Vérification des propriétés temporisées des automates programmables industriels

Houda Bel Mokadem

THÈSE

PRÉSENTÉE À L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

POUR OBTENIR LE GRADE DE

DOCTEUR DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

par : Houda BEL MOKADEM

Spécialité : INFORMATIQUE

VÉRIFICATION DES PROPRIÉTÉS TEMPORISÉES DES AUTOMATES PROGRAMMABLES INDUSTRIELS

Soutenue le 28 septembre 2006 devant un jury composé de :

Béatrice BÉRARD	Professeur des Universités	Directeur de thèse
Serge HADDAD	Professeur des Universités	Rapporteur
François LAROUSSINIE	Maître de conférences (HDR)	Directeur de thèse
Olivier ROUX	Professeur des Universités	Président du jury
Françoise SIMONOT-LION	Professeur des Universités	Rapporteur
François VERNADAT	Professeur des Universités	Examinateur

Thèse préparée au sein du Laboratoire Spécification et Vérification.

Table des matières

Chapitre 1

Introduction

Il n'est plus nécessaire d'insister aujourd'hui sur l'omniprésence des systèmes embarqués dans notre quotidien. Ces systèmes, souvent critiques, mettent en jeu des coûts importants, mais surtout parfois des vies humaines. Ils sont de plus en plus autonomes, complexes et interagissent directement avec leur environnement. Ils existent dans les transports, l'électroménager, l'informatique, l'automatique, les équipements médicaux, etc.
Comme l'illustrent ces domaines d'applications, des défaillances de ces systèmes peuvent avoir des conséquences graves et leur fiabilité est donc primordiale. Des "bugs" célèbres témoignent des enjeux importants de la vérification des systèmes embarqués. Par exemple, l'explosion de la fusée *Ariane V* et le bug du processeur *Pentium* ont causé des pertes économiques très importantes, et l'appareil médical *Therac-25* [LT93] a provoqué la mort de plusieurs patients. Ces illustrations malheureuses sont toutes dues à des erreurs dans des composants embarqués.

La vérification formelle propose des techniques qui permettent de se convaincre qu'un système est conforme à sa spécification et/ou qu'il satisfait des propriétés de sûreté. Ces méthodes peuvent être rangées dans trois principales catégories.
– **La génération de test**, souvent utilisée par les concepteurs, produit des ensembles (les plus larges possible) de *scénarios*, dans le but de falsifier la spécification.
– **La démonstration automatique** consiste à décrire le système réel et sa spécification dans un système de preuve, puis à utiliser des règles de déduction pour décider si ce système satisfait ou non sa spécification.
– **Le model-checking** consiste, étant donnés un modèle du système (ou d'une abstraction de ce système) et une formule, exprimant une propriété dans un langage de spécification, à vérifier algorithmiquement si la formule est satisfaite par le modèle.

Les références bibliographiques concernant les notions décrites dans cette introduction seront données dans le **chapitre 2** et au début du **chapitre 3**, ces deux parties étant aussi des introductions plus détaillées au cadre scientifique de cette thèse.

