure temporisée, mais peut être modélisé, pour un événement a, par une transition représentant le début de a, une transition représentant la fin de a et un état représentant les instants pendant lesquels a est effectué.

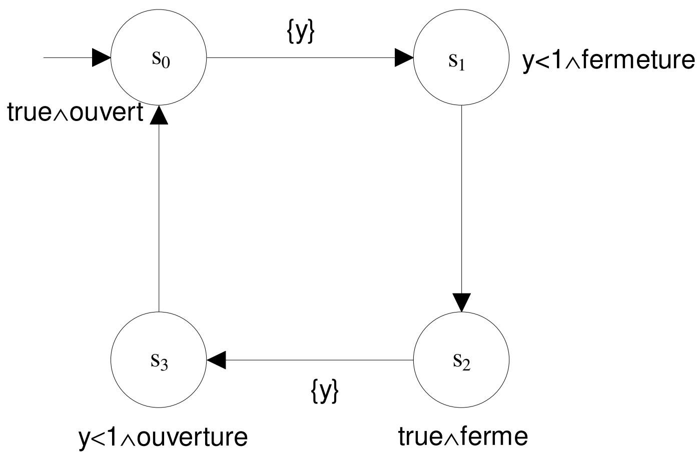



Figure 5: structure temporisée représentant le comportement d'une barrière d'un passage à niveau

La figure 5 montre une structure temporisée représentant le comportement d'une barrière d'un passage à niveau. Le sommet $s_0$ représente l'état où la barrière est ouverte et aucun train n'est annoncé. La structure temporisée permet de rester dans ce sommet indéfiniment, ceci correspondant au cas où il n'y aurait jamais de train passant par le passage à niveau. Lorsqu'un train est annoncé, la barrière commence à se fermer. Ceci correspond à effectuer une transition vers le sommet $s_1$. Comme la barrière met au plus une unité de temps pour être effectivement fermée, on se retrouve ensuite dans le sommet $s_2$, représentant l'état où la barrière est fermée, au plus une unité de temps plus tard. Pour une raison ou une autre, un train rentrant dans le passage à niveau peut ne jamais en ressortir (s'il tombe en panne par exemple). Il est donc possible de rester dans cet état indéfiniment et la barrière peut ne jamais se relever. Ceci est représenté dans la structure temporisée par le fait qu'il est possible de rester indéfiniment dans le sommet $s_2$.Lorsque le train sort du passage à niveau, la barrière commence à se relever. Ceci correspond à effectuer la transition vers le sommet $s_3$. De nouveau, la barrière mettra au plus une unité de temps pour être rouverte. Après être rentré dans le sommet $s_3$, on se retrouvera donc de nouveau dans le sommet $s_0$ après au plus une unité de temps.

**Graphe temporisé [Yov93]**

Pour un graphe temporisé, $\Psi(H)$ est l'ensemble des contraintes possibles sur les horloges de H, une contrainte étant définie par la grammaire :

$\varphi \rightarrow true \mid x < c \mid x \leq c' \mid x - y < c'' \mid x - y \leq c'' \mid \neg\varphi \mid \varphi \wedge \varphi$

où $c \in \angle \setminus \{0\}$ .

  $c' \in \angle$.

  $c'' \in 9$.

De nouveau, certaines contraintes comme $x = c$, $x > c$, true ou $\varphi \vee \varphi$ peuvent être définie comme des abréviations.

La formule $x - y$ op $c''$ (op $\in \{<, \leq\}$) permet d'avoir ici des contraintes sur les horloges légèrement différentes par rapport aux contraintes permises dans le cadre des structures temporisées.

Notons que par rapport à [Yov93], on interdit des contraintes qui n'ont pas de sens, comme $x \leq -5$ ou $x < 0$, en limitant le domaine des constantes c dans les formules x op c.

Un graphe temporisé est un 5-uple $(S, s_{init}, H, \Delta, \delta)$ où

- S est l'ensemble fini des sommets du graphe.
- $s_{init} \in S$ est le sommet initial.
- H est l'ensemble de ses horloges.
- $\Delta : S \rightarrow \Psi(H)$ est une fonction qui associe à chaque sommet une condition appelée invariant du sommet. Contrairement aux structures temporisées, un invariant doit satisfaire la propriété de fermeture en arrière, c'est à dire $\forall\ s \in S, \nu \in V, t \in R^+.\ \nu + t$ satisfait $\Delta(s) \Rightarrow \nu$ satisfait $\Delta(s)$. Cela veut dire que si on peut se trouver dans un état s à un moment donné (caractérisé par une valuation) alors on pouvait s'y trouver également avant.
- $\delta \subseteq S \times A \times \psi(H) \times 2^H \times S$ est un ensemble d'arcs labellés par une action, une contrainte sur les horloges et un ensemble d'horloges.

Les graphes temporisés sont similaires aux structures temporisées, à quelques détails près. Une différence entre ces deux types d'automates, outre les invariants de sommets possibles, est que les transitions d'un graphe temporisé sont labellées par une action et une contrainte sur les horloges. Pour pouvoir effectuer une transition, les valeurs des horloges doivent ici, en plus de satisfaire l'invariant du sommet d'arrivée, satisfaire la condition de la transition. De plus, effectuer une transition dans un graphe temporisé est instantané, c'est à dire que si une transition est effectuée à un instant t, on se trouvera toujours à l'instant t après la transition. Lorsqu'on arrive dans l'état d'arrivée, les horloges devant être remises à zéro valent effectivement zéro. Contrairement à une structure temporisée, on peut donc se trouver ici dans plusieurs états différents à un instant donné.

La figure 6 représente le graphe temporisé d'un contrôleur de température. Celui-ci est relié à un capteur qui envoie au contrôleur un signal lorsque la température dépasse un certain seuil. A ce moment-là, le contrôleur active un système de refroidissement qui sera actif pendant deux unités de temps. Pendant cette période aucun signal ne peut venir du capteur. Si cela se produit tout de même, le système rentre dans un état d'erreur. Après les deux unités de temps, le contrôleur retombe dans son état initial et attend un nouveau signal. De plus, on considère

que le capteur n'est pas fiable et peut tomber en panne, un processus de refroidissement sera pour cela réenclenché automatiquement lorsqu'un signal s n'a plus été reçu depuis dix unités de temps.
Le graphe temporisé de la figure 6 débute dans le sommet initial avec x et y valant zéro. L'envoi du signal par le capteur fait basculer le graphe temporisé dans le sommet $s_1$. Comme la garde de ce sommet est y = 0 et que y est remis à zéro lors de la transition, le processus de refroidissement est immédiatement enclenché et le graphe temporisé transite instantanément vers l'état $s_2$ où il reste pendant la durée du processus, c'est à dire deux unités de temps. Si un signal apparaît pendant ce temps, le graphe temporisé bascule vers le sommet $s_3$, pour spécifier une erreur. Après les deux secondes, le système revient au sommet initial $s_0$ et attend le prochain signal. La variable y permet de déterminer le temps écoulé depuis le dernier signal reçu. Lorsque y vaut 10, le système commence automatiquement un refroidissement.

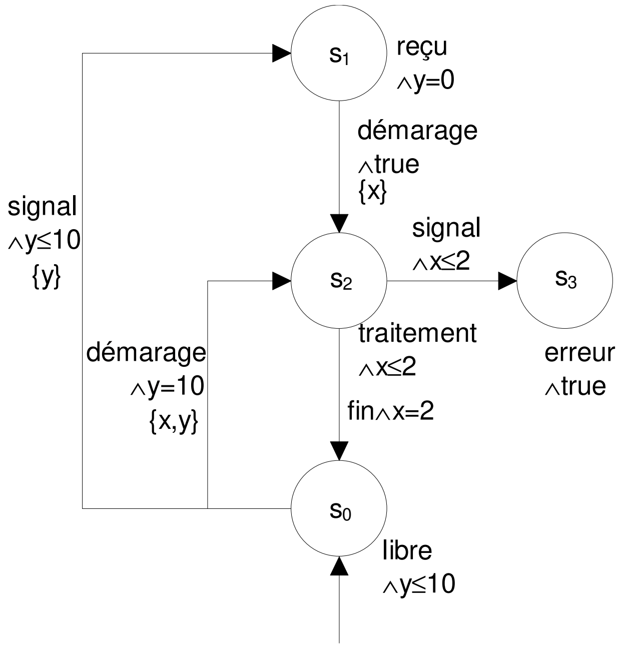


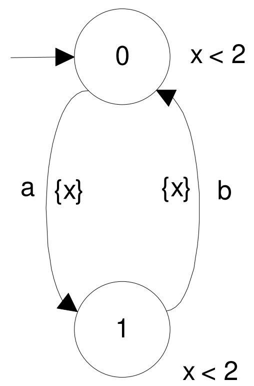

Figure 6:graphe temporisé représentant un contrôleur de température

A partir d'un graphe temporisé G, nous pouvons construire un LTS, généralement infini vu la densité du temps, donnant les transitions et changements d'états possibles de G.
Nous donnons ici les idées principales de la construction d'un tel LTS pour un graphe temporisé (voir [Yov93] pour la définition complète).
L'ensemble des états du LTS est l'ensemble des états $<s, \nu>$ de G. Il existe deux types de transitions :

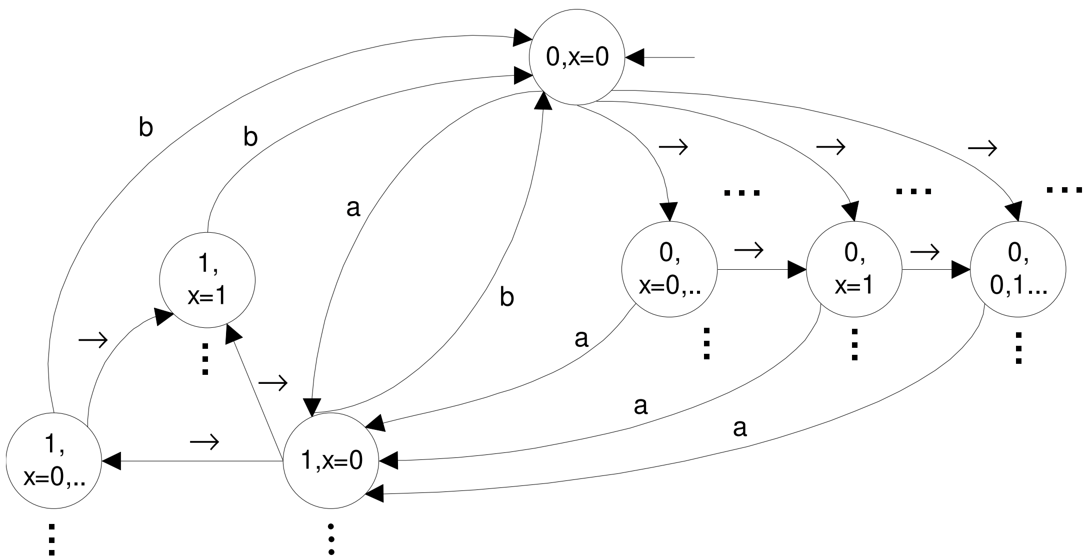
1) Les transitions effectuant une action a. Une telle transition existe entre deux états $<s, \nu>$ et $<s', \nu'>$ s'il existe un arc (s, a, φ, H', s') dans G tel que ν satisfasse φ et $\nu' = \nu\,[H' \rightarrow 0]$.
2) Le vieillissement de toutes les horloges d'une même valeur. La nouvelle valuation doit toujours satisfaire l'invariant de s. Si le graphe temporisé peut se trouver dans un état s avec des valeurs d'horloges données par la valuation ν et vieillir de t, nous aurons dans le LTS des transitions de l'état $<s, \nu>$ vers les états $<s, \nu + t'>$ avec $t' \leq t$. Ceci étant justifié par la propriété de fermeture arrière des invariants de sommets.

Partant d'un sommet s avec des valeurs d'horloges données par ν, lors d'une transition dans une structure temporisée, vu la densité du temps et le fait qu'une transition n'est pas

instantanée, on ne pourra pas déterminer les valeurs d'horloges après transition. Il n'est pour cela pas possible, pour une structure temporisée, de déterminer les transitions du premier type et donc de construire un LTS à partir de tels automates temporisés.
La figure 7.b montre une partie du LTS du graphe temporisé représenté à la figure 7.a. Les transitions labellée par $\rightarrow$ sont des transitions liées à un vieillissement.




a)graphe temporisé




b)LTS du graphe temporisé

Figrue 7

## 4. Traduction d'automates

L'analyse d'un système modélisé sous la forme d'un automate, suivant le type de propriétés que nous voulons regarder, peut se traduire par l'analyse de chaque exécution de l'automate de manière indépendante par rapport aux autres ou par l'analyse séparée des ensembles des exécutions possibles partant d'un état commun de l'automate, ces ensembles étant dans ce cas mis sous forme d'arbre.
Cela peut être fait, suivant les cas, en utilisant des séquences, des arbres, d'états ou d'action.

Pour un automate non temporisés :

- l'ensemble des traces ou des séquences d'actions de l'automate, suivant les cas, est l'ensemble des séquences d'états ou de transitions possibles décrit par l'automate.
  Chacune de ces traces ou séquences d'actions satisfait les conditions suivantes :
  - Elles commencent par un état initial de l'automate.
  - Dans le cas de traces, si un état $s_1$ précède un état $s_2$ dans une trace, alors il existe une transition de $s_1$ vers $s_2$ dans l'automate. Dans le cas de séquences d'actions, si une séquence contient la transition ($s_1$, a, $s_2$) alors il existe une transition de $s_1$ vers $s_2$ labellée par l'action a dans l'automate.
  - Les traces ou séquences d'actions satisfont les conditions d'acceptation de l'automate.
- L'ensemble des arbres de branchements ou d'actions de l'automate est un ensemble d'arbres dont les racines sont des états initiaux de l'automate et reprenant pour chaque état s accédé, les transitions possibles et les états voisins accédés via ces transitions. De tels arbres sont donc une sorte de dépliage de l'automate.

Si, pour être représentées par un automate, les séquences d'états ou d'actions (éventuellement mises sous forme d'arbres) doivent être infinies et que ce n'est pas le cas, on s'arrange pour que ce le soit. Certains types d'automates ne permettant pas d'accepter des séquences finies, il faut alors construire si nécessaire de tels automates de telle manière que certaines de leurs séquences infinies puissent être associées de manière naturelle à une séquence finie. Par exemple, en construisant l'automate comme si les séquences finies étaient acceptées, puis en rajoutant un état " final" vers lequel on transite à la fin d'une séquence finie en effectuant si besoin est une action "finale". Lorsqu'on arrive dans cet état, on ne peut plus que faire des transitions vers l'état "final" lui-même en effectuant de nouveau si besoin est l'action "finale". De cette manière, des séquences infinies de l'automate auraient comme préfixe les séquences finies désirées, suivie, suivant les cas, soit d'une infinité d'état " final", soit d'une infinité d'action "finale".

Dans le cas d'un automate temporisé, les exécutions peuvent être représentées par un ensemble de séquences ou d'arbres temporisés :

- L'ensemble des séquences temporisées de l'automate est l'ensemble contenant, suivant les cas, toutes les séquences temporisées de branchement ou d'actions satisfaisant les conditions suivantes
  - Les séquences commencent dans un état initial de l'automate.
  - Pour tout sommet $s_i$ et intervalle $I_i$ d'une séquence temporisée, il faut pouvoir atteindre ce sommet dans l'automate temporisé et y rester pendant l'intervalle $I_i$ en passant auparavant par les sommets $s_0$, $s_1$,…,$s_{i-1}$ le précédant dans la séquence temporisée et en y restant durant les intervalles qui leur sont associés dans la séquence temporisée. De plus, dans le cadre d'une séquence temporisée d'actions, toutes les actions effectuées pour atteindre $s_i$ dans l'automate doivent correspondre aux actions effectuées dans la séquence temporisée d'actions pour atteindre $s_i$.
- L'ensemble des arbres temporisés de l'automate est un ensemble d'arbres dont les racines sont des états initiaux de l'automate temporisé et reprenant pour chaque sommet s accédé à un instant t les transitions possibles et les états accédés via ces transitions ainsi que l'intervalle de temps durant lequel on peut rester dans s. On peut avoir deux types de transitions à partir de chaque nœud
  - une transition représentant un changement de sommet dans le modèle. Pour une transition donnée de l'automate, les instants pendant lesquels on peut rester dans le sommet d'arrivée vont dépendre de l'instant où cette transition est effectuée. On va donc avoir en général dans l'arbre une infinité de transitions pour une transition dans l'automate temporisé, une pour chaque instant où celle ci peut être effectuée.

- une transition liée au fait qu'une transition de l'automate devienne ou cesse d'être disponible dans l'automate. Ceci correspond à un vieillissement jusqu'à un moment où les horloges de l'automate prennent des valeurs telles que les contraintes liées à la possibilité de faire une transition change de valeur. Pour un nœud $s_i$, la borne de droite de l'intervalle $I_i$ sera le plus petit instant entre l'instant où l'invariant du sommet $s_i$ dans l'automate devient faux et le premier instant où une transition devient disponible ou cesse de l'être. Il n'y a donc pas dans ce cas de changement de sommet.

# Chapitre 3

# Les logiques temporelles

On présente dans ce chapitre quelques logiques temporelles classiques qui ont été abordées dans le cadre de ce travail. Pour chaque logique, des explications sur les différents opérateurs sont données ainsi que des commentaires sur l'expressivité. Ces commentaires sont basés sur une réflexion personnelle.

## 1. Présentation des logiques temporelles

Les logiques temporelles que nous allons étudier peuvent être vues comme une extension des logiques propositionnelles du premier ordre. En effet, une logique propositionnelle s'applique à un ensemble de propositions dont les valeurs de vérité sont données par une interprétation (fonction qui va de l'ensemble des propositions possibles dans {Vrai, Faux}). Une formule de logique temporelle est généralement évaluée soit sur une trace (ce sont les logiques temporelles linéaires ou linear time temporal logic), soit sur un arbre de branchement (ce sont les logiques de branchement ou branching time temporal logic), un état donnant la valeur de vérité des propositions à un moment donné. La différence entre les traces et les arbres de branchement est que dans le premier cas, la formule est évaluée sur une des "exécutions" possibles du système tandis que dans le second cas, la formule peut raisonner sur toutes les évolutions futures [1] possibles. Il existe également certaines logiques hybrides dans lesquelles se mélangent les aspects linéaires et de branchement. Nous ne parlerons pas des logiques de ce genre dans ce travail.
Il existe d'autres différences entre les logiques.
Certaines logiques temporelles, appelées logiques temporisées, introduisent une notion quantitative de temps dans leur sémantique. Par contre, dans les logiques non temporisées, bien que les valeurs de vérités des propositions varient en passant d'un état au suivant dans une trace ou un arbre de branchement, aucune notion quantitative de temps dans la

[1] Nous ne regardons ici que les logiques n'ayant pas d'opérateur(s) sur le passé.

sémantique[1] n'est généralement présente. Cet ajout de temps quantitatif est intéressant pour effectuer de la vérification sur des systèmes temporisés.
La vérification de propriétés avec les logiques temporelles temporisées peut se faire en utilisant des automates temporisés. On peut considérer que le temps est discret (ex. : les valeurs du temps sont isomorphes à ∠) ou dense (ex. : les valeurs du temps sont isomorphes à 3 ou Θ). Ceci influence la logique considérée. Par exemple, avec un temps dense la notion de moment suivant n'a plus de sens et des opérateurs tel que ❍ (état suivant) peut ne plus être relevant.
Une autre différence qui peut survenir entre les logiques temporelles est l' "orientation". Il y a les logiques orientées propositions : ce sont les logiques évaluées sur des arbres ou des séquences d'états labellés par des propositions. Il y a également les logiques orientées actions où ce ne sont pas les états qui sont labellés, mais les transitions avec des actions. Ceci permet d'analyser un système en regardant les séquences d'actions possibles et non les séquences d'états possibles du système et leurs propriétés associées. Il existe des logiques permettant de combiner des propriétés sur des états et sur des actions. De nouveau, nous ne parlerons pas des logiques de ce genre dans ce travail.
Dans le cadre de ce travail, les logiques temporelles sont utilisées pour spécifier des propriétés à vérifier sur un système. Les formules de logiques temporelles ayant une sémantique bien définie, il est possible de trouver des méthodes systématiques pour les vérifier sur un automate représentant un système. De l'expressivité de la logique utilisée va dépendre l'ensemble des propriétés vérifiables à l'aide de ces méthodes systématiques définies. Dans le cadre du Model Checking, il est intéressant de pouvoir exprimer à l'aide de logiques temporelles certaines classes de propriétés. Ces propriétés sont les propriétés d'atteignabilité (reachability properties) et d'équité (fairness properties).
Les propriétés d'atteignabilité portent sur l'atteignabilité potentielle ou inévitable d'états ou d'actions. On peut aussi distinguer les propriétés de vivacité (liveness properties) et de sûreté (safety properties). Les premières expriment le fait qu'un état désirable sera atteint ou qu'une action désirable sera effectuée. Par exemple, le fait qu'une demande d'accès à une ressource partagée est toujours suivie de l'accès à cette ressource en un temps fini. Les propriétés de sûreté expriment le fait qu'un état indésirable ne sera jamais atteint ou qu'une action indésirable ne sera jamais effectuée. Par exemple, le fait qu'on ne se trouvera jamais dans une situation de deadlock.
Les propriétés d'équité portent sur la répétition infinie de certains états ou actions. On distingue l'équité faible de l'équité forte. L'équité faible exprime le fait que si une propriété est continuellement vérifiée, alors une autre le sera infiniment souvent. Continuellement vérifiée ne veut pas dire que la propriété est invariante mais qu'à partir d'un moment donné, la propriété est toujours satisfaite (avant ce moment, certains états peuvent ne pas satisfaire la propriété). Il peut bien sûr ne pas y avoir de moments précédants.
Par exemple, si un processus est continuellement prêt à être exécuté, alors il le sera infiniment souvent. Plusieurs actions ne pouvant être effectuées à la fois, demander qu'une action a continuellement satisfaite implique qu'une autre action b le soit infiniment souvent n'a pas de sens. On ne parle donc pas d'équité faible exprimée en terme d'actions.
L'équité forte exprime quant à elle le fait que si une propriété est infiniment souvent vérifiée ou une action infiniment souvent effectuée, alors une autre propriété ou action le sera également. Par exemple, si un processus demande l'accès à une ressource infiniment souvent, il l'obtiendra infiniment souvent.

[1] Excepté pour certains formalismes où il existe par exemple toujours une transition par unité de temps; le nombre de transitions donne alors le temps écoulé. Notons cependant que dans ce cas le nombre de transitions peut être très important et que le temps est discret.

## 2. Exemples de logiques

Nous avons mis en évidence trois caractéristiques majeures que peuvent avoir les différentes logiques temporelles :

- Les logiques peuvent être dites linéaires ou de branchement.
- Les logiques peuvent être temporisées ou non.
- Les logiques peuvent être orientées actions ou propositions.

Théoriquement, ces trois caractéristiques peuvent se mélanger de n'importe quelle manière, néanmoins des logiques de certains types semblent ne pas avoir été définies.

| **linéaire / de branchement** | **Temporisée / non temporisée** | **action / proposition** | **exemple** |
|---|---|---|---|
| linéaire | non temporisée | action | TLA |
| linéaire | non temporisée | proposition | LTL |
| linéaire | Temporiséee | action | |
| linéaire | Temporisée | proposition | MITL |
| branchement | non temporisée | action | ACTL |
| branchement | non temporisée | proposition | CTL |
| branchement | Temporisée | action | |
| branchement | Temporisée | proposition | TCTL |

Nous allons présenter ici des exemples pour les différents types de logiques. Cette liste d'exemples étant loin d'être exhaustive, le lecteur pourra se référer par exemple à [MP92] pour la présentation du $\mu$-calcul et d'autres logiques.

Pour tout nœud, état ou sommet s d'un arbre ou d'une séquence, $\mu(s)$ donne l'ensemble des propositions labellant s dans ce qui suit.

**La logique LTL ([Pnu77])**
**Description :**
La logique LTL est un exemple de logique temporelle linéaire non temporisée (untimed linear temporal logic).
En voici la syntaxe:
Une formule logique $\phi$ est une formule LTL si elle peut être construite à partir de la grammaire suivante:

$$\phi \rightarrow p \mid \neg p \mid \mathrm{TRUE} \mid \mathrm{FALSE} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \bigcirc\phi \mid \phi\, U\, \phi \mid \phi \wr \phi$$

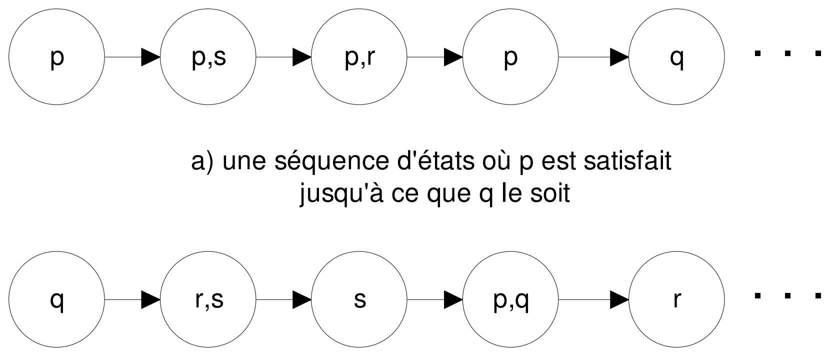
Nous introduisons maintenant la sémantique des opérateurs de LTL:
Une formule LTL est évaluée sur une séquence d'états infinie $\sigma = s_0\, s_1\, s_2 \ldots$

Pour tout $\sigma$ nous avons $\sigma \models \mathrm{TRUE}$ et $\sigma \not\models \mathrm{FALSE}$.
$\sigma \models p$ ssi $p \in \mu(\sigma_0)$.
$\sigma \models \neg p$ ssi $\sigma \not\models p$.
$\sigma \models \phi_1 \wedge \phi_2$ ssi $\sigma \models \phi_1$ et $\sigma \models \phi_2$.
$\sigma \models \phi_1 \vee \phi_2$ ssi $\sigma \models \phi_1$ ou $\sigma \models \phi_2$.
$\sigma \models \bigcirc\phi$ ssi $\sigma_1 \models \phi$ avec $\sigma_1 = s_1\, s_2\, s_3 \ldots$
$\sigma \models \phi_1\, U\, \phi_2$ ssi $s_j \models \phi_2$ pour un $j \geq 0$ et $s_k \models \phi_1$ pour tout k avec $0 \leq k < j$.
$\sigma \models \phi_1 \wr \phi_2$ ssi pour tout $i \geq 0$ tel que $s_i \not\models \phi_2$, il existe $0 \leq j < i$ tel que $s_j \models \phi_1$.

Notons que l'opérateur $\tilde{U}$ présenté ici et repris par Wolper dans [Wol98] ne correspond pas à l'opérateur $\tilde{U}$ défini par Manna et Pnueli [MP92].
La négation d'une formule n'existe pas telle quelle en LTL, mais peut être définie pour chaque type de formule. Ainsi

$\neg(\phi_1 \wedge \phi_2) \equiv \neg\phi_1 \vee \neg\phi_2$
$\neg(\phi_1 \vee \phi_2) \equiv \neg\phi_1 \wedge \neg\phi_2$
$\neg(\bigcirc\, \phi_1) \equiv \bigcirc\, \neg\phi_1$
$\neg(\phi_1\, U\, \phi_2) \equiv \neg\phi_1\, \tilde{U}\, \neg\phi_2$
$\neg(\phi_1\, \tilde{U}\, \phi_2) \equiv \neg\phi_1\, U\, \neg\phi_2$

La formule $\phi_1\ U\ \phi_2$ demande qu'un état de la trace satisfasse la propriété $\phi_2$ et que tous les états le précédant dans la trace satisfassent la propriété $\phi_1$. Par exemple, la figure 8.a, où chaque état contient les propositions qu'il vérifie, représente une séquence satisfaisant la formule p *U* q. La figure 8.b satisfait également cette formule. Il n'est, en effet, pas demandé dans la sémantique de LTL qu'il y ait obligatoirement au moins un état satisfaisant p qui précède l'état satisfaisant q.




a) une séquence d'états où p est satisfait jusqu'à ce que q le soit

b) q est directement satisfait

Figure 8: séquence satisfaisant p*U*q

Une formule du type $\phi_1\ \tilde{U}\ \phi_2$ demande que dans une trace tous les états, et plus particulièrement le premier, ne satisfaisant pas $\phi_2$ soient précédés d'un état satisfaisant $\phi_1$. Une séquence d'états satisfaisant $\phi_1\ \tilde{U}\ \phi_2$ ne doit donc pas nécessairement contenir d'états satisfaisant $\phi_1$, si tous les états de la séquence satisfont $\phi_2$.
L'opérateur $\bigcirc$ permet de spécifier une propriété devant être satisfaite par l'état successeur d'un état donné. La figure 9 montre une séquence d'états satisfaisant true *U* (p $\wedge$ $\bigcirc(\neg$q)). Cette formule demande qu'il y ait dans la trace un état satisfaisant p dont le successeur ne satisfait pas q.

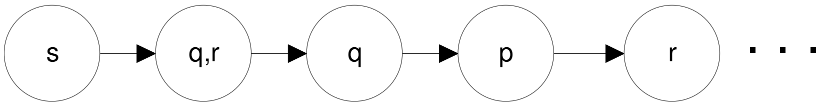



Figure 9: séquence d'états satisfaisant true $U$ (p $\wedge$ $\bigcirc$($\neg$q))

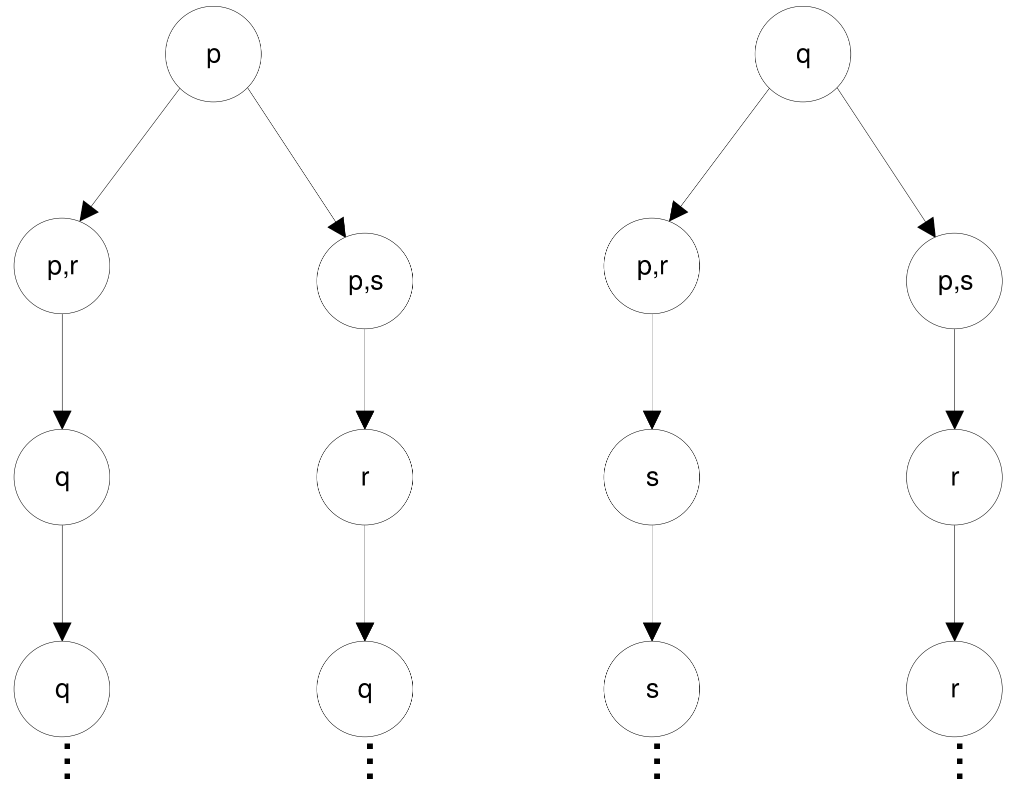
**Expressivité :**
LTL étant une logique linéaire, il n'est pas possible de différencier l'atteignabilité potentielle de l'atteignabilité inévitable, c'est à dire qu'un état ayant un seul état suivant dans une trace infinie, cela revient au même de demander s'il est possible ou inévitable d'atteindre dans une trace un état ayant une propriété donnée. Dans le cadre de vérifications de certaines propriétés simples, cela ne pose pas de problème. Pour un automate donné, savoir s'il est possible d'atteindre un état satisfaisant une certaine propriété p ne peut être exprimé directement en LTL. Mais on peut vérifier si $\neg$p est invariant, et de cette évaluation peut être déduite la valeur de vérité de la propriété désirée. Par contre, la propriété selon laquelle il est possible d'atteindre un état dans l'automate à partir duquel tout les états accessibles satisfont p ne peut être exprimée en LTL et on ne peut trouver de subterfuge pour déterminer la valeur de vérité de cette propriété.
Les propriétés de sûreté s'expriment sans problème. Exprimer le fait qu'on n'atteigne jamais un état ne satisfaisant pas p s'exprime sous la forme d'invariance avec la formule $\neg$(true $U$ ($\neg$p)). Les propriétés d'atteignabilité bornée, c'est à dire le fait qu'on puisse atteindre un état avant c unités de temps, ne peuvent en principe pas être exprimées en LTL puisque cette logique n'est pas temporisée. Grâce à l'opérateur $\bigcirc$, on peut néanmoins simuler un passage de temps discret. Si on se donne comme convention qu'on se trouve dans un état différent à chaque unité de temps alors, en prenant une séquence d'états, le premier état est l'état dans lequel on se trouve à l'instant un, le deuxième état est l'état dans lequel on se trouve à l'instant deux, etc. On peut de cette manière exprimer le fait qu'il est inévitable d'atteindre un état satisfaisant la propriété p avant deux unités de temps avec la formule p $\vee$ $\bigcirc$p $\vee$ $\bigcirc\bigcirc$p. Il est évident que cette manière de procéder génère des formules très lourdes. Cela peut être évité en se définissant des notations appropriées. La possibilité de pouvoir exprimer des propriétés temporelles avec un temps discret peut être intéressante pour exprimer des propriétés de systèmes où une action n'est effectuée que lors d'un tick d'une horloge contenue dans le système.
Les propriétés d'équités, comme les propriétés de sûreté, n'ont pas de mal à être exprimée en LTL. La propriété d'équité faible selon laquelle une propriété p continuellement vraie implique qu'une autre propriété q l'est infiniment souvent s'exprime en LTL comme
true $U$ $\neg$(true $U$ $\neg$p) $\rightarrow$ $\neg$(true $U$ $\neg$(true $U$ q)) et la propriété d'équité forte selon laquelle le fait que p soit vérifié infiniment souvent implique que q l'est aussi s'exprime en LTL comme $\neg$(true $U$ $\neg$(true $U$ p)) $\rightarrow$ $\neg$(true $U$ $\neg$(true $U$ q)), l'implication étant définie classiquement.

**La logique CTL ([Alu91], [CES96])**
**Description :**
La logique CTL est un exemple de logique temporelle de branchement non temporisée (untimed branching time logic).
Sa syntaxe se définit comme suit:
Une formule $\phi$ est une formule CTL si elle peut être construite à partir de la grammaire suivante:

$$\phi \rightarrow p \mid \neg\phi \mid \phi \wedge \phi \mid \exists\bigcirc\phi \mid \forall\bigcirc\phi \mid \phi \,\exists U\, \phi \mid \phi \,\forall U\, \phi$$

Nous introduisons maintenant la sémantique des opérateurs de CTL:
Une formule CTL est évaluée par rapport à un arbre de branchement n'ayant pas de branches finies.
Ayant un arbre de branchement de racine $s_0$

$s_0 \models p$ ssi $p \in \mu(s_0)$.
$s_0 \models \neg\phi$ ssi $s_0 \not\models \phi$.
$s_0 \models \phi_1 \wedge \phi_2$ ssi $s_0 \models \phi_1$ et $s_0 \models \phi_2$.
$s_0 \models \exists\bigcirc\phi$ ssi $s' \models \phi$ pour un état s' successeur de $s_0$.
$s_0 \models \forall\bigcirc\phi$ ssi $s' \models \phi$ pour tout état s' successeur de $s_0$.
$s_0 \models \phi_1\ \exists U\ \phi_2$ ssi pour une branche de l'arbre de branchement $(s_0, s_1, s_2,...)$, $\exists i \geq 0$ tel que $s_i \models \phi_2$ et $\forall\ 0 \leq j < i$, $s_j \models \phi_1$.
$s_0 \models \phi_1\ \forall U\ \phi_2$ ssi pour toute branche de l'arbre de branchement $(s_0, s_1, s_2,...)$, $\exists i \geq 0$ tel que $s_i \models \phi_2$ et $\forall\ 0 \leq j < i$, $s_j \models \phi_1$.
L'idée des opérateurs $\exists\bigcirc$ et $\forall\bigcirc$ est la même que pour l'opérateur LTL $\bigcirc$. Une formule CTL étant évaluée sur des arbres, un état peut avoir plusieurs successeurs. Dans le cas de l'opérateur $\exists\bigcirc$, au moins un successeur doit satisfaire une propriété donnée tendis que dans le cas de $\forall\bigcirc$, ce sont tous les successeurs.
L'opérateur $\exists U$ demande qu'on puisse atteindre, de l'état racine de l'arbre, un état satisfaisant une propriété donnée avec les états intermédiaires satisfaisant une seconde propriété.




a)les états d'une branche satisfont p jusqu'à ce que q soit satisfait

b) q est directement satisfait

Figure 10: arbre satisfaisant p $\exists U$ q

L'arbre de branchement représenté à la figure 10.a satisfait la formule p $\exists U$ q. Partant de l'état racine, la branche gauche permet d'atteindre un état satisfaisant q avec tous les états intermédiaires, y compris l'état racine, satisfaisant p. Par contre cet arbre ne satisfait pas la formule p $\forall U$ q. Pour cela, il faut que toutes les branches de l'arbre permettent d'atteindre un état satisfaisant q, avec les états intermédiaires satisfaisant p. Or, pour atteindre un état satisfaisant q par la branche droite de l'arbre, on passe par un état ne satisfaisant que r (et donc pas p).
L'arbre de la figure 10.b, quant à lui, satisfait également la formule p $\exists U$ q. En effet, rien n'oblige dans la sémantique de CTL de transiter par au moins un état satisfaisant p et l'état racine peut directement satisfaire q. Pour cette même raison, cet arbre satisfait également la formule p $\forall U$ q.