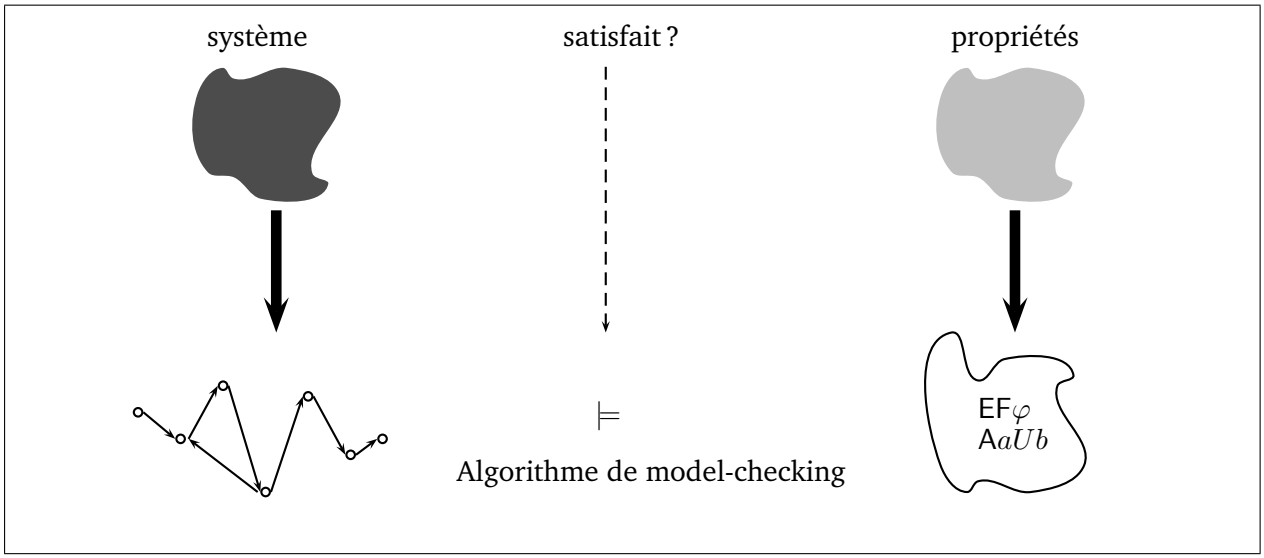
1.1 Le model-checking

La méthode du model-checking (vérification du modèle) est fondée sur trois étapes :

– **La phase de modélisation du système** a pour but de fournir une représentation formelle du système étudié. De nombreux modèles ont été proposés, dont la sémantique est définie sous la forme de systèmes de transitions où les noeuds représentent les états du système, tandis que les transitions décrivent les évolutions possibles d'un état à un autre. On peut citer les structures de Kripke, les automates finis, les automates temporisés, les réseaux de Petri,...

– **La phase de spécification de la propriété** consiste à traduire dans un langage formel de spécification la propriété à vérifier, écrite en langage naturel. Parmi les nombreux formalismes proposés, citons les logiques temporelles de temps linéaire ou de temps arborescent (LTL,CTL,TCTL,...) , le μ-calcul,...

– **La phase de vérification** applique à ces données un algorithme qui vérifie si le modèle du système satisfait ou non le modèle de sa spécification. Cet algorithme dépend de la nature des modèles choisis pour le système et la propriété.



Parmi les modèles possibles pour décrire un système et une propriété donnés, le choix est souvent un compromis entre l'expressivité et la facilité d'analyse (la complexité).
Les travaux présentés dans cette thèse s'inscrivent dans le cadre des techniques de model-checking "temporisé", qui concernent plus particulièrement les automates temporisés comme formalisme de modélisation et la logique TCTL comme formalisme de spécification. Nous décrivons brièvement dans ce qui suit ces deux formalismes, ainsi que les systèmes particuliers que nous avons étudiés : les automates programmables industriels.

1.2 Modèle et logique temporisés

Automates temporisés. Un automate temporisé est un automate fini où le temps intervient via des variables réelles, appelées *horloges*. Les horloges avancent toutes à la même vitesse et peuvent être comparées à des constantes et remises à zéro. Les configurations associées à un automate temporisé ont deux types d'évolution possibles :

– soit le temps s'écoule et l'automate reste dans le même état. Les valeurs des horloges vont toutes être augmentées du délai d'attente.
– soit l'automate franchit une transition discrète, ainsi il passera à l'état suivant. Des contraintes sur les valeurs des horloges sont associées à cette transition. Elles doivent alors être satisfaites par les valeurs courantes de ces horloges avant le franchissement. Les transitions discrètes sont aussi accompagnées par des remises à zéro d'un sous-ensemble d'horloges.

Logique temporisée. Les logiques temporelles sont des langages de spécification qui permettent d'énoncer des propriétés faisant intervenir des notions d'ordonnancement dans le temps, par exemple "Après la pluie, le beau temps". Une logique temporelle est dite de temps arborescent quand elle permet d'examiner les états et leur arbre d'exécution, contrairement à la logique de temps linéaire qui énonce des propriétés sur une exécution donnée. La logique temporisée arborescente TCTL ajoute à ce cadre des informations quantitatives sur les délais séparant les actions. Elle est construite à partir de propositions atomiques, par exemple *pluie* et *beau temps*, de connecteurs logiques et de combinateurs temporisés qui utilisent la modalité U (*until*).

Parmi les problématiques de recherche concernant ces logiques, on peut citer les points suivants.
– **Les questions des sémantiques** ont pour objet de définir de nouvelles modalités pour décrire de manière naturelle des classes de propriétés plus larges.
– **Les problèmes d'expressivité** concernent les comparaisons entre les classes de propriétés décrites par différentes logiques. Le choix d'une logique dans un problème de vérification dépend en partie de son pouvoir d'expression.
– **Les problèmes algorithmiques** s'attachent à la décidabilité et à la complexité du model-checking, ainsi qu'à des techniques d'optimisation.

Les nombreux travaux sur les modèles temporisés et les logiques temporisées ont conduit au développement d'outils pour le model-checking temporisé, comme HyTech, Uppaal, Kronos, etc.

1.3 Contexte : Plan Pluri-Formations *VSMT*

Cette thèse a été en partie réalisée dans le cadre du projet *VSMT* (Vérification de Systèmes Multi-tâches Temps réel). Ce projet rassemble des automaticiens et des informaticiens autour de la problématique de la vérification formelle des Automates Programmables Industriels.

Un automate programmable industriel (API). est un matériel utilisant un microprocesseur composé d'une unité centrale, d'une mémoire pour le stockage interne des fonctions d'automatisme et de ports d'entrées et de sorties. L'API est équipé d'un système d'exploitation qui permet d'exécuter un programme (les fonctions) sous un mode cyclique. Le programme est écrit et compilé par des utilisateurs sur un matériel externe grâce à des langages adaptés, normalisés par la norme IEC 61131-3. Il est téléchargé ensuite dans l'API.
Les automates programmables industriels sont donc des systèmes embarqués, réactifs et temps-réel.

De manière spécifique, l'objectif du projet *VSMT* est de mettre au point des modèles formels qui prennent en compte :
• l'hétérogénéité des langages de programmation (IEC 61131-3) ;
• les aspects temps-réel ;
• les aspects multi-tâches de ces systèmes ;
Ces modèles doivent se prêter à la vérification automatique par la technique du model-checking.

Une partie de cette thèse concerne la modélisation et à la vérification d'un fragment d'un langage de programmation pour les APIs par la technique du model-checking.

1.4 Contributions de cette thèse

Mes contributions portent sur les trois aspects mentionnés plus haut : modélisation, spécification et vérification.
Modélisation. Lors d'une étude de cas, la phase de modélisation est difficile car il n'y a pas de règles générales. La première étape consiste à comprendre le système réel à partir de son cahier des charges, et à lever les ambiguïtés qui peuvent y être présentes.
Nous avons réalisé une étude de cas concernant la plate-forme MSS (Mechatronic Standard System) du groupe Bosch. Le rôle de cette plate-forme est de trier des pignons selon leurs matériaux et de leur ajouter ou de leur retirer un palier. Ces pignons sont transportés par un composant, appelé le chariot, circulant sur un rail. Des capteurs et une tige sont placés au dessus du rail pour ces diverses opérations. La plate-forme est contrôlée par un API dont le programme est écrit en Ladder Diagram (un des langages de la norme IEC 61131-3), comportant plusieurs temporisateurs. Une des propriétés que le programmeur souhaite garantir concerne la précision de l'arrêt du chariot lorsqu'il atteint une position prédéfinie.
En utilisant ce problème, nous proposons une sémantique formelle pour un fragment du langage Ladder avec des temporisateurs du type TON (Timed On Delay) par un réseau d'automates temporisés. Nous utilisons cette sémantique pour modéliser le programme de contrôle de la plate-forme. Nous modélisons aussi les autres composants physiques de ce système. En utilisant l'outil de model-checking temporisé Uppaal, nous vérifions certaines propriétés de la plate-forme. Cette étude a permis de trouver un problème concernant le programme de la plate-forme. Malgré la taille du modèle, les temps des vérifications sont raisonnables grâce à une forte hypothèse d'atomicité.