**Expressivité :**

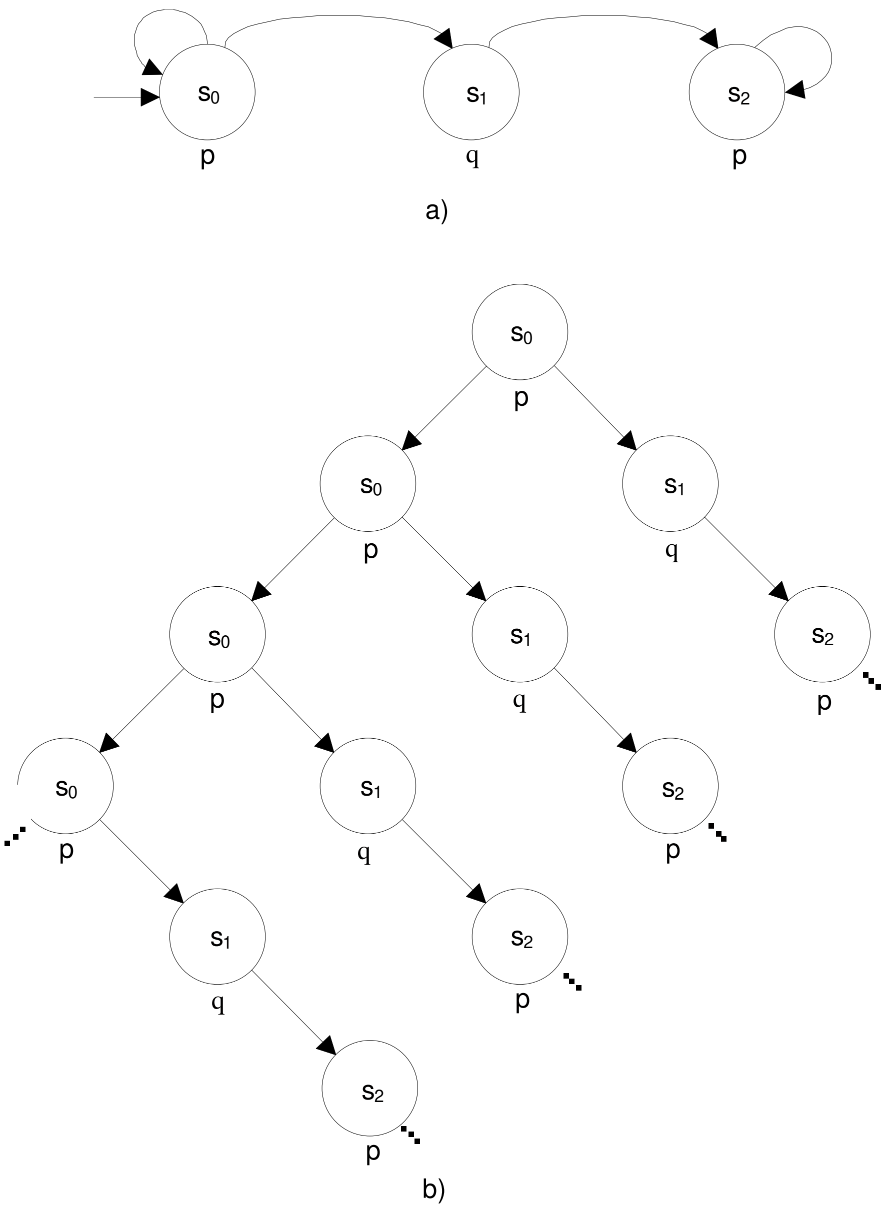



Figure 11

Les opérateurs $\forall U$ et $\exists U$ permettent, contrairement à LTL, de pouvoir différencier l'atteignabilité potentielle de l'atteignabilité inévitable. Par contre, il est dit dans [CES86] que les propriétés d'équité faible ne sont pas exprimables en CTL. En effet, la propriété exprimant le fait qu'il est inévitable d'atteindre un état à partir duquel une propriété p est toujours vérifiée n'est pas exprimable en CTL. On pourrait penser pouvoir exprimer cette propriété grâce à la formule true $\forall U$ $\neg$(true $\exists U$ $\neg$p). Cette formule spécifie le fait qu'il est inévitable d'atteindre un état à partir duquel tous les états accessibles satisfont p.
Si on regarde la structure de Kripke représentée par la figure 11.a, on peut remarquer que quelle que soit l'exécution de l'automate, on atteint inévitablement lors de l'exécution un état à partir duquel tous les états traversés ensuite sont labellés par p. Cependant, si on déploie ce modèle en un arbre de branchement (représenté à la figure 11.b) et qu'on évalue la formule sur celui-ci, l'évaluation aboutit à false. En effet, chaque état de la branche de gauche a deux états successeurs, l'un satisfaisant p, l'autre q. La formule ne représente donc pas la propriété voulue. Pour spécifier des propriétés d'équité faible pour des systèmes non temporisés, on utilisera plutôt une logique linéaire comme LTL. Par contre, la simplicité d'un algorithme de Model Checking CTL (présenté au chapitre suivant) rend cette logique attrayante d'un point de vue pratique.
Il n'y a par contre pas de problème à exprimer le fait qu'une propriété p est infiniment souvent vérifiée. Ceci est fait grâce à la formule $\neg$(true $\exists U$ $\neg$(true $\forall U$ p)). Il est donc tout à fait possible d'exprimer une propriété d'équité forte.
De la même manière qu'en LTL, on peut simuler un temps discret en posant certaines conditions sur les arbres de branchement. Grâce aux opérateurs $\forall\bigcirc$ et $\exists\bigcirc$, il est maintenant possible de spécifier le fait qu'il est possible d'atteindre un état satisfaisant p avant deux unités de temps avec la formule p $\vee$ $\exists\bigcirc$p $\vee$ $\exists\bigcirc\exists\bigcirc$p. Pour spécifier que la propriété est inévitable, on utilise l'opérateur $\forall\bigcirc$.

**La logique TLA ([Lam94a], [Lam94b])**
**Description :**
La logique TLA est un exemple de logique temporelle linéaire non temporisée orientée action (action linear temporal logic).

VAR étant un ensemble de variables, on considère ici que les états d'une trace ne sont pas labellés par un ensemble de propositions mais associent une valeur à chaque variable de VAR.
Un pas est constitué d'un état de départ et d'un état d'arrivée. Une trace peut donc être vue comme une suite de pas.

Une action est une expression booléenne formée de variables, de variables primées, de constantes et d'opérateurs tels que {+, -, =, $\in$, $\wedge$, …}.
Une action représente une relation entre un ancien état et un nouvel état, où les variables non primées font référence aux valeurs données par l'ancien état et les variables primées font références aux valeurs données par le nouvel état. Par exemple, x' = x +1 signifie que la valeur de x dans le nouvel état est la valeur de x dans l'ancien état augmentée de 1.
Une action a est vraie pour un pas si les valeurs données par l'état de départ et d'arrivée aux variables de VAR satisfont l'action a. Un tel pas est appelé un a-pas.
Les états de départ et d'arrivée d'un pas peuvent attribuer les mêmes valeurs aux variables. Ces pas sont appelés bégaiements.
Une action a est disponible dans un état i s'il existe un état j tel que le pas (i, j) est un a-pas.

Un prédicat est une fonction booléenne construite à l'aide de variables, constante et d'opérateurs booléens et arithmétiques. Par exemple, $x^2 = y - 3 \wedge x \in$ Nat donne une relation entre les variables x et y et spécifie que x est un naturel.

Pour un pas, un ensemble f de variables et f' l'ensemble contenant les variables de f primées, on définit f = f' ssi les valeurs données aux variables de f par les ensembles de départ et d'arrivée du pas sont égales. On écrit $f \neq f'$ pour exprimer $\neg( f = f')$.

Une formule est une formule TLA si elle peut être écrite à partir de la grammaire suivante
$\phi \rightarrow P \mid \ [a]_f \mid$ €$\phi \mid \exists x : \phi \mid \neg\phi \mid \phi \wedge \phi \mid WF_f(a) \mid SF_f(a) \mid enabled(a)$
où P est un prédicat.
f est une liste de variables non primée.

Si f désigne, une liste de variables non primées, P un prédicat, $\sigma = s_0, s_1,s_2,\ldots$ une trace et x une variable, la sémantique de TLA est définie comme suit :

$\sigma \models P$ ssi l'état initial de $\sigma$ satisfait le prédicat P.
$\sigma \models \ [a]_f$ ssi chaque pas de la trace est un a-pas ou laisse les variables de f inchangées.
$\sigma \models enabled(a)$ ssi a est disponible dans l'état initial de $\sigma$.
$\sigma \models \neg\phi$ ssi $\sigma \not\models \phi$.
$\sigma \models \phi_1 \wedge \phi_2$ ssi $\sigma \models \phi_1$ et $\sigma \models \phi_2$.
$\sigma \models \ \phi$ ssi $\phi$ est vrai pour tout suffixe de $\sigma$.
$\sigma \models \exists x : \phi$ ssi il existe une trace $\sigma' = s'_0, s'_1,s'_2,\ldots$ tel que $\forall i \in R^+$ : les valeurs données aux variables par $s_i$ et $s'_i$ sont les mêmes sauf pour x et $\sigma' \models \phi$.
$\sigma \models WF_f(a)$ ssi $\sigma \models \ \lozenge\neg(enabled(a \wedge (f' \neq f)))$ ou $\sigma \models \ \neg \ [\neg a]_f$.
$\sigma \models SF_f(a)$ ssi $\sigma \models \lozenge \ \neg(enabled(a \wedge (f' \neq f)))$ ou $\sigma \models \ \neg \ [\neg a]_f$.
où $\lozenge\phi \equiv \neg \ \neg\phi$.

L'opérateur $\exists$x. permet de ne pas tenir compte de la valeur donnée par les différents états d'un comportement à une variable x. Cela est intéressant, par exemple, lorsqu'on est obligé d'utiliser des variables "de travail" pour spécifier la propriété voulue. Ces variables ne sont utilisées que dans un but de définition et aucune contrainte sur les valeurs données à ces variables ne doit être satisfaite par une trace.
Par exemple, si on veut modéliser une barque navigant constamment de la berge gauche à la berge droite d'une rivière et vice versa, on peut modéliser cela simplement en utilisant deux variables booléennes gauche et droite qui prennent alternativement la valeur vrai pour spécifier la berge où se trouve la barque. Si la barque part de la rive gauche, on a ainsi la formule : (on ne spécifie pas le type des variables pour ne pas surcharger la formule)
$gauche \wedge \neg droite \wedge \ _{[gauche,\ droite]}((gauche \rightarrow (droite' \wedge \neg gauche')) \wedge (droite \rightarrow (gauche' \wedge \neg droite')))$

Si on souhaite spécifier que la barque passe par le milieu de la rivière, on aura besoin d'une variable de travail pour indiquer le sens de progression de la barque lorsqu'elle se trouve effectivement au milieu. Si on appelle cette variable sens et qu'elle vaut vrai lorsque la barque va vers la rive gauche et faux lorsque la barque va vers la rive droite, alors on a la formule

$$
\begin{aligned}
\phi \equiv\ & \text{gauche} \wedge \neg\text{droite} \wedge \neg\text{milieu} \wedge {}_{[\text{gauche, droite, milieu,sens}]} (\text{gauche} \rightarrow (\text{milieu}' \wedge \neg\text{gauche}' \wedge \neg\text{droite}' \wedge \neg\text{sens}')) \\
& \wedge (\text{droite} \rightarrow (\text{milieu}' \wedge \neg\text{gauche}' \wedge \neg\text{droite}' \wedge \text{sens}')) \\
& \wedge ((\text{milieu} \wedge \neg\text{sens}) \rightarrow (\text{droite}' \wedge \neg\text{milieu}' \wedge \neg\text{gauche}')) \\
& \wedge ((\text{milieu} \wedge \text{sens}) \rightarrow (\text{gauche}' \wedge \neg\text{milieu}' \wedge \neg\text{droite}'))
\end{aligned}
$$

On pourrait avoir une trace où les variables gauche, milieu et droite prennent chacune la valeur vrai dans le bon ordre, mais la trace ne satisfait pas la formule $\phi$ à cause des valeurs données à sens dans ses états. Or, sens ne correspond à rien dans le comportement qu'on spécifie (le trajet de la barque).
L'opérateur $\exists x.$ permet de ne pas avoir ce problème. En ayant $\exists$sens. $\phi$, on demande que partant de la trace $\sigma$ sur laquelle on évalue la formule, on puisse obtenir une trace $\sigma'$ satisfaisant $\phi$ en modifiant dans $\sigma$ les valeurs attribuées à sens par ses états.
Les opérateurs WF et SF permettent de spécifier des propriétés d'équité faible et forte. Dans ce cadre-ci, la notion de propriété d'équité est légèrement différente de celle qui a été introduite auparavant en faisant intervenir la notion de disponibilité d'une action. $WF_f(a)$ signifie que si l'action $a \wedge f \neq f'$ est continuellement disponible, cela implique qu'il y a un nombre infini de $(a \wedge f \neq f')$-pas. $SF_f(a)$ signifie que $a \wedge f \neq f'$ est infiniment souvent disponible implique qu'il y a un nombre infini de $(a \wedge f \neq f')$-pas.
L'opérateur $[a]_f$ permet de définir des propriétés de sûreté. Exprimer que l'action a avec modification des variables de f ne sera jamais effectuée s'exprime en TLA avec la formule $[\neg a]_f$. Les propriétés de vivacité sont également exprimable grâce à cet opérateur. Dire qu'on effectuera une action a et qu'on modifiera les valeurs des variables d'un ensemble f s'exprime en TLA par la formule $\neg\ [\neg a]_f$.
**Expressivité :**
La possibilité de pouvoir définir les actions de manière précise ne prédestine pas la logique TLA à être utilisée dans le cadre du Model Checking mais dans celui de la vérification d'algorithmes de traitement. Faire de la vérification avec TLA consiste à transformer une formule TLA du type $\phi_1 \rightarrow \phi_2$ en une formule d'un second formalisme (LP) et de faire ensuite la vérification de la formule ainsi trouvée, $\phi_1$ étant une formule donnant le comportement de l'algorithme à vérifier et $\phi_2$ étant la propriété à vérifier. Pour plus de précisions, le lecteur pourra se référer à [LUP92].
Bien que Leslie Lamport présente TLA comme une logique orientée action, il est à noter qu'il est possible d'énoncer des propriétés purement orientées états. En effet, la possibilité d'exprimer des prédicats sur les états permet, en combinant ces prédicats avec l'opérateur , des conjonctions, négations, etc., d'exprimer des propriétés de sûreté sur les états, par exemple le fait que x ne sera jamais plus petit ou égal à y avec la formule $(\neg(x \leq y))$, des propriétés de vivacité sur les états, par exemple le fait que x sera à un moment donné égal à zéro avec la formule $\neg( (x \neq 0))$. On peut également spécifier des propriétés d'équité sur les états. La propriété d'équité faible spécifiant le fait que si un prédicat p est continûment vérifié, alors le prédicat q est infiniment souvent vérifié peut s'exprimer en TLA grâce à la formule $\Diamond\ p \rightarrow\ \Diamond q$ et la propriété d'équité forte spécifiant le fait que le prédicat p est infiniment souvent vérifié implique que le prédicat q l'est aussi est exprimable en TLA par la formule $\Diamond p \rightarrow\ \Diamond q$. On conclut donc que cette logique n'est pas purement orientée action. Ceci ne semble pas avoir été la volonté de son auteur car Leslie Lamport ne semble avoir introduit

cette possibilité de spécifier des prédicats sur les états que pour spécifier des propriétés devant être satisfaites au départ d'une trace.

**La logique ACTL ([NFGR91])**
**Description :**
La logique ACTL est un exemple de logique temporelle de branchement orientée action (action branching time logic).

ACTL est, comme son nom le laisse suggérer, une extension de la logique CTL présentée ci-avant. ACTL ajoute un aspect orienté action par rapport à CTL. Ceci est fait en ne posant plus des conditions sur les propositions labellant les sommets mais sur les actions effectuées lors des changements d'états. Ces conditions sont rajoutées en modifiant la syntaxe et la sémantique des opérateurs temporels ($\bigcirc$, $\exists U$, $\forall U$).
Un certain nombre de contraintes sur les actions, appelées formules d'actions, peuvent être spécifiées. Une formule est une formule d'actions si elle peut être construite à partir de la grammaire suivante :

$\chi \rightarrow a \mid \neg\chi \mid \chi \wedge \chi$

La sémantique des formules d'actions est la suivante :
$a \models b$ ssi $a = b$
$a \models \neg\chi$ ssi $a \not\models \chi$
$a \models \chi \wedge \chi'$ ssi $a \models \chi$ et $a \models \chi'$

Une formule $\phi$ est une formule ACTL si elle peut être construite à partir de la grammaire suivante :

$\phi \rightarrow true \mid \neg\phi \mid \phi \wedge \phi \mid \exists\bigcirc_\chi\phi \mid \exists\bigcirc_\tau\phi \mid \phi\ {}_\chi\exists U_{\chi'}\ \phi \mid \phi\ {}_\chi\forall U_{\chi'}\ \phi \mid \phi\ {}_\chi\exists U\ \phi \mid \phi\ {}_\chi\forall U\ \phi$

où $\chi$ et $\chi'$ sont des formules d'actions.
$\tau$ est l'action invisible.
Nous définissons maintenant la sémantique des opérateurs d'ACTL :
Une formule ACTL est évaluée par rapport à un arbre d'actions de racine $s_0$. Notons qu'ici les branches de l'arbre peuvent être finies.

$s_0 \models true$.
$s_0 \models \neg\phi$ ssi $s_0 \not\models \varphi$.
$s_0 \models \phi_1 \wedge \phi_2$ ssi $s_0 \models \phi_1$ et $s_0 \models \phi_2$.
$s_0 \models \exists\bigcirc_\tau\phi$ ssi il existe une séquence d'actions $\theta$ commençant par la transition invisible ($s_0$, $\tau$, s') où s' $\models \phi$.
$s_0 \models \exists\bigcirc_\chi\phi$ ssi il existe une séquence d'actions $\theta$ commençant par la transition ($s_0$, a, s') où s' $\models \phi$ et $a \models \chi$.
$s_0 \models \phi_1\ {}_\chi\exists U\ \phi_2$ ssi il existe une séquence d'actions $\theta = (s_0, a_0, s_1)\ (s_1, a_1, s_2)\ (s_2, a_2, s_3) \ldots$ tel qu'il existe $(s_i, a_i, s_{i+1})$ avec $s_i \models \phi_2$ et $\forall 0 \leq j < i$ : $s_j \models \phi_1$ et ($a_j \models \chi$ ou $a_j \models \tau$).
$s_0 \models \phi_1\ {}_\chi\exists U_{\chi'}\ \phi_2$ ssi il existe une séquence d'actions $\theta = (s_0, a_0, s_1)\ (s_1, a_1, s_2)\ (s_2, a_2, s_3) \ldots$ tel qu'il existe $(s_i, a_i, s_{i+1})$ avec $s_{i+1} \models \phi_2$, $a_i \models \chi'$, $s_i \models \phi_1$ et $\forall 0 \leq j < i$ : $s_j \models \phi_1$ et ($a_j \models \chi$ ou $a_j \models \tau$).
$s_0 \models \forall\bigcirc_\tau\phi$ ssi pour toute séquence d'actions $\theta$ commençant par ($s_0$, $\tau$, s'), s' $\models \phi$.
$s_0 \models \forall\bigcirc_\chi\phi$ ssi pour toute séquence d'actions $\theta$ commençant par ($s_0$, a, s'), s' $\models \phi$ et $a \models \chi$.
$s_0 \models \phi_1\ {}_\chi\forall U\ \phi_2$ ssi pour toute séquence d'actions $\theta = (s_0, a_0, s_1)\ (s_1, a_1, s_2)\ (s_2, a_2, s_3) \ldots$, il existe i avec $(s_i, a_i, s_{i+1})$ et $s_i \models \phi_2$ et $\forall 0 \leq j < i$ : $s_j \models \phi_1$ et ($a_j \models \chi$ ou $a_j \models \tau$).

$s_0 \models \phi \ {}_{\chi}\forall U_{\chi'} \ \phi'$ ssi pour toute séquence d'actions $\theta = (s_0, a_0, s_1)\ (s_1, a_1, s_2)\ (s_2, a_2, s_3) \ldots$, il existe i avec $(s_i, a_i, s_{i+1})$ et $s_{i+1} \models \phi_2$, $a_i \models \chi$, $s_i \models \phi_1$ et $\forall 0 \leq j < i :\ s_j \models \phi_1$ et ($a_j \models \chi$ ou $a_j \models \tau$).

Comme il a été dit plus haut, les opérateurs d'ACTL sont les opérateurs de CTL auxquels ont été rajoutées des contraintes sur les actions. Par exemple, une formule $\phi_1 \ {}_{\chi}\exists U_{\chi'} \ \phi_2$ spécifie que, partant de l'état racine $s_0$ de l'arbre, il existe une branche de l'arbre permettant d'atteindre un état satisfaisant $\phi_2$, tous les états intermédiaires satisfaisants $\phi_1$. Mais il faut également qu'il y ait une action a, satisfaisant $\chi'$, effectuée pour atteindre l'état satisfaisant $\phi_2$. Les actions effectuées avant a doivent quant à elle satisfaire $\chi$ ou être l'action invisible. Si la contrainte $\chi'$ n'est pas spécifiée, alors il n'est pas demandé qu'au moins une action soit effectuée. Si des actions doivent tout de même être effectuées pour atteindre un état satisfaisant $\phi_2$, celles-ci doivent soit être l'action invisible, soit satisfaire $\chi$. La syntaxe oblige de toujours spécifier la contrainte $\chi$, néanmoins, si $\chi$ est true, aucune contrainte sur les actions ne sera effectivement spécifiée. Par exemple, la formule true ${}_{true}\exists U_a$ true demande qu'il soit possible d'effectuer à un moment donné une action a, n'importe quelle action pouvant être effectuée avant a. Les modifications effectuées à l'opérateur $\forall U$ par rapport à CTL sont les mêmes pour l'opérateur $\exists U$. Les opérateurs $\exists\bigcirc$ et $\forall\bigcirc$ permettent également de spécifier une contrainte sur l'action effectuée pour atteindre l'état suivant.

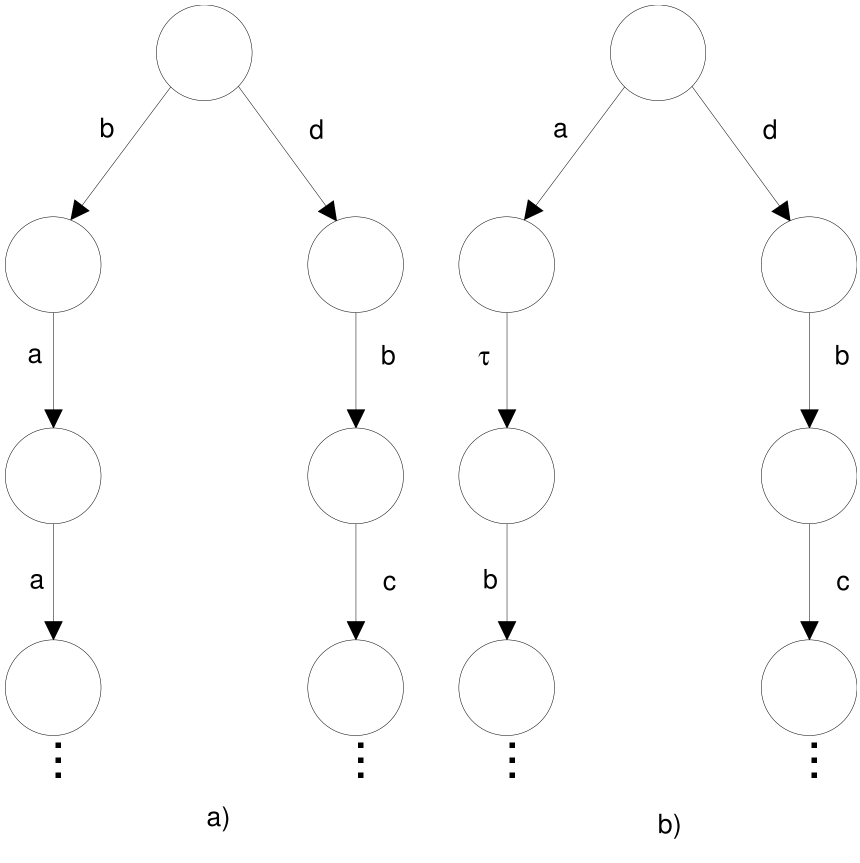



Figure 12: arbres satisfaisant true ${}_a\exists U(\exists\bigcirc_b$ true)

La figure 12 représente deux arbres satisfaisant la formule true ${}_a\exists U$ ($\exists\bigcirc_b$ true). L'arbre de la figure 12.a satisfait bien cette formule car l'état suivant l'état racine se trouvant dans la branche gauche de l'arbre est accessible en faisant une action b et la sémantique ne demande pas qu'au moins une action a soit effectuée. Cette formule est équivalente à la formule true ${}_a\exists U_b$ true. Elles demandent toutes les deux qu'à partir de l'état racine s de l'arbre, il soit possible d'effectuer une action b tendis que les actions la précédant, s'il en existe, sont soit a, soit $\tau$.

**Expressivité :**

Il est possible d'exprimer en ATCTL des propriétés de vivacité, de sûreté et d'équité de la même manière qu'en CTL.

Par exemple, exprimer la propriété d'équité forte selon laquelle une action a infiniment souvent effectuée implique qu'une seconde action b l'est également se fait en ACTL grâce à la formule $\neg(\text{true}\ {}_{\text{true}}\exists U\ \neg(\text{true}\ {}_{\text{true}}\exists U_{a}\ \text{true})) \rightarrow \neg(\text{true}\ {}_{\text{true}}\exists U\ \neg(\text{true}\ {}_{\text{true}}\exists U_{b}\ \text{true}))$.

Comme le montre cet exemple, un problème lié à la syntaxe d'ACTL est que les formules peuvent être considérablement allongées par un nombre important de sous-formules true. Pour pallier ce problème, il est intéressant de définir des abréviations comme par exemple $<\chi>\exists U$ $<\chi'>$, qui correspond à la formule true ${}_{\chi}\exists U_{\chi'}$ true, ou $\exists U <\chi'>$, qui correspond à la formule true ${}_{\text{true}}\exists U_{\chi'}$ true.

La possibilité d'exprimer des propriétés sur des actions est intéressante pour énoncer les propriétés d'un système défini, par exemple, sous la forme d'une spécification LOTOS. En effet, de telles spécifications décrivent l'ensemble des états dans lesquels peut se trouver le système et les différentes actions qui peuvent être effectuées à partir de ces états.

**La logique MITL ([Alu91])**

La logique MITL est un exemple de logique temporelle linéaire temporisée (timed linear temporal logic).

Sa syntaxe est définie somme suit :

$\phi \rightarrow p \mid \neg\phi \mid \phi \wedge \phi \mid \phi\ U_{I}\ \phi$

où I est un intervalle non singulier (différent de [t, t]) dont les bornes sont des entiers. Intuitivement, cet intervalle représente un intervalle de temps dans lequel la seconde sous-formule doit être satisfaite.

Le suffixe $\delta^{t}$ d'une séquence temporisée d'états $\delta = (\sigma, I)$ pour un moment $t \in I_j$ est défini comme étant la séquence temporisée d'états

$(\sigma', I')$ avec

$\sigma' = \sigma_j , \sigma_{j+1} , \ldots$

$I' = I'_j , I'_{j+1} , \ldots$

Où $\forall\ k \geq j : I'_k = I_k - t$

La sémantique des opérateurs de MITL est définie comme suit:

Une formule MITL est évaluée sur une séquence temporisée d'états $\delta = (\sigma, I)$. Notons qu'un intervalle I d'une formule $\phi_1\ U_I\ \phi_2$ ne doit pas (mais peut) être un intervalle de la séquence temporisée d'états sur laquelle elle est évaluée et que les intervalles singuliers sont autorisés dans une séquence temporisée d'états.

$\delta \models p$ ssi $p \in \mu(\sigma_0)$.

$\delta \models \neg\phi$ ssi $\sigma \not\models \phi$.

$\delta \models \phi_1 \wedge \phi_2$ ssi $\sigma \models \phi_1$ et $\sigma \models \phi_2$.

$\delta \models \phi_1\ U_I\ \phi_2$ ssi pour au moins un $t \in I$, $\delta^{t} \models \phi_2$ et $\forall \min(I) \leq t' < t$ avec $t' > 0$, $\delta^{t'} \models \phi_1$ ou $\delta^{t'} \models \phi_2$ et $\forall 0 < t' < \min(I)$, $\delta^{t'} \models \phi_1$.

où $\delta^{t}, \delta^{t'}$ sont des suffixes de $\delta$.

Une séquence temporisée d'états satisfait une formule du type $\phi_1$ $U_I$ $\phi_2$ si la séquence contient un état s satisfaisant $\phi_2$ dans l'intervalle de temps I, tous les états le précédant dans l'intervalle I satisfaisant $\phi_1$ ou $\phi_2$ et les autres états prédécesseurs de s satisfaisant $\phi_1$. Cet opérateur est dit strict en son premier argument, c'est à dire que si $\phi_2$ est satisfait à l'instant initial et que l'instant 0 se trouve dans I, alors la séquence temporisée d'états satisfait la formule $\phi_1$ $U_I$ $\phi_2$, mais dans le cas contraire, il n'est pas demandé que $\phi_1$ soit satisfait à l'instant initial.
L'idée de l'opérateur est d'avoir $\phi_1$ continuellement vérifié jusqu'à ce que $\phi_2$ le soit. Si, après l'instant initial, $\phi_1$ est satisfait jusqu'à l'instant 5 compris, puis la propriété $\phi_2$ est vérifiée alors, vu la densité du temps, on ne pourra pas trouver le premier instant où $\phi_2$ est satisfait. Pour cette raison, on autorise que des états intermédiaires dans I satisfassent $\phi_1$ ou $\phi_2$. Le fait qu'on ait des états intermédiaires satisfaisant $\phi_2$ n'est pas incompatible avec l'idée de l'opérateur.

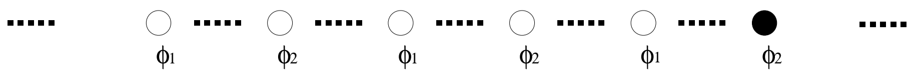



Figure 13

La figure 13 représente une alternance d'états satisfaisant $\phi_1$ ou $\phi_2$ durant un certain intervalle, par exemple [1, 3]. Les points représentent le fait qu'il n'y ait pas de changement de propriétés des états. L'état plein satisfait $\phi_2$ et tous les états le précédant dans l'intervalle [1, 3] dans l'intervalle satisfont soit $\phi_1$, soit $\phi_2$. En supposant que tous les états de la séquence avant l'instant 1, sauf peut-être à l'instant initial, satisfont $\phi_1$, la propriété $\phi_1$ $U_{[1,\ 3]}$ $\phi_2$ est vérifiée. On peut constater qu'effectivement, on a une suite d'états satisfaisant $\phi_1$ jusqu'au moment où un état dans l'intervalle [1, 3] satisfait $\phi_2$, bien que cet état ne soit pas l'état plein.
Si on autorisait les intervalles singuliers dans une formule du type $\phi_1$ $U_I$ $\phi_2$, on aurait dans ces cas-là de nouveau des problèmes liés à la densité du temps. C'est pour cela que cette possibilité est interdite. C'est également à cause de la densité du temps qu'on ne retrouve pas d'opérateur état suivant.

**Expressivité :**

Comme on peut avoir un intervalle [0, ∞[, il est possible d'exprimer en MITL des propriétés de vivacité, de sûreté et d'équité de la même manière qu'en LTL. Ainsi, exprimer le fait qu'on n'atteigne jamais un état ne satisfaisant pas p s'exprime en MITL avec la formule $\neg$(true $U_{[0,\infty[}$ $\neg$p). La différence avec LTL vient du fait que si les états intermédiaires doivent satisfaire une propriété autre que true, cette propriété ne sera pas forcément vérifiée à l'instant initial. Notons qu'à partir de l'opérateur strict en son premier argument, il est possible de définir le cas non strict. Si l'intervalle I contient l'instant initial, il est possible de définir le cas non strict avec la formule $\phi_2 \vee (\phi_1 \wedge \phi_1\ U_I\ \phi_2)$, sinon le cas non strict correspond à la formule $(\phi_1 \wedge \phi_1\ U_I\ \phi_2)$. Ainsi, la formule MITL $q \vee (p \wedge p\ U_{]0,\infty[}\ q)$ et la formule LTL p *U* q exprime toutes les deux que q est vérifié à un moment donné, et si q n'est pas satisfait au départ, p est satisfait entre temps, y compris au départ. Définir le cas strict à l'aide d'un opérateur non strict n'est, par contre, pas possible.
De plus, l'intervalle I rajouté à l'opérateur U permet d'exprimer des propriétés d'atteignabilité bornée avec un temps continu, ce qui permet de spécifier des propriétés temporelles de systèmes temporisés ne contenant pas d'horloges obligeant les changements d'états à se faire à des instants précis.

Il est à noter que, dans [Alu91], la sémantique de MITL ne tient pas compte de la densité du temps. En effet, la sémantique de l'opérateur $U_I$ y est définie de la manière suivante
$\sigma \models \phi_1 \, U_I \, \phi_2$ ssi pour au moins un $t \in I$, $\sigma^t \models \phi_2$ et $\sigma^{t'} \models \phi_1 \; \forall t' \in \,]0, t[$.
On a donc effectué ici une correction de la sémantique lors de sa présentation pour pouvoir tenir compte de ce problème.

**La logique TCTL ([Alu91])**
La logique TCTL est un exemple de logique temporelle de branchement temporisée (timed branching time logic).

La logique TCTL, comme ACTL, est une extension de la logique CTL. Une notion de temps quantifié est rajoutée dans la syntaxe et la sémantique de TCTL. Contrairement à MITL, les contraintes de temps ne se font pas en donnant un intervalle de temps pendant lequel des propriétés doivent être vérifiées, mais en rajoutant un type de formule permettant de spécifier des contraintes sur des horloges (qui permettent ainsi de définir des intervalles).

La syntaxe de TCTL est définie comme suit :
Une formule $\phi$ est une formule TCTL si elle peut être construite à partir de la grammaire suivante:

$$\phi \rightarrow p \mid (x + c) \leq (y + d) \mid \neg\phi \mid \phi \wedge \phi \mid \phi \, \exists U \, \phi \mid \phi \, \forall U \, \phi \mid x.\phi$$

où $c, d \in \angle$.
x, y sont des horloges de la logique, c'est à dire des horloges distinctes des horloges utilisées dans un automate temporisé quelconque.

La sémantique de TCTL est normalement définie par rapport à des arbres temporisés de branchement. Au chapitre suivant, néanmoins, nous présentons un algorithme de Model Checking TCTL où les formules sont évaluées sur une structure temporisée. Montrer que l'évaluation d'une formule TCTL sur un tel modèle est équivalent à la vérifier sur des arbres temporisés n'est pas évident. Pour cette raison, nous donnons ici la définition de la sémantique de TCTL directement par rapport à une structure temporisée.

On considère ici qu'une valuation donne également une valeur aux horloges de la logique. Les horloges de la logique et de l'automate vieillissent donc à la même vitesse.

Une exécution r d'une structure temporisée $(S, s_{init}, \mu, H', \Delta, \delta)$ est une séquence temporisée infinie d'états étendus par des ensembles d'horloges, valuations d'horloges et initialisation d'horloges du type
$\rightarrow_{\nu 0} (s_0, I_0) \,^{\lambda 1}\!\!\rightarrow_{\nu 1} (s_1, I_1) \,^{\lambda 2}\!\!\rightarrow_{\nu 2} (s_2, I_2) \,^{\lambda 3}\!\!\rightarrow_{\nu 3} \ldots$
où $s_i \in S$.
$I_i$ est un intervalle.
$\lambda_i \subseteq H'$.
$\nu_i$ est une valuation.
Cette séquence infinie satisfaisant les contraintes suivantes:

- pour tout $i \geq 0$, soit $< s_i , s_{i+1} , \lambda_i > \in \delta$ ou $s_{i+1} = s_i$ avec $\lambda_i = \emptyset$.
- $\nu_{i+1} = (\nu_i + Max(I_i) - Min(I_i)) \, [\lambda_{i+1} \rightarrow 0]$ pour tout $i \geq 0$.
- pour tout $t \in I_i$ , les valeurs des horloges, données par $\nu_i + t - Min(I_i)$, satisfont l'invariant de $s_i$.

Un calcul $\rho_r$ associé à une exécution r est une fonction qui associe à chaque élément de $R^+$ un état et est défini comme,
Pour tout $t \in I_j$, $\rho_r(t) = \langle s_j, \nu_j + t - \min(I_j) \rangle$

La sémantique de TCTL est définie comme suit :
Soit une structure temporisée $T = (S, s_{init}, \mu, H, \Delta, \delta)$ et un état $q = \langle s, \nu \rangle$ de T et une formule TCTL $\phi$, la relation de satisfaction $q \models \phi$ est définie de manière inductive comme suit

$q \models p$ ssi $p \in \mu(s)$
$q \models (x + c) \leq (y + d)$ ssi $(\nu(x) + c) \leq (\nu(y) + d)$
$q \models \neg\phi$ ssi $\langle s, \nu \rangle \not\models \phi$
$q \models \phi_1 \wedge \phi_2$ ssi $\langle s, \nu \rangle \models \phi_1$ et $\langle s, \nu \rangle \models \phi_2$
$q \models x.\phi$ ssi $\langle s, \nu [x \rightarrow 0] \rangle \models \phi$
$q \models \phi_1 \, \exists U \, \phi_2$ ssi il existe une exécution r de T ayant son calcul $\rho_r$ tel que $\rho_r(0) = q$ et satisfaisant : $\rho_r(t) \models \phi_2$ pour un $t \geq 0$, et $\rho_r(t') \models \phi_1 \vee \phi_2$ pour tout $0 \leq t' < t$.
$q \models \phi_1 \, \forall U \, \phi_2$ ssi toute les exécutions r de T ayant leur calcul $\rho_r$ tel que $\rho_r(0) = q$ et satisfaisant : $\rho_r(t) \models \phi_2$ pour un $t \geq 0$, et $\rho_r(t') \models \phi_1 \vee \phi_2$ pour tout $0 \leq t' < t$.
où x, y sont des horloges de la logique.