Spécification. Nous nous sommes intéressés à des sémantiques particulières pour les logiques temporisées, afin de les rendre plus adéquates pour la spécification de systèmes réels ; plus précisément, nous avons cherché à intégrer aux propriétés une notion d'abstraction des états transitoires (au sens où le système y reste un temps négligeable). De tels états sont produits d'une manière naturelle lors de la modélisation des systèmes réels par des automates temporisés, en particulier lorsque des APIs sont considérés. Une logique a été proposée pour l'abstraction de séquences d'états de durées inférieures à un certain seuil (paramétré). Cette logique étend la logique TCTL avec les modalités U^k où k est un paramètre entier. Nous définissons deux sémantiques possibles pour ces modalités, en distinguant les abstractions locales et les abstractions globales, et nous notons TCTL^{Δ} et $\mathsf{TCTL}^{\Delta}_{\Sigma}$ les deux logiques ainsi obtenues. Nous prouvons plusieurs résultats d'expressivité concernant TCTL^{Δ}.

Vérification. Nous avons ensuite étudié le problème du model-checking pour ces différentes sémantiques. Nous montrons que le model-checking est indécidable pour la sémantique "globale", alors qu'il est décidable pour la sémantique "locale", avec la même complexité que celui de TCTL : nous proposons un algorithme PSPACE-complet. Pour montrer ce résultat de décidabilité, nous introduisons une relation d'équivalence sur les exécutions temporisées, plus fine que celle qui est utilisée habituellement.

1.5 Plan de la thèse

Cette thèse commence par un chapitre de rappels (**chapitre 2**), où est défini précisément le cadre du model-checking temporisé dans lequel nos travaux vont se placer. Nous y présentons d'une manière plus détaillée les automates temporisés, leur sémantique et la logique TCTL. Ensuite, nous rappelons les principaux résultats concernant le model-checking temporisé. Finalement, nous présentons brièvement l'outil UPPAAL, qui sera utilisé pour l'étude de cas.

La suite du manuscrit est organisée en deux parties :

La première concerne la modélisation des APIs. Elle est composée d'un seul chapitre (le **chapitre 3**), où nous présentons l'étude de cas réalisée. Ce chapitre commence par une introduction détaillée des automates programmables industriels, de leurs particularités, de leurs modes de programmation et de la norme IEC 61131-3. Ensuite, nous présentons brièvement les travaux existant concernant la vérification des APIs. Finalement, nous décrivons notre modélisation de la plate-forme MSS, les expériences effectuées ainsi que les résultats obtenus. Cette étude de cas est décrite dans l'article [BBG+05].

La deuxième partie concerne la spécification des systèmes réels (aussi bien des automates programmables industriels). Son contenu a fait l'objet des articles [BBBL05, BBBL06]. Elle est composée de deux chapitres.

• Au **chapitre 4**, nous définissons la logique qui permet d'abstraire les états transitoires pendant la vérification. Nous proposons deux sémantiques différentes : une sémantique "locale" -TCTL^{Δ}-, et une sémantique globale -$\mathsf{TCTL}^{\overset{\Delta}{\Sigma}}$-. Nous prouvons plusieurs résultats d'expressivité sur la logique TCTL^{Δ}. Enfin, nous donnons une preuve de l'indécidabilité du model-checking de $\mathsf{TCTL}^{\overset{\Delta}{\Sigma}}$, obtenue par la simulation d'une machine de Minsky.

• Au **chapitre 5**, nous présentons l'algorithme de model-checking pour TCTL^{Δ}. Dans un premier temps, nous donnons une nouvelle relation d'équivalence sur les exécutions temporisées. Nous l'utilisons pour montrer que les régions décrites par Alur, Courcoubetis et Dill restent correctes pour la sémantique de TCTL^{Δ}. Ensuite, nous adaptons le graphe des régions donné par Alur, Courcoubetis et Dill et nous détaillons l'algorithme d'étiquetage. Nous terminons par une preuve complète de la correction de cet algorithme.

Chapitre 2

Modélisation et spécification de systèmes temporisés

La vérification de systèmes temporisés nécessite un formalisme adapté aux contraintes temporelles quantitatives. Dans ce chapitre, nous rappelons les *systèmes de transitions temporisés* (paragraphe 2.1) qui nous serviront par la suite à définir la sémantique d'un modèle maintenant classique, celui des *automates temporisés* (paragraphe 2.3), puis nous présentons un formalisme de spécification : la logique TCTL (paragraphe 2.2.2) qui est une extension de la logique CTL (paragraphe 2.2.1). Ensuite, nous citons les résultats fondamentaux de la vérification pour les automates temporisés. Et enfin, nous terminons ce chapitre introductif par une présentation des outils de vérification, en particulier UPPAAL [LPY97].