Une formule $\phi_1 \, \exists U \, \phi_2$ ($\phi_1 \, \forall U \, \phi_2$) spécifie qu'il doit être possible (inévitable) d'atteindre, en vieillissant et en effectuant des changements de sommet, un état satisfaisant la propriété $\phi_2$ avec tous les états intermédiaires satisfaisant $\phi_1 \vee \phi_2$. L'idée de l'opérateur, comme en CTL, est que les états visités pour atteindre l'état satisfaisant $\phi_2$ doivent satisfaire $\phi_1$, mais vu la densité du temps, comme dans le cas de MITL, on autorise que les états intermédiaires satisfassent $\phi_1 \vee \phi_2$. De nouveau, ceci n'est pas incompatible avec l'idée de base de l'opérateur.
L'opérateur x. correspond à une remise à zéro d'horloge. En effet, un état $\langle s, \nu \rangle$ satisfait la formule $x.\phi$ si l'état $\langle s, \nu' \rangle$ satisfait $\phi$ où $\nu'$ donne les mêmes valeurs aux horloges que $\nu$ à l'exception de x, une variable de la logique, qui vaut zéro. Cet opérateur est, en général, utilisé avec des sous-formules $(x + c) \leq (y + d)$, ce qui permet de définir des intervalles de temps pendant lesquels des propriétés doivent être vérifiées.
Par exemple, la propriété : "il est possible que p soit atteint avant l'instant 2." peut être spécifiée en TCTL par de la formule $x.(\text{ true } \exists U \, y.(p \wedge x \leq y + 2))$.
Un état satisfait cette formule si on peut trouver une exécution commençant par le même état excepté que x vaut zéro dans laquelle on traverse des états pouvant avoir n'importe quelle propriété. Mais il faut qu'à un moment donné, on atteigne un état s satisfaisant p. Il faut également que dans l'état qui est l'état s où la valeur de y a été mise à zéro, x soit plus petit ou égal à y + 2. Comme x est mis à zéro dans l'état de départ, la formule spécifie bien la propriété demandée.
**Expressivité :**
Les propriétés de vivacité, sûreté et d'équité sont exprimables en TCTL de la même façon qu'en CTL. De plus, les contraintes sur les horloges et l'opérateur d'initialisation d'horloge permettant en plus de spécifier des propriétés d'atteignabilité bornée avec un temps continu. Ceci permet d'énoncer des propriétés de systèmes temporisés pour lesquels le temps a cette propriété. Notons néanmoins la disparition de l'opérateur ❍ par rapport à CTL.

Comme pour MITL, lors de la définition de la sémantique de TCTL, Alur a oublié de tenir compte du problème de la densité du temps puisque, par exemple il définit la relation d'équivalence pour l'opérateur $\exists U$ de la manière suivante

$\langle s, \nu \rangle \models \phi_1 \ \exists U\ \phi_2$ ssi pour une exécution r de T ayant un calcul $\rho_r$ tel que $\rho_r(0) = \langle s, \nu \rangle$ et satisfaisant : $\rho_r(t) \models \phi_2$ pour un $t \geq 0$, et $\rho_r(t') \models \phi_1$ pour tout $0 \leq t' < t$.

On a donc ici effectué une correction lors de la définition de la sémantique.

Notons que la sémantique de la formule $(x + c) \leq (y + d)$ est définie de manière différente par Alur. En effet, dans [Alu91], la sémantique de cette formule est définie comme

$q \models (x + c) \leq (y + d)$ ssi $\nu(x) + d \geq \nu(y) + c$

c'est à dire que ce type de formule ne revient pas, comme ici, à comparer la valeur réelle de deux horloges, mais les derniers instants où ces horloges ont été remises à zéro. Si on note ces derniers instants comme étant, respectivement pour x et y, $\xi(x)$ et $\xi(y)$, la contrainte spécifiée est $(\xi(x) + c) \leq (\xi(y) + d)$. Grâce à la relation $\nu(x) = t - \xi(x)$, où $\nu$ est une valuation, valable pour toute horloge x de la logique, on a

$(\xi(x) + c) \leq (\xi(y) + d) \equiv (t - \xi(y) + c) \leq (t - \xi(x) + d) \equiv (\nu(y) + c) \leq (\nu(x) + d)$.

Le fait qu'on teste les derniers instants où les horloges ont été remises à zéro plutôt que leur valeur réelle est le résultat d'un choix arbitraire. La relation $\nu(x) = t - \xi(x)$ montre qu'il n'y a pas de différence d'expressivité entre ces deux manières de poser des contraintes sur les variables.

Alur utilise dans sa thèse cette méthode car il définit deux fois la sémantique de TCTL : une fois par rapport à des arbres temporisés de branchement (en fait un ensemble de séquences temporisées d'états ayant certaines propriétés qui font qu'on peut le comparer à des arbres temporisés de branchement) et une fois par rapport à une structure temporisée. La première fois, il définit également l'opérateur x. comme étant non pas une remise à zéro, mais comme l'opérateur "freeze", c'est à dire que cet opérateur assigne la valeur de l'instant courant à x. Donc, si cet opérateur est effectué dans un nœud d'un arbre temporisé à l'instant t, x reçoit la valeur t. S'il avait utilisé l'initialisation d'horloges et la comparaison des valeurs réelles des horloges, il aurait dû modifier la valeur de ses variables à chaque vieillissement lorsqu'il parcourt un arbre temporisé de branchement. L'utilisation de l'opérateur "freeze" et la comparaison des derniers instants où les variables ont été initialisées lui évite de faire cela.

Lorsqu'il redéfinit la sémantique par rapport à une structure temporisée, il transforme l'opérateur x. en opérateur d'initialisation d'horloges, modifie la sémantique de $(x + c) \leq (y + d)$, mais pour rester cohérent avec sa première définition, ne change pas le sens de la formule (cela correspond toujours à comparer les derniers instants où les horloges ont été remises à zéro).

Comme nous avons ici défini la sémantique de TCTL directement sur une structure temporisée, que les deux méthodes de comparaisons d'horloges sont équivalentes, et pour ne pas choquer le lecteur, on a choisi ici de comparer les valeurs réelles des horloges.

# Chapitre 4

# Model Checking

En théorie, ayant une propriété $\phi$ exprimée sous forme de formule d'une logique temporelle et un système S, vérifier que S satisfait $\phi$, dénoté $S \models \phi$, consiste à trouver l'ensemble $\Gamma$ des traces ou des arbres de transitions, suivant les cas, représentant les différentes exécutions possibles de S, puis à vérifier que chacune de ces traces ou chacun de ces arbres satisfont la propriété ( $\forall\, \sigma \in \Gamma : \sigma \models \phi$ ).
En pratique, étant donné qu'à la fois le nombre et la taille des traces ou arbres considérés est infini, l'utilisation d'automates représentant de manière finie les évolutions possibles d'un système, et de logiques temporelles permet de faire la vérification de systèmes. On présente ici plusieurs algorithmes de vérifications utilisant différentes logiques temporelles et modèles présentés dans les chapitres précédants. Les algorithmes repris ici ne sont pas que de simples retranscriptions d'algorithmes décrits dans la littérature. Des modifications ont été faites par rapport aux écrits de référence à des fins d'amélioration, de correction et d'adaptation, l'algorithme de Model Checking LTL présenté ici s'inspirant d'un algorithme pour une autre logique.

## 1. Algorithmes

**Algorithme pour la logique LTL ([MP92])**
Pour vérifier une formule $\phi$, l'idée est de créer un automate de Büchi représentant l'ensemble des séquences d'états satisfaisant $\neg\phi$ et de vérifier qu'aucune de ces séquences n'est une séquence d'états possible de l'automate de Büchi S représentant le système à vérifier. Le Model Checking pour cette logique linéaire revient donc à vérifier que l'ensemble des traces infinies de S est inclus dans l'ensemble des traces infinies qui vérifient la formule $\phi$.

Etant donné que ce sont les transitions dans un automate de Büchi qui sont labellées, on définit l'ensemble des traces d'un automate de Büchi comme l'ensemble des strings acceptés par cet automate. Comme dans le cadre de cet algorithme-ci, les transitions sont labellées par des ensembles de propositions, les strings acceptés sont des séquences infinies d'ensembles de propositions, chaque ensemble étant considéré comme un état.

Nous définissons la fermeture $cl(\phi)$ d'une formule $\phi$ comme étant le plus petit ensemble de formules satisfaisant les conditions suivantes :

- $\phi \in cl(\phi)$
- $\phi_1 \wedge \phi_2 \in cl(\phi) \rightarrow \phi_1, \phi_2 \in cl(\phi)$
- $\phi_1 \vee \phi_2 \in cl(\phi) \rightarrow \phi_1, \phi_2 \in cl(\phi)$
- $\phi_1 \in cl(\phi) \rightarrow \neg\, \phi_1 \in cl(\phi)$
- $\bigcirc\, \phi_1 \in cl(\phi) \rightarrow \phi_1 \in cl(\phi)$
- $\phi_1\ U\ \phi_2 \in cl(\phi) \rightarrow \phi_1, \phi_2 \in cl(\phi)$
- $\phi_1 \wr \phi_2 \in cl(\phi) \rightarrow \phi_1, \phi_2 \in cl(\phi)$
- $\neg\phi_1 \in cl(\phi) \rightarrow \phi_1 \in cl(\phi)$

$cl(\phi)$ est donc l'ensemble des sous formules de $\phi$ et leur négation.

L'automate de Büchi représentant l'ensemble des traces satisfaisant une formule $\phi$ est créé à partir de deux autres automates de Büchi : l'automate local (local automaton) et l'automate des éventualités (eventuality automaton).

L'automate local accepte des séquences d'états où chaque état est un ensemble de formules représentant les sous formules de $\phi$ qui sont satisfaites dans cet état, le premier état de chaque séquence contenant $\phi$.
L'automate Local est un automate de Büchi L ($N_L$, $\rho_L$, $N_\phi$, $N_L$) où

- $N_L$ est l'ensemble des états s qui sont des sous-ensembles de $cl(\phi)$ et qui satisfont les conditions suivantes :
  - Pour tout $\phi_1 \in cl(\phi)$, nous avons $\phi_1 \in s$ ssi $\neg\phi_1 \notin s$.(1)
  - Pour tout $\phi_1 \wedge \phi_2 \in cl(\phi)$, nous avons $\phi_1 \wedge \phi_2 \in s$ ssi $\phi_1, \phi_2 \in s$.(2)
  - Pour tout $\phi_1 \vee \phi_2 \in cl(\phi)$, nous avons $\phi_1 \vee \phi_2 \in s$ ssi $\phi_1$ ou $\phi_2 \in s$.(3)
  - Pour tout $\phi_1\ U\ \phi_2 \in cl(\phi)$, nous avons $\phi_1\ U\ \phi_2 \in s$ si $\phi_1$ ou $\phi_2 \in s$ et si $\phi_2 \in s$ alors $\phi_1\ U\ \phi_2 \in s$.(4)
  - Pour tout $\phi_1 \wr \phi_2 \in cl(\phi)$, nous avons $\phi_1 \wr \phi_2 \in s$ si $\phi_2 \in s$ et si $\phi_1 \wedge \phi_2 \in s$ alors $\phi_1 \wr \phi_2 \in s$.(5)

  Ces conditions empêchent des contradictions entre les formules contenues dans les états de l'automate et rendent l'ensemble des formules satisfaites dans les états cohérent. Si $\phi_1 \vee \phi_2$ est satisfait dans un état, il faut également que $\phi_1$ ou $\phi_2$ soit satisfait dans cet état.
- La fonction de transition $\rho_L$ définit l'ensemble des transitions, partant d'un état s vers un état t, satisfaisant les conditions suivantes :
  - Pour tout $\bigcirc\, \phi_1 \in cl(\phi)$, nous avons $\bigcirc\, \phi_1 \in s$ ssi $\phi_1 \in t$.
  - Pour tout $\phi_1\ U\ \phi_2 \in cl(\phi)$, nous avons $\phi_1\ U\ \phi_2 \in s$ ssi soit $\phi_2 \in s$, soit $\phi_1 \in s$ et $\phi_1\ U\ \phi_2 \in t$.
  - Pour tout $\phi_1 \wr \phi_2 \in cl(\phi)$, nous avons $\phi_1 \wr \phi_2 \in s$ ssi soit $\phi_1$ et $\phi_2 \in s$, soit $\phi_2$ et $\phi_1 \wr \phi_2 \in t$.

  Les transitions respectent ainsi les conditions imposées par les formules temporelles des états ; elles sont labellées par l'ensemble des formules de leur état d'origine.
- L'ensemble des états initiaux $N_\phi$ est l'ensemble des états qui contiennent la formule $\phi$.
- L'ensemble des états finaux est l'ensemble des états lui-même.

On a rajouté ici, par rapport à [MP92], plusieurs conditions devant être satisfaites par les ensembles de formules de $N_L$. Dans [MP92], les ensembles de formules contenues dans $N_L$ ne doivent satisfaire que des conditions par rapport aux formules qu'ils contiennent construites autours d'une conjonction, disjonction ou d'une négation (conditions 1,2 et 3). Des conditions sur les formules à base d'opérateurs temporels (4 et 5) sont ici rajoutées. Les conditions que doivent respecter les transitions de $\rho_L$ interdisent à ces transitions d'avoir un état ne satisfaisant pas ces conditions 4 et 5 comme origine. Donc, si on gardait dans $N_L$ les ensembles de formules ne satisfaisant pas les conditions 4 et 5, une séquence d'états contenant un tel état ne pourrait qu'être finie. Or, une séquence d'états d'un automate de Büchi n'est valide que si elle est infinie. Eliminer ces états permet donc d'alléger l'automate local sans modifier son ensemble de séquences d'états infinies. La figure 14 montre l'automate local construit pour la formule p *U* q. Les transitions de l'automate doivent normalement être labellées par les formules de leur état de départ.

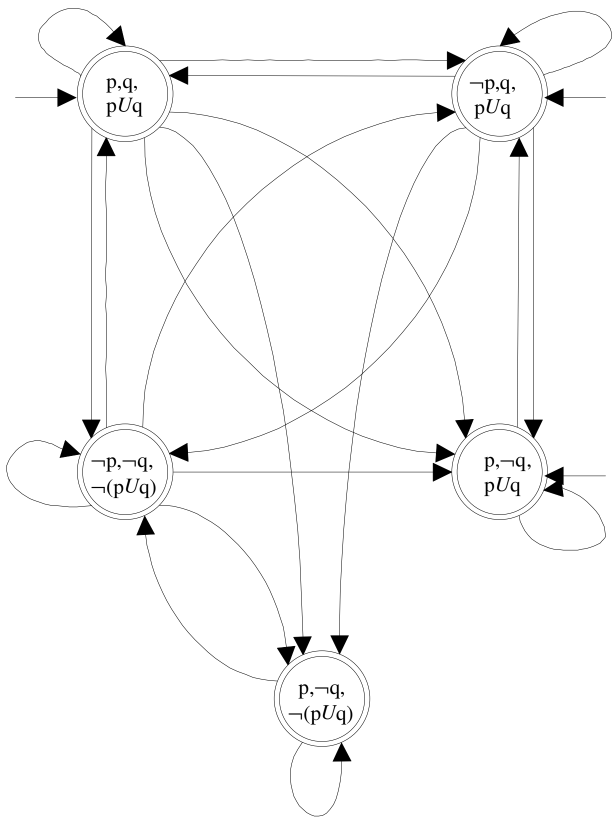
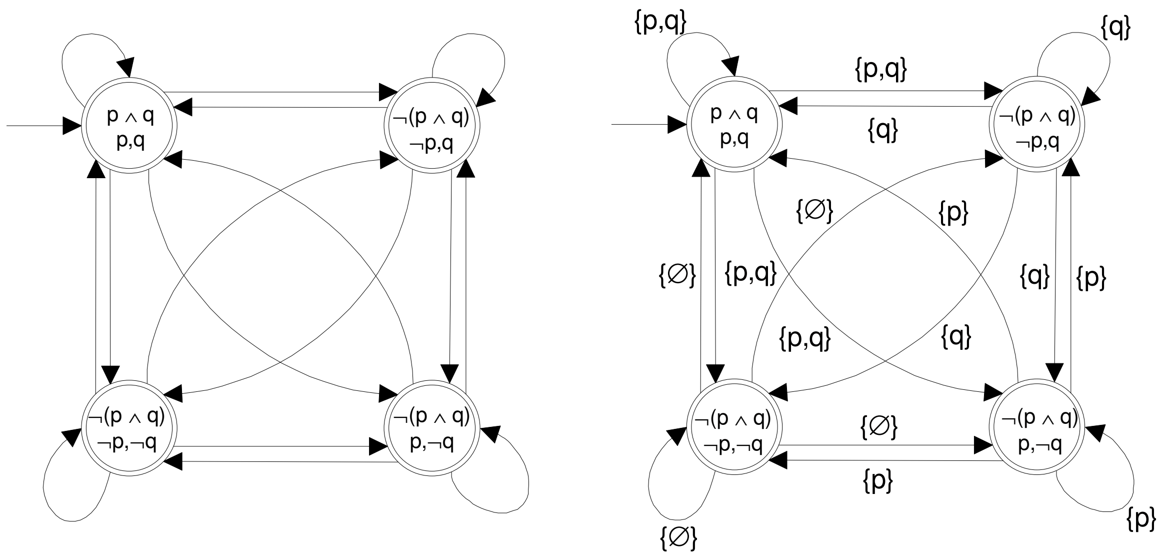



Figure 14: automate local pour la formule p*U*q

On a écrit plus haut que dans les séquences d'états de l'automate local, les états étaient constitués de l'ensemble des formules qui y étaient satisfaites. Or, en regardant l'automate de la figure 14, on peut constater que ce n'est pas tout à fait vrai : partant de l'état {p, ¬q, p *U* q}, on peut effectuer continuellement la transition aboutissant à ce même état. On a donc des séquences d'états où des états contenant la formule p *U* q et ¬q ne sont pas suivis par un état contenant q. Pour éviter cela, il faut construire un deuxième automate, l'automate des éventualités.

Si l'automate local a été construit pour une formule $\phi$ ne contenant pas de sous-formule du type $\phi_1 \; U \; \phi_2$ (ou $\neg(\phi_1 \; \tilde{U} \; \phi_2)$), par exemple $p \wedge q$, l'automate représentant l'ensemble des traces satisfaisant $\phi$ est l'automate trouvé à partir de l'automate local en retirant des labels des transitions les formules qui ne sont pas simplement des propositions. La figure 15.a représente l'automate local de la formule $p \wedge q$, les transitions devant être labellées par les formules contenues dans leur état d'origine. La figure 15.b représente l'automate de Büchi acceptant toutes les séquences d'états satisfaisant $p \wedge q$. Dans ce cas, il n'est donc pas nécessaire de construire l'automate des éventualités.




a) automate local de la formule $p \wedge q$

b) automate de Büchi acceptant l'ensemble des séquences d'états satisfaisant $p \wedge q$.

Figure 15

Un état de l'automate des éventualités est un ensemble de sous-formules de $\phi$ du type $\phi_1 \; U \; \phi_2$ (ou $\neg(\phi_1 \; \tilde{U} \; \phi_2)$) qui joue un rôle de mémoire permettant de savoir quelles sont les formules de ce type qui ont été satisfaites. Les transitions sont labellées par des formules de $cl(\phi)$. Contrairement à l'automate local, une transition labellée par les formules $\phi_1 \; U \; \phi_2$ et $\neg\phi_2$ est automatiquement suivie d'une transition labellée par $\phi_2$, les transitions intermédiaires étant labellées par $\phi_1$. Pour cela, les transitions labellées par des formules $\phi_1 \; U \; \phi_2$ et $\neg\phi_2$ aboutissent automatiquement à un état contenant la formule $\phi_1 \; U \; \phi_2$. Partant d'un tel état, on atteint un état ne contenant plus cette formule uniquement en effectuant une transition labellée par $\phi_2$. L'ensemble des états finaux ne contenant que l'ensemble vide, une transition labellée par $\phi_2$ doit être effectuée après une transition labellée par des formules $\phi_1 \; U \; \phi_2$ et $\neg\phi_2$. Si ce n'est pas le cas, tous les états de l'automate traversés après la transition labellée par $\phi_1 \; U \; \phi_2$ et $\neg\phi_2$ contiendraient la formule $\phi_1 \; U \; \phi_2$ et l'état final ne serait pas infiniment souvent traversé.

Donc, l'ensemble des traces de l'automate des éventualités est l'ensemble des traces dont les états sont labellés par les formules de $cl(\phi)$ qui sont satisfaites dans ces états. Contrairement aux traces de l'automate local, tout état d'une trace de l'automate des éventualités labellé par des formules du type $\phi_1 \; U \; \phi_2$ (ou $\neg(\phi_1 \; \tilde{U} \; \phi_2)$) et $\neg\phi_2$ est obligatoirement suivit d'un état labellé par $\phi_2$. Par contre, les états initiaux des traces ne sont pas obligatoirement labellés par $\phi$.

Nous définissons ev($\phi$) comme étant l'ensemble des éléments de cl($\phi$) de la forme $\phi_1\ U\ \phi_2$ (ou $\neg(\phi_1 \wr \phi_2)$).

L'automate des éventualités est l'automate de Büchi ($2^{ev(\phi)}$, $\rho_F$, {$\varnothing$}, {$\varnothing$}) où

- L'ensemble des états est l'ensemble des sous-ensembles de ev($\phi$).
- La fonction de transition $\rho_F$ définit un ensemble de transitions labellées par un ensemble de formules $A \in 2^{cl(\phi)}$. Contrairement à [MP92], on impose que l'ensemble A satisfasse les mêmes conditions que les états de l'automate local. Il n'y a donc pas de contradictions entre les formules de A, c'est à dire qu'on ne pourra pas avoir de transition labellée, par exemple, par la formule $\phi_1 \wedge \phi_2$ et la formule $\neg\phi_2$, ce qui voudrait dire que les formules $\phi_1 \wedge \phi_2$ et $\neg\phi_2$ pourrait être vérifiées en même temps. De plus, les transitions satisfont les conditions suivantes où s est l'état d'origine de la transition et t est l'état d'arrivée :
  - Si $s = \varnothing$ alors pour tout $\phi_1\ U\ \phi_2$ (ou $\neg(\neg\phi_1 \wr \neg\phi_2)) \in A$, on a $\phi_1\ U\ \phi_2$ (ou $\neg(\neg\phi_1 \wr \neg\phi_2)) \in t$ ssi $\phi_2 \notin A$.
  - Si $s \neq \varnothing$ alors pour tout $\phi_1\ U\ \phi_2$ (ou $\neg(\neg\phi_1 \wr \neg\phi_2)) \in s$, on a $\phi_1\ U\ \phi_2$ (ou $\neg(\neg\phi_1 \wr \neg\phi_2)) \in t$ ssi $\phi_2 \notin A$.

  Ces conditions imposent que lorsque l'on effectue une transition labellée par $\phi_1\ U\ \phi_2$ (ou $\neg(\neg\phi_1 \wr \neg\phi_2)$)et $\neg\phi_2$, il faut que l'état d'arrivée contienne cette formule pour retenir que $\phi_2$ doit labeller une transition future. De plus, si $\phi_2$ labelle une transition, l'état d'arrivée ne peut contenir la formule $\phi_1\ U\ \phi_2$ (ou $\neg(\neg\phi_1 \wr \neg\phi_2)$).
- L'ensemble des états initiaux se résume à l'état $\varnothing$. En effet, au départ, aucune formule n'a encore été vérifiée. Par conséquent, aucune formule temporelle ne doit se trouver dans l'état de départ.
- L'ensemble des états finaux se résume aussi à l'état $\varnothing$. En effet, lorsque on passe par cet état, si une formule $\phi_1\ U\ \phi_2$ a été vérifiée (c'est à dire si une transition labellée par cette formule a été effectuée), alors la formule $\phi_2$ a ensuite, ou en même temps, été vérifiée (alors la même transition ou une transition effectuée ensuite était labellée par $\phi_2$).

La figure 16 montre l'automate des éventualités pour la formule p *U* q.

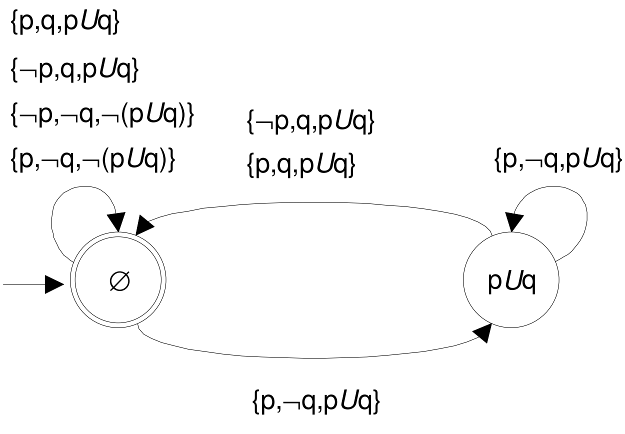



Figure 16: automate des éventualités pour la formule p*U*q

Ayant l'automate local et l'automate des éventualités, on crée un automate qui est l'intersection entre ces deux automates. Ce nouvel automate est appelé automate modèle (model automaton).

Cet automate est $M = (N_M, \rho_M, N_{M0}, F_M)$ où

- $N_M = N_L \times 2^{ev(\phi)}$ où pour $A \in N_L$ et $A' \in 2^{ev(\phi)}$, on a le sommet $(A,A')$ si pour toute formule $\phi_i \in A'$, on a $\neg\phi_i \notin A$. Notons que cela implique que $A' \subset A$.
- Ayant la transition de s vers p labellée par $a \in \rho_L$ et la transition de t vers q labellée par $a \in \rho_F$ alors on a la transition de (s, t) vers (p, q) labellée par $a \in \rho_M$.
- $N_{M0} = N_\phi \times \{\emptyset\}$, c'est à dire les états constitués d'états initiaux des deux modèles.
- $F_M = N_L \times \{\emptyset\}$, c'est à dire les états constitués d'états finaux des deux modèles.

Contrairement à [MP92], on impose des conditions sur les états. Le but de ceci est de nouveau d'alléger l'automate en enlevant des états qui ne servent à rien. En effet, prenant un état de l'automate des éventualités contenant une formule $\phi_1$ *U* $\phi_2$, toutes les transitions partant de cet état seront labellées par cette même formule. Par contre, toutes les transitions partant d'un état de l'automate local contenant une formule $\neg(\phi_1$ *U* $\phi_2)$ seront labellées par cette formule. Pour cette raison, un état de l'automate modèle qui serait la composition de deux états dont l'un contiendrait une formule et l'autre sa négation n'aurait pas de transitions qui en partirait. De nouveau, aucune trace infinie (et donc valide) de l'automate ne contiendrait de tels états et ceux-ci peuvent donc être omis sans modifier l'ensemble des traces infinies de l'automate.
Notons également que, de nouveau, contrairement à [MP92], il est demandé ici que les ensembles de formules labellant les transitions de l'automate des éventualités satisfassent les mêmes conditions que les états de l'automate local, c'est à dire une certaine cohérence. Ceci est fait pour correspondre aux idées de l'automate (les états des traces de l'automate des éventualités sont labellés par les formules de $cl(\phi)$ qui y sont satisfaites) ; de plus, les transitions ne satisfaisant pas ces conditions n'auraient servi à rien lors de la construction de l'automate modèle. Toutes les transitions de l'automate local suivent, en effet, également ces règles.

Pour une formule $\phi$, l'automate modèle génère un ensemble de séquences d'états dont les états représentent les sous formules de $\phi$ satisfaites dans ces états. Toutes les traces de cet automate commencent par un état contenant $\phi$ et si elles contiennent un état labellé par une formule $\phi_1$ *U* $\phi_2$, contrairement à l'automate local, alors cet état sera obligatoirement suivi d'un état labellé par $\phi_2$ s'il ne contient pas lui-même cette formule. La figure 17 montre l'automate modèle pour la formule p *U* q où les transitions doivent être labellées par les formules contenues dans leur état d'origine.

Ce dont on a besoin, c'est un automate représentant l'ensemble des séquences d'états satisfaisant une formule $\phi$. Pour obtenir cet automate, on doit reprendre l'automate modèle que l'on nettoie en enlevant les formules labellant les transitions qui ne sont pas des propositions.

On peut ainsi construire l'automate de Büchi représentant l'ensemble des séquences d'états satisfaisant $\phi$, c'est à dire satisfaisant la négation de la formule à vérifier. Ayant par ailleurs l'automate de Büchi donnant le comportement du système à vérifier, il ne reste plus qu'à faire l'intersection entre ces deux automates et regarder si l'automate ainsi trouvé génère l'ensemble vide. Si c'est le cas, les ensembles de séquences d'états représentés par les deux automates de Büchi n'ont pas de séquences d'états communes et on peut conclure que le système satisfait la formule à vérifier. Si ce n'est pas le cas, les ensembles de séquences d'états représentés par les deux automates ont des séquences d'états communes. Ceci implique qu'au moins une séquence d'états de l'automate représentant le comportement du système ne satisfait pas la formule à vérifier. On en conclut dans ce cas que le système ne satisfait pas la formule à vérifier.

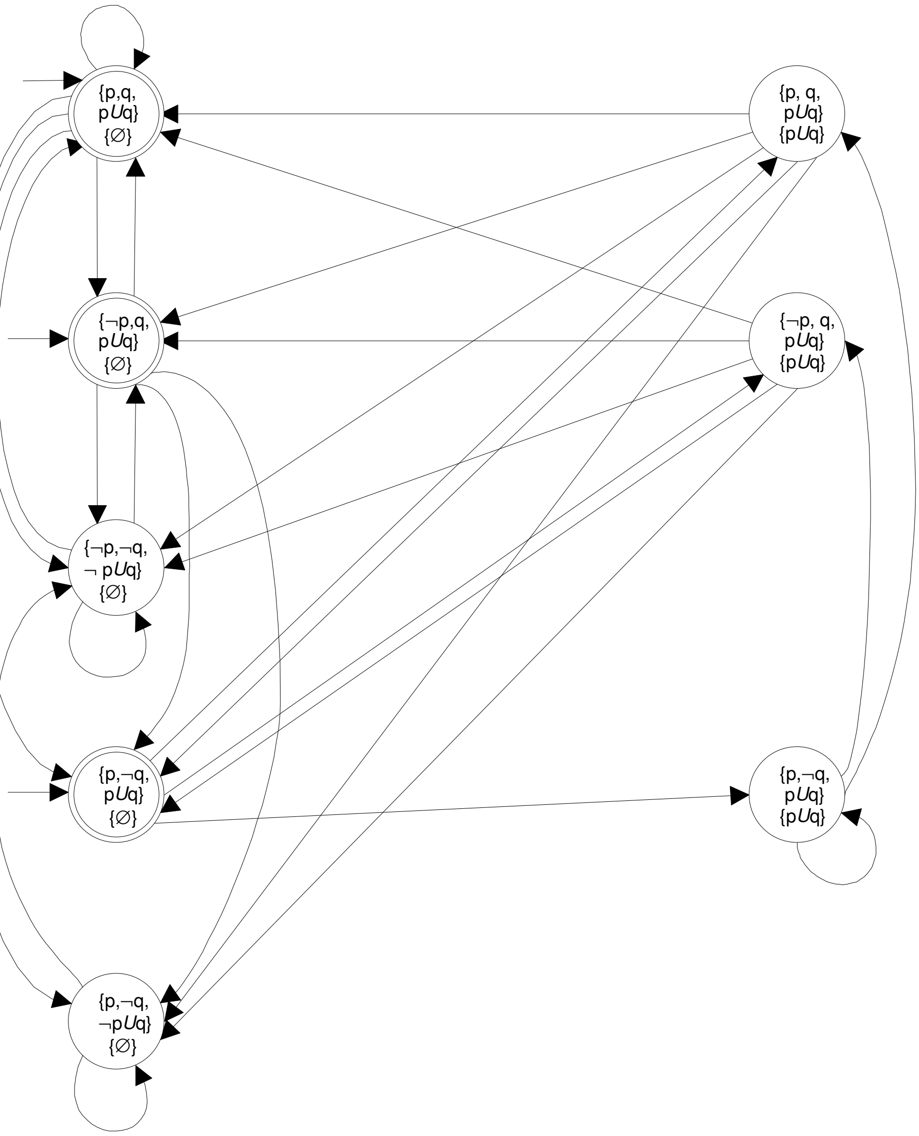



Figure 17: Automate modèle pour la formule p*U*q

Vérifier qu'un automate de Büchi ne représente pas l'ensemble vide de séquences d'états se fait en regardant si il existe un état final se trouvant dans un cycle et accessible à partir d'un état de départ. Cela peut être fait en appliquant l'algorithme de Tarjan pour déterminer les composantes fortement connexes de l'automate. Si des états finaux se trouvent dans ces composantes fortement connexes, un parcours, en largeur ou en profondeur, est effectué à partir des états initiaux jusqu'à ce qu'on atteigne un de ces états. Il existe d'autres algorithmes particulièrement efficaces pour déterminer si le langage accepté par un automate de Büchi est vide.

L'intersection entre N automates de Büchi se calcule plus ou moins de la même manière que l'intersection de simples automates.
Soit $L_1, L_2, \ldots, L_N$ des automates de Büchi. $L_1 \cap L_2 \cap \ldots \cap L_N$, est également un automate de Büchi où les états sont du type $(s_1, s_2, \ldots, s_N, k)$ où $s_1$ est un état de $L_1$, $s_2$ est un état de $L_2, \ldots$ et k est un compteur dont la valeur évolue entre 1 et N. Nous avons une transition de l'état $(s_1, s_2, \ldots, s_N, k)$ vers l'état $(s_1', s_2', \ldots, s_N, k')$ labellée par a ssi il y a une transition de $s_1$ vers $s_1'$ labellée par a dans $L_1$ et il y a une transition de $s_2$ vers $s_2'$ labellée par a dans $L_2, \ldots$ et $k' = (k \bmod N) + 1$ (N est le nombre d'automate) ssi $s_k \in F_k$ (l'ensemble des états d'acceptation de l'automate k) sinon $k' = k$. Les état initiaux sont du type $(s_1, s_2, \ldots, 1)$ où $s_1$ est un état de départ de $L_1$ et $s_2$ un état de départ de $L_2$, … L'ensemble des états d'acceptation F contient tous les états du type $(s_1, s_2, \ldots, N)$ où $s_N \in F_N$. Grâce au compteur k, on sait que lorsqu'on arrive dans un état de F, on est passé auparavant, pour chaque automate i ($i \in [1, N-1]$), par un état $(s_1, s_2, \ldots, k)$ où $s_i \in F_i$. Comme $k = 1$ pour chaque successeur d'un état de F, on sait grâce à ce compteur que si on passe un nombre infini de fois par un état de F, alors pour tout $i \in [1, N]$, on passe un nombre infini de fois par un état $(s_1, s_2, \ldots, k)$ où $s_i \in F_i$. et la composition des N automates de Büchi satisfait les conditions d'acceptations des automates qui le composent.

Notons le cas particulier où on évalue une formule du type $\phi \vee \neg\phi$. Dans ce cas, l'automate local est construit pour une formule du type $\neg\phi \wedge \phi$ et ne contient pas d'états initiaux. L'intersection de cet automate avec un second automate quelconque donne un automate n'ayant également pas d'états initiaux et donc n'acceptant aucun string. On en conclut que le système vérifié satisfait bien la propriété $\phi \vee \neg\phi$, quelle que soit ce système.

L'algorithme présenté ici est une adaptation de l'algorithme pour la logique PLTTL (propositionnal linear time temporal logic) présenté dans [MP92], les logiques PLTTL et LTL ayant les mêmes caractéristiques, c'est à dire que les deux logiques sont des logiques temporelles linéaires non temporisées et orientées états ayant des opérateurs plus ou moins similaires. On a repris les idées de l'algorithme de PLTTL et on les a adaptées à LTL pour créer l'algorithme présenté ici. En fait, seuls les opérateurs temporels diffèrent entre les deux logiques. L'opérateur $\wr$ n'existe pas en PLTTL et l'opérateur $U$ diffère légèrement entre les deux logiques. En effet, une séquence infinie d'états satisfaisant $\phi_1$ et non $\phi_2$ satisfait la formule $\phi_1 \; U \; \phi_2$ en PLTTL mais pas en LTL. La manière de créer l'automate local et l'automate des éventualités présentée ici tient compte de ces différences entre les logiques PLTTL et LTL et leur construction est optimisée, en ne générant pas d'états et de transitions inutiles.

**Algorithme pour la logique CTL ([CES86])**
La vérification d'une formule CTL $\phi$ se fait sur une structure de Kripke (nettoyé des états sans successeurs pour ne tenir compte que des traces infinies) décrivant le système. L'idée de base de l'algorithme de vérification est d'étendre la fonction $\mu$ aux sous-formules de $\phi$. De manière inductive, on calcule la valeur de vérité des sous-formules de $\phi$ dont les valeurs de vérités des sous-formules ont déjà été calculées. Pour chaque sommet, la valeur de vérité des propositions associées au sommet par la fonction $\mu$ est vraie et la valeur de vérité des propositions qui ne sont pas associées au sommet par $\mu$ est fausse. Ainsi, au départ, on recherche la valeur de vérité des sous-formules de $\phi$ dont les sous-formules sont des propositions ou leur négation. Si la valeur de vérité d'une formule est vraie dans un état, alors la formule est rajoutée à l'ensemble des formules associées à l'état par $\mu$. On continue ensuite en calculant la valeur de vérité des sous-formules de $\phi$ ayant comme sous-formules les

formules dont on vient de calculer la valeur de vérité ou une proposition, ou la négation d'une proposition,…
Pour finir, on va rechercher la valeur de vérité de $\phi$. Le système satisfera la formule $\phi$ si $\mu$ associe $\phi$ à tous les états initiaux de la structure de Kripke représentant le système à vérifier.
Pour savoir si une formule $\varphi \in \mu(s)$, c'est à dire si la formule $\varphi$ est satisfaite dans l'état s, il faut faire des vérifications suivant le type de $\varphi$ :

- si $\varphi \equiv \neg\varphi'$, alors $\varphi \in \mu(s)$ ssi $\varphi' \notin \mu(s)$.
- si $\varphi \equiv \varphi_1 \wedge \varphi_2$, alors $\varphi \in \mu(s)$ ssi $\varphi_1 \in \mu(s)$ et $\varphi_2 \in \mu(s)$.
- si $\varphi \equiv \exists\bigcirc\varphi'$, alors $\varphi \in \mu(s)$ ssi $\exists$ s' tel que $(s, s') \in \delta \wedge \varphi' \in \mu(s')$. (il existe un voisin de s satisfaisant $\varphi'$)
- si $\varphi \equiv \forall\bigcirc\varphi'$, alors $\varphi \in \mu(s)$ ssi $\forall$ s' tel que $(s, s') \in \delta \wedge \varphi' \in \mu(s')$. (tous les voisins de s satisfont $\varphi'$)
- si $\varphi \equiv \varphi_1 \ \exists U \ \varphi_2$, alors $\varphi \in \mu(s)$ ssi $\exists\ \sigma = s_0, s_1, \ldots, s_{n-1}, s_n, \ldots$ avec $s = s_0$ et $\varphi_2 \in \mu(s_n)$ et $\forall i = 0..n-1 : \varphi_1 \in \mu(s_i)$. (il existe une séquence d'états partant de s dans laquelle un état satisfait $\varphi_2$ et tous les états le précédant dans la séquence d'états satisfont $\varphi_1$).
- si $\varphi \equiv \varphi_1 \ \forall U \ \varphi_2$, alors $\varphi \in \mu(s)$ ssi $\forall\ \sigma = s_0, s_1, \ldots, s_{n-1}, s_n, \ldots$ avec $s = s_0$, $\exists n$ tel que $\varphi_2 \in \mu(s_n)$ et $\forall i = 0..n-1 : \varphi_1 \in \mu(s_i)$. (toutes les séquences d'états partant de s sont telles qu'il existe un état satisfaisant $\varphi_2$ et tous les états le précédant dans la séquences d'états satisfont $\varphi_1$)

En pratique, pour $\varphi \equiv \varphi_1 \ \exists U \ \varphi_2$ et $\varphi \equiv \varphi_1 \ \forall U \ \varphi_2$, on peut écrire un algorithme qui parcourt au plus une fois le graphe donné par la structure de Kripke et qui calcule pour tous les états $s \in S$ si $\varphi \in \mu(s)$.