2.1 Systèmes de transitions temporisés

Nous notons $\mathbb{N}$ l'ensemble des entiers naturels et $\mathbb{R}_+$ l'ensemble des réels positifs ou nuls. Dans tout ce document, le domaine du temps est $\mathbb{R}_+$. Nous supposons donnés un ensemble de propositions atomiques AP et un alphabet fini Σ.
Les systèmes de transitions temporisés (STT) permettent de décrire la dynamique des systèmes combinant des évolutions discrètes et des évolutions continues. Un STT est un graphe orienté étiqueté (éventuellement infini). Formellement :

Définition 1 (Système de transitions temporisé)
Un système de transitions temporisé est un quadruplet $\mathcal{T} = (S, s_{init}, \rightarrow, L)$ où :
- *S est un ensemble (éventuellement infini) de configurations (ou états) et s_{init} est la configuration initiale,*
- *$\rightarrow \subseteq S \times (\Sigma \cup \mathbb{R}_+) \times S$ est une relation de transition. Une transition est notée $\xrightarrow{\alpha}$ avec $\alpha \in \Sigma \cup \mathbb{R}_+$.*
- *$L : S \rightarrow 2^{\mathsf{AP}}$ est une application qui étiquette chaque configuration par un sous-ensemble de* AP*.*

Une transition (s, α, s') est notée $s \xrightarrow{\alpha} s'$ et nous lui associons une *durée* définie par α si $\alpha \in \mathbb{R}_+$ et 0 sinon. Si $\alpha \in \Sigma$, la transition est appelée transition d'action, sinon elle est appelée transition de durée.

Pour les systèmes que nous considérons, les transitions de durée doivent de plus vérifier les propriétés ci-dessous :
déterminisme : si $s \xrightarrow{d} s'$ et $s \xrightarrow{d} s''$, avec $d \in \mathbb{R}_+$, alors $s' = s''$,
additivité : si $s \xrightarrow{d} s'$ et $s' \xrightarrow{d'} s''$, avec $(d, d') \in \mathbb{R}_+^2$, alors $s \xrightarrow{d+d'} s''$,
continuité : si $s \xrightarrow{d} s'$, alors pour tout $(d', d'') \in \mathbb{R}_+^2$ avec $d = d' + d''$, $\exists s'' \in S$ tel que $s \xrightarrow{d'} s'' \xrightarrow{d''} s'$,
durée 0 : pour tout $s \in S$, $s \xrightarrow{0} s$.

Définition 2 (Chemin d'un système de transitions temporisé)
Un chemin *d'un système de transitions temporisé est une séquence infinie de transitions consécutives :*

$$\rho = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \dots$$

où $\alpha_i \in \mathbb{R}_+ \cup \Sigma$, pour tout $i \geq 0$.

Grâce aux propriétés ci-dessus, tout chemin contenant une infinité de transitions d'action, peut être décrit par la forme particulière suivante :

$$\rho = s_0 \xrightarrow{d_0}\xrightarrow{a_0} s_1 \xrightarrow{d_1}\xrightarrow{a_1} \cdots \text{ où } d_i \in \mathbb{R}_+ \text{ et } a_i \in \Sigma$$

Un tel chemin contient toutes les configurations intermédiaires entre s_i et s_{i+1} obtenues à partir de s_i par les transitions $\xrightarrow{d}$, où $0 \leq d \leq d_i$.

Nous associons à un chemin $\rho = s_0 \xrightarrow{d_0}\xrightarrow{a_0} s_1 \xrightarrow{d_1}\xrightarrow{a_1} \cdots$, une séquence de dates absolues $(t_i)_{i\geq 0}$ définie par $t_0 = 0$ et $t_i = \sum_{j\leq i} d_j$. Cette séquence correspond aux dates des transitions d'action. Nous pouvons ainsi décrire un chemin par la séquence $\big((s_i, t_i)\big)_{i\geq 0}$.

Pour un chemin $\rho = \big((s_i, t_i)\big)_{i\geq 0}$ d'un STT, une configuration s peut être rencontrée plusieurs fois le long de ρ. Chacune de ces occurrences est située à une certaine *position* le long de ρ. On notera $p \in \rho$ pour désigner une position p de ρ et $\mathbb{P}(\rho)$ pour désigner l'ensemble des positions de ρ. Pour toute position p, nous notons s_p la configuration correspondante. Le long d'une transition de durée $s \xrightarrow{d} s'$ avec $d > 0$, il y a une infinité de positions différentes. Les notations $\rho^{\leq p}$, $\rho^{\geq p}$ et $p \overset{\sigma}{\mapsto} p'$ désignent respectivement le *préfixe* de ρ jusqu'à la position p, le *suffixe* de ρ depuis la position p et le *sous-chemin* menant de p à p'. Nous notons $SC(\rho)$ l'ensemble des sous-chemins de ρ. Nous associons à un préfixe $\rho^{\leq p}$ une *durée* notée $Time(\rho^{\leq p})$ définie par la somme des durées de toutes les transitions de $\rho^{\leq p}$. Nous définissons un ordre total sur l'ensemble $\mathbb{P}(\rho)$ des positions de ρ de la manière suivante : p *précède strictement* p' (nous noterons $p <_\rho p'$) s'il existe un sous-chemin $p \overset{\sigma}{\mapsto} p'$ (le long de ρ) non réduit à la transition $\xrightarrow{0}$, c'est-à-dire qu'il contient au moins une transition d'action ou une transition de durée $\xrightarrow{d}$ avec d un réel strictement positif. Si $\sigma \in SC(\rho)$, on écrit $\sigma <_\rho p$ quand toutes les positions de σ précèdent strictement la position p le long de ρ.

Définition 3 (chemin divergent)
Un chemin ρ est dit divergent *si pour tout $t \in \mathbb{R}_+$, il existe une position p de ρ tel que $Time(\rho^{\leq p}) > t$.*

Un tel chemin est parfois appelé dans la littérature un chemin *non-zénon*, en référence au paradoxe énoncé par Zénon d'Elée.

Définition 4 (Exécution d'un système de transitions temporisé)
Une exécution *d'un système de transitions temporisé $\mathcal{T}$ est un chemin divergent de $\mathcal{T}$, comprenant un nombre infini de transitions d'action.*

Nous notons $Exec(s)$ l'ensemble des exécutions issues de la configuration s.

2.2 Logiques temporelles

Les logiques temporelles sont des formalismes adaptés pour énoncer des propriétés faisant intervenir la notion d'ordonnancement dans le temps, par exemple : "une barrière s'ouvre à l'approche d'une voiture". Ces logiques ont été proposées pour la spécification des systèmes réactifs, pour la première fois en 1977 par A.Pnueli [Pnu77].

2.2.1 La logique CTL

La logique CTL (pour Computation Tree Logic) a été définie au début des années 1980 dans [QS82, CES86]. Elle est interprétée sur des structures de Kripke, c'est-à-dire des automates finis dont les états sont étiquetés par des propositions atomiques. Elle permet d'exprimer des propriétés sur ces états, faisant intervenir l'arbre des exécutions issues de cet état. Elle utilise des combinaisons booléennes, des modalités "next" (EX et AX) et des modalités "until" : E_U_ et A_U_.

2.2.1.1 Syntaxe de CTL

La syntaxe de la logique CTL est donnée par la grammaire suivante :

$$\varphi, \psi ::= P_1 \mid P_2 \mid \ldots \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{EX}\varphi \mid \mathsf{AX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$$

où les P_i sont des propositions atomiques de AP.

La modalité $\mathsf{EX}\varphi$ signifie "il existe un successeur qui vérifie φ", la modalité $\mathsf{AX}\varphi$ signifie "tout successeur vérifie φ", la modalité $\mathsf{E}\varphi\mathsf{U}\psi$ signifie "il existe un chemin vérifiant φ jusqu'à ce que ψ soit vraie" et $\mathsf{A}\varphi\mathsf{U}\psi$ signifie "tout chemin vérifie φ jusqu'à ce que ψ soit vraie".