- Pour $\varphi \equiv \varphi_1 \ \exists U \ \varphi_2$, on peut par exemple marquer chaque état s où $\varphi_2 \in \mu(s)$ comme satisfaisant $\varphi$ et marquer tous les états s' accessibles à partir de s par un chemin inverse dont tous les états intermédiaires (s' y compris) satisfont $\varphi_1$, comme état satisfaisant $\varphi$.
- Pour $\varphi \equiv \varphi_1 \ \forall U \ \varphi_2$, on peut par exemple faire un depth first. Partant d'un état non encore traité, on arrête l'exploration d'un chemin soit lorsqu'on accède à un état où ni $\phi_1$, ni $\phi_2$ est satisfait, soit lorsqu'on s'aperçoit de l'existence d'un cycle, soit lorsqu'on accède à un état satisfaisant $\phi_2$. Dans les premier et second cas, la formule est fausse pour tous les états du chemin en cours d'exploration. Ceux-ci sont marqués comme tel et traités. Dans le second cas, on revient à l'état précédant le dernier état visité et on continue le parcours en profondeur. La formule sera vraie pour un état s'il est possible, partant de cet état, d'effectuer le parcours, en accédant toujours à un état satisfaisant $\phi_2$ avant d'accéder à un état où ni $\phi_1$, ni $\phi_2$ ne sont satisfaits ou de se rendre compte de l'existence d'un cycle.

On a remarqué qu'il n'est pas possible d'utiliser ce genre d'algorithme pour l'évaluation de formules de logiques linéaires. En effet, pour ces logiques, les formules sont vérifiées, pour un ensemble de traces, sur chaque trace de manière indépendante par rapport aux autres alors que l'évaluation de formules en parcourant un graphe ne permet pas d'évaluer une formule sur une trace isolée.
Par exemple, en LTL, on pourrait essayer d'évaluer la formule $\phi \equiv \text{true}\ U\ \neg(\text{true}\ U\ d)$ sur la structure de Kripke K représentée par la figure 18 en utilisant un algorithme de parcours de graphe similaire à celui de CTL présenté ici. On pourrait ainsi définir $\phi_1\ U\ \phi_2 \in \mu(s)$ ssi tous les chemins partant de s dans le graphe permettent d'atteindre un état satisfaisant $\phi_2$, avec les états intermédiaires satisfaisant $\phi_1$. On remarque qu'il est possible dans K d'atteindre un état satisfaisant d de l'état $s_0$, $s_1$ et $s_2$. $\mu$ n'associerait donc pas la sous-formule $\neg(\text{true}\ U\ d)$ à ces trois états. Partant de l'état initial, il serait donc possible de ne jamais atteindre un état labellé

par ¬(true *U* d) et ainsi μ n'associerait pas φ à l'état initial, c'est à dire que l'algorithme évaluerait que φ n'est pas satisfait par K, or K satisfait bien φ.
Notons que l'évaluation de la négation ne peut se faire comme dans le cas de CTL, en effet, ce n'est pas parce que toutes les traces partant d'un état ne satisfont pas une formule qu'aucune de ces traces ne satisfait cette formule. On suppose donc ici que la négation est évaluée de manière différente (en faisant un parcours de graphe par exemple).

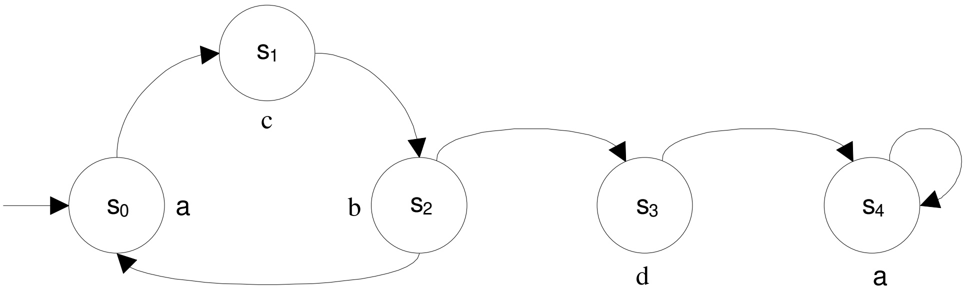
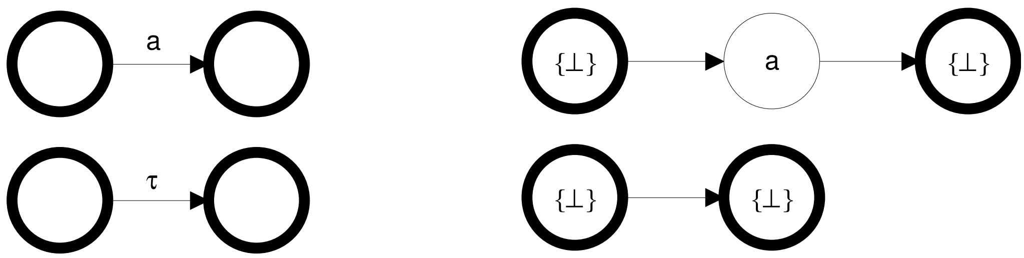



Figure 18

**Algorithme pour la logique ACTL ([NFGR91])**
La structure utilisée pour la vérification de formules ACTL est un système de transitions labellées. Sont considérée comme valides les séquences d'actions infinies du LTS ainsi que les séquences d'actions finies qui aboutissent à un états sans successeur. De plus, l'ensemble des états initiaux du LTS est restreint à un seul état.

L'algorithme consiste à prendre la formule ACTL à vérifier, la transformer en formule CTL, prendre le système de transitions labellées, le transformer en structure de Kripke, puis appliquer l'algorithme de Model Checking pour CTL.
Pour cela, on définit deux fonctions :

- $f_g$ : prend un système de transitions labellées et renvoie une structure de Kripke. La fonction transforme un modèle ayant des transitions labellées par des actions en un modèle où les états sont labellés par des propositions. Les deux modèles représentent le même système, ont le "même" comportement.
- $f_f$ : prend une formule ACTL et renvoie une formule CTL. Cette transformation se fait en fonction de la manière dont se fait la transformation du modèle. De cette manière, la formule CTL représente la même propriété à vérifier sur le modèle que la formule ACTL de départ.




a)Système de transitions labellées b)Structure de Kripke

Figure 19: Modification du modèle

Définissons ces deux fonctions :

- Comme l'idée est de se ramener au Model Checking CTL, les actions effectuées doivent être spécifiées aux niveaux des états du modèle. La transformation qu'effectue $f_g$ est de remplacer chaque transition (s, a, s') par un état <s, a, s'> et deux transitions (s, <s, a, s'>) et (<s, a, s'>, s') avec s et s' labellé par $\{\perp\}$. Pour chaque transition (s, $\tau$, s'), on labelle simplement s et s' avec $\{\perp\}$. La figure 19 montre cette transformation. Les états du modèle de départ sont représentés avec un bord épais. Le fait de labeller les états qui se trouvaient déjà dans le modèle de départ permet de pouvoir distinguer ces états par rapport aux états rajoutés.

  Formellement, soit $\Phi = (S, \delta, s_0)$ un LTS, $S_d$ l'ensemble des états de S sans successeurs, $\perp$ est le symbole vide et on suppose que $\perp \notin L$. La structure de Kripke $f_g(\Phi)$ est définie comme $(S', \delta', s_0', \mu')$ où :
  - $S' = S \cup \{<r, a, s> \text{ tel } (r, a, s) \in \delta\} \cup N$ avec $N = \{s_f\}$ (avec $s_f \notin S$) si $S_d \neq \emptyset$, $N = \emptyset$ sinon. Un système de transitions labellés peut accepter des séquences d'actions finies alors que toutes les séquences d'états d'une structure de Kripke sont infinies. Pour remédier à ce problème, si le LTS accepte effectivement des séquences d'actions finies, alors on rajoute un état $s_f$. Cet état permettra de faire correspondre des séquences d'états infinies à ces séquences d'actions finies.
  - $\delta' = \{(r, s) \text{ tel que } (r, \tau, s) \in \delta\} \cup \{(r, (r, a, s)) \text{ tel que } (r, a, s) \in \delta\} \cup \{((r, a, s), s) \text{ tel que } (r, a, s) \in \delta\} \cup T$, où $T = \{(t, s_f) \text{ tel que } t \in S_d\} \cup \{(s_f, s_f) \text{ si } S_d \neq \emptyset\}$ ($T = \emptyset$ si $S_d = \emptyset$). Si des états du LTS $\Phi$ n'ont pas de successeurs, cela signifie que les séquences d'actions contenant ces états sont finies. Pour associer de telles séquences d'actions à des séquences d'états infinies de la structure de Kripke, on rajoute une transition partant de chaque état de $S_d$ et aboutissant à l'état $s_f$ et une transition partant et aboutissant dans $s_f$.
  - $s_0' = s_0$.
  - Pour $r, s \in S$ : $\mu(s) = \{\perp\}$, si $\exists(r, a, s) \in \delta : \mu(<r, a, s>) = \{a\}$.
  - $\mu(s_f) = \{\perp\}$

- $f_f$ est un mapping qui se définit de manière inductive comme suit :

  $f_f(\text{true}) = \text{true}$
  $f_f(\neg\varphi) = \neg f_f(\varphi)$
  $f_f(\varphi \wedge \varphi') = f_f(\varphi) \wedge f_f(\varphi')$
  $f_f(\varphi\ {}_{\chi}\exists U_{\chi'}\ \varphi') = (((\perp \wedge f_f(\varphi)) \vee (\neg\perp \wedge \chi))\ \exists U\ ((\neg\perp \wedge \chi') \wedge \exists\bigcirc(\perp \wedge f_f(\varphi'))))$
  $f_f(\varphi\ {}_{\chi}\exists U\ \varphi') = (((\perp \wedge f_f(\varphi)) \vee (\neg\perp \wedge \chi))\ \exists U\ (\perp \wedge f_f(\varphi')))$
  $f_f(\exists\bigcirc_{\chi}\varphi) = \exists\bigcirc(\neg\perp \wedge \chi \wedge \exists\bigcirc(f_f(\varphi)))$
  $f_f(\exists\bigcirc_{\tau}\varphi) = \perp \wedge \exists\bigcirc(\perp \wedge f_f(\varphi))$
  $f_f(\varphi\ {}_{\chi}\forall U_{\chi'}\ \varphi') = (((\perp \wedge f_f(\varphi)) \vee (\neg\perp \wedge \chi))\ \forall U\ ((\neg\perp \wedge \chi') \wedge \forall\bigcirc(\perp \wedge f_f(\varphi'))))$
  $f_f(\varphi\ {}_{\chi}\forall U\ \varphi') = (((\perp \wedge f_f(\varphi)) \vee (\neg\perp \wedge \chi))\ \forall U\ (\perp \wedge f_f(\varphi')))$
  $f_f(\forall\bigcirc_{\chi}\varphi) = \forall\bigcirc(\neg\perp \wedge \chi \wedge \forall\bigcirc(f_f(\varphi)))$
  $f_f(\forall\bigcirc_{\tau}\varphi) = \perp \wedge \forall\bigcirc(\perp \wedge f_f(\varphi))$

  On transforme la formule en tenant compte de ce que les actions ne sont plus spécifiées aux niveaux des transitions mais des états. Une formule ACTL exprimant le fait que l'action a d'une transition effectuée doit satisfaire une propriété $\chi$ devient une formule CTL exprimant le fait qu'on doit passer par un état dont la proposition a doit satisfaire une propriété $\chi$. Pour les opérateurs temporels, il faudra donc distinguer dans une séquence d'états les états représentant le fait qu'une action est effectuée et les états se trouvant déjà dans le LTS de départ. Les états du premier type devront satisfaire les conditions posées sur les actions dans les formules ACTL et les états du second type devront satisfaire les propriétés demandées pour les états du LTS de départ.

La fonction $f_f$ a été simplifiée par rapport à [NFGR91]. Par exemple, la formule ACTL $\varphi\ _{\chi}\forall U\ \varphi'$ est transformée dans [NFGR91] en la formule CTL $(\perp \wedge f_f(\varphi')) \vee (((\perp \wedge f_f(\varphi)) \vee (\neg\perp \wedge \chi))\ \forall U\ (\perp \wedge f_f(\varphi')))$. Or, par définition de la sémantique des opérateurs de CTL, les états satisfaisant une formule $\phi_2$ font partie de l'ensemble caractéristique des formules du type $\phi_1\ \forall U\ \phi_2$ (et $\phi_1\ \exists U\ \phi_2$). Ainsi, les états satisfaisant $(\perp \wedge f_f(\varphi'))$ feront partie de l'ensemble caractéristique de la formule $((\perp \wedge f_f(\varphi)) \vee (\neg\perp \wedge \chi))\ \forall U\ (\perp \wedge f_f(\varphi'))$ et la disjonction entre cette formule et $(\perp \wedge f_f(\varphi'))$ est donc inutile.

**Algorithme pour la logique TCTL ([ALU91])**

Etant donné la densité du temps, un algorithme de vérification pour la logique TCTL doit utiliser une approche symbolique, plutôt qu'énumérative, c'est à dire que l'algorithme ne va pas énumérer tous les états pour lesquels une formule $\phi$ est vraie, mais va les représenter à l'aide de contraintes et de sommets.

Ce qui suit décrit brièvement le premier algorithme de Model checking symbolique pour TCTL, proposé par Alur dans [ALU91].

L'algorithme vérifie une formule TCTL $\phi$, sur une structure temporisée $T = (S, s_{init}, \mu, H, \Delta, \delta)$.

Pour vérifier cette formule $\phi$, l'idée de base de l'algorithme est qu'on peut trouver une relation d'équivalence entre les valuations. Cette relation permet de définir des ensembles d'états, appelés classes d'équivalences, où les valeurs de vérités des sous-formules de $\phi$ sont les même pour tous les états. Ceci repose sur l'idée que les invariants d'états et les contraintes sur les horloges dans une formule TCTL ne font intervenir que des constantes naturelles alors que les horloges prennent leurs valeurs dans $R^+$. Ainsi, pour une horloge x de T et la contrainte $x \leq 2 \wedge x \geq 1$, le fait qu'une valuation d'horloges assigne les valeurs 1.2 ou 1.3 n'interférera pas sur la valeur de vérité de la contrainte. Pour toutes les valuations donnant une valeur à x dans l'intervalle [1, 2], la valeur de vérité de la contrainte est vraie. L'idée est donc de découper, pour chaque horloge de T, $R^+$ en intervalle d'une unité de temps. Comme il est également possible de spécifier des contraintes du type x = 2, les entiers vont faire l'objet d'intervalle spécifique. Pour chaque horloge x de T, on obtient une découpe

$[0, 0]\ \ ]0, 1[\ \ [1, 1]\ \ ]1,2[\ \dots\ ]c_x - 1, c_x[\ \ [c_x\ c_x]\ \ ]\ c_x, \infty[$

où $c_x$ est le plus grand naturel utilisé dans une contrainte sur x dans un invariant d'un sommet de T. En effet, une horloge x ayant dépassé cette valeur $c_x$ peut vieillir autant qu'on veut, cela ne modifiera pas les valeurs de vérités de toutes les contraintes sur x spécifiées au niveau des sommets de T. Si on a deux contraintes $x \leq 3$ et $x > 2$ alors $c_x = 3$. Effectivement, lorsque x dépasse la valeur 3, elle peut vieillir de n'importe quel valeur, $x \leq 3$ sera toujours faux et $x > 2$ sera toujours vrai. Avec cette découpe de $R^+$ pour chaque horloge, on sait que des valuations donnant une valeur à x dans un même intervalle donne la même valeur de vérité aux contraintes sur x dans T.

Pour chaque horloge x utilisée dans $\phi$, on fait une même découpe de $R^+$ où $c_x$ est cette fois-ci le plus grand écart autorisé par une sous-formule de $\phi$ entre la variables x et une autre variable. A priori, une formule du type $x + c \leq y + d$ ne demande pas de savoir la valeur exacte de x et y mais demande simplement que les valeurs des deux horloges respectent cette relation. Mais couplé à l'opérateur de remise à zéro d'horloges, comme pour les horloges de T, il faut savoir dans quel intervalle se trouvent les variables pour connaître la valeur de vérité d'une formule du type $x + c \leq y + d$ si x ou y est remis à zéro. Si on a la contrainte $x \leq y + 2$, avec x et y dans l'intervalle ]1, 2[, lors de la remise à zéro de y, on sait que la contrainte sera encore vérifiée car on sait que x ne peut être plus grand que y de deux unités. Si x et y étaient plus grand que 2 alors quelle que soit la valeur de x, la contrainte n'aurait plus été vérifiée après la mise à zéro de y.

Nous définissons maintenant de manière plus formelle cette relation d'équivalence entre les valuations.

Soit $H_\phi$, l'ensemble des variables apparaissant dans la formule $\phi$ à vérifier.
Pour $x \in H$, $c_x$ est la plus grande constante d tel que $x \leq d$ ou $x \geq d$ est une sous formule d'un invariant de sommet de T. Pour $x \in H_\phi$, soit $c_x$ la plus grande valeur $| c - d |$ tel que $(x + c) \leq (y + d)$ ou $(y + c) \leq (x + d)$ est une sous formule de $\phi$, $| z |$ représentant la valeur absolue de z.

*fract*(z) donne la partie fractionnaire du réel z et $H' = H \cup H_\phi$.

Prenant deux valuations $\nu$ et $\nu'$ sur H', $\nu \sim \nu'$ ssi toutes les conditions suivantes sont vérifiées :

- Pour chaque $x \in H'$, soit $\lfloor\nu(x)\rfloor$ et $\lfloor\nu'(x)\rfloor$ sont les mêmes, soit les valeurs $\nu(x)$ et $\nu'(x)$ sont plus grandes que $c_x$.
- Pour chaque $x, y \in H'$ tel que $\nu(x) \leq c_x$ et $\nu(y) \leq c_y$, *fract*$(\nu(x)) \leq$ *fract*$(\nu(y))$ ssi *fract*$(\nu'(x)) \leq$ *fract*$(\nu'(y))$
- Pour chaque $x \in H'$ tel que $\nu(x) \leq c_x$, *fract*$(\nu(x)) = 0$ ssi *fract*$(\nu'(x)) = 0$
- Pour une sous formule $(x + c) \leq (y + d)$ de $\phi$, $(\nu(x) + c) \leq (\nu(y) + d)$ ssi $(\nu'(x) + c) \leq (\nu'(y) + d)$

La première condition impose que les valeurs données aux horloges de deux valuations satisfaisant la relation d'équivalence entre elles doivent avoir la même partie entière.
La deuxième condition impose un ordre sur les parties fractionnaires des valeurs des horloges.
La troisième condition impose que si une valuation donne une valeur entière à une horloge alors les valuations qui satisfont la relation d'équivalence avec celle-ci devront également assigner une valeur entière à cette même horloge.
Ces trois conditions ne doivent être satisfaites que pour des horloges x pour lesquelles des valeurs plus petites que $c_x$ ont été assignées.
En suivant les trois premières conditions, si on a
$H' = \{x, y, z\}$
$C_x = 3$, $C_y = 7$, $C_z = 10$
$\nu$ tel que $\nu(x) = 2.1$, $\nu(y) = 4.2$ et $\nu(z) = 5$
$\nu'$ tel que $\nu'(x) = 2.3$, $\nu'(y) = 4.4$ et $\nu'(z) = 5$
alors $v \sim v'$.
Supposons maintenant qu'on a également s et t dans H' et que ces deux horloges sont utilisées dans la formule $(s + 2) \leq t$ à vérifier. En ne suivant toujours que les trois premières conditions, on pourrait encore avoir $\nu \sim \nu'$ avec $\nu$ et $\nu'$ qui assignent des valeurs à s et t dépassant $c_s$ et $c_t$, telles que pour $\nu$ la sous-formule soit vraie et pour $\nu'$ la formule soit fausse. Pour éviter cela, on impose la quatrième condition.

Une région d'horloges est une classe d'équivalence de valuation induite par ~. Une région d'horloges est caractérisée par des contraintes déterminant les valeurs possibles pour chaque horloge.
Une région d'horloges est appelée région de fin si $x > c_x$ pour tout $x \in H'$.
Une région limite est une région d'horloges satisfaisant $x = c$ pour au moins un $x \in H'$ et pour au moins un $c \leq c_x$ (avec $c \in \angle$).

Soit $\alpha$ et $\beta$ deux régions d'horloges distinctes, $succ(\alpha) = \beta$ ssi pour chaque $\nu \in \alpha$, il existe un $t \in R^+$ différent de zéro tel que pour tout $t' < t$, $\nu + t' \in \alpha \cup \beta$. La région successeur est définie pour toutes les régions sauf pour la région de fin.

La région $\alpha[H' \to 0]$ est la région $\alpha' = \{\cup_{\nu \in \alpha} \nu[H' \to 0]\}$.

L'algorithme va créer un "graphe des régions" à partir de la structure temporisée T.

L'ensemble des états du graphe des régions est constitué de l'ensemble des couples $<s, \alpha>$ où $s \in S$ et $\alpha$ est une région d'horloges dont les valuations satisfaisont $\Delta(s)$.

Il y a deux types de transitions :

- les transitions liées à une action.
  Nous considérons deux cas :
  - Si nous avons $<s, \alpha>$ où $\alpha$ n'est pas une région limite. Pour chaque transition $s \to s'$ labellée par l'ensemble d'horloges H', nous aurons trois transitions du type $<s, \alpha>$ vers $<s', \alpha [H' \to 0]>$, $<s', succ(\alpha [H' \to 0])>$ et $<s', succ( \alpha ) [H' \to 0]>$. Le premier type de transition représente une transition effectuée après un intervalle de temps $|t, t'[$ ($| \in \{[, ]\}$) passé dans s. Dans ce cas on fait d'abord le changement de sommet puis les remises à zéros des horloges et on se retrouve dans s' avec les horloges de H' à zéro. Le second type de transition représente une transition effectuée après un intervalle de temps $|t, t']$ passé dans s. Dans ce cas, la remise à zéro des horloges est d'abord effectuée, puis le changement de sommet et on se retrouve dans le sommet s' avec les horloges de H' ayant une valeur juste supérieure à zéro. Enfin, le troisième type de transition représente une transition effectuée en même temps que le changement de région d'horloges. Cette transition étant faite après avoir passé un intervalle de temps $|t, t'[$ ouvert à droite dans le sommet s. Dans ce cas, comme dans le premier cas, on effectue d'abord le changement de sommet, ainsi que le changement de région d'horloges, puis la remise à zéro. Dans le cas d'un intervalle fermé, on se rapporte aux transitions du second type.
  - Si nous avons $<s, \alpha>$ où $\alpha$ est une région limite. Pour chaque transition $s \to s'$ labellée par l'ensemble d'horloges H', nous aurons une transition de $<s, \alpha>$ vers $<s', succ(\alpha [H' \to 0])>$. On distingue ce cas-ci du cas précédent car ici, on a toujours un intervalle fermé à droite et la transition est faite en même temps qu'un changement de région.

- les transitions liées à l'écoulement du temps et le passage d'une région d'horloges à une autre. Pour un état s, nous avons une transition de $<s, \alpha>$ vers $<s, succ(\alpha)>$.

Les transitions existent effectivement si les régions d'arrivées sont des régions légales, c'est à dire que la région d'horloges satisfait l'invariant du sommet, et lors du calcul de succ($\alpha$), $\alpha$ ne peut être la région de fin.

Le graphe des régions n'est pas construit de manière générale mais par rapport à une formule $\phi$. En effet, les régions sont trouvées par rapport aux valuations $\nu$ qui donnent les valeurs pour les variables d'une formule $\phi$ à vérifier. Il est donc nécessaire de créer ce graphe des régions pour chaques formules $\phi$ à vérifier.

La construction du graphe des régions permet d'analyser, à partir de ce graphe, un système temporisés en oubliant ses aspects temporels. On évalue une formule TCTL de la même manière que si c'était une formule CTL, c'est à dire que l'évaluation consiste à associer, grâce à une fonction $\mu$, des formules aux états du graphe des régions. De manière inductive, on évalue une formule lorsque ses sous-formules ont été évaluées. Si une formule est satisfaite par un état, la fonction $\mu$ l'associe à l'état. Néanmoins, deux nouveaux types de formules ont été rajoutés dans la logique TCTL par rapport à CTL.
Lorsqu'on veut vérifier si une formule $\phi$ est satisfaite dans un état, celle-ci peut être de six types possibles:

- si $\varphi \in \mathtt{P}$, alors $\varphi \in \mu(<s, \alpha>)$ ssi $\varphi \in \mu(s)$.
- si $\varphi \equiv \neg\varphi'$, alors $\varphi \in \mu(<s, \alpha>)$ ssi $\varphi' \notin \mu(<s, \alpha>)$.
- si $\varphi \equiv \varphi_1 \wedge \varphi_2$, alors $\varphi \in \mu(<s, \alpha>)$ ssi $\varphi_1 \in \mu(<s, \alpha>)$ et $\varphi_2 \in \mu(<s, \alpha>)$.
- si $\varphi \equiv x.\varphi'$, alors $\varphi \in \mu(<s, \alpha>)$ ssi $\varphi' \in \mu(<s, \alpha\,[x \rightarrow 0]>)$.
- $\varphi \equiv (x + c) \leq (y + d)$, alors $\varphi \in \mu(<s, \alpha>)$ ssi $\alpha \wedge x + c \leq y + d \neq$ false.
- si $\varphi \equiv \varphi_1 \; \exists U \; \varphi_2$, alors $\varphi \in \mu(<s, \alpha>)$ ssi $\exists$ une trace $\sigma = <s_0, \alpha_0>, <s_1, \alpha_1>, \ldots, <s_{n-1}, \alpha_{n-1}>, <s_n, \alpha_n>, \ldots$ avec $<s, \alpha> = <s_0, \alpha_0>$, $\exists n$ tel que $\varphi_2 \in \mu(<s_n, \alpha_n>)$ et $\forall i = 0 \ldots n - 1 : \varphi_1 \in \mu(<s_i, \alpha_i>)$.
- si $\varphi \equiv \varphi_1 \; \forall U \; \varphi_2$, alors $\varphi \in \mu(<s, \alpha>)$ ssi $\forall$ trace $\sigma = <s_0, \alpha_0>, <s_1, \alpha_1>, \ldots, <s_{n-1}, \alpha_{n-1}>, <s_n, \alpha_n>, \ldots$ avec $<s, \alpha> = <s_0, \alpha_0>$, $\exists n$ tel que $\varphi_2 \in \mu(<s_n, \alpha_n>)$ et $\forall i = 0 \ldots n - 1 : \varphi_1 \in \mu(<s_i, \alpha_i>)$.

Notons que lors de l'évaluation des formules $\phi_1 \; \exists U \; \phi_2$ et $\phi_1 \; \forall U \; \phi_2$, il n'est pas demandé que les régions traversées pour atteindre une région satisfaisant $\phi_2$ satisfassent $\phi_1$ ou $\phi_2$, mais simplement $\phi_1$. En effet, l'évaluation ne se fait plus directement au niveau des états mais au niveau des régions (des ensembles d'états) et ainsi la densité du temps ne pose plus problème. Traverser une suite de régions satisfaisant $\phi_1$, puis une région satisfaisant $\phi_2$ correspond à un ensemble d'exécutions de la structure temporisée de départ où $\phi_1$ est continuellement vérifié jusqu'à ce que $\phi_2$ le soit.

En pratique la mise en œuvre d'algorithmes pour les formules du type $\varphi \equiv \varphi_1 \; \forall U \; \varphi_2$ et $\varphi \equiv \varphi_1 \; \exists U \; \varphi_2$ sont les mêmes que ceux mis en œuvre dans le cadre de la logique CTL.

L'algorithme présenté ici diffère légèrement de celui présenté dans [Alu91]. En effet, les régions limites y sont marquées par une proposition spéciale $p_r$. Pour qu'une formule $\varphi \equiv \varphi_1 \exists U \varphi_2$ (ou $\varphi_1 \forall U \varphi_2$) soit satisfaite dans $<s, \alpha>$, Alur demande qu'il existe une trace (toutes les traces) $\sigma = <s_0, \alpha_0>, <s_1, \alpha_1>, \ldots, <s_{n-1}, \alpha_{n-1}>, <s_n, \alpha_n>, \ldots$ avec $<s, \alpha> = <s_0, \alpha_0>$ pour laquelle $\exists n$ tel que ($\varphi_2 \in \mu(<s_n, \alpha_n>)$ et( $\varphi_1$ ou $p_r \in \mu(<s_n, \alpha_n>)$)) et $\forall i = 0..n-1$ : $\varphi_1 \in \mu(<s_i, \alpha_i>)$. Ceci est faux. En effet, une formule $\varphi \equiv \varphi_1 \exists U \varphi_2$ (ou $\varphi_1 \forall U \varphi_2$) pourrait ainsi être considérée comme non satisfaite par l'algorithme parce que les états satisfaisant $\varphi_2$ ne satisfont pas $\varphi_1$, ce qui n'est aucunement justifié d'après la sémantique de TCTL.
De plus, la définition des valeurs $c_x$ diffère de celle donnée par Alur : Pour une horloge x de T, Alur ne tient compte que des sous-formules $x \leq d$ d'une contrainte associée à un sommet par $\Delta$. Ce qui est incomplet puisque les sous formules du type $x \geq d$ ne sont pas prises en compte.
Si nous avions une horloge x et deux contraintes $x \leq 2$ et $x \geq 3$ sur les sommets, $c_x = 2$ pour Alur. On pourrait avoir deux valuations $\nu$ et $\nu'$ donnant une valeur pour x plus grande que $c_x$, par exemple 2.5 pour $\nu$ et 7 pour $\nu''$. On aurait dans ce cas une valeur de vérité de $x \geq 3$ différente pour $\nu$ et $\nu'$ alors que $\nu \sim \nu''$, ce qui est contradictoire avec l'idée des classes d'équivalence.
De plus, pour une horloge x de la formule, Alur définit la valeur $c_x$ comme étant la plus grande valeur d tel que $(x + c) \leq (y + d)$ ou $(x + d) \leq (y + c)$ est une sous-formule de la formule à vérifier. L'idée de prendre la plus grande constante d n'est pas fausse puisque le plus grand écart autorisé par la formule pour chaque horloge x est plus petit ou égale à cette valeur d. Mais cela augmente le nombre de régions d'horloges, impliquant une augmentation du nombre d'états du graphe des régions. Mais tel que marqué dans [Alu91], Alur ne tient pas compte des formules du type $(y + c) \leq (x + d)$ et $(y + d) \leq (x + c)$, ce qui pose les mêmes problèmes que dans le cas des horloges de T.
Rappelons finalement que la logique TCTL utilisée ici est légèrement différente de celle utilisées dans [Alu91]. La présentation de l'algorithme en tient compte.

## 2. Problème lié à l'algorithme de Model Checking symbolique TCTL

Dans le cadre de l'implantation d'un algorithme de Model Checking permettant de vérifier des propriétés de systèmes temporisés, l'algorithme de Model Checking TCTL présenté dans ce chapitre aurait pu être intéressant. Malheureusement, le graphe des régions généré par cet algorithme est en général beaucoup plus grand que la structure temporisée de départ. Ceci entraîne une augmentation du besoin de mémoire par l'algorithme et également une augmentation du temps d'exécution. Or le Model Checking est un problème lourd à résoudre et pour cela un algorithme ne créant pas ce graphe des régions pourrait être intéressant. Un tel algorithme utilisant des états symboliques plus élaborés est présenté au chapitre suivant.

# Chapitre 5

# Model Checking symbolique pour TCTL

Nous présentons de manière plus détaillée dans ce chapitre un algorithme de Model Checking symbolique pour la logique TCTL tel que défini dans [Yov93] et [Oli94]. Des différences existent entre les algorithmes présentés dans ces deux thèses. Ce qui est présenté ici est une fusion de ces deux travaux. De plus, certaines corrections ont été effectuées lorsque cela s'avérait nécessaire. L'implantation de cet algorithme a été effectuée dans le cadre de ce travail, bien que TCTL soit une logique orientée états. Aucune logique temporelle temporisée orientée action n'a été trouvée. Le souhait de pouvoir spécifier des propriétés en terme d'actions n'est donc pas réalisé à ce stade. On montrera au chapitre 7 la manière dont ce problème a été résolu.

## 1. Définition préliminaire

Avant de présenter l'algorithme de Model Checking, un certain nombre de définitions utiles à la compréhension de ce qui suit doivent être données.

Un état symbolique est un binôme $(s, \varphi)$ où $s$ est un sommet et $\varphi \in \Psi(H)$ (où $\Psi(H)$ est l'ensemble des contraintes possibles sur les horloges). Un état symbolique représente l'ensemble des états ayant tous le même sommet et dont la valuation assigne des valeurs aux horloges qui satisfont la contrainte $\varphi$. Pour un graphe temporisé $G = (S, H, \delta, s_{init}, \Delta)$, si $\varphi \Rightarrow \Delta(s)$ alors l'état symbolique représente un ensemble d'états de G. On considère dans ce qui suit que cette implication sera toujours vérifiée.

La contrainte $\varphi + t$ est la contrainte obtenue à partir de $\varphi$ où pour tout $x \in H$, toutes les occurrences de x ont été substituées par $x + t$. Par exemple, si $\varphi = x < 2 \wedge y \geq 3$ alors
$\varphi + t = x < 2 - t \wedge y \geq 3 - t$.

Pour une contrainte $\varphi$ et un ensemble d'horloges H', $\varphi$ [H' $\rightarrow$ 0] est la contrainte $\varphi$ où pour tout $x \in$ H' on a remplacé toutes les occurrences de x par 0. Cette opération est appelée une initialisation d'horloges à zéro pour une contrainte.
Contrairement à ce qu'il pourrait sembler, le but de cet opérateur n'est pas de spécifier dans la contrainte que les horloges de H' valent zéro. Ayant un état symbolique $<s, \varphi>$, on aimerait connaître les états d'un graphe temporisé G qui atteignent un état de l'ensemble représenté par $<s, \varphi>$ en effectuant une transition e. Les valeurs des horloges sont les mêmes avant et après une transition, sauf pour les horloges d'un ensemble H' remis à zéros. La valeur de ces horloges avant d'effectuer e n'a que peu d'importance pour cette raison. Par contre, $\varphi$ donne des relations qui doivent être satisfaites par les valeurs de toutes les horloges après transition. Partant d'un état $<s', \nu>$ et effectuant e pour arriver dans un état représenté par l'état symbolique $<s, \varphi>$, il faut que les valeurs données aux horloges par $\nu$ satisfassent des contraintes telles que $\nu$ [H' $\rightarrow$ 0] satisfasse $\varphi$. Ce sont ces contraintes qui sont données par l'initialisation d'horloges $\varphi$ [H' $\rightarrow$ 0]. L'utilité de cet opérateur vient du fait que l'algorithme présenté dans ce chapitre parcourt le graphe représentant le modèle en sens inverse.

Nous définissons les opérateurs suivant sur des ensembles d'états symboliques comme suit :
$\cup_{s\in S}$ (s, $\varphi_s$) $\wedge$ $\cup_{s\in S}$ (s, $\varphi'_s$) = $\cup_{s\in S}$ (s, $\varphi_s \wedge \varphi'_s$)
$\cup_{s\in S}$ (s, $\varphi_s$) $\vee$ $\cup_{s\in S}$ (s, $\varphi'_s$) = $\cup_{s\in S}$ (s, $\varphi_s \vee \varphi'_s$)
$\neg$ $\cup_{s\in S}$ (s, $\varphi_s$) = $\cup_{s\in S}$ (s, $\neg\varphi_s \wedge \Delta(s)$)
($\cup_{s\in S}$ (s, $\varphi_s$) [x $\rightarrow$ 0]) = $\cup_{s\in S}$ (s, $\varphi_s$ [x $\rightarrow$ 0] $\wedge$ $\Delta(s)$)
où $\Delta(s)$ est l'invariant du sommet s.
Remarquons que comme $\varphi_s \Rightarrow \Delta(s)$ et $\varphi'_s \Rightarrow \Delta(s)$, on a $\varphi_s \wedge \varphi'_s \Rightarrow \Delta(s)$ et $\varphi_s \vee \varphi'_s \Rightarrow \Delta(s)$. Il ne faut donc pas rajouter explicitement que les invariants de sommet doivent être satisfait par les valuations des états de l'ensemble solution pour les opérateurs $\wedge$ et $\vee$.

La logique TCTL utilisée dans ce chapitre, bien que très proche, est différente de la logique TCTL présentée au chapitre 3. Nous donnons pour cela une nouvelle définition de TCTL. Lorsqu'on parlera de TCTL dans les chapitres suivant, on fera référence à la logique présentée ici.
Une formule est une formule de la logique TCTL ssi elle peut être construite à partir de la grammaire

$$\phi \rightarrow p \mid \neg\phi \mid x \text{ op } c \mid x - y \text{ op } c \mid \phi \wedge \phi \mid \phi\, \exists U\, \phi \mid \phi\, \forall U\, \phi \mid x.\phi$$

$$\text{op} \rightarrow < \mid \leq$$

où $c \in 9$.

Notons que des contraintes comme $x \leq -2$ ou $x < 0$ peuvent être spécifiées alors qu'elles n'ont pas beaucoup de sens puisqu'une horloge est de domaine $3^+$. Ces contraintes dénuées de sens ne sont pas interdites par la syntaxe car la contrainte x op c est équivalente à la contrainte y.(x – y op c) et jouer sur la constante c de x op c ne permet pas de les interdire complètement. C'est à cause de cette difficulté à interdire ces contraintes dénuées de sens qu'elles sont autorisées par la syntaxe.

Par rapport à la logique TCTL présentée au chapitre 3, les formules x op c et x – y op c' (op $\in$ {<, $\leq$}) ont remplacé la formule $(x + c) \leq (y + d)$. Comme ces formules représentent toutes des contraintes sur les valeurs réelles d'horloges et que les opérateurs x. et $\neg$ sont les mêmes dans les deux logiques, on peut montrer que les ensembles des contraintes sur les horloges exprimables dans les deux logiques sont les mêmes. En effet, on a

$x - y \leq c' \equiv (x + c) \leq (y + d)$
$x - y < c' \equiv \neg((y + d) \leq (x + c))$
$x \leq c' \equiv y.((x + c) \leq (y + d))$
$x < c' \equiv \neg(y.((y + d) \leq (x + c)))$
avec $c' = d - c$.

Notons également que x et y ne sont pas des horloges d'un modèle mais des horloges de la logique, utilisées pour spécifier des contraintes de temps dans la formule.

On va présenter un algorithme de Model Checking TCTL où les formules sont évaluées sur des graphes temporisés. Or, de nouveau, il n'est pas évident de montrer que l'évaluation d'une formule TCTL sur un graphe temporisé correspond à la vérification de cette formule sur des arbres temporisés. On définit pour cela, comme au chapitre 3, la sémantique de TCTL directement par rapport à un graphe temporisé.

De nouveau, on considère ici qu'une valuation donne également une valeur aux horloges de la logique.

Soit un graphe temporisé G ; on définit un pas $q \succ_t q'$ de durée t ($t \in 3^+$) où $q = $ <s, ν> et $q'$ = <s', ν'> comme un écoulement de t unités de temps à partir de l'état <s, ν>, c'est à dire le passage par tous les états <s, ν> + t' avec $t' \leq t$, suivi éventuellement d'une transition liée à une action (y compris l'action invisible $\tau$) . L'état finalement atteint est q'.
Il est à noter que le vieillissement peut être nul. Si aucune action n'est effectuée, nous avons le pas $q \succ_0 q$.

Une séquence $\sigma$ de pas est une séquence infinie
$\sigma = q_0 \succ_{t0} q_1 \succ_{t1} q_2 \succ_{t2} \ldots$
avec $t_i \in 3^+$.

Pour définir la sémantique de TCTL, nous avons besoin d'un outil permettant de définir une séquence ordonnée d'état à partir d'une séquence de pas :
Le support d'une séquence de pas $\sigma$ est une paire $(\chi(\sigma), \leq)$ où

- $\chi(\sigma) = \cup_{i \in \angle} \chi_i(\sigma)$
  où pour tout $i \in \angle$,
  $\chi_i(\sigma) = \{(q_i + t, i) \mid t \leq t_i\}$
- $\leq$ est un ordre total défini par :
  $\forall i, j \in \angle, t, t' \in 3^+. (q_i + t, i) \leq (q_j + t', j)$ ssi $(i < j) \vee (i = j \wedge t \leq t')$

Soit un graphe temporisé G (S, H, $\delta$, $s_{init}$, $\Delta$), un état q = <s ,$\nu$> de G et une formule TCTL $\phi$, la relation de satisfaction $q \models \phi$ est définie de manière inductive comme suit :
$q \models p$ ssi $p \in \mu(s)$.
$q \models x$ op c ssi $\nu(x)$ op c.
$q \models x - y$ op c ssi $\nu(x) - \nu(y)$ op c.
$q \models \neg\phi$ ssi $q \not\models \phi$.
$q \models \phi_1 \wedge \phi_2$ ssi $q \models \phi_1$ et $q \models \phi_2$.
$q \models x.\phi$ ssi $<s, \nu [x \rightarrow 0]> \models \phi$.
$q \models \phi_1 \exists U \ \phi_2$ ssi $\exists\sigma$ dans G avec $q_0 = q$ tel que $\exists(q', i) \in \chi(\sigma)$ tel que $q' \models \phi_2$ et $\forall(q'', j) \leq (q', i)$, $q'' \models \phi_1 \vee \phi_2$.
$q \models \phi_1 \forall U \ \phi_2$ ssi $\forall\sigma$ de G avec $q_0 = q$, $\exists(q', i) \in \chi(\sigma)$ tel que $q' \models \phi_2$ et $\forall(q'', j) \leq (q', i)$, $q'' \models \phi_1 \vee \phi_2$.
où op $\in \{<, \leq\}$.
c $\in$ 9.
x, y sont des horloges et x, y $\notin$ H car la logique n'agit pas sur le comportement du modèle mais permet de spécifier des propriétés que devront satisfaire le modèle.
$\sigma$ est une séquence de pas et $\chi(\sigma)$ son support.

On appelle l'ensemble caractéristique d'une formule $\phi$, l'ensemble des états de G satisfaisant la formule $\phi$.
Lors d'un pas, les vieillissements nuls n'étant pas interdits, il en résulte que contrairement à la logique TCTL présentée au chapitre 3, il est possible ici que plusieurs états soient visités à un instant t.
Des séquences de pas considérées pourraient donc être convergentes, c'est à dire où le temps ne progresse plus au bout d'un moment.
Exemple :$((0, x = 0) \rightarrow^{a} (1, x = 0) \rightarrow^{b})^{+}$
De plus, dans une séquence de pas, bien qu'il y ait un vieillissement non nul lors de chaque pas, le temps pourrait ne pas progresser au-delà d'un certain instant. Par exemple, lorsque lors du premier pas, il y a un vieillissement de t, et lors des pas suivants des vieillissements de t/2, t/4, t/8,…
Cela peut poser problème lors de la vérification d'une propriété dont la traduction en une formule TCTL contient un opérateur $\forall U$.
Par exemple, pour un graphe temporisé G où il est inévitable d'atteindre un état satisfaisant une propriété p, comme il est toujours possible d'effectuer des pas où on ne vieillit pas et où on ne fait pas d'action, il est possible de trouver une séquence de pas convergente où, avant d'atteindre un état satisfaisant p, on effectue une infinité de tels pas. De plus, s'il est possible, pour atteindre un état satisfaisant p, de se retrouver dans un sommet s de G où on peut vieillir (ce qui est courant), on peut trouver une séquence de pas où on se retrouve dans s et au lieu de vieillir une (ou quelques) fois avant de changer de sommet, on reste dans s et on vieillit une infinité de fois mais jamais assez pour que $\Delta(s)$ ne soit plus satisfait par les valeurs des horloges et ainsi ne pas atteindre un état satisfaisant p dans la séquence de pas.

De plus, le temps pour un système réel évolue au delà de tout instant et donc, seules les séquences de pas d'un graphe temporisé où le temps évolue également au delà de tout instant sont intéressantes.

Pour cela, lors de l'évaluation de formules TCTL, on impose de ne tenir compte que des séquences de pas où le temps s'écoule au delà de tout instant.

Notons que dans le cadre de la logique TCTL présentée au chapitre 3, la définition d'une exécution d'une structure temporisée force le temps à évoluer au delà de tout instant. En effet, il est demandé que tout instant $t \in 3^+$ se trouve dans un intervalle d'une exécution d'une structure temporisée.

## 2. Caractérisation par point fixe des opérateurs $\forall U$ et $\exists U$

Les opérateurs $\forall U$ et $\exists U$ peuvent être caractérisés par un point fixe. Ceci a été fait dans le cadre de la logique CTL. Les fonctionnelles définies dans ce cadre utilisent un opérateur "état suivant" ❍. Dans le cadre de TCTL, vu la densité du temps, lors d'un vieillissement, on peut toujours trouver un état intermédiaire entre deux états "successifs". L'idée d'état suivant présente en CTL n'a, pour cette raison, plus de sens en TCTL. On utilise pour cela un autre opérateur : l'opérateur prédécesseurs en un pas.

**Prédécesseurs en un pas**

L'opérateur prédécesseurs en un pas, $Q_1 \succ Q_2$, avec $Q_i$ (i = 1,2) deux ensembles d'états, caractérise l'ensemble des états pour lesquels il existe un état se trouvant dans $Q_2$, accessible en un pas, avec tous les états intermédiaires se trouvant dans $Q_1$ ou $Q_2$.

Par exemple, si $Q_1$ est l'ensemble des états dont les valuations associent une valeur à l'horloge x inférieure à 2 et $Q_2$ est l'ensemble des états dont les valuations associent une valeur à x égale ou supérieure à 2 et dont le sommet est labellé par une proposition p. Soit un état $(s, \nu)$ pour lequel $\nu$ associe la valeur 0 à x et s n'est pas labellé par p. Cet état satisfera $Q_1 \succ Q_2$ s'il est possible, partant de cet état, de rester pendant deux unités de temps dans le sommet s et ensuite faire une transition vers un sommet labellant p (sans que x soit réinitialisé).

Il est à noter que si un état $(s, \nu)$ satisfait $Q_1 \succ Q_2$, alors $(s, \nu) \in Q_1 \cup Q_2$.

Un pas est constitué de deux parties, un vieillissement suivi éventuellement d'une transition. On donne ici une manière de calculer ces deux parties.

Nous définissons les notions de prédécesseurs instantanés et de prédécesseurs temporels.

- Soit un arc $e = (s_1, a, \varphi, H', s_2) \in \delta$ et soit l'état symbolique $(s_2, \varphi_2)$. On définit les prédécesseurs instantanés de $(s_2, \varphi_2)$ par l'arc e :

  $pre_e(s_2, \varphi_2) = (s_1, \varphi \wedge \Delta(s_1) \wedge (\varphi_2)[H' \rightarrow 0])$ .

  Intuitivement, cet opérateur caractérise l'ensemble des états à partir desquels, en n'effectuant que la transition e, il est possible d'atteindre un état de $(s_2, \varphi_2)$. Ce sont donc des états de sommet $s_1$ dont la valuation satisfait l'invariant de $s_1$ et la contrainte liée à la transition e, pour pouvoir être en droit de la faire. Lors de la transition, on remet les horloges de H' à zéro. Avant la transition, les valeurs des horloges doivent satisfaire des contraintes telles que après la remise à zéro d'horloges, $\varphi_2$ est satisfait. C'est ce que demande la troisième contrainte.

  L'ensemble des états symboliques caractérisant les prédécesseurs instantanés de $(s, \varphi)$ est :

  $Pre(s, \varphi) = \cup_{e \in \delta_s} pre_e(s, \varphi)$

  où $\delta_s$ est l'ensemble des transition ayant s comme sommet d'arrivée.

- L'opérateur prédécesseurs temporels pour un sommet s et deux contraintes $\varphi_1$ et $\varphi_2 \in \Psi(H)$ est défini comme suit :
  $(s, \varphi_1) \succ_t (s, \varphi_2) = (s, \exists t \in R^+. (\varphi_2 + t \wedge \forall t' \leq t. (\varphi_1 \vee \varphi_2) + t'))$
  Intuitivement, cet opérateur donne l'ensemble des états de sommet s à partir desquels, en vieillissant, il est possible d'atteindre un état satisfaisant la contrainte $\varphi_2$ et pendant tout ce vieillissement, la contrainte $\varphi_1 \vee \varphi_2$ est satisfaite.
  On donne plus loin la manière d'éliminer les quantificateurs et obtenir ainsi une contrainte équivalente sans quantificateur.

Nous pouvons maintenant définir l'opérateur prédécesseurs en un pas entre deux ensembles d'états :

$(\cup_{s\in S}(s, \varphi_s^1)) \succ (\cup_{s\in S}(s, \varphi_s^2)) = \cup_{s\in S}(s, \varphi_s^1) \succ_t (s, \xi_s \vee \varphi_s^2)$

où $\xi_s = \vee_{(s,\varphi)\in P_s} \varphi$

et $P_s = \{(s, \varphi \wedge \varphi_s^1)$ tel que $\exists(s', \varphi_{s'}^2)$ avec $(s, \varphi) \in Pre(s', \varphi_{s'}^2)\}$

$\xi_s$ est l'ensemble des contraintes des états symboliques de sommet s, prédécesseurs instantanés d'un état de $\cup_{s\in S}(s, \varphi_s^2)$ et satisfaisant $\varphi_s^1$. On impose cette dernière contrainte car :
- Soit un prédécesseur instantané $(s, \nu)$ ne satisfait pas $\varphi_s^2$, et dans ce cas il doit satisfaire $\varphi_s^1$.
- Soit un prédécesseur instantané $(s, \nu)$ satisfait $\varphi_s^2$. Dans ce cas, même s'il ne satisfait pas $\varphi_s^1$, il est repris dans $(s, \xi_s \vee \varphi_s^2)$.

Les ensembles d'états sont représentés sous forme d'ensemble d'états symboliques. Il est montré dans [Yov93] que cela n'est pas une restriction dans le cadre de l'évaluation de formules TCTL.

**Opérateur $\exists U$**

Dans le cadre de CTL, l'ensemble caractéristique de $\phi_1 \exists U \phi_2$ peut être défini comme le plus petit point fixe de la fonctionnelle
$F(X) = \mu X. (Q_2 \vee (Q_1 \wedge \bigcirc X))$
où $Q_i$ est l'ensemble des sommets satisfaisant la contrainte $\phi_i$ (i = 1, 2)
En effet, le processus itératif commence avec un ensemble solution contenant tous les états du modèle satisfaisant $\phi_2$. A chaque étape, le processus itératif ajoute les états du modèle satisfaisant $\phi_1$ et à partir desquels on peut atteindre un état se trouvant déjà dans l'ensemble solution en effectuant une transition. Lorsqu'on met un état dans l'ensemble solution, par induction, on sait que cet état, en plus de satisfaire $\phi_1$, permet d'atteindre un état satisfaisant $\phi_2$ avec tous les états intermédiaires satisfaisant $\phi_1$.
Nous définissons ainsi, dans le cadre de TCTL, l'ensemble caractéristique de $\phi_1 \exists U \phi_2$ comme étant le plus petit point fixe de la fonctionnelle
$F(X) = \mu X. (Q_2 \vee (Q_1 \succ X))$
où $Q_i$ est l'ensemble des états satisfaisant la contrainte $\phi_i$ (i = 1, 2).
De nouveau, au départ, l'ensemble solution contient tous les états satisfaisant $\phi_2$. A chaque étape du processus itératif, on rajoute tous les états du modèle à partir desquels on peut atteindre un état de l'ensemble solution en un pas avec tous les états intermédiaires soit satisfaisant $\phi_1$, soit se trouvant dans l'ensemble solution. De cette manière, lorsqu'on rajoute un état dans l'ensemble solution, on sait qu'il est possible d'atteindre un état satisfaisant $\phi_2$, avec tous les états intermédiaires satisfaisant $\phi_1 \vee \phi_2$.

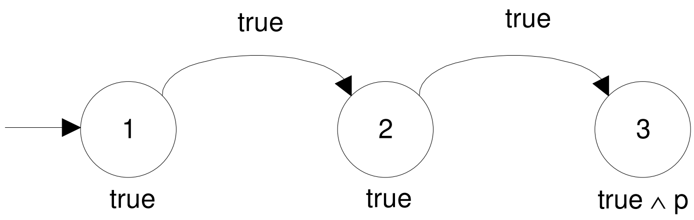



Figure 20

Le calcul de l'ensemble caractéristique de la formule true $\exists U$ p sur le graphe temporisé montré à la figure 20 se fait en trois étapes.
Au départ, l'ensemble solution contient tous les états de sommets 3.
A la première étape de l'algorithme itératif, on rajoute tous les états de sommet 2. En effet, dès qu'on se trouve au niveau du sommet 2, il est possible d'atteindre en un pas un état de sommet 3 avec tous les états intermédiaires ayant leur valuation satisfaisant true ou leur sommet étant 3.
A la deuxième étape, on rajoute tous les états de sommet 1. A partir de ces états, il est en effet possible d'atteindre un état de sommet 2 en un pas avec tous les états intermédiaires ayant leur valuation satisfaisant true ou se trouvant dans l'ensemble solution.
A la troisième étape, aucun nouvel état n'est rajouté, l'algorithme a convergé et les états satisfaisant true $\exists U$ p se trouvent bien tous dans l'ensemble solution.

**Opérateur $\forall U$**
Dans le cadre de CTL, l'ensemble caractéristique de $\phi_1 \ \forall U \ \phi_2$ a été défini comme étant le plus petit point fixe de la fonctionnelle
$F(X) = \mu X.\ (Q_2 \vee (Q_1 \wedge \bigcirc X \wedge \neg \bigcirc \neg X)$
où $Q_i$ est l'ensemble des sommets satisfaisant la contrainte $\phi_i$ (i = 1, 2)

Malheureusement, nous ne pouvons pas nous inspirer de cette fonctionnelle dans le cadre de TCTL en remplaçant l'opérateur état suivant par l'opérateur prédécesseur en un pas. En effet, cela nous donnerait la fonctionnelle
$F(X) = \mu X.\ (Q_2 \vee (Q_1 \triangleright X \wedge \neg(Q_1 \triangleright \neg X)))$
Considérons un état q qui n'a que des successeurs temporels, c'est à dire qu'aucune action ne pourra être effectuée à partir de q. Supposons que q ait un successeur temporel q + t dont la valuation assigne la valeur 2 à x. De ce fait, q satisfait true $\forall U$ x = 2. Mais q ne fera pas partie de l'ensemble solution calculé avec F(X). En effet, tous les états qu'on atteint à partir de q pour arriver à q + t ont leur valuation qui assigne à x une valeur plus petite que 2. De ce fait, en un pas, on peut également atteindre un état satisfaisant x ≠ 2 avec les états intermédiaires satisfaisant true ∨ x ≠ 2.

Une autre fonctionnelle doit être trouvée. Celle-ci est obtenue à partir d'une fonctionnelle caractérisant l'inévitabilité bornée.
L'inévitabilité bornée est la propriété exprimée comme :
"Il est inévitable de satisfaire $\phi_2$ avant c unités de temps, $\phi_1 \vee \phi_2$ étant auparavant continûment satisfait."
Ceci peut être exprimé en TCTL de la manière suivante

$$z.\ (\phi_1\ \forall U\ (\phi_2 \wedge z \leq c))$$

On peut exprimer cette propriété d'une seconde manière :
" Il n'est pas possible d'atteindre un état ne satisfaisant ni $\phi_1$, ni $\phi_2$ ou de vieillir de plus de c sans atteindre un état satisfaisant $\phi_2$."
Ceci justifie l'équivalence suivante

$$z.\ (\phi_1\ \forall U\ (\phi_2 \wedge z \leq c)) \equiv \neg z.\ (\neg\phi_2\ \exists U\ (\neg(\phi_1 \vee \phi_2) \vee z > c))$$

où la seconde formule est la traduction en TCTL de l'inévitabilité bornée exprimée de la seconde manière.

Comme il est possible d'exprimer l'inévitablilité bornée en TCTL avec l'opérateur $\exists U$, on peut définir l'ensemble caractéristique de cette propriété par une fonctionnelle, basée sur la fonctionnelle de cet opérateur.
$F(X) = \neg z.\mu X.\ (\neg( Q_1 \vee Q_2) \vee z > c \vee \neg Q_2$ ➢ $X)$
Où $Q_i$ est l'ensemble des états satisfaisant la contrainte $\phi_i$ $(i = 1, 2)$
$c \in \angle\backslash\{0\}$. Rien n'empêche c d'être réel, mais cela n'aurait rien d'avantageux.
Nous définissons l'ensemble caractéristique de $\phi_1\ \forall U\ \phi_2$ à partir de cette fonctionnelle. L'idée est que, partant de l'ensemble solution égal à $Q_2$, la fonctionnelle va rajouter à chaque étape les états pour lesquels $z.\ (\phi_1\ \forall U\ (X \wedge z \leq c))$ est vrai, X caractérisant l'ensemble des états se trouvant déjà dans l'ensemble solution, c'est à dire l'ensemble des états à partir desquels il est inévitable d'atteindre avant c unités de temps un état de l'ensemble solution avec tous les états intermédiaires dans l'ensemble solution ou satisfaisant $\phi_1$. D'itération en itération, on rajoute donc les états à partir desquels il faut vieillir de plus en plus pour atteindre un état satisfaisant $\phi_2$, les états intermédiaires satisfaisant $\phi_1$ ou $\phi_2$. On en conclut que c peut prendre n'importe quelle valeur, mais il est intéressant de prendre une grande valeur.
L'ensemble caractéristique de $\phi_1\ \forall U\ \phi_2$ peut être défini comme le plus petit point fixe de la fonctionnelle
$F(X) = \mu X.\ (Q_2 \vee \neg z.\ \mu Y.\ (\neg(Q_1 \vee X) \vee z > c \vee \neg X$ ➢ $Y))$
Où $c \in \angle$.
$Q_i$ est l'ensemble des états satisfaisant la contrainte $\phi_i$ $(i = 1, 2)$.
Le calcul de l'ensemble caractéristique de la formule true $\forall U$ p sur le graphe temporisé de la figure 21 se fait en quatre étapes. On considère que c = 1. Comme aucun état ne satisfait la contrainte $\neg(\text{true} \vee p)$, dans ce cas ci, la fonctionnelle calcule à chaque étape l'ensemble des états à partir desquels on ne peut vieillir de plus d'une unité de temps sans atteindre un état de l'ensemble solution.
Au départ, l'ensemble solution contient tous les états de sommet 3.
A la première étape, les états rajoutés sont les états de sommet 2 dont la valuation donne une valeur à x comprise dans l'intervalle [2,3].
A la seconde étape, on rajoute les états de sommet 2 dont la valuation assigne à x une valeur comprise entre 1 et 2. Sont également rajoutés les états de sommet 1 dont la valuation assigne également une valeur à x entre 1 et 2. De ces états, on ne peut en effet que vieillir de moins

d'une unité de temps, puis effectuer l'action a, et ainsi atteindre un état de sommet 2 se trouvant dans l'ensemble solution depuis l'étape 1.
A la troisième étape, …

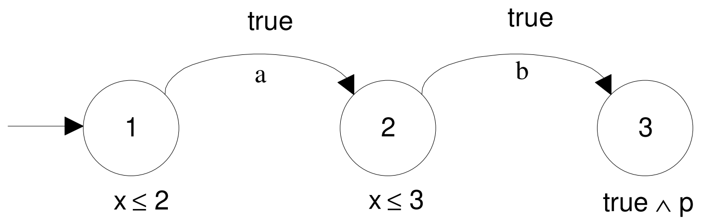



Figure 21

On pourrait se demander si la contrainte sur la variable z est nécessaire et si l'ensemble caractéristique de la formule $\phi_1 \forall U \phi_2$ ne peut pas être trouvé avec la fonctionnelle
$F(X) = \neg \mu X. (\neg(Q_1 \vee Q_2) \vee \neg Q_2 \succ X))$
où $Q_i$ est l'ensemble des états satisfaisant la contrainte $\phi_i$ (i = 1, 2).
Cette fonctionnelle revient à calculer l'ensemble caractéristique de la formule
$\neg(\neg\phi_2 \exists U (\neg\phi_1 \wedge \neg\phi_2))$.

Comme on joue dans ce cas avec des négations, on pourrait être amené à rechercher les états à partir desquels il n'est pas possible d'atteindre un état satisfaisant false.
Par exemple, si on réessaye de calculer à partir du modèle de la figure 21 l'ensemble caractéristique de la formule true $\forall U$ p avec cette fonctionnelle, cela va revenir à calculer l'ensemble caractéristique de le formule $\neg(\neg p \exists U \text{ false})$. Cela n'a aucun sens puisque aucun état ne satisfait la contrainte false. Cette fonctionnelle ne calcule donc pas ce qu'on veut.

Il est démontré dans [Yov93] que les fonctionnelles présentée ici calculent bien les ensembles caractéristiques des formules $\phi_1 \exists U \phi_2$ et $\phi_1 \forall U \phi_2$.

## 3. Algorithme de Model Checking symbolique

**Algorithme**
Pour un graphe temporisé $G = (S, H, \delta, s_{init}, \Delta)$, soit $S(\phi)$ est l'ensemble des états symboliques satisfaisant la formule $\phi$.
$S(\phi)$ est calculé par induction sur la structure de $\phi$ de la façon suivante :

$S(p) = \cup_{s\in S, p\in \nu(s)} (s, \Delta(s)) \cup \cup_{s\in S, p\notin \nu(s)} (s, \text{false})$
$S(x \text{ op } c) = \cup_{s\in S} (s, \Delta(s) \wedge x \text{ op } c)$
$S(x - y \text{ op } c) = \cup_{s\in S} (s, \Delta(s) \wedge x - y \text{ op } c)$
$S(x . \phi_1) = S(\phi_1) [x \to 0]$
$S(\neg \phi_1) = \neg S(\phi_1)$
$S(\phi_1 \wedge \phi_2) = S(\phi_1) \wedge S(\phi_2)$
$S(\phi_1 \exists U \phi_2) = \vee_{k\in N} X_k$
Où $X_0 = S(\phi_2)$ et $X_{k+1} = X_k \vee (S(\phi_1) \succ X_k)$

$S\ (\phi_1\ \forall U\ \phi_2) = \vee_{k\in N}\ X_k$
Où $X_0 = S\ (\phi_2)$ et $X_{k+1} = X_k \vee (\neg\ (\neg(X_k)\ \exists U\ (\neg(S(\phi_1) \vee X_k) \vee z > c)))\ [z \rightarrow 0]$

Où op $\in \{<, \leq\}$.
c $\in \angle\backslash\{0\}$. Comme dit plus haut, il est intéressant que c soit grand. On considère donc que c est le plus grand entier codable sur la machine où l'algorithme est implanté.
La preuve de cet algorithme est donnée dans [Yov93].

**Explication de l'algorithme**
L'opérateur x. est calculé en effectuant une initialisation d'horloges sur des contraintes d'états symbolique. Effectivement, on retrouve la même idée qu'au niveau des transitions lors du parcours en sens inverse d'un graphe temporisé. Connaissant les états satisfaisant une formule $\phi$, on aimerait savoir quel sont les états pour lesquels, en mettant la variable x à zéro, on obtient un état de l'ensemble caractéristique de $\phi$. La valeur de x n'a pas d'importance avant la mise à zéro. Par contre, des relations entre les horloges doivent être satisfaites pour qu'après initialisation de x, l'état obtenu satisfasse bien $\phi$.
Pour les opérateur $\exists U$ et $\forall U$, l'algorithme utilise les fonctionnelles définies ci-avant pour leur calcul.

## 4. Représentation des contraintes

L'algorithme de Model Checking TCTL qu'on vient de présenter est basé sur la manipulation de contrainte sur les horloges. Il existe un moyen intéressant pour implanter de telles contraintes. Cette implantation utilise des "matrices de bornes", pour lesquelles des opérateurs sont définis. Nous allons maintenant les décrire.

**Borne**
Définissons tout d'abord ce qu'est une borne. L'ensemble des bornes est l'ensemble $\beta$ défini comme suit :
$\beta = 9 \times \{<, \leq\} \cup \{(-\infty, <), (\infty, <)\}$.
Les bornes vont permettre de représenter des contraintes sur des horloges. On n'a pas de borne $(-\infty, \leq)$ et $(\infty, \leq)$ car cela permettrait de dire qu'une horloge vaut l'infini. Or, l'infini n'étant pas un nombre, cela n'aurait pas de sens.

Nous pouvons définir un ordre total $\subseteq$ sur les éléments de cet ensemble $\beta$. Nous posons tout d'abord que < est strictement inférieur à $\leq$. Pour deux éléments quelconques $(c_1, \Lambda_1)$ et $(c_2, \Lambda_2)$, nous définissons $(c_1, \Lambda_1) \subseteq (c_2, \Lambda_2)$ si $c_1 < c_2$ ou $c_1 = c_2$ et si $\Lambda_1$ est inférieur ou égal à $\Lambda_2$. Nous définissons l'opposé d'une borne :$\neg(c, <)$ est la borne $(-c, \leq)$ et $\neg(c, \leq)$ est la borne $(-c, <)$.

Les opérateurs $\otimes$ et $\oplus$ sont définis comme suit :
$(c_1, \Lambda_1) \otimes (c_2, \Lambda_2) = (c_1 + c_2\ , \min(\Lambda_1, \Lambda_2))$
$(c_1, \Lambda_1) \oplus (c_2, \Lambda_2) = \min((c_1, \Lambda_1), (c_2, \Lambda_2))$
où min (a, b) donne le minimum entre a et b.
+ est la somme sur les entiers.

**Matrice de bornes**

L'horloge **0** est une horloge spéciale pour laquelle toutes les valuations lui associent la valeur 0. Une matrice de borne est une matrice dont les éléments sont des bornes et dont chaque ligne et chaque colonne i sont associées à une horloge. Une borne (c, •) en colonne i associée à y et en colonne j associée à x, représente la contrainte x – y • c. Une matrice de borne représente donc une contrainte sur les horloges qui est la conjonction des contraintes représentées par ses bornes.

Ex. :

$$\begin{array}{cc} & \begin{array}{ccc}\mathbf{0} & x & y\end{array} \\ \begin{array}{c}\mathbf{0}\\ x\\ y\end{array} & \begin{pmatrix} (0,\le) & (0,<) & (-1,\le) \\ (2,<) & (0,\le) & (1,<) \\ (\infty,<) & (\infty,<) & (0,\le)\end{pmatrix}\end{array}$$

La matrice représente la contrainte $y \geq 1 \wedge 0 < x < 2 \wedge x - y < 1$.

La contrainte x – x • c n'est vrai que si • = ≤ et c = 0. Les bornes de la diagonale d'une matrice sont pour cette raison en général la borne (0, ≤). Si ce n'est pas le cas, on se retrouve en présence d'une matrice représentant la contrainte false.

L'ensemble caractéristique d'une matrice de borne est l'ensemble des valuations qui associent aux horloges des valeurs satisfaisant les contraintes représentées par la matrice.

Nous pouvons étendre l'ordre total $\subseteq$ des bornes aux matrices de bornes. Pour toutes matrices de bornes M et M', nous avons $M \subseteq M'$ si $M_{x,y} \subseteq M'_{x,y}$ pour tout x, y $\in$ H. Notons qu'ici l'ordre n'est plus total.

Si $M \subseteq M'$ alors l'ensemble caractéristique de M est inclus dans celui de M'.

Un certain nombre de matrices spéciales sont définies :

E est la matrice où pour tout x, y $\in$ H,

$E_{x,y}$ = (0, ≤) si x = y

(∞, <) sinon.

O est la matrice où pour tout x, y $\in$ H,

$O_{x,y}$ = (-∞, <).

La matrice O permet de représenter de manière unique l'ensemble caractéristique vide (contrainte false).

U est la matrice où pour tout x, y $\in$ H,

$U_{x,y}$ = (0, ≤) si x = y ou x = 0

(∞, <) sinon.

La matrice U représente la contrainte true.

Les opérateurs $\otimes$ et $\oplus$ peuvent également être étendus aux matrices de bornes.

Pour deux matrices de bornes M et M', $M \otimes M'$ est la matrice M'' telle que pour tout $x, y \in H$,
$M''_{x,y} = \oplus_{z \in H} M_{x,z} \otimes M'_{z,y}$ .

Cet opérateur permet de faire ressortir des contraintes implicitement données par une matrice M. Ceci est fait en calculant $M \otimes M$.
Ex. :

$$\begin{pmatrix} (0,\leq) & (0,<) & (-1,\leq) \\ (2,<) & (0,\leq) & (\infty,<) \\ (\infty,<) & (\infty,<) & (0,\leq) \end{pmatrix} \otimes \begin{pmatrix} (0,\leq) & (0,<) & (-1,\leq) \\ (2,<) & (0,\leq) & (\infty,<) \\ (\infty,<) & (\infty,<) & (0,\leq) \end{pmatrix} = \begin{pmatrix} (0,\leq) & (0,<) & (-1,\leq) \\ (2,<) & (0,\leq) & (1,<) \\ (\infty,<) & (\infty,<) & (0,\leq) \end{pmatrix}$$

En ayant l'horloge x correspondant à la ligne et la colonne 2 et l'horloge y correspondant à la ligne et la colonne 3, la matrice de bornes M donne la contrainte $y > 1 \wedge 0 < x < 2$. Mais ces contraintes impliquent que $x - y < 1$. La matrice résultant du calcul de $M \otimes M$ donne cette contrainte explicitement.
Pour mieux comprendre pourquoi cette contrainte apparaît dans la matrice résultat. Calculons la borne représentant la contrainte $M''_{x,y}$.
D'après la définition de l'opérateur $\otimes$,
$M''_{x,y} = (M_{x,0} \oplus M_{0,y}) \otimes (M_{x,x} \oplus M_{x,y}) \otimes (M_{x,y} \oplus M_{y,y})$
Pour chaque horloge z, on prend les contraintes $x - z \bullet^1 c$ et $z - y \bullet^2 d$ spécifiée par les bornes de la matrice. De ces deux contraintes, il est possible de trouver une contrainte du type $x - y \bullet c + d$. En effet,
$x - z \bullet^1 c \equiv x - c \bullet^1 z$
$z - y \bullet^2 d \equiv z \bullet^2 d + y$
De là, on a
$x - c \bullet^1 z \bullet^2 d + y \equiv x - c \bullet d + y \equiv x - y \bullet c + d$
où $\bullet = <$ si $\bullet^1$ ou $\bullet^2 = <$.
$= \leq$ sinon.
C'est ce qui est fait en calculant les opérateurs $\oplus$ sur les bornes dans la formule. Ayant toutes les contraintes $x - y \bullet c$ de la matrice M, on ne garde que celle qui est la plus contraignante. C'est ce qui est fait en calculant les opérateurs $\otimes$ sur les bornes. Ce sera la nouvelle contrainte $x - y \bullet c$, représentée par la borne $M''_{x,y}$ de M''.

Il se peut qu'après avoir effectué $M \otimes M$, toutes les contraintes ne soient pas explicitement données par les bornes de la matrice résultante. Si on veut faire apparaître toutes les contraintes implicites d'une matrice M, il faut alors réitérer l'opération jusqu'au moment où on ne fait plus apparaître de nouvelles contraintes. Lorsque cela est fait, la matrice obtenue est dite en forme normale ou canonique.

$M^* = \oplus_{k \in N} M^k$ où $M^{k+1} = M^k \otimes M$ et $M^0 = E$

La forme canonique d'une matrice de borne M, notée cf (M), est définie par :
cf (M) = O si l'ensemble caractéristique de M est vide
$= M^*$ sinon.

Une propriété intéressante de cf (M) est que pour deux matrices de bornes M et M', nous avons que l'ensemble caractéristique de M est égal à celui de M' ssi cf (M) = cf (M').
Ceci est démontré dans [Yov93].

Pour deux matrices de bornes M et M', $M \oplus M'$ est la matrice M'' tel que pour tout $x, y \in H$, $M''_{x,y} = M_{x,y} \oplus M'_{x,y}$.

Nous pouvons facilement montrer que l'ensemble caractéristique de $M \oplus M'$ est l'intersection des ensembles caractéristiques de M et M'. En effet, Pour tout $x, y \in H$, la borne en $M''_{x,y}$ représentera la contrainte la plus contraignante entre les contraintes représentées par les bornes $M_{x,y}$ et $M'_{x,y}$.
Ex. :

$$\begin{pmatrix} (0,\le) & (0,<) & (-1,\le) \\ (2,<) & (0,\le) & (1,<) \\ (\infty,<) & (\infty,<) & (0,\le) \end{pmatrix} \oplus \begin{pmatrix} (0,\le) & (0,<) & (-4,<) \\ (\infty,<) & (0,\le) & (1,<) \\ (\infty,<) & (\infty,<) & (0,\le) \end{pmatrix} = \begin{pmatrix} (0,\le) & (0,<) & (-4,<) \\ (2,<) & (0,\le) & (1,<) \\ (\infty,<) & (\infty,<) & (0,\le) \end{pmatrix}$$

De nouveau, en ayant l'horloge x correspondant à la ligne et la colonne 2 et l'horloge y correspondant à la ligne et la colonne 3,
La première matrice représente la contrainte $y > 1 \wedge 0 < x < 2 \wedge x - y < 1$.
La seconde matrice représente la contrainte $x > 0 \wedge y > 4 \wedge x - y < 1$.
La troisième matrice représente la contrainte $y > 4 \wedge 0 < x < 2 \wedge x - y < 1$, qui est bien la conjonction des contraintes des deux premières matrices.

Si nous prenons un ensemble de matrices de bornes K, nous pouvons définir l'ensemble caractéristique de cet ensemble comme étant l'union des ensembles caractéristiques des matrices qui le composent et la forme canonique de cet ensemble comme étant l'ensemble des formes canoniques des matrices composant K, $cf(K) = \{\cup_{M \in K} cf(M)\}$.

Nous définissons maintenant l'opérateur d'initialisation d'horloges. Pour un ensemble H' d'horloges à remettre à zéro, nous définissons les ensembles $L(x_i)$.

$$L(x_i) = \begin{cases} H' \cup \{\mathbf{0}\} & \text{si } x_i = \mathbf{0}. \\ \varnothing & \text{si } x_i \in H'. \\ \{x_i\} & \text{si } x_i \notin H'. \end{cases}$$

L'opérateur d'initialisation d'horloges noté $M[H' \to 0]$ pour une matrice M est défini comme suit :
Posons $M' = M[H' \to 0]$.

$$M'_{xy} = \begin{cases} \oplus_{z \in L(x), w \in L(y)} M_{zw} & \text{si } L(x) \neq \varnothing \text{ et } L(y) \neq \varnothing. \\ U_{xy} & \text{sinon.} \end{cases}$$

L'idée de l'opérateur est que les occurrences des horloges remises à zéro sont remplacées par zéro dans la contrainte représentée par la matrice M.
Si M représente la contrainte
$x < 2 \wedge y > 1 \wedge x - y < 1$
l'initialisation de x donnerait la contrainte
$0 < 2 \wedge y > 1 \wedge 0 - y < 1$, c'est à dire $y > 1$.
Si on initialise une horloge z à zéro, on peut constater que les contraintes du type $z \bullet c$ ($\bullet \in \{<, \le\}$) vont disparaître ainsi que les contraintes $z \bullet c$ (avec $\bullet \in \{>, \ge\}$), sauf si, dans ce

dernier cas, ce ne sont pas les contraintes $x \geq 0$, les contraintes devenant alors false. Les contraintes du type $z - v \bullet c$ et $v - z \bullet c$ ($\bullet \in \{<, \leq\}$) vont se transformer en contraintes du type $v \bullet c$ et $-v \bullet c$. Comme dans l'exemple où on a les contraintes $y > 1$ et $y > -1$, on risque de se retrouver avec plusieurs contraintes du type $v \bullet c$ ou $-v \bullet c$. Dans la matrice résultat, il ne faut garder que les bornes représentant les contraintes les plus fortes. De plus, les bornes représentant les contraintes ayant disparu, ainsi que les contraintes qui sont devenues false, sont remplacées par la borne se trouvant au même emplacement dans la matrice U (par exemple, la borne $M'_{x,y}$ sera la borne $U_{x,y}$). Si une contrainte devient la contrainte false, le calcul de la borne $M'_{0,0}$ ne donnera pas la borne $(0, \leq)$ et la matrice M' représentera bien la contrainte false. Les bornes $M'_{v,z}$ où v et z ne sont pas remises à zéros sont les bornes $M_{v,z}$. On peut remarquer que les matrices doivent être en forme canonique pour que le calcul de l'opérateur fonctionne. Bien sûr, des contraintes implicites peuvent apparaître dans la matrice après initialisation.

Toute contrainte sur les horloges peut être spécifiée sous forme d'ensemble de matrices de bornes. En effet, une borne dans une matrice représente une contrainte sur les valeurs de deux horloges. Une matrice de borne représente une conjonction de telles contraintes et un ensemble de matrice représente une disjonction des contraintes représentées par les matrices de bornes de l'ensemble.
C'est cette représentation des contraintes qui a été utilisée lors de l'implantation de l'algorithme présenté dans ce chapitre pour représenter les contraintes des états symboliques et les invariants de sommet.

Un certain nombre de modifications ont été cependant apportées par rapport à [Yov93] et [Oli94].
Tout d'abord au niveau la définition des opérateurs sur les ensembles d'états symboliques.
Pour la négation et la remise à zéro d'horloges, la définition présentée ici rajoute par rapport à [Yov93] le fait que les états résultants de l'opération doivent satisfaire leur invariant de sommet. Partant d'états satisfaisant déjà cette contrainte, cette contrainte est automatiquement vérifiée pour la conjonction et la disjonction, mais pas nécessairement pour les autres opérateurs.
Les opérateurs prédécesseurs instantanés et prédécesseurs temporels ont été simplifiés par rapport à [Yov93] et [Oli94]. En effet, le prédécesseur instantané d'un état $(s_1, \varphi_1)$ par un arc $e = (s_1, a, \varphi, H', s_2)$ y est défini comme
$pre_e(s_2, \varphi_2) = (s_1 , \varphi \wedge \Delta(s_1) \wedge (\varphi_2 \wedge \Delta(s_2)) [H' \rightarrow 0])$ .
et le prédécesseur temporel pour un sommet s et deux contraintes $\varphi_1$ et $\varphi_2$ y est défini comme suit
$(s, \varphi_1) \succ_t (s, \varphi_2) = (s, \exists t \in R^+. ((\varphi_2 \wedge \Delta(s_2)) + t \wedge \forall t' \leq t. (\varphi_1 \vee \varphi_2) + t'))$.
Dans les deux cas, rajouter explicitement l'invariant de $s_2$ dans les contraintes est inutile puisque cela est impliqué par $\varphi_2$.
Pour les besoins de l'algorithme, des fonctionnelles ont été définies pour le calcul de l'ensemble caractéristique de formules du type $\varphi_1 \exists U \varphi_2$ et $\varphi_1 \forall U \varphi_2$.
La fonctionnelle pour $\varphi_1 \forall U \varphi_2$ a été définie comme
$F(X) = \mu X. (Q_2 \vee \neg z. \mu Y. (\neg(Q_1 \vee X) \vee z > c \vee \neg X \succ Y))$
Lors de la définition de l'algorithme donnée dans [Yov93] et [Oli94], la contrainte c a été remplacée par 1.
Comme il a été expliqué, ici on considère que c est la plus grande valeur possible, c'est à dire le plus grand entier codable sur la machine où l'algorithme est implanté.

Enfin, le calcul de l'opérateur de remise à zéro d'horloges présenté ici est une simplification par rapport à la méthode présentée dans [Oli94], où on permet l'initialisation d'horloges à n'importe quelle valeur. La simplification est justifiée par le cadre dans lequel on travaille ici : pour le Model Checking TCTL, seule la mise à zéro d'horloges est utilisée.

De plus, le calcul de l'opérateur d'initialisation d'horloges présenté dans [Yov93] ne marche pas. Tout d'abord, l'opérateur, $M \downarrow x$ y est défini :
Pour une matrice de borne M, nous posons M' = cf (M) et nous définissons $M \downarrow x$, où $x \in H$, comme étant la matrice O si M' = O sinon comme étant cf (M'') où M'' est définie comme :
$M''_{y,z} = (0, \leq)$ si y = x et z = **0**
$\quad\quad = M'_{y,z}$ sinon.

L'opérateur d'initialisation d'horloge, noté M[x := 0], est ensuite défini comme suit :
posons $M' = M \downarrow x$, alors M[x := 0] est O si M' = O sinon M'' où
$M''_{y,z} = (\infty, <)$ si y = x et z = **0**
$\quad\quad = M'_{y,z}$ sinon.

Donc, dans [Yov93], l'idée du calcul de la mise à zéro d'une horloge x est de mettre la borne $M_{x,0}$ comme étant égale à $(0, \leq)$. Une horloge ne pouvant pas être négative, cela revient à imposer explicitement que x vaut 0. Ensuite, la matrice est mise en forme normale pour faire propager cette information dans la matrice. Finalement, la borne $M_{x,0}$ est mise à $(\infty, <)$ pour enlever la contrainte x = 0 explicitement donnée par la matrice. Le problème avec cette méthode est que les contraintes du type $x - y \bullet d$ et $y - x \bullet d$ existent toujours dans la matrice. De l' "ancienne " information se trouve donc toujours dans la matrice, ce qui bien évidemment pose problème.

# 5. Elimination des quantificateurs

Pour une matrice de borne M, il est démontré dans [Oli94] qu'une formule du type
$\exists t \in 3^+ . (M + t)$ contenant un quantificateur existentiel peut se ramener à une formule sans quantificateur.
Plus précisément,
$\exists t \in 3^+ . (M + t) \equiv \wedge_{x \in H}\, 0 \leq x \bullet M_{x0} \wedge x - y \bullet M_{xy}$
où $M_{xy}$ est la valeur entière de la borne se trouvant à l'emplacement x, y dans la matrice M.
$\bullet$ est l'opérateur de la borne se trouvant à l'emplacement x, y dans la matrice M.
Un état satisfaisant cette contrainte est un état pour lequel en vieillissant de t, on atteint un état satisfaisant la contrainte M. Pour cela, avant vieillissement, les contraintes du type $x - y \bullet c$ doivent déjà être satisfaites puisque toutes les horloges vieillissent à la même vitesse. De plus, les horloges ne peuvent pas non plus dépasser une valeur limite imposée par M. Quant aux valeurs minimales possibles pour les horloges, comme explicité dans l'équivalence, il n'y en a pas avant vieillissement. Si les autres contraintes sont satisfaites alors il est possible de vieillir d'un certain t pour que les contrainte de M soient satisfaites. Il faut pour cela qu'il n'y ait pas de contraintes implicites, c'est à dire que la matrice M doit être en forme normale.
Par exemple, prenons la contrainte $C_1 = y \geq 2 \wedge 0 < x < 3 \wedge x - y < 1$, tous les états atteignant un état satisfaisant $C_1$ par un vieillissement sont les états satisfaisant $x < 3 \wedge x - y < 1$. Même dans le cas limite où la valeur de x est juste inférieure à 3, y ne pouvant être inférieure à x d'une unité de temps ou plus, il est donc possible de vieillir pour que y égale 2 sans que x ne dépasse 3.

A partir de ce résultat, il est également démontré dans [Oli94] qu'une formule du type $\exists t \in 3^+ . (M + t) \wedge \forall t' . t' \leq t \rightarrow (M \vee M') + t'$ peut se ramener à une formule sans quantificateur. Plus précisément,

$\exists t \in 3^+ . (M + t) \wedge \forall t' . t' \leq t \rightarrow (M \vee M') + t' \equiv M'' \wedge \neg(\exists t' . (M'' \wedge \neg M \wedge \neg M') + t')$.

où $M''$ est la matrice obtenue en enlevant le quantificateur $\exists$ de $\exists t \in 3^+ . (M + t)$.

Notons que le quantificateur $\exists$ peut être éliminé par la méthode présentée ci-avant.

L'idée est qu'une contrainte $\forall c$ est équivalente à la contrainte $\neg\exists\, \neg c$ et que la contrainte $t' \leq t$ peut être transformée en une contrainte représentée par une matrice de borne. L'idée de la formule est qu'il faut être capable de vieillir de t pour atteindre un état satisfaisant une contrainte représentée par M, tous les états atteint lors de ce vieillissement satisfaisant une autre contrainte représentée par $M \vee M'$. Tous les états intermédiaires étant des états à partir desquels il est également possible d'atteindre un état satisfaisant M en vieillissant de $t' \leq t$, tous ces états intermédiaires sont caractérisés par la contrainte $\exists t \in 3^+. (M + t)$.

Ce raisonnement permet donc le remplacement de la contrainte $t' \leq t$ par la contrainte $\exists t \in 3^+. (M + t)$.

# Chapitre 6

# Implantation de l'algorithme

L'algorithme de Model Checking présenté au chapitre précédant a été implanté dans le cadre de ce travail. Nous abordons dans ce chapitre quelques aspects de cette implantation.

## 1. Parsing de la logique TCTL

La syntaxe de la logique TCTL présentée au chapitre précédant doit être retravaillée pour pouvoir implanter un analyseur syntaxique efficace. On a pour cela transformé cette grammaire en grammaire LALR(1)
Nous partons de la grammaire

$$
\begin{aligned}
\phi \;\; & \rightarrow p \\
& \mid \neg\phi \\
& \mid x \text{ op } c \\
& \mid x - y \text{ op } c \\
& \mid \phi \wedge \phi \\
& \mid \phi \; \exists U \; \phi \\
& \mid \phi \; \forall U \; \phi \\
& \mid x.\phi \\
\text{op} \;\; & \rightarrow < \\
& \mid \leq
\end{aligned}
$$

Cette grammaire est ambiguë (exemple : $p \wedge q \wedge r$ a plusieurs arbres de dérivation ) . Pour la rendre non ambiguë, nous allons la transformer en une grammaire donnant un ordre de priorité aux opérateurs et imposant l'associativité à gauche pour les opérateurs binaires. La logique ainsi trouvée est

$$
\begin{aligned}
\phi \;\; & \rightarrow \phi_1 \\
\phi_1 \;\; & \rightarrow \phi_1 \wedge \phi_2 \\
& \mid \phi_2 \\
\phi_2 \;\; & \rightarrow \phi_2 \; \exists U \; \phi_3 \\
& \mid \phi_3
\end{aligned}
$$

$\phi_3 \;\rightarrow \phi_3 \;\forall U \;\phi_4$

$\quad | \;\phi_4$

$\phi_4 \;\rightarrow ( \;\neg\phi_1 \;)$

$\quad | \;x. \;( \;\phi_1 \;)$

$\quad | \;x \;op \;c$

$\quad | \;x - y \;op \;c$

$\quad | \;( \;\phi_1 \;)$

$op \;\rightarrow <$

$\quad | \;\leq$

Les formule state(sommet) et TRUE sont rajoutées pour faciliter l'expression de propriétés avec
$q \models state(s)$ ssi $q = (s', \nu)$ et $s' = s$.
$q \models TRUE$ est toujours vrai.

Le programme implanté dans le cadre de ce mémoire intègre un analyseur lexical et syntaxique permettant d'analyser les formules TCTL qui lui sont données en input. Ces analyseurs n'ont pas été implantés à la main, mais avec les outils lex et yacc. Ce choix est motivé par le fait que :

- Il est beaucoup plus rapide d'écrire des spécifications lex et yacc que d'implanter les analyseurs à la main. De plus, le code généré par ces outils est exempt d'erreurs.
- Les spécifications lex et yacc étant simples à faire et à comprendre, il est rapide et facile si le besoin s'en présente, de modifier les analyseurs pour répondre à de nouveaux besoins.

Notons qu'aucune technique de gestion d'erreurs n'a été implantée.
Le fichier yacc finalement écrit contient les spécifications suivantes :

```
%%
FORMDEPART  : FORMET
            ;
FORMET      : FORMET ET FORMEU
            | FORMEU
            ;
FORMEU      : FORMEU EU FORMAU
            | FORMAU
            ;
FORMAU      : FORMAU AU FORMBASE
            | FORMBASE
            ;
FORMULE4    : CARACT
            | VRAI
            | STATE POUVRE NUMBER PFERME
            | CARACT OP NUMBER
            | CARACT MOINS CARACT OP NUMBER
            | POUVRE FORMET PFERME
            | NOT POUVRE FORMET PFERME
            | CARACT POINT POUVRE FORMET PFERME
            ;
OP          : PP
            | PPE
            ;
NUMBER      : INTEGER
            ;
CARACT      : STRING
            ;
%%
```

Le fichier lex finalement écrit contient les spécifications suivantes :

```
%%
[\n\t]+                    {}
[0-9]+                     {return INTEGER;}
state                      {return STATE;}
not                        {return NOT;}
et                         {return ET;}
au                         {return AU;}
eu                         {return EU;}
TRUE                       {return VRAI;}
"."                        {return POINT;}
"-"                        {return MOINS;}
"("                        {return POUVRE;}
")"                        {return PFERME;}
"<"                        {return PP;}
"<="                       {return PPE;}
%%
```

## 2. Analyse orientée objet d'un Model Checker symbolique pour TCTL

**Schéma**

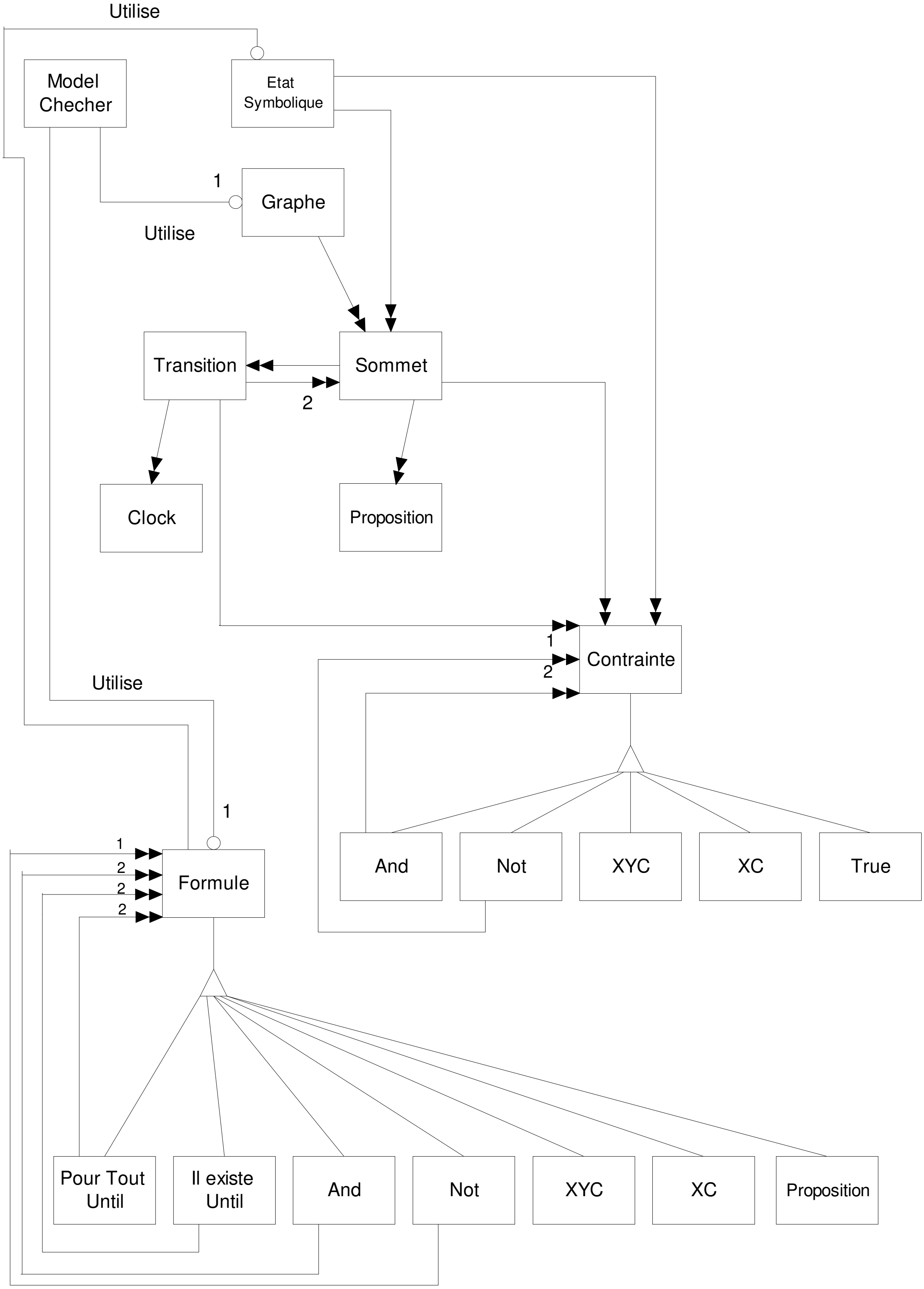



Figure 22

**Explication**

Le Model Checker utilise un graphe représentant le système sur lequel on va faire la vérification et une formule TCTL qui représente la propriété à vérifier. Une formule TCTL peut être de sept types différents ($\forall U$, $\exists U$, and, not, x-y • c, x • c et une simple proposition) et peut être constituée de zéro, une ou deux sous-formules. Le type de formule state(s) est considéré comme un cas particulier de proposition (chaque sommet étant labellé par une proposition spéciale qui est le numéro de sommet le caractérisant, state(s) correspond à vérifier si le sommet est labellé par s) et le type TRUE est considéré comme n'existant pas vraiment dans le système. Une formule TRUE en input du système sera évaluée par celui-ci en manipulant des propositions, négations et conjonctions. Une formule utilise des états symboliques pour le calcul et la représentation de la solution. Un état symbolique étant constitué d'un sommet et de contraintes sur les horloges. Le graphe est également constitué d'un certain nombre de sommets. Chaque sommet est constitué d'un certain nombre de transitions (pouvant être nul) vers d'autres sommets, de contraintes sur les horloges et de propositions. Une transition est constituée d'un ensemble d'horloge à remettre à zéro, de sommets extrémités et de contraintes. Une contrainte peut être de cinq types différents (and, not, x - y • c, x • c et true avec • $\in$ {<, ≤}) et peut être constituée de zéro, une ou deux sous-contraintes.

## 3. Implantation des opérateurs de potentialité et d'inévitabilité

Nous avons vu que les quantificateurs pouvaient être définis comme le plus petit point fixe d'une fonctionnelle. Nous allons voir maintenant comment ces fonctionnelles ont été implantées.

**Implantation de l'opérateur $\exists U$**

Nous avons vu qu'un algorithme pour évaluer l'opérateur $\exists U$ était

$S\,(\phi_1 \,\exists U\, \phi_2) = \vee_{k\in N} X_k$

où $X_0 = S\,(\phi_2)$ et $X_{k+1} = X_k \vee S\,(\phi_1) \succ X_k$

$S(\phi_i)$ représentant l'ensemble des états solution de la formule $\phi_i$.

L'implantation d'un tel algorithme peut se faire comme suit, de manière immédiate :

**Fonction** $\exists U(P_1, P_2)$
**Var** $\chi,\chi'$ ;
**Début**
  $\chi := P_2$ ;
  $\chi' := P_1 \succ \chi$ ;
  **Tant que** $\chi' \neq \chi$ **faire**
    $\chi := \chi \vee \chi'$ ;
    $\chi' := P_1 \succ \chi'$ ;
  **Ftantque**
  **Retourner** $(\chi)$ ;
**Fin**

Notons qu'en théorie, comme à chaque étape sont rajoutés les états à partir desquels il est possible d'atteindre, en vieillissant et en effectuant si nécessaire une transition, un état de l'ensemble solution (les états intermédiaires satisfaisant certaines propriétés) et le nombre de transitions du modèle étant fini, on est sûr que l'algorithme converge en un temps fini. Si le modèle contient n transitions alors dans le pire des cas il faudra n itérations de l'algorithme pour que l'ensemble solution contienne tous les états satisfaisant la formule à vérifier.

En pratique, il faut s'assurer qu'on a une bonne fonction testant l'équivalence entre deux ensembles d'états symboliques, sinon, bien qu'ayant convergé, l'algorithme peut ne pas s'en apercevoir et ne jamais s'arrêter. La fonction utilisée lors de l'implantation consiste à vérifier que les états symboliques ayant le même sommet ont également les mêmes matrices de bornes représentant leurs contraintes. Pour que cette fonction marche effectivement, il faut veiller à ce que les matrices soient toujours en forme normale et il faut que les ensembles de matrices de borne ne contiennent pas de matrices inutiles, c'est à dire des matrices de bornes représentant des contraintes impliquées par une autre matrice de l'ensemble. Pour cela, la méthode d'insertion de matrices dans un ensemble de matrices joue sur la relation d'ordre définie sur les matrices de bornes. Quatre cas peuvent se présenter :

- La matrice à rajouter se trouve déjà dans l'ensemble et dans ce cas elle n'est pas rajoutée une seconde fois.
- Une matrice plus petite se trouve déjà dans l'ensemble. L'ensemble caractéristique de la matrice à rajouter inclus l'ensemble caractéristique de cette matrice. La matrice plus petite est donc retirée de l'ensemble et la matrice à rajouter est effectivement rajoutée.
- Une matrice plus grande se trouve déjà dans l'ensemble. L'ensemble caractéristique de l'ensemble ne serait donc pas modifié si la matrice à rajouter est effectivement rajoutée. Pour cela, la matrice n'est pas rajoutée dans l'ensemble.
- Les autres cas. Dans ce cas, on rajoute la matrice dans l'ensemble.

**Implantation de l'opérateur $\forall U$**

Nous avons vu qu'un algorithme pour évaluer l'opérateur $\forall U$ était

$S(\phi_1 \ \forall U \ \phi_2) = \ \vee_{k\in N} X_k$
où $X_0 = S(\phi_2)$ et $X_{k+1} = X_k \vee (\neg\, (\neg(X_k) \ \exists U \ \neg((S(\phi_1) \vee X_k) \wedge z \leq c)))\ [z \rightarrow 0]$

Pour l'implantation, nous calculons les variables $\gamma_k$ tel que $\gamma_k = \neg X_k$. L'implantation de l'algorithme se fait comme suit :

**Fonction** $\forall U(P_1, P_2)$
**Var** $\gamma,\gamma'$ ;
**Début**
  $\gamma := \neg P_2$ ;
  $\gamma' := \exists U(\gamma, (\neg P_1 \wedge \gamma) \vee z > 1)$ ;
  $\gamma' := \gamma' \ [z := 0]$ ;
  **Tant que** $\gamma \neq \gamma'$ **faire**
    $\gamma := \gamma \wedge \gamma'$ ;
    $\gamma' := \exists U(\gamma, (\neg P_1 \wedge \gamma) \vee z > c)$ ;
    $\gamma' := \gamma' \ [z \rightarrow 0]$ ;
  **Ttantque**
  **Retourner** $(\neg\gamma)$ ;
**Fin**

On peut se demander si un état à partir duquel il est possible d'atteindre un état de $P_2$ en un temps infini (avec les états intermédiaires satisfaisant les propriétés voulues) peut faire converger l'algorithme en un temps infini. Ceci ne se peut puisque, à partir d'un tel état, il est aussi possible de ne jamais atteindre un état de $P_2$. De tels états ne devront donc jamais se retrouver dans l'ensemble solution de l'algorithme et on est sûr que l'algorithme converge en un temps fini.

## 4. Code

Pour l'implantation, le choix du langage s'est porté sur un langage orienté objet à cause des particularités de ces langages qui peuvent être particulièrement bien exploitées ici :

- l'encapsulation permet d'augmenter la modularité du code. Pour cela, aucune donnée ne peut être accédée sans passer par une méthode. Dans le cadre d'optimisations futures, par exemple, avec le code produit, il est aisé de modifier la structure de donnée utilisée pour modéliser une matrice de borne en mémoire sans qu'il y ait de répercussions dans le reste du code.
- l'héritage permet de définir facilement des nouvelles classes. Par exemple, beaucoup de types de listes (ensemble de matrices de bornes, ensemble d'états symbolique, ensemble de propositions,…) sont utilisés dans le programme. On a donc défini une classe virtuelle pure LISTE où un certain nombre de méthodes liées à la gestion de liste sont définies (insertion, suppressions, parcours,…). Lorsqu'on a dû par la suite définir une nouvelle classe de liste, pour laquelle des traitements spécifiques sont définis (la conjonction, négation,… définies pour une liste de matrice de bornes par exemple), en héritant de LISTE, on ne devait s'occuper que des méthodes spécifiques au type de la liste. De plus, les méthodes de la classe LISTE ayant déjà été testées, on était sûr de ne pas avoir de problème avec ces méthodes.
- le polymorphisme a permi, par exemple, lors de l'évaluation d'une formule, de ne pas se soucier du type des sous-formules. Chaque classe représentant un type de formule a une méthode start qu'elle a héritée de sa classe mère FORMULE, où elle est définie en tant que méthode virtuelle pure. Pour chacun de ces types, cette méthode a dû être définie et contient la méthode d'évaluation du type de formule. Lorsqu'on est amené à devoir évaluer une sous-formule, il suffit d'appeler la méthode start de l'objet représentant la sous-formule et, suivant le type de formule, non connu à l'endroit où start est appelée, la méthode correspondant au bon type est effectuée.

Comme le but n'est pas de produire un code optimisé, aucune optimisation importante n'a été réalisée, tant au niveau occupation mémoire, les matrices sont des tableaux en mémoire, qu'au niveau temps d'exécution, rien d'important n'a été fait pour rendre les traitements plus rapides.

Le code de l'algorithme se trouve en annexe de ce travail et est disponible à la bibliothèque du Département Informatique de l'Université Libre de Bruxelles.

## 5. Exemple

A titre d'exemple, on détaille dans cette section l'évaluation de la formule p $\forall U$ q sur le graphe temporisé présenté à la figure 23, en utilisant l'algorithme présenté au chapitre 5.

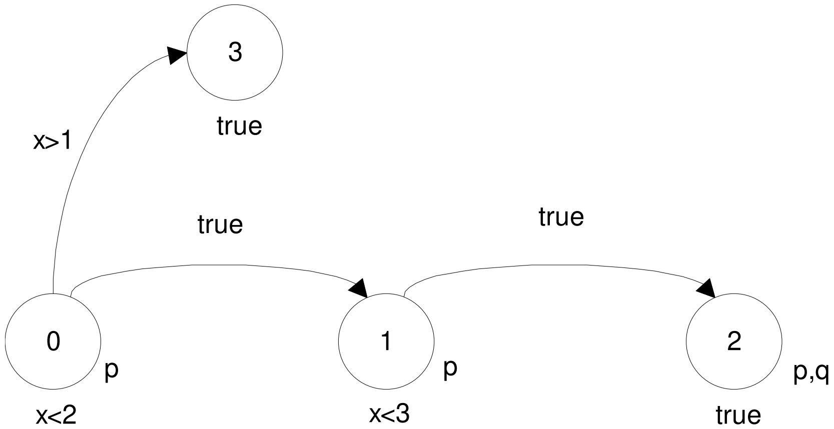



Figure 23

**Appel à l'algorithme $\forall U$**
On calcule d'abord l'ensemble des états où p est vrai ($P_1$) et où q est vrai ($P_2$)
$P_1 = \{(0, x < 2) \vee (1, x < 3) \vee (2, \text{true}) \vee (3, \text{false})\}$
$P_2 = \{(0, \text{false}) \vee (1, \text{false}) \vee (2, \text{true}) \vee (3, \text{false})\}$

$\gamma = \neg P_2 = \{(0, x < 2) \vee (1 ,x < 3) \vee (2, \text{false}) \vee (3 ,\text{true})\}$
$\neg P_1 = \{(0, \text{false}) \vee (1, \text{false}) \vee (2, \text{false}) \vee (3, \text{true})\}$
$\neg P_1 \wedge \gamma = \{(0, \text{false}) \vee (1, \text{false}) \vee (2, \text{false}) \vee (3, \text{true})\}$
$\neg P_1 \wedge \gamma \vee z > 1 = \{(0, x < 2 \wedge z > 1 \wedge x - z < 1) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\}$

$\gamma' = \gamma \exists U \neg P_1 \wedge \gamma \vee z > 1$

**Appel à l'algorithme $\exists U$**
$\chi = \{(0, x < 2 \wedge z > 1 \wedge x - z < 1) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\}$

$P^{\chi}_1 = \{(0, x < 2) \vee (1, x < 3) \vee (2, \text{false}) \vee (3, \text{true})\}$
$\chi' = P^{\chi}_1 \succ \chi$

$\text{Pre}(1, x < 3 \wedge z > 1 \wedge x - z < 2) = (0, x < 2 \wedge z > 1 \wedge x - z < 1)$
$\text{Pre}(2, z > 1) = (1, x < 3 \wedge z > 1 \wedge x - z < 2)$
$\text{Pre}(3, \text{true}) = (0, x > 1 \wedge x < 2)$

$\xi_0 = (x < 2 \wedge z > 1 \wedge x - z < 1) \vee (x > 1 \wedge x < 2)$
$\xi_1 = x < 3 \wedge z > 1 \wedge x - z < 2$

$(0, x < 2) \succ_t (0, (x < 2 \wedge z > 1 \wedge x - z < 1) \vee (x > 1 \wedge x < 2)) = (0 , (x < 2 \wedge x - z < 1) \vee (x < 2)) = (0, x < 2)$
$(1, x < 3) \succ_t (1, x < 3 \wedge z > 1 \wedge x - z < 2) = (1, x < 3 \wedge x - z < 2)$
$(2, \text{false}) \succ_t (2, z > 1) = \succ_t (2, z > 1)$
$(3, \text{true}) \succ_t (3, \text{true}) = (3, \text{true})$

$\chi' = \{(0, x < 2) \vee (1, x < 3 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\} \neq \chi$

Un second tour de boucle est entamé.

$\chi = \chi \vee \chi' = \{(0, x < 2) \vee (1, x < 3 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\}$
$\chi' = P^{\chi}{}_1 \succ \chi$

$\text{Pre}(1, x < 3 \wedge x - z < 2) = (0, x < 2 \wedge x - z < 2)$
$\text{Pre}(2, z > 1) = (1, x < 3 \wedge z > 1 \wedge x - z < 2)$
$\text{Pre}(3, \text{true}) = (0, x > 1 \wedge x < 2)$

$\xi_0 = (x < 2 \wedge x - z < 2) \vee (x > 1 \wedge x < 2)$
$\xi_1 = x < 3 \wedge z > 1 \wedge x - z < 2$

$(0, x < 2) \succ_t (0, x < 2) = (0, x < 2)$
$(1, x < 3) \succ_t (1, x < 3 \wedge z > 1 \wedge x - z < 2) = (1, x < 3 \wedge x - z < 2)$
$(2, \text{false}) \succ_t (2, z > 1) = \succ_t (2, z > 1)$
$(3, \text{true}) \succ_t (3, \text{true}) = (3, \text{true})$

$\chi' = \{(0, x < 2) \vee (1, x < 3 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\} = \chi$

**Retour à l'algorithme $\forall U$**
On trouve ainsi la valeur pour $\gamma$ :
$\gamma' = \chi = \{(0, x < 2) \vee (1, x < 3 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\}$
$\gamma'[z \rightarrow 0] = \{(0, x < 2) \vee (1, x < 2) \vee (2, \text{false}) \vee (3, \text{true})\} \neq \gamma$

il faut donc faire un second tour de boucle.
$\gamma = \gamma \wedge \gamma' = \{(0, x < 2) \vee (1, x < 2) \vee (2, \text{false}) \vee (3, \text{true})\}$

$\neg P_1 \wedge \gamma = \{(0, \text{false}) \vee (1, \text{false}) \vee (2, \text{false}) \vee (3, \text{true})\}$
$\neg P_1 \wedge \gamma \vee z > 1 = \{(0, x < 2 \wedge z > 1 \wedge x - z < 1) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\}$

$\gamma' = \gamma \, \exists U \, \neg P_1 \wedge \gamma \vee z > 1$

**Appel à l'algorithme $\exists U$**
$\chi = \{(0, x < 2 \wedge z > 1 \wedge x - z < 1) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\}$

$P^{\chi}_1 = \{(0, x < 2) \vee (1, x < 2) \vee (2, \text{false}) \vee (3, \text{true})\}$
$\chi' = P^{\chi}_1 \succ \chi$

$\text{Pre}(1, x < 3 \wedge z > 1 \wedge x - z < 2) = (0, x < 2 \wedge z > 1 \wedge x - z < 1)$
$\text{Pre}(2, z > 1) = (1, x < 3 \wedge z > 1 \wedge x - z < 2)$
$\text{Pre}(3, \text{true}) = (0, x > 1 \wedge x < 2)$

$\xi_0 = (x < 2 \wedge z > 1 \wedge x - z < 1) \vee (x > 1 \wedge x < 2)$
$\xi_1 = x < 3 \wedge z > 1 \wedge x - z < 2$

$(0, x < 2) \succ_t (0, (x < 2 \wedge z > 1 \wedge x - z < 1) \vee (x > 1 \wedge x < 2)) = (0 , (x < 2 \wedge x - z < 1) \vee (x < 2)) = (0, x < 2)$
$(1, x < 2) \succ_t (1, x < 3 \wedge z > 1 \wedge x - z < 2) = (1, x < 3 \wedge z > 1 \wedge x - z < 2)$
$(2, \text{false}) \succ_t (2, z > 1) = \succ_t (2, z > 1)$
$(3, \text{true}) \succ_t (3, \text{true}) = (3, \text{true})$

$\chi' = \{(0, x < 2) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\} \neq \chi$

Un second tour de boucle est entamé.

$\chi = \chi \vee \chi' = \{(0, x < 2) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\}$
$\chi' = P^{\chi}_1 \succ \chi$

$\text{Pre}(1, x < 3 \wedge z > 1 \wedge x - z < 2) = (0, x < 2 \wedge z > 1 \wedge x - z < 1)$
$\text{Pre}(2, z > 1) = (1, x < 3 \wedge z > 1 \wedge x - z < 2)$
$\text{Pre}(3, \text{true}) = (0, x > 1 \wedge x < 2)$

$\xi_0 = (x < 2 \wedge z > 1 \wedge x - z < 1) \vee (x > 1 \wedge x < 2)$
$\xi_1 = x < 3 \wedge z > 1 \wedge x - z < 2$

$(0, x < 2) \succ_t (0, x < 2) = (0, x < 2)$
$(1, x < 2) \succ_t (1, x < 3 \wedge z > 1 \wedge x - z < 2) = (1, x < 3 \wedge z > 1 \wedge x - z < 2)$
$(2, \text{false}) \succ_t (2, z > 1) = \succ_t (2, z > 1)$
$(3, \text{true}) \succ_t (3, \text{true}) = (3, \text{true})$

$\chi' = \{(0, x < 2) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\} = \chi$

**Retour à l'algorithme $\forall U$**
On trouve ainsi la valeur pour $\gamma$ :
$\gamma' = \chi = \{(0, x < 2) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, \text{true})\}$
$\gamma'[z \rightarrow 0] = \{(0, x < 2) \vee (1, \text{false}) \vee (2, \text{false}) \vee (3 ,\text{true})\} \neq \gamma$

il faut donc faire un second tour de boucle.
$\gamma = \gamma \wedge \gamma' = \{(0, x < 2) \vee (1, false) \vee (2, false) \vee (3, true)\}$

$\neg P_1 \wedge \gamma = \{(0, false) \vee (1, false) \vee (2, false) \vee (3, true)\}$
$\neg P_1 \wedge \gamma \vee z > 1 = \{(0, x < 2 \wedge z > 1 \wedge x - z < 1) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, true)\}$

$\gamma' = \gamma \, \exists U \, \neg P_1 \wedge \gamma \vee z > 1$

**Appel à l'algorithme $\exists U$**
$\chi = \{(0, x < 2 \wedge z > 1 \wedge x - z < 1) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, true)\}$

$P^{\chi}_1 = \{(0, x < 2) \vee (1, false) \vee (2, false) \vee (3, true)\}$
$\chi' = P^{\chi}_1 \succ \chi$

$Pre(1, x < 3 \wedge z > 1 \wedge x - z < 2) = (0, x < 2 \wedge z > 1 \wedge x - z < 1)$
$Pre(2, z > 1) = (1, x < 3 \wedge z > 1 \wedge x - z < 2)$
$Pre(3, true) = (0, x > 1 \wedge x < 2)$

$\xi_0 = (x < 2 \wedge z > 1 \wedge x - z < 1) \vee (x > 1 \wedge x < 2)$
$\xi_1 = x < 3 \wedge z > 1 \wedge x - z < 2$

$(0, x < 2) \succ_t (0, (x < 2 \wedge z > 1 \wedge x - z < 1) \vee (x > 1 \wedge x < 2)) = (0, (x < 2 \wedge x - z < 1) \vee (x < 2)) = (0, x < 2)$
$(1, false) \succ_t (1, x < 3 \wedge z > 1 \wedge x - z < 2) = (1, x < 3 \wedge z > 1 \wedge x - z < 2)$
$(2, false) \succ_t (2, z > 1) = \succ_t (2, z > 1)$
$(3, true) \succ_t (3, true) = (3, true)$

$\chi' = \{(0, x < 2) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, true)\} \neq \chi$

Un second tour de boucle est entamé.

$\chi = \chi \vee \chi' = \{(0, x < 2) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, true)\}$
$\chi' = P^{\chi}_1 \succ \chi$

$Pre(1, x < 3 \wedge z > 1 \wedge x - z < 2) = (0, x < 2 \wedge z > 1 \wedge x - z < 1)$
$Pre(2, z > 1) = (1, x < 3 \wedge z > 1 \wedge x - z < 2)$
$Pre(3, true) = (0, x > 1 \wedge x < 2)$

$\xi_0 = (x < 2 \wedge z > 1 \wedge x - z < 1) \vee (x > 1 \wedge x < 2)$
$\xi_1 = (x < 3 \wedge z > 1 \wedge x - z < 2)$

$(0, x < 2) \succ_t (0, x < 2) = (0, x < 2)$
$(1, false) \succ_t (1, x < 3 \wedge z > 1 \wedge x - z < 2) = (1, x < 3 \wedge z > 1 \wedge x - z < 2)$
$(2, false) \succ_t (2, z > 1) = \succ_t (2, z > 1)$
$(3, true) \succ_t (3, true) = (3, true)$

$\chi' = \{(0, x < 2) \vee (1, x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, true)\} = \chi$

**Retour à l'algorithme $\forall U$**
On trouve ainsi la valeur pour $\gamma$ :
$\gamma' = \chi = \{(0, x < 2) \vee (1, \ x < 3 \wedge z > 1 \wedge x - z < 2) \vee (2, z > 1) \vee (3, true)\}$
$\gamma'[z \rightarrow 0] = \{(0, x < 2) \vee (1, false) \vee (2, false) \vee (3, true)\} = \gamma$

L'ensemble des états du graphe satisfaisant p $\forall U$ q est l'inverse de $\gamma$, c'est à dire
$\{(0, false) \vee (1, x < 3) \vee (2, true) \vee (3, false)\}$

Notons que les itérations des processus itératifs peuvent être plus nombreuses. Par exemple, si la valeur de c était plus grande que 1, si le graphe avait plus de sommets,…

## 6. Comparaison avec Kronos

Les travaux de Sergio Yovine [Yov93] et d'Alfredo Olivero [Oli94] sont à la base d'un outil de vérification de systèmes temporisés implantant l'algorithme de Model Checking TCTL qui a été implanté dans le cadre de ce travail. Cet outil est appelé Kronos. On a effectué une comparaison entre cet outil et le programme implanté dans le cadre de ce mémoire. Cette comparaison permet de déterminer où se situent les performances de l'implantation effectuée pour ce travail par rapport à un outil équivalent utilisé dans l'industrie et permet de savoir si des optimisations importantes peuvent être envisagée dans un développement futur ou non.
C'est la version 2.1a de Kronos qui a été utilisée lors de cette comparaison. Cette version implante une autre méthode d'évaluation de formules TCTL, appelée méthode "en avant". Cette méthode n'ayant rien à voir avec la méthode implantée pour ce travail (dite méthode "en arrière"), elle n'a pas été utilisée dans le cadre de cette comparaison.
On a utilisé pour cette comparaison deux graphes temporisés représentatifs des graphes temporisés sur lesquels on peut être amené à vérifier des propriétés. Les figures 24 et 25 représentent ces deux graphes temporisés. Lors des évaluations de formules, les deux programmes ont toujours donné la réponse attendue. Lors de l'évaluation de formules contenant un opérateur $\forall U$, la constante c utilisée dans l'algorithme de cet opérateur est initialisée à la même valeur pour les deux outils. On a choisi la valeur 180. Les résultats sont reportés dans deux tableaux, suivant le graphe temporisé utilisé, et les valeurs sont données en secondes. La colonne rapport contient, pour chaque ligne, le rapport entre le temps mis par notre outil et le temps pris par Kronos pour évaluer la formule.
Ces évaluations ont été effectuées sur une Ultra sparc 1/140 tournant sous solaris 2.5 (SunOS 5.5) disposant de 64 Mb de ram.

Graphe temporisé 1 :
Nombre de sommets : 8
Nombre de transitions : 11
Horloges : 1

| **Formule** | **Kronos** | **Outil** | **Rapport** |
|---|---|---|---|
| $x < 2$ | 0.03 | 0.05 | 1.7 |
| $\neg(\neg p \wedge \neg r)$ | 0.03 | 0.06 | 2 |
| $\neg(\neg p \wedge \neg r)\ \exists U$ CS | 0.04 | 0,51 | 12.75 |
| $\neg(\neg p \wedge \neg r)\ \forall U$ CS | 0.06 | 1.45 | 24.2 |
| $x.(p\ \exists U\ (r \wedge x < 2))$ | 0.04 | 0.36 | 9 |
| $x.(p\ \forall U\ (r \wedge x < 2))$ | 0.05 | 1 | 20 |

Graphe temporisé 2 :
Nombre de sommets : 16
Nombre de transitions : 30
Horloges : 5

| **Formule** | **Kronos** | **Outil** | **Rapport** |
|---|---|---|---|
| $\neg$(CONNECT $\wedge$ $\neg$(x.(true $\forall U$ (LIBRE $\wedge$ x $\leq$ 180)))) | 0.31 | 6min 31.44 | 1262.7 |
| x < 100 | 0.06 | 3.81 | 63.5 |
| true $\exists U$ ($\neg$($\neg$(CHIFF1) $\wedge$ $\neg$(LIBRE))) | 0.09 | 16.75 | 186.1 |
| $\neg$($\neg$(CHIIF1) $\wedge$ $\neg$(CHIFF2)) | 0.05 | 3.05 | 61 |
| CHIFF1 | 0.05 | 2.28 | 45.6 |
| true $\forall U$ LIBRE | 6.76 | 7min 14.81 | 64.3 |

La comparaison sur le premier graphe temporisé permet de conclure que, sur de très petits graphes Kronos évalue déjà plus vite, de manière significative, les opérateurs $\exists U$ et $\forall U$. Les performances des deux programmes pour les autres opérateurs sont, elles, proches. En regardant les résultats du second graphe, on constate que les rapports s'accentuent avec l'augmentation d'états et d'horloges et les temps d'exécution de notre programme deviennent déjà fort important lors d'évaluations de formules contenant les opérateurs $\exists U$ et $\forall U$. De plus, on peut constater que notre outil, lors de l'évaluation de formules simples ne nécessitant que des manipulations simples de contraintes, est beaucoup plus lent que Kronos et on en conclut donc que la représentation des contraintes dans Kronos est optimisée par rapport à de simples matrices de bornes et permet de mieux gérer l'ensemble des horloges.
On peut constater le rapport considérable entre les performances des deux programmes lors du calcul de la formule $\neg$(CONNECT $\wedge$ $\neg$(x.(true $\forall U$ (LIBRE $\wedge$ x $\leq$ 180)))). Cela est d'autant plus surprenant lorsqu'on regarde le rapport de performance lié à l'évaluation de la formule true $\forall U$ LIBRE. Cette différence entre les deux rapports s'explique par le fait que Kronos évalue la propriété d'inévitabilité bornée de manière beaucoup rapide que la propriété d'inévitablité non bornée. Or, l'évaluation est, en suivant l'algorithme implanté pour ce travail, dans les deux cas plus ou moins longue.
Dans les deux cas, lors de la première itération du processus itératif calculant le $\forall U$, le processus imbriqué commence avec un ensemble solution d'états symboliques dont les contraintes sont les invariants de sommet et z > 180 (où z est l'horloge utilisée dans la fonctionnelle du $\forall U$). La contrainte de l'état symbolique de sommet 7 est donc $x_1 \leq 60 \wedge x_2 \leq 3 \wedge z > 180$. A la première itération, le calcul du prédécesseur en un pas rajoute une matrice dans l'état symbolique de sommet 7. Cette matrice est de l'information provenant de l'état symbolique de sommet 8 et contient, entre autre, la contrainte $57 \leq x_1 \leq 60 \wedge x_2 \leq 3 \wedge z > 177$. De ce sommet 7, il est possible d'effectuer une transition remettant $x_2$ à zéro chaque fois que cette variable vaut 3. Cette transition permet de rajouter lors de la seconde itération une matrice contenant $54 \leq x_1 \leq 57 \wedge z > 174$ dans l'état symbolique de sommet 7, etc. Il faut donc vingt itérations avant qu'il n'ait plus de matrice rajoutée. La convergence du processus n'est donc pas rapide dans les deux cas et on peut donc ainsi expliquer le fait que notre outil n'évalue pas la formule d'inévitabillité bornée de manière beaucoup plus rapide. On ne peut donc que conclure que dans le cadre d'évaluation de propriétés d'inévitabilité bornée, Kronos permet une évaluation très rapide.
En conclusion, comme on pouvait s'y attendre, les performances de Kronos sont toujours supérieures à celle de notre outil et les performances de celui-ci sont en général mauvaises.

On a pu mettre en évidence le fait que la gestion des contraintes dans Kronos a été fortement optimisée par rapport à de simples matrices de bornes, comme implantées dans le cadre de ce travail. De plus, certains résultats laissent à penser que des techniques spéciales ont été mises en œuvre pour une évaluation extrêmement rapide dans des cas particuliers.

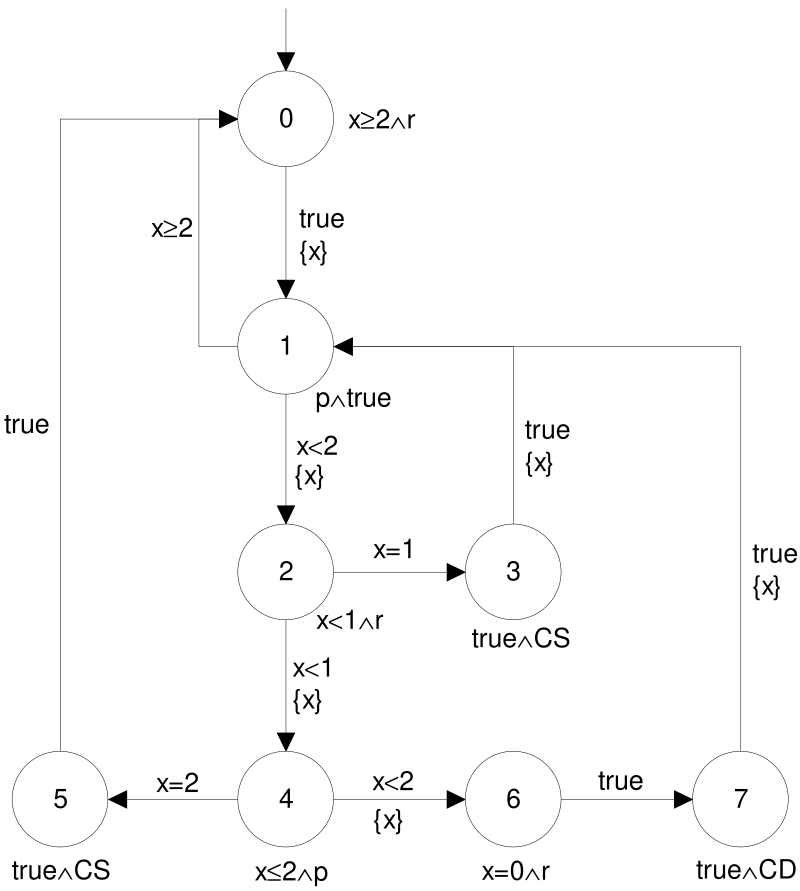



Figure 24:Graphe temporisé d'une souris

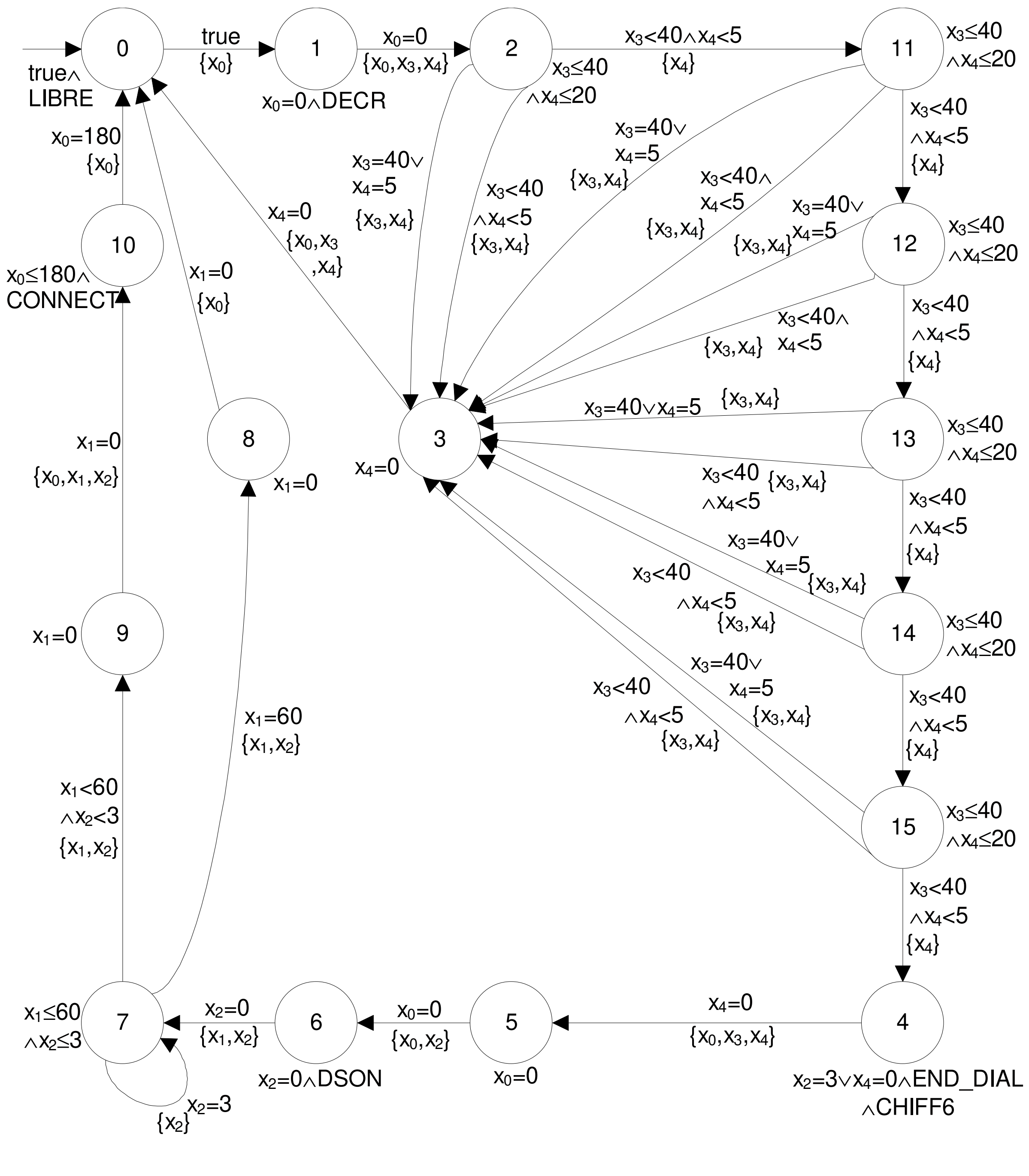



Figure 25:graphe temporisé d'un
central téléphonique

# Chapitre 7

# Une logique temporisée orientée action

Aucune logique temporelle temporisée orientée action n'a été trouvée dans le cadre de ce travail. Pour cette raison, on a créé une nouvelle logique ayant ces propriétés. Deux algorithmes de Model Checking utilisant cette nouvelle logique pour spécifier les propriétés à vérifier ont également été créés. Ces algorithmes se basent sur l'algorithme de Model Checking présenté au chapitre 5.
On présente dans ce chapitre ce nouveau formalisme et les deux algorithmes associés.

## 1. Motivation

Les algèbres de processus (temporisées) permettent de définir des systèmes en décrivant les ordonnancements d'actions possibles. Il semble donc naturel que les logiques utilisées pour spécifier des propriétés à vérifier soient des logiques orientées actions. Or les logiques temporelles orientées actions existantes semblent, comme nous l'avons vu, n'être que des logiques, linéaires ou de branchement, non temporisées. Le fait que ces logiques soient non temporisées limite le champ d'utilisation de telles logiques à la vérification de systèmes dits non temps réel. Il serait pour cette raison intéressant de créer une logique temporisée orientée action. Pour cela, nous allons nous inspirer des versions temporisée et orientée action de CTL, TCTL et ACTL.
La logique que nous présentons ici a été baptisée ATCTL.

## 2. Présentation de la logique ATCTL

La logique ATCTL est un exemple de logique temporelle de branchement temporisée orientée actions.

L'ensemble des contraintes spécifiables sur les actions est l'ensemble des formules d'actions défini au chapitre trois dans le cadre de la présentation de la logique ACTL.

La syntaxe de la logique ATCTL est définie comme suit :
Une formule est une formule de la logique ATCTL si elle peut être construite à partir de la grammaire :

$\phi \rightarrow \text{TRUE} \mid x \text{ op } c \mid x - y \text{ op } c \mid \neg\phi \mid \phi \wedge \phi \mid \phi\ {}_{\chi}\exists U_{\chi'}\ \phi \mid \phi\ \exists U_{\chi}\ \phi \mid \phi\ {}_{\chi}\forall U_{\chi'}\ \phi \mid \phi\ \forall U_{\chi}\ \phi$
$\mid x.\ \phi$

$\text{op} \rightarrow < \mid \leq$

où $c \in 9$.
$\chi$, $\chi'$ sont des formules d'actions.

Comme la sémantique de TCTL, la sémantique d'ATCTL est ici directement définie sur un graphe temporisé et les valuations associent également une valeur aux horloges de la logique.
On associe une action à chaque pas d'une séquence de pas :
$\forall i \in \angle$ : $a_i$ est l'action associée au $i^{\text{ème}}$ pas d'une séquence de pas et est :
- l'action effectuée lors de ce pas si une action est effectuée (y compris $\tau$).
- $\tau$ sinon.

Pour un graphe temporisé G (S, H, $\delta$, $s_{init}$, $\Delta$), un état $q = <s, \nu>$ et une formule $\phi$ de ATCTL, la relation de satisfaction $q \models \phi$ est définie de manière inductive comme suit :

$q \models \text{TRUE}$ est toujours vrai.
$q \models \neg\phi$ ssi $q \not\models \phi$.
$q \models x \text{ op } c$ ssi $\nu(x) \text{ op } c$.
$q \models x - y \text{ op } c$ ssi $\nu(x) - \nu(y) \text{ op } c$.
$q \models \phi_1 \wedge \phi_2$ ssi $q \models \phi_1$ et $q \models \phi_2$.
$q \models z.\ \phi$ ssi $<s, \nu\ [z \rightarrow 0]> \models \phi$.
$q \models \phi_1\ {}_{\chi}\exists U_{\chi'}\ \phi_2$ ssi $\exists\sigma$ dans G où $q_0 = q$, $\exists(q', i) \in \chi(\sigma)$ avec $i > 0 \mid q' \models \phi_2$ et $a_{i-1} \models \chi'$ et $\forall(q'', j) \leq (q', i)$, $q'' \models \phi_1 \vee \phi_2 \wedge \forall\ 0 \leq j < i - 1 : (a_j \models \chi \vee a_j = \tau)$.
$q \models \phi_1\ {}_{\chi}\forall U_{\chi'}\ \phi_2$ ssi $\forall\sigma$ dans G où $q_0 = q$, $\exists(q', i) \in \chi(\sigma)$ avec $i > 0 \mid q' \models \phi_2$ et $a_{i-1} \models \chi'$ et $\forall(q'', j) \leq (q', i)$, $q'' \models \phi_1 \vee \phi_2 \wedge \forall\ 0 \leq j < i - 1 : (a_j \models \chi \vee a_j = \tau)$.
$q \models \phi_1\ {}_{\chi}\exists U\ \phi_2$ ssi $\exists\sigma$ dans G où $q_0 = q$, $\exists(q', i) \in \chi(\sigma) \mid q' \models \phi_2$ et $\forall(q'', j) \leq (q', i)$, $q'' \models \phi_1 \vee \phi_2 \wedge \forall\ 0 \leq j \leq i - 1 : (a_j \models \chi \vee a_j = \tau)$.
$q \models \phi_1\ {}_{\chi}\forall U\ \phi_2$ ssi $\forall\sigma$ dans G où $q_0 = q$, $\exists(q', i) \in \chi(\sigma) \mid q' \models \phi_2$ et $\forall(q'', j) \leq (q', i)$, $q'' \models \phi_1 \vee \phi_2 \wedge \forall\ 0 \leq j \leq i - 1 :(a_j \models \chi \vee a_j = \tau)$.

où $c \in 9$.
op $\in \{<, \leq\}$.
z est une horloge qui n'est pas utilisée dans G.
$\sigma$ est une séquence de pas et $\chi(\sigma)$ est son support.

Comme dans le cadre de TCTL, on ajoute que les séquences de pas sur lesquelles sont évaluées les formules ATCTL sont des séquences où le temps s'écoule au-delà de tout instant. Ceci est rajouté pour les mêmes raisons qu'en TCTL.

La formule $\phi_1\ {}_{\chi}\exists U_{\chi'}\ \phi_2$ ($\phi_1\ {}_{\chi}\forall U_{\chi'}\ \phi_2$) spécifie le fait qu'il est possible (inévitable) d'atteindre un état satisfaisant un état satisfaisant $\phi_2$ en effectuant une action satisfaisant la formule d'actions $\chi'$, tous les états intermédiaires satisfaisant $\phi_1$ et les actions intermédiaires effectuées satisfaisant $\chi$ ou étant l'action invisible. La différence entre ${}_{\chi}\exists U_{\chi'}$ et ${}_{\chi}\exists U$ est que le premier opérateur impose qu'au moins une action soit effectuée, le second n'imposant qu'une contrainte sur les actions éventuellement effectuées.
ATCTL étant un mélange entre TCTL et ACTL, il est possible de spécifier des propriétés de sûreté, vivacité et d'équité de la même manière qu'en ACTL. Le rajout d'une notion de temps quantifié permet en plus de pouvoir spécifier des propriétés d'atteignabilité bornée. Il est maintenant possible, par exemple, de spécifier qu'une action a inévitablement atteinte est suivie inévitablement avant dix unités de temps par l'action b avec la formule true ${}_{\text{true}}\forall U_{\text{a}}$ (x.(true ${}_{\text{true}}\forall U_{\text{b}}$ x < 10)).
Comme en ACTL, les formules intéressantes comportent souvent un certain nombre de sous-formules true. De nouveau, il est intéressant d'utiliser des notations spéciale pour les raccourcis. Par exemple, on peut définir $\{\bullet c\}\exists U\langle\chi'\rangle$ comme étant x.(true ${}_{\text{true}}\exists U_{\chi'}$ (x • c)) .

## 3. Algorithme de vérification

L'idée de l'algorithme de vérification pour la logique ATCTL est la même que celle de l'algorithme de vérification de la logique ACTL. Deux fonctions de transformations sont définies :
- La première ($f_g$) prend un graphe temporisé et le transforme en un second graphe temporisé sur lequel on va pouvoir faire la vérification d'une formule TCTL. Les deux graphes représentent les mêmes systèmes, ayant le "même" comportement.
- La deuxième ($f_f$) prend une formule de la logique ATCTL et la transforme en une formule de la logique TCTL. La transformation d'une formule ATCTL, spécifiant une propriété $\alpha$, tient compte de la manière dont est transformé le modèle pour que la formule TCTL résultante représente cette même propriété $\alpha$.

Après application des deux fonctions, l'algorithme de vérification pour la logique TCTL est appliqué sur le graphe temporisé donné par $f_g$ et vérifie la formule TCTL résultant de la fonction $f_f$.

Tout repose donc sur les deux fonctions sus-citées.

**La fonction $f_g$**
L'idée de l'algorithme étant de se ramener à TCTL, le graphe temporisé sur lequel on veut vérifier des propriétés doit être modifié. TCTL ne travaillant pas au niveau des transitions mais des états, le fait qu'une action soit effectuée ne doit plus être spécifiée au niveau des transitions mais des sommets. Comme en ACTL, l'idée est de placer un état intermédiaire au niveau de toutes les transitions labellées par une action différente de $\tau$. Ces états n'étant pas des états du système mais des états rajoutés pour la vérification de propriétés, on rajoute une nouvelle horloge t au système. Cette nouvelle horloge permet de spécifier le fait qu'on passe par ces états intermédiaires de manière instantanée. Des contraintes doivent être placées au niveau des transitions et de l'état intermédiaire pour ne pas modifier le comportement du modèle. De plus, on labelle les états qui se trouvaient déjà dans le modèle de départ par un

symbole $\perp$ pour pouvoir les différencier des états rajoutés. Les états rajoutés sont, eux, labellés par l'action effectuée. Pour une formule ATCTL $\phi_1\ {}_{\chi}\exists U_{\chi'}\ \phi_2$, les contraintes $\chi$ et $\chi'$ devront être vérifiées pour des états rajoutés et $\phi_1$ et $\phi_2$ devront être vérifiées par des états qui se trouvaient déjà dans le modèle de départ. Il est pour cela important de pouvoir distinguer les deux types d'états. $\tau$ étant une action interne à laquelle on ne s'intéresse pas, lorsqu'on rencontre une transition labellée par une telle action, on ne touche pas à la transition. On labelle simplement les états extrémités par $\perp$. La figure 26.a montre les deux types de transitions qu'on pourra retrouver dans le graphe temporisé de départ et la figure 26.b montre ces mêmes transitions après transformation. $H_T$ et les $C_i$ représentent, respectivement, un ensemble d'horloges et des invariants de sommet. Les états du modèle de départ sont représentés par un bord épais.

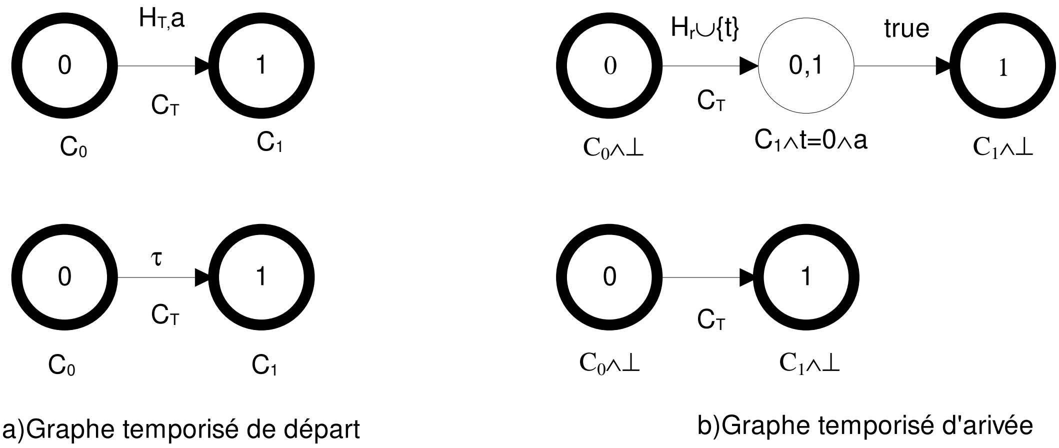



Figure 26: Transformation du graphe temporisé

De manière plus formelle. En appliquant la fonction $f_g$ sur un graphe temporisé $<S, H, \delta, s_{init}, \Delta>$, on obtient le graphe temporisé $<S', H', \delta', s'_{init}, \Delta'>$ où

- $S' = S$ où les états sont labellés par $\{\perp\} \cup \{(s, s')$ labellé par $\{a\} \mid <s, a, C_T, H_T, s'> \in \delta\}$.
- $H' = H \cup \{ t \}$ où t est une horloge ne se trouvant pas dans H. Cette horloge t est utilisée lorsqu'on passe par un sommet rajouté lors de la transformation. Comme elle est remise à zéro lorsqu'on effectue une transition vers un tel sommet rajouté et qu'on ne peut rester dans de tels sommets qu'avec $t = 0$, elle permet d'obliger un passage instantané par ces sommets rajoutés. Après être passé par un sommet rajouté, la valeur de t n'étant plus utilisée, elle peut être réutilisée lors d'un nouveau passage par un sommet rajouté.
- $\delta' = \{<s, \emptyset, C_T, H_T \cup \{t\}, (s, s')>, <(s, s'), \emptyset, TRUE, \emptyset, s'> \mid <s, a, C_T, H_T, s'> \in \delta\} \cup \{<s, \emptyset, C_T, H_T, s'>$ tel que $<s, \tau, C_T, H_T, s'> \in \delta\}$.
- $s'_{init} = s_{init}$.
- $\Delta' : \forall s \in S : \Delta'(s) = \Delta(s)$.
  $\forall s \notin S : s = (s_1, s_2) \rightarrow \Delta'(s) = \Delta(s_2) \wedge t = 0$.

**La fonction $f_f$**

La fonction $f_f$, qui transforme une formule de la logique ATCTL en une formule TCTL, est définie de manière inductive de la manière suivante :

$f_f(\text{TRUE}) = \bot$
$f_f(x \text{ op } c) = \bot \wedge x \text{ op } c$
$f_f(x - y \text{ op } c) = \bot \wedge x - y \text{ op } c$
$f_f(\neg\phi) = \bot \wedge \neg f_f(\phi)$
$f_f(x.\ \phi) = x.\ f_f(\phi)$
$f_f(\phi_1 \wedge \phi_2) = f_f(\phi_1) \wedge f_f(\phi_2)$
$f_f(\phi_1\ {}_{\chi}\exists U_{\chi'}\ \phi_2) = \bot \wedge (\neg(\neg(\chi \wedge \neg\bot) \wedge \neg f_f(\phi_1)))\ \exists U\ (\neg\bot \wedge ((\chi' \wedge \neg\bot)\ \exists U\ f_f(\phi_2)))$
$f_f(\phi_1\ \exists U_{\chi}\ \phi_2) = \bot \wedge (\neg(\neg(\chi \wedge \neg\bot) \wedge \neg f_f(\phi_1)))\ \exists U\ (\bot \wedge f_f(\phi_2))$
$f_f(\phi_1\ {}_{\chi}\forall U_{\chi'}\ \phi_2) = \bot \wedge (\neg(\neg(\chi \wedge \neg\bot) \wedge \neg f_f(\phi_1)))\ \forall U\ (\neg\bot \wedge ((\chi' \wedge \neg\bot)\ \exists U\ f_f(\phi_2)))$
$f_f(\phi_1\ \forall U_{\chi}\ \phi_2) = \bot \wedge (\neg(\neg(\chi \wedge \neg\bot) \wedge \neg f_f(\phi_1)))\ \forall U\ (\bot \wedge f_f(\phi_2))$
où op $\in \{\leq, <\}$
Remarque : étant donné que tout état rajouté n'a qu'un successeur, la formule $\neg\bot \wedge ((\chi' \wedge \neg\bot)\ \exists U\ f_f(\phi_2))$ pourrait être remplacée par $\neg \bot \wedge ((\chi' \wedge \neg\bot)\ \forall U\ f_f(\phi_2))$.

L'idée est de transformer les formules d'actions en sous-formules d'une formule TCTL, les actions se trouvant maintenant au niveau des sommets. Les formules TCTL résultant de la transformation doivent tenir compte qu'un chemin dans le modèle transformé contient des états du modèle de départ et des états qui sont là pour spécifier qu'une action a été effectuée. L'intégration des contraintes des formules ATCTL doit se faire dans une formule TCTL en spécifiant sur quel type d'état elles doivent être vérifiées. On veille également à ce que seuls les états qui se trouvaient dans le modèle de départ se trouvent dans l'ensemble caractéristique des formules TCTL.
Regardons la transformation
$f_f(\phi_1\ {}_{\chi}\exists U_{\chi'}\ \phi_2) = \bot \wedge (\neg(\neg(\chi \wedge \neg\bot) \wedge \neg f_f(\phi_1)))\ \exists U\ (\neg\bot \wedge ((\chi' \wedge \neg\bot)\ \exists U\ f_f(\phi_2)))$
Ce que demande la formule ATCTL, c'est d'atteindre finalement un état satisfaisant $\phi_2$ en effectuant une action satisfaisant la contrainte $\chi'$. Pour cela, en TCTL, on calcule l'ensemble caractéristique de $(\neg\bot \wedge ((\chi' \wedge \neg\bot)\ \exists U\ f_f(\phi_2)))$. Comme partant d'un état rajouté (satisfaisant $\neg\bot$), on ne peut effectuer qu'une transition vers un état qui se trouvait dans le graphe temporisé de départ, cet ensemble caractéristique contient tous les états rajoutés à partir desquels, en effectuant une transition, on atteint un état du modèle de départ satisfaisant $\phi_2$.
De plus, la formule ATCTL demande que toutes les actions effectuées avant d'effectuer l'action satisfaisant $\chi'$ satisfassent $\chi$ et que tous les états précédemment visités satisfassent $\phi_1$. Cela se traduit dans la formule TCTL par la recherche des états à partir desquels on atteint un état satisfaisant $(\neg\bot \wedge ((\chi' \wedge \neg\bot)\ \exists U\ f_f(\phi_2)))$ avec tous les états intermédiaires satisfaisant soit $\chi$ si l'état est un état représentant une action, soit $\phi_1$ sinon. Ceci est spécifié par $\neg(\neg(\chi \wedge \neg\bot) \wedge \neg f_f(\phi_1))$. On ne demande pas explicitement que les états satisfaisant $S(\phi_1)$ et $S(\phi_2)$ soient des états du modèle de départ car seuls de tels états peuvent se trouver dans l'ensemble caractéristique de ces formules.
Finalement, on demande de ne garder dans l'ensemble caractéristique de la formule TCTL que les états du modèle de départ (labellés par $\bot$).

Les fonctions de transformation auraient pu être définies de manière différente. Comme le montre la figure 27, la transformation du modèle aurait pu se faire en dédoublant les sommets de telle manière qu'il n'y ait qu'une action possible pour atteindre chaque sommet et en labellant chaque sommet par l'action effectuée pour y accéder. A côté de cela, la fonction $f_f$ doit pouvoir transformer, entre autre, une formule ATCTL $\phi_1\ {}_{\chi}\exists U_{\chi'}\ \phi_2$ en une formule TCTL correspondante. On pourrait penser que la transformation donne $\neg(\phi_2 \wedge \chi') \wedge ((\phi_1 \wedge (\chi \vee \tau))\ \exists U\ (\phi_2 \wedge \chi'))_{(1)}$ auquel il faut ajouter une partie de formule spécifiant qu'on doit également retrouver dans l'ensemble caractéristique de la formule les états, satisfaisant $\phi_1$ mais n'étant pas labellé par une action satisfaisant $\chi$, à partir desqules il est possible d'atteindre un état satisfaisant (1) en effectuant une transition (après éventuellement avoir vieilli). Or, la sémantique des opérateurs temporels TCTL $\exists U$ et $\forall U$ ne permet pas de jouer sur le nombre de transitions effectuées. Cette partie de formule à rajouter n'est pour cela pas spécifiable en TCTL et il n'est pas possible, même en essayant de transformer les formules ATCTL d'une autre manière, de définir $f_f$.
On pourrait penser à rajouter en plus, dans chaque sommet, des propositions spéciales permettant de savoir quelles sont les actions disponibles dans un état. Cette méthode ne résout pas le problème car elle ne permet pas de savoir quelle transition correspond à quelle action.
Pour cette raison, on pourrait finalement penser à dédoubler les sommets de telle manière qu'il n'y ait au plus qu'une transition possible à partir de chaque sommet. Malheureusement, on perd la propriété de pouvoir choisir l'action à entreprendre dans un état. Par exemple, si un graphe temporisé contient un sommet à partir duquel on peut soit effectuer une action b permettant d'aboutir à ce même sommet, soit une action a aboutissant à un autre sommet, la transformation du graphe ne sera pas possible.

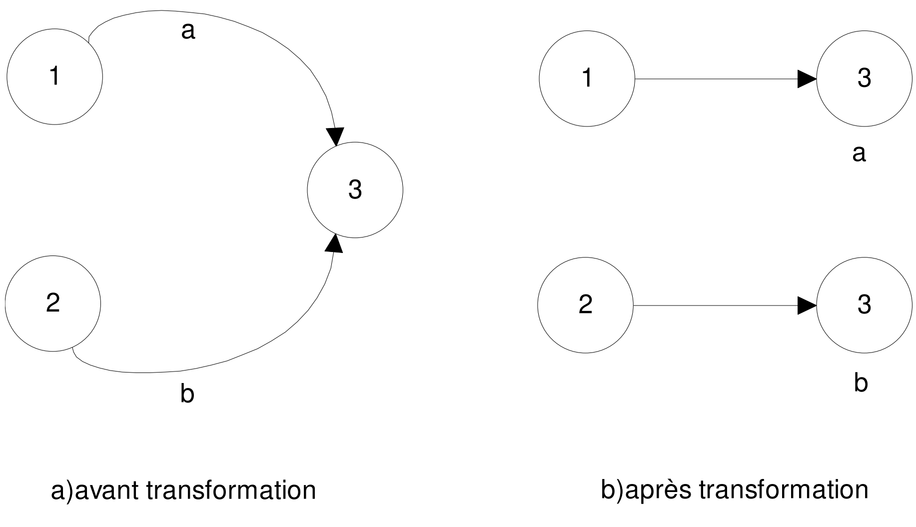



Figure 27

Le fait de se ramener à quelque chose de déjà connu est intéressant d'un point de vue à la fois pratique et théorique. Dans les deux cas, on ne doit pas s'occuper du plus difficile, c'est à dire l'évaluation en elle-même. Dans le cadre de ce travail, on a implanté un précompilateur implantant $f_f$ et $f_g$, dont l'output est utilisé comme input du Model Checker TCTL qui a été implanté précédemment. Néanmoins, la méthode entraîne une augmentation importante du nombre d'états du graphe temporisé. Si on admet qu'en moyenne une action sur deux dans un graphe temporisé est l'action invisible alors, partant d'un graphe ayant n sommets et m

transitions, après application de la fonction $f_g$, le graphe obtenu aura, en moyenne, n + m/2 sommets et 3m/2 transitions. Dans le pire des cas, c'est à dire où toutes les actions sont des actions invisibles, ce qui n'est heureusement pas courant, le graphe aura n + m sommets et 2m transitions.
On remarque que la fonction de transformation augmente également le nombre d'opérateurs d'une formule. On va tenter de donner un poids à $f_f$ pour chaque opérateur et formule de base.
Lors de la comparaison avec Kronos, on a pu constater que les opérateurs TCTL $\forall U$ et $\exists U$ étaient bien plus lourds à calculer que les autres opérateurs. En effet, l'évaluation de ces autres opérateurs ne consiste à faire qu'une opération alors que les quantificateurs sont calculés à l'aide d'un processus itératif où un certain nombre d'opérations sont effectuées à chaque étape. Pour cette raison, les opérateurs autres que $\forall U$ et $\exists U$ utilisés lors de la transformation des opérateurs ATCTL sont considérés comme négligeables.
Comme, lors de la transformation d'une formule ATCTL de conjonction, négation, remise à zéro d'horloges, true et d'une contrainte sur des horloges, on ne rajoute qu'au plus une conjonction et la formule $\perp$, le poids de la transformation dans ces cas-là est considéré comme nul.
Si on regarde la sémantique de l'opérateur ATCTL ${}_\chi\exists U$, on constate que celle-ci est fort proche de la sémantique de l'opérateur TCTL $\exists U$. En effet, l'évaluation des deux opérateurs est la même sauf que dans le cas de l'opérateur ATCTL on regarde les actions effectuées, ce qui a priori ne semble pas être une opération qui nécessite du temps. De plus, la transformation liée à l'opérateur ${}_\chi\exists U$ d'une formule ATCTL consiste à remplacer cet opérateur par un opérateur $\exists U$ et un certain nombre d'autres opérateurs considérés comme négligeables. Pour ces raisons, le surcoût de la transformation pour cet opérateur est négligeable. Le même raisonnement peut être tenu pour l'opérateur ${}_\chi\forall U$.
En ce qui concerne les opérateurs ATCTL ${}_\chi\exists U_{\chi'}$ et ${}_\chi\forall U_{\chi'}$, ceux-ci sont respectivement remplacés dans la formule TCTL par deux opérateurs $\exists U$, et un opérateur $\forall U$ et $\exists U$. Le remplacement de ces opérateurs se fait donc par deux opérateurs lourds à évaluer de TCTL et dans ces cas-ci, le surcoût de la transformation pourrait être considéré comme non négligeable. Néanmoins, l'opérateur $\exists U$ se calcule de manière beaucoup plus rapide que l'opérateur $\forall U$ et le calcul du $\exists U$ dans la sous-formule $(\chi' \wedge \neg\perp)\ \exists U\ f_f(\phi_2)$ converge rapidement (en une itération) puisqu'il calcule l'ensemble contenant les états satisfaisant $\chi'$ à partir desquels on peut atteindre en un pas un état satisfaisant la propriété $f_f(\phi_2)$. On peut donc considérer que la transformation dans ces cas-ci n'entraîne qu'un faible surcoût.
En conclusion, on ne considère pas la transformation des formules comme étant un facteur qui alourdit la résolution du problème. Par contre, le nombre d'états rajoutés n'est pas à négliger. En effet, le temps de résolution d'une formule dépend fortement du nombre de sommets du modèle et également du nombre d'horloges (on en rajoute une).
Pour des raisons de performance, il serait donc intéressant d'avoir un algorithme qui ne touche pas au modèle.

**Code**
Le code de l'algorithme se trouve en annexe de ce travail et est disponible à la bibliothèque du Département Informatique de l'Université Libre de Bruxelles.

## 4. Second algorithme de vérification

La seule différence entre la vérification d'une formule TCTL et d'une formule ATCTL est que, dans le second cas, les transitions doivent satisfaire certaines propriétés. Il serait donc naturel de baser l'algorithme de vérification de formules ATCTL sur celui de formules TCTL, en modifiant les opérateurs prédécesseurs instantanés et prédécesseurs en un pas, de telle manière qu'ils permettent la vérification de conditions sur les transitions effectuées.

Une formule d'actions étendue est une formule d'actions dans laquelle on peut retrouver $\tau$, c'est à dire une formule générée à partir de la grammaire

$\chi \rightarrow a \mid \neg\chi \mid \chi \wedge \chi$

ou $a \in \mathtt{A}$ (Rappelons que $\tau \in \mathtt{A}$)

Pour un état symbolique $<s, \varphi>$ et une formule d'actions étendue $\chi$, l'opérateur prédécesseurs instantanés sous la condition $\chi$ de $<s, \varphi>$, noté $pre_{\{\chi\}}(<s, \varphi>)$, caractérise l'ensemble des états à partir desquels on peut atteindre un état de l'ensemble représenté par $<s, \varphi>$ en effectuant une action satisfaisant $\chi$.

Calculer cet opérateur revient à calculer les prédécesseurs instantanés en ne tenant compte que des transitions labellées par une action satisfaisant $\chi$. Cela revient à calculer

$pre_{\{\chi\}}(<s, \varphi>) = \cup_{(s',a,\varphi,H',s)\in\delta \wedge a \models \chi}\, pre_{(s',a,\varphi,H',s)}(<s, \varphi>)$

Pour une formule d'actions étendue $\chi$ et deux ensembles d'états $Q_1$ et $Q_2$, le prédécesseurs en un pas sous la condition $\chi$ de $Q_1$ et $Q_2$, noté $Q_1 \succ_{\{\chi\}} Q_2$, caractérise l'ensemble des états à partir desquels il est possible d'atteindre en un pas un état de $Q_2$ avec tous les états intermédiaires appartenant à $Q_1 \vee Q_2$ et si une action est effectuée lors du pas, celle ci satisfait $\chi$.

Cette opérateur étant très similaire à l'opérateur prédécesseur en un pas présenté dans le cadre du Model Checking TCTL, la manière de calculer les deux opérateurs est également très proche :

$\cup_{s\in S}(s, \varphi_s^1) \succ_{\{\chi\}} \cup_{s\in S}(s, \varphi_s^2) = \cup_{s\in S}(s, \varphi_s^1) \succ_t (s, \xi_s \vee \varphi_s^2)$

où $\xi_s = \vee_{(s,\varphi)\in P_s}\, \varphi$

et $P_s = \{(s, \varphi \wedge \varphi_s^1)$ tel que $\exists(s', \varphi_{s'}^2). (s, \varphi) \in pre_{\{\chi\}}(s', \varphi_{s'}^2)\}$

$\xi_s$ est l'ensemble des contraintes des états symboliques de sommet s caractérisant les états prédécesseurs instantanés sous la condition $\chi$ d'un état de $\cup_{s\in S}(s, \varphi_s^2)$ et dont la valuation satisfait $\varphi_s^1$.

A partir de ces deux nouveaux opérateurs, on adapte l'algorithme de Model Checking TCTL afin d'obtenir un algorithme de Model Checking ATCTL :

Pour un graphe temporisé $G = (S, H, \delta, s_{init}, \Delta)$, soit $S$ $(\phi)$ est l'ensemble des états symboliques satisfaisant la formule $\phi$, $S$ $(\phi)$ est calculé par induction sur la structure de $\phi$ de la façon suivante :

$S$ $(x \bullet c) = \cup_{s\in S} (s, \Delta(s) \wedge x \bullet c)$
$S$ $(x - y \bullet c) = \cup_{s\in S} (s, \Delta(s) \wedge x - y \bullet c)$
$S$ $(x . \phi_1) = S (\phi_1) [x \to 0]$
$S$ $(\neg \phi_1) = \neg S (\phi_1)$
$S$ $(\phi_1 \wedge \phi_2) = S (\phi_1) \wedge S (\phi_2)$
$S$ $(\phi_1 \, {}_{\chi}\exists U_{\chi'} \, \phi_2) = \vee_{k\in N} X_k$
Où $X_0 = Pre_{\{\chi'\}} (S (\phi_2))$ et $X_{k+1} = X_k \vee S (\phi_1) \succ_{\{\chi\vee\tau\}} X_k$
$S$ $(\phi_1 \, {}_{\chi}\exists U \, \phi_2) = \vee_{k\in N} X_k$
Où $X_0 = S (\phi_2)$ et $X_{k+1} = X_k \vee S (\phi_1) \succ_{\{\chi\vee\tau\}} X_k$
$S$ $(\phi_1 \, {}_{\chi}\forall U_{\chi'} \, \phi_2) = \vee_{k\in N} X_k$
Où $X_0 = Pre_{\{\chi'\}} (S(\phi_2)) \wedge \neg Pre_{\{\neg\chi'\}} (S (true)) \wedge \neg Pre_{\{true\}} (S(\neg\phi_2))$
et $X_{k+1} = X_k \vee ((\neg (\neg(X_k) \, {}_{true}\exists U \, \neg((S(\phi_1) \vee X_k) \wedge z \le c))) [z \to 0]$
$\wedge \neg(\neg X_k \, {}_{true}\exists U_{(\neg\chi\wedge\neg\tau)} \, true)$
$S$ $(\phi_1 \, \forall U_{\chi} \, \phi_2) = \vee_{k\in N} X_k$
Où $X_0 = S (\phi_2)$
et $X_{k+1} = X_k \vee ((\neg (\neg(X_k) \, {}_{true}\exists U \, \neg((S(\phi_1) \vee X_k) \wedge z \le c))) [z \to 0]$
$\wedge \neg(\neg X_k \, {}_{true}\exists U_{(\neg\chi\wedge\neg\tau)} \, true)$
Où $\delta_s$ est l'invariant du sommet s.

L'adaptation est naturelle et l'implantation de cet algorithme peut se faire à partir de l'implantation de l'algorithme de Model Checking TCTL présenté au chapitre 5 en utilisant la propriété d'héritage des langages orientés objets. A l'exception des opérateurs ${}_{\chi}\forall U$ et ${}_{\chi}\forall U_{\chi'}$, l'algorithme calcule les opérateurs de la même manière qu'en TCTL, les transitions devant satisfaire en plus certaines propriétés dans le cas des opérateurs ${}_{\chi}\exists U_{\chi'}$ et ${}_{\chi}\exists U$.
Nous allons montrer que dans le cas des opérateurs ${}_{\chi}\forall U$ et ${}_{\chi}\forall U_{\chi'}$, l'algorithme calcule bien ce qu'il doit calculer. Dans les deux cas, ceci peut être montré par induction. Dans le cas de ${}_{\chi}\forall U_{\chi'}$, les états qui, au départ, se trouvent dans $X_0$ sont tous les états à partir desquels on ne peut faire que des actions satisfaisant $\chi'$ sinon les états ne se trouveraient pas dans l'ensemble caractéristique de $\neg Pre_{\{\neg\chi'\}}$ $S$ (true) et les états ainsi atteints satisfont $\phi_2$. Dans le cas contraire, les états ne se trouveraient pas dans l'ensemble caractéristique de $\neg$ $Pre_{\{true\}}$ $S(\neg\phi_2)$. De plus, au moins une telle action est effectivement faisable à partir de ces états, sinon ils ne se trouveraient pas dans l'ensemble caractéristique de $Pre_{\{\chi'\}}S(\phi_2)$.
Les états se trouvant dans $X_0$ satisfont donc bien la formule $\phi_1 \, {}_{\chi}\forall U_{\chi'} \, \phi_2$.
En considérant que cela est également le cas après N itérations, on peut montrer que les états rajoutés à N +1$^{ième}$ itération satisfont également $\phi_1 \, {}_{\chi}\forall U_{\chi'} \, \phi_2$.
En effet, admettons qu'un état rajouté ne satisfasse pas la formule $\phi_1 \, {}_{\chi}\forall U_{\chi'} \, \phi_2$. Dans ce cas, il se pourrait qu'une action effectuée ne satisfasse pas $\chi$ et ne soit pas $\tau$. Dans ce cas, ce ne peut être une action effectuée pendant un pas partant de l'état rajouté et permettant d'atteindre un état se trouvant déjà dans $X_N$ sinon l'état ne ferait pas partie de l'ensemble solution de $\neg(\neg X_N \, {}_{true}\exists U_{(\neg\chi\wedge\neg\tau)} \, true)$ et par induction, toutes les actions effectuées ensuite satisfont $\chi$ ou sont $\tau$ sauf les actions permettant d'atteindre un état satisfaisant $\phi_2$ qui, elles, satisfont $\chi'$. Ce ne sont donc pas les actions qui peuvent poser problème. Un état accédé pour atteindre l'état satisfaisant $\phi_2$ ne doit alors pas satisfaire $\phi_1 \vee \phi_2$. Si cet état est atteint pendant un pas partant de l'état rajouté et permettant d'atteindre un état de $X_N$, alors l'état rajouté ne peut pas se

trouver dans l'ensemble solution de $((\neg\ (\neg(X_N)\ _{true}\exists U\ \neg((S(\phi_1) \vee X_N) \wedge z \leq c)))\ [z \rightarrow 0]$. Ce dernier est l'ensemble des états à partir desquels il est inévitable d'atteindre un état de l'ensemble $X_N$ avant c unités de temps, tous les états intermédiaires satisfaisant $\phi_1$. De plus, par induction, un état non désiré ne peut être atteint ensuite. Donc, les états atteints durant les séquences de pas permettant d'atteindre un état satisfaisant $\phi_2$ ne peuvent également pas poser problème. On en conclut que tous les états qui se retrouveront dans $X_{N+1}$ satisfont $\phi_1\ _{\chi}\forall U_{\chi'}\ \phi_2$. On peut montrer, de manière semblable, que l'algorithme calculant l'opérateur $\forall U_{\chi'}$ calcule bien ce qu'il doit calculer.

L'avantage de cet algorithme est qu'il gomme les défauts du précédant, c'est à dire qu'on ne touche pas au graphe (ni à la formule). On peut donc espérer un gain de temps lors d'évaluation de formules, bien qu'il soit à noter que dans le cas d'une formule du type $\phi_1\ _{\chi}\forall U_{\chi'}\ \phi_2$ ou $\phi_1\ _{\chi}\forall U\ \phi_2$ à chaque étape du processus itératif, deux processus itératifs sont calculés. Aucune implantation de cet algorithme n'ayant été faite dans le cadre de ce mémoire, par manque de temps, on ne peut donner ici de constatation expérimentale sur ses performances.

# Chapitre 8

# Application

Au chapitre précédent, on a présenté un nouveau formalisme et deux algorithmes de Model Checking l'utilisant. On va mettre en application dans ce chapitre le premier algorithme en vérifiant plusieurs propriétés d'un système à l'aide du Model Checker implanté dans le cadre de ce travail.

**Le système**
Le système est constitué d'un circuit circulaire sur lequel se déplace un petit train. A un endroit donné du circuit, il y a un passage à niveau entouré de deux senseurs. Lorsque le train passe par le premier senseur (action sensor1), la barrière du passage à niveau commence à se baisser et met entre 8 et 16 secondes pour être fermée (action closed). Lors du passage devant le deuxième senseur (action sensor2), la barrière commence à se lever et prend entre 10 et 20 secondes pour être ouverte (action open). Le train, lui, prend entre 15 et 20 secondes pour rentrer (action enter) dans le passage à niveau après être passé par le premier senseur, entre 10 et 15 secondes pour ensuite atteindre le second senseur et finalement entre 100 et 150 secondes pour revenir au senseur 1.

**Résultat**

La figure 28 représente un graphe temporisé G représentant le système décrit ci-avant. Les actions y sont spécifiées au niveau des transitions.
En appliquant la fonction $f_g$ à G, nous obtenons un graphe temporisé G', représenté à la figure 29, sur lequel des vérifications de formules TCTL peuvent être effectuées.

| | Graphe avant transformation | Graphe après transformation |
|---|---|---|
| Nombre de sommets | 19 | 30 |
| Nombre de transitions | 27 | 38 |

On constate que l'augmentation du nombre de sommets lors de la transformation de graphe temporisé est de 58%, ce qui est important. Dans ce cas-ci, le nombre de sommets après transformation n'est pas encore important, vu la petite taille de graphe temporisé de départ, mais sur un graphe temporisé un peu plus conséquent de 1000 états, la transformation rajouterait 580 sommets.

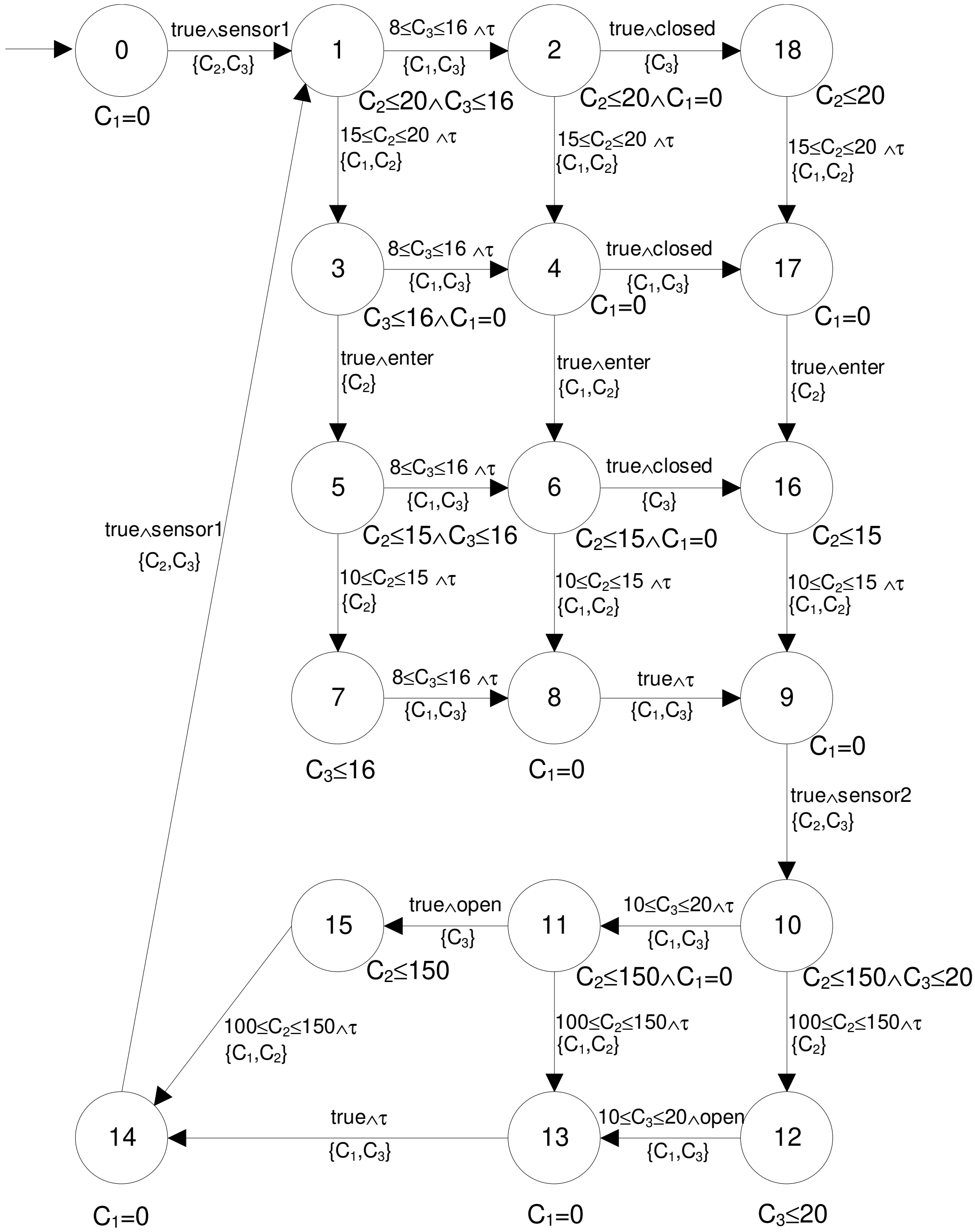



Figure 28

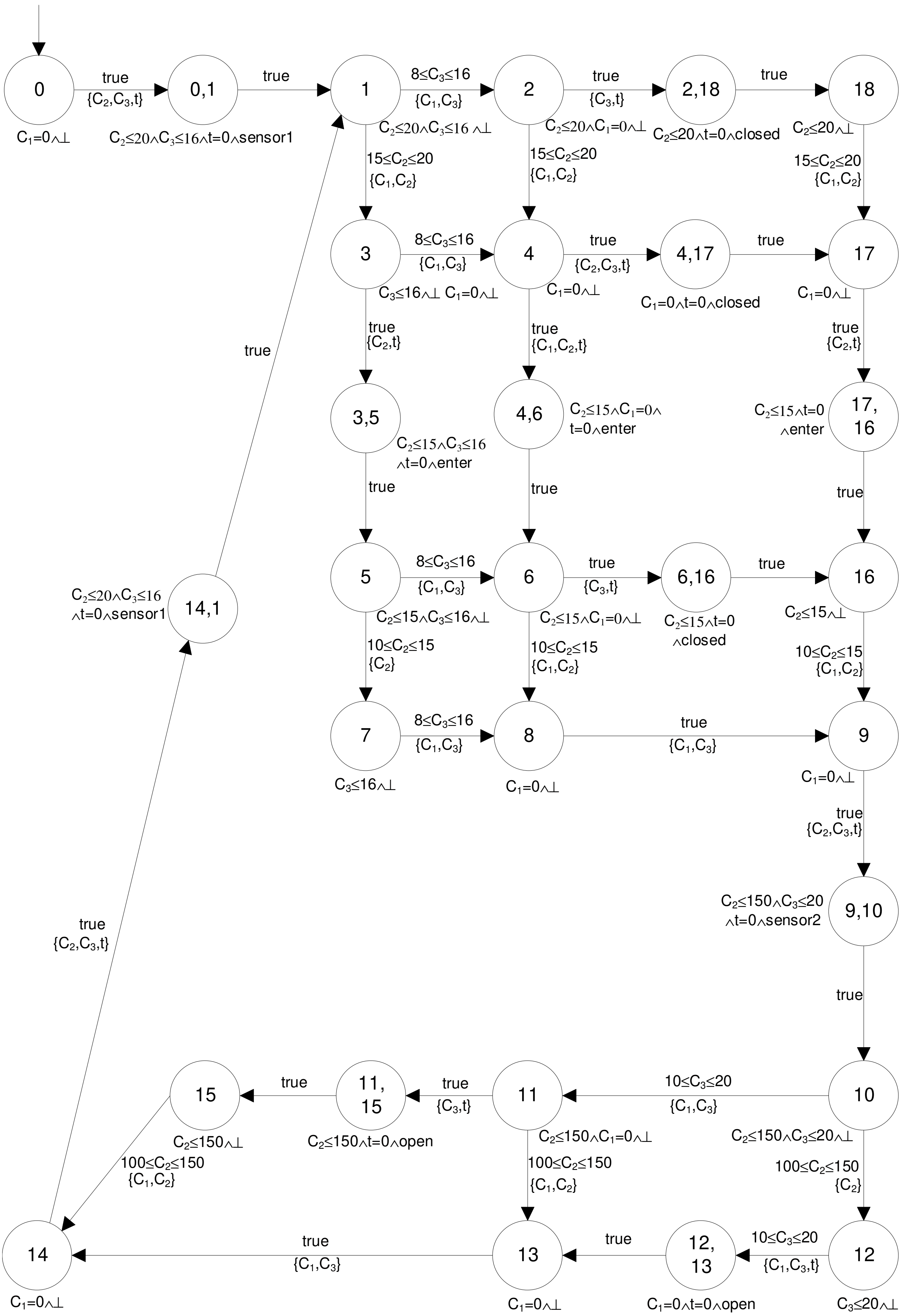

0
0,1
1
2
2,18
18
3
4
4,17
17
3,5
4,6
17, 16
5
6
6,16
16
14,1
7
8
9
9,10
10
11
11, 15
15
12
12, 13
13
14
true
{C2,C3,t}
8≤C3≤16
{C1,C3}
{C3,t}
15≤C2≤20
{C1,C2}
10≤C2≤15
{C2}
10≤C3≤20
100≤C2≤150
{C1,C3,t}
C1=0∧⊥
C2≤20∧C3≤16∧t=0∧sensor1
C2≤20∧C3≤16 ∧⊥
C2≤20∧C1=0∧⊥
C2≤20∧t=0∧closed
C2≤20∧⊥
C3≤16∧⊥
C1=0∧t=0∧closed
C2≤15∧C3≤16 ∧t=0∧enter
C2≤15∧C1=0∧ t=0∧enter
C2≤15∧t=0 ∧enter
C2≤15∧C3≤16∧⊥
C2≤15∧C1=0∧⊥
C2≤15∧t=0 ∧closed
C2≤15∧⊥
C2≤150∧C3≤20 ∧t=0∧sensor2
C2≤150∧C1=0∧⊥
C2≤150∧C3≤20∧⊥
C2≤150∧⊥
C2≤150∧t=0∧open
C1=0∧t=0∧open
C3≤20∧⊥

Figure 29

Une première propriété vérifiée est que la barrière est toujours fermée lorsque le train rentre dans le passage à niveau.
Ceci peut être exprimé par la formule ATCTL
$\neg(\text{true}\ {}_{\text{true}}\exists U_{\text{sensor1}}(\text{true}\ {}_{\neg(\text{closed})}\exists U_{\text{enter}}\ \text{true}))$
En effet, la formule spécifie qu'il n'est pas possible d'effectuer à un moment donné l'action sensor1, aucune contrainte n'étant spécifiée sur ce qu'il est fait avant cela, puis d'effectuer un certain nombre d'actions, autre que l'action closed, et finalement faire l'action enter.

Après application de la formule $f_f$, nous obtenons la formule TCTL
$\perp \wedge \neg(\perp \wedge (\neg(\neg(\text{TRUE} \wedge \neg(\perp)) \wedge \neg(\perp))\ \exists U\ (\neg(\perp) \wedge ((\text{sensor1} \wedge \neg(\perp))\ \exists U\ ((\perp \wedge (\neg(\neg(\neg(\text{closed}) \wedge \neg(\perp)) \wedge \neg(\perp))\ \exists U\ (\neg(\perp) \wedge ((\text{enter} \wedge \neg(\perp))\ \exists U\ (\perp)))))))))$

L'évaluation de cette formule sur G' donne FALSE. Si le train prend 15 secondes pour rentrer dans le passage à niveau et la barrière 16 secondes pour se fermer, le train a en effet la possibilité de rentrer dans le passage à niveau alors que la barrière n'est pas fermée.
Si nous posons que la barrière met entre 8 et 14 secondes pour se fermer, l'évaluation de la formule donne dans ce cas TRUE.
On constate bien en regardant la figure 28 que le calcul de l'opérateur $\exists U$ de la sous-formule $(\text{enter} \wedge \neg(\perp))\ \exists U\ (\perp)$ converge en une itération. Il n'y a donc bien dans cette formule que deux opérateurs $\exists U$ sur quatre qui risquent de prendre réellement du temps. La formule ATCTL de départ contenant deux opérateurs du type ${}_{\chi}\exists U_{\chi'}$, le poids de la transformation est effectivement faible.

Il se pourrait que le passage à niveau ne puisse pas empêcher un trafic de voitures traversant la ligne à cet endroit pendant une durée n'excédant pas 60 unités de temps.
Cette propriété est spécifiée par la formule ATCTL
$\text{true}\ {}_{\text{true}}\forall U_{\text{sensor1}}\ z.(\text{true}\ {}_{\text{true}}\forall U_{\text{open}}\ z < 60)$
En effet, cette formule spécifie qu'il est inévitable d'effectuer l'action sensor1. Cette action permettant d'aboutir dans un état s à partir duquel il est inévitable d'effectuer l'action open et de se retrouver dans un état sans avoir vieilli depuis s de soixante unités de temps ou plus.

Cette formule, transformée par $f_f$, donne la formule TCTL
$\perp \wedge (\neg(\neg(\text{TRUE} \wedge \neg(\perp)) \wedge \neg(\perp)))\ \forall U\ (\neg(\perp) \wedge (\text{sensor1} \wedge \neg(\perp)\ \exists U$
$((z.(\perp \wedge (\neg(\neg(\text{TRUE} \wedge \neg(\perp)) \wedge \neg(\perp)))\ \forall U\ (\neg(\perp) \wedge (\text{open} \wedge \neg(\perp)\ \exists U\ (\perp \wedge z < 60))))))))$

L'évaluation de cette formule sur G' donne TRUE. En effet, il faut au maximum 55 unités de temps après que le train soit passé par le senseur1 pour que la barrière soit de nouveau ouverte.

Nous vérifions finalement que la barrière est obligatoirement ouverte lorsque le train passe par le senseur 1.
Cette propriété peut être exprimée en ATCTL par la formule
$\neg(\text{true}\ {}_{\text{true}}\exists U_{\text{closed}}\ (\text{true}\ {}_{\neg\text{open}}\exists U_{\text{sensor1}}\ \text{true}))$
En effet, cette formule énonce le fait qu'il n'est pas possible d'effectuer un certain nombre d'actions quelconques jusqu'au moment où l'action closed est effectuée et qu'ensuite le système puisse faire un certain nombre d'actions autre que open jusqu'au moment où l'action sensor1 est effectuée.

La fonction $f_f$ appliquée à cette formule donne la formule TCTL
$\perp \wedge \neg(\perp \wedge (\neg(\neg(\text{TRUE} \wedge \neg(\perp)) \wedge \neg(\perp))\ \exists U\ (\neg(\perp) \wedge (\text{closed} \wedge \neg(\perp)$
$\exists U\ ((\perp \wedge(\neg(\neg(\neg(\text{open}) \wedge \neg(\perp)) \wedge \neg(\perp))\ \exists U\ (\neg(\perp) \wedge (\text{sensor1} \wedge \neg(\perp)\ \exists U\ (\perp))))))))))$

L'évaluation de cette formule sur G' donne TRUE. Après être passé par le senseur 2, il faut entre 100 et 150 secondes au train pour passer par le senseur 1, alors que la barrière prend au maximum 20 secondes pour être ouverte.

Les temps d'évaluations des formules sont repris (en secondes) dans le tableau suivant. Ces évaluations ont été effectuées sur la même machine que celles effectuées dans le cadre de la comparaison avec Kronos.

| **Formule** | Temps d'évaluation |
|---|---|
| $\neg(\text{true}\ {}_{\text{true}}\exists U_{\text{sensor1}}(\text{true}\ {}_{\neg(\text{closed})}\exists U_{\text{enter}}\ \text{true}))$ | 1min21.84 |
| $\text{true}\ {}_{\text{true}}\forall U_{\text{sensor1}}\ z.(\text{true}\ {}_{\text{true}}\forall U_{\text{open}}\ z < 60)$ | 15moin57.84 |
| $\neg(\text{true}\ {}_{\text{true}}\exists U_{\text{closed}}\ (\text{true}\ {}_{\neg\text{open}}\exists U_{\text{sensor1}}\ \text{true}))$ | 22.14 |

# Chapitre 9

# Conclusion

On commencera cette conclusion par dire que tous les objectifs qu'on s'était fixé au départ ont été atteints. A savoir :

- la compréhention et la familiarisation des concepts, idées, méthodes, etc. liés au Model Checking. Pour cela, on s'est familiarisé avec un certain nombre de logiques classiques et compris le sens de leurs opérateurs. On s'est également familiarisé avec quelques automates classiques et compris comment ils fonctionnaient. On s'est finalement intéressé à des algorithmes de Model Checking mettant en œuvre les logiques et automates étudiés auparavant. On doit soulever quelques problèmes qui ont été rencontrés pendant cette phase d'étude. Tout d'abord, un grand nombre de documents qui ont été lus et étudiés étaient peu clairs, obscures, avec peu d'explications. Certains points ont donc nécessité du temps avant d'être compris. Il faut également ajouter à cela le fait que chaque document avait sa manière propre de présenter et de définir différents concepts. Il a donc fallu effectuer un travail d'harmonisation pour pouvoir parler des logiques, modèles, algorithmes de manières cohérentes. On note finalement les erreurs disséminées dans les différents documents qui ont rendu la compréhension longue et difficile.
- La définition d'une nouvelle logique. Pour répondre au souhait que l'algorithme de Model checking implanté dans le cadre de ce mémoire utilise une logique temporisée orientée action, une logique ayant ces propriétés a dû être créée. En effet, aucune logique répondant à ces critères n'a pu être trouvée. Cette nouvelle logique, appelée ATCTL, a été définie en s'inspirant de deux extensions de CTL, à savoir TCTL et ACTL. Chacune de ces deux extensions rajoutant par rapport à CTL un des deux critères recherchés, ATCTL peut être vu comme une sorte de fusion entre ces deux logiques. Deux algorithmes de Model Checking ATCTL ont également été définis. Ces deux algorithmes se basent ou s'inspirent du Model Checking TCTL.
- L'implantation d'un algorithme de Model Checking. L'algorithme finalement implanté utilise la nouvelle logique ATCTL pour spécifier les propriétés à vérifier. Cet algorithme se compose de deux parties. La première partie consiste à transformer la formule à vérifier et le graphe temporisé représentant le système en respectivement une formule TCTL et un graphe temporisé permettant la vérification de cette formule TCTL. La seconde partie consiste à effectivement faire la vérification. Cette partie utilise un algorithme de Model Checking TCTL. C'est évidemment cette dernière partie qui a nécessité le plus de travail

et de temps. Il a fallu tout d'abord lire la thèse de Sergio Yiovine. Un travail de compréhension a permis de mettre en évidence certaines méthodes qui ne marchent pas. Il a fallu lire pour cela la thèse d'Alfredo Olivero et mettre des informations trouvées dans ces deux thèses ensembles pour pouvoir déterminer ce qu'il faudrait implanter. De nouveau, on a rencontré un problème lié au manque d'explications qui a rendu la phase de compréhension préliminaire à l'implantation proprement dite fastidieuse. Finalement, l'implantation a pu être faite. Comme ce n'était pas demandé dans les objectifs, on n'a pas essayé d'optimiser l'implantation, d'avoir une interface élaborée,…Par contre, on s'est attaché à produire un code propre, mettre un certain nombre de commentaires et à produire un code modulaire, facilement modifiable. L'utilisation d'un langage orienté objet a été utilisé dans le but d'atteindre ce dernier point.

Quelques activités annexes ont également été réalisées. Une comparaison entre le Model Checker implanté dans le cadre de ce travail et Kronos, un outil implantant également l'algorithme présenté dans les thèses de Sergio Yiovine et Alfredo Olivero. Cette comparaison a pu mettre en évidence que l'implantation effectuée dans le cadre de ce travail peut être grandement optimisée. Un regard sur l'expressivité de certaines logiques a été effectué. Les propriétés auxquelles on s'est intéressé sont les propriétés d'atteignabilité et d'équité, les grandes classes de propriétés vérifiées à l'aide du Model Checking.

A partir de ce qui a été fait dans le cadre de ce travail, certains points seraient intéressant à approfondir.

Il serait intéressant de chercher à optimiser l'algorithme de Model Checking TCTL qui a été implanté. Lors de la comparaison avec l'outil Kronos, on a constaté qu'il était possible de grandement améliorer les performances. On pourrait s'intéresser à optimiser la représentation et la manipulation de contraintes sur des horloges. Comme la manipulation de contraintes est à la base de l'algorithme, si ceci pouvait être fait, les performances de l'algorithme seraient déjà bien meilleures. D'autres optimisations peuvent être envisagées comme des techniques permettant une évaluation rapide dans des cas bien précis mais fréquents.

D'autres part, il existe un outil permettant, a partir d'une spécification ET-LOTOS, de générer un graphe temporisé étendu. De ce graphe temporisé étendu, il est possible créer un graphe temporisé utilisable par Kronos. On pourrait envisager de créer et implanter un algorithme qui transforme une spécification ET-LOTOS en un graphe temporisé sur lequel pourraient être effectuées des vérifications de propriétés spécifiées en ATCTL grâce à l'algorithme qui a été implanté. Cet outil permettrait de donner une nouvelle dimension au Model Checkeur ATCTL. Pour l'instant, il n'est possible que de vérifier des systèmes simples. Ceci est lié au fait que le graphe doit être créé "à la main" et un outil permettant de faciliter la spécification de systèmes en rajoutant un niveau d'abstraction gommerait ce problème. L'avantage d'ATCTL est que la logique, grâce à son orientation action, permet de spécifier des propriétés de spécifications ET-LOTOS de manière plus naturelles qu'en TCTL.

On pourrait également améliorer l'interface du Model Checkeur ATCTL, qui est actuellement assez rudimentaire et peu pratique.

Il serait également intéressant d'implanter le second algorithme de Model Checking ATCTL présenté ici. On pourrait ainsi faire des constatations expérimentales sur ses performances et comparer ces mêmes performances avec celles du premier algorithme. Il serait également intéressant de réfléchir à la possibilité de définir d'autres algorithmes de Model Checking ATCTL qui seraient plus performant que ceux présenté ici.

La syntaxe d'ATCTL pose quelques petits problèmes. On a constaté que spécifier une propriété intéressante donne en général des formules assez longues et compliquées. Ceci est probablement dû à l'héritage des logiques CTL, ACTL et TCTL dans la syntaxe et la sémantique d'ATCTL. Il serait intéressant de rechercher des méthodes d'expressions simples

permettant de ne pas avoir une explosion de la taille des formules dès qu'on essaye de spécifier des propriétés intéressantes.
Un certain nombre d'autres points intéressant pourraient être également approfondis :
On pourrait continuer à investiguer les logiques existantes à la recherche de bonnes idées qui pourrait être à la base de travaux, étudier de manière approfondie l'expressivité d'ATCTL, créer une extension d'ATCTL permettant d'utiliser celle-ci dans un champ d'application qui pour l'instant ne lui est pas propice,…
On a vu qu'un certain nombre de choses avaient été faites dans le cadre de ce travail telles des travaux de compréhensions, d'implantation, comparaison, création,… Tout cela a ouvert, comme on vient de le voir, des champs d'investiguations qui pourront faire l'objet de travaux futurs.

# Bibliographie


[Alu91] : Rajeev Alur : Techniques for Automatic verification of real-time systems, thèse de doctorat, Stanford University, 1991.

[ASU89] : Alfred Aho, Ravi Sethi, Jeffrey Ullman : Compilateur : Principe, technique et outil, InterEdition

[CES86] : E.M.Clarke, E.A.Emerson, A.P.Sistla, Automatic verification of finite-state concurent system using temporal logic specifications, 1996.

[Lam94a] : Leslie Lamport, Introduction to TLA, SRC Technical Note, digital system research center, 1994.

[Lam94b] : Leslie Lamport, The temporal logic of action, revised version of SRC research report 79, 1994.

[Léo97] : L.Léonard, An extended LOTOS for the design of time-sensitive systems, PhD thésis, Université de Liège, 1997

[LUP92] : Leslie Lamport, Urban Engberg, Peter Gronning, Mechanical verification of concurrent systems with TLA, 14 september 1992, version 2.

[Mil80] : Robin Milner, A Calculus of Concurency Systems, volume 92 of LMCS, springler-verlag, 1980.

[Mil89] : Communication and concurency, Prentice Hall, 1989.

[MP92] : Zohar Manna, Amir Pnueli, The temporal logic of reactive and concurent system, Springler-Verlag, 1992

[NFGR91] : R. De Nicola, A. Fantechi, S. Gnesi, G. Ristori : An action based framework for verifying logical and behavioural properties of concurent systems, Università La Sapienza Roma

[Oli94] : Alfredo Olivero, Modélisation et analyse de système temporisés et hybrides, thèse de doctorat, Institut national polytechnique de Grenoble, 1994.

[Pnu77] : A.Pnueli, The temporal logic of programs, proceeding of the eighteenth symposium on foundations of computer science, Providence, RI, November 1977

[RO97] : Krzysztof R.Apt, Ernst-Rüdiger Olderog, Verification of Sequential and Concurent Programs, Springler-Verlag, second édition, 1997

[Wol98] : Pierre Wolper Lecture 4 : Temporal logic and their relation to automata, transparent du cour the algorithmic verification of reactive system, 1998.

[Yov93] : Sergio Yovine, Méthodes et outils pour la vérification symbolique de systèmes temporisés, thèse de doctorat, Institut national polytechnique de Grenoble, 1993