Nous pouvons définir d'autres formules à partir de ces modalités de base :

$$\mathsf{EF}\,\varphi \stackrel{def}{=} \mathsf{E}(\top\,\mathsf{U}\,\varphi) \qquad \mathsf{AF}\,\varphi \stackrel{def}{=} \mathsf{A}(\top\,\mathsf{U}\,\varphi)$$
$$\mathsf{EG}\,\varphi \stackrel{def}{=} \neg\mathsf{AF}\neg\varphi \qquad \mathsf{AG}\,\varphi \stackrel{def}{=} \neg\mathsf{EF}\neg\varphi$$

où $\top$ est la constante "vrai".

Ainsi, la formule $\mathsf{EF}\,\varphi$ signifie qu'il est **possible** d'atteindre un état vérifiant φ. La formule $\mathsf{AF}\,\varphi$ signifie que φ est **inévitable**, c'est-à-dire φ sera vraie un jour. La formule $\mathsf{AG}\,\varphi$ indique que φ est **toujours** vraie. Enfin, la formule $\mathsf{EG}\,\varphi$ signifie qu'**il existe** un chemin le long duquel φ est **toujours** vraie. Ces formules sont illustrées dans la figure 2.1.

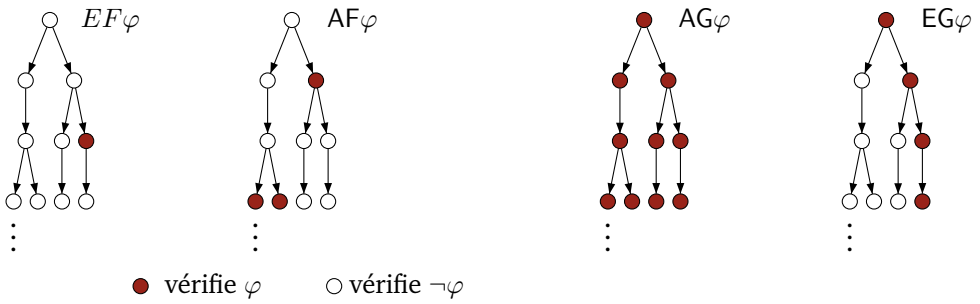


FIG. 2.1 – Illustration de certaines modalités de CTL

2.2.2 La logique TCTL

La logique TCTL est une extension proposée dans [ACD93] de la logique CTL, afin d'énoncer des propriétés temporisées, i.e qui font intervenir des informations **quantitatives** sur le temps, par exemple :

"La barrière s'ouvre **en moins de 5 secondes** à l'approche d'une voiture" (2.1)

2.2.2.1 Syntaxe de TCTL

La syntaxe de la logique TCTL est donnée par la grammaire suivante :

$$\varphi, \psi ::= P_1 \mid P_2 \mid \ldots \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{E}\varphi\mathsf{U}_{\sim c}\psi \mid \mathsf{A}\varphi\mathsf{U}_{\sim c}\psi$$

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

Les opérateurs E_U_ et A_U_ de CTL correspondent respectivement aux opérateurs E_U$_{\geq 0}$_ et A_U$_{\geq 0}$_ de la logique TCTL. Nous pouvons définir de la même manière que dans CTL les formules ci-dessous :

$$\mathsf{EF}_{\sim c}\,\varphi \stackrel{def}{=} \mathsf{E}(\top\,\mathsf{U}_{\sim c}\,\varphi) \qquad \mathsf{AF}_{\sim c}\,\varphi \stackrel{def}{=} \mathsf{A}(\top\,\mathsf{U}_{\sim c}\,\varphi)$$
$$\mathsf{EG}_{\sim c}\,\varphi \stackrel{def}{=} \neg\mathsf{AF}_{\sim c}\neg\varphi \qquad \mathsf{AG}_{\sim c}\,\varphi \stackrel{def}{=} \neg\mathsf{EF}_{\sim c}\neg\varphi$$

La formule $\mathsf{EF}_{\sim c}\,\varphi$ signifie qu'il est **possible** d'atteindre une configuration vérifiant φ située à une durée d depuis la configuration courante, telle que $d \sim c$. La formule $\mathsf{AF}_{\sim c}\,\varphi$ signifie que φ est **inévitable** (le long de tout chemin) après une durée d de la configuration courante telle que $d \sim c$. La formule $\mathsf{AG}_{\sim c}\,\varphi$ indique que φ est **toujours vraie** après une durée d de la configuration courante telle que $d \sim c$. Enfin, la formule $\mathsf{EG}_{\sim c}\,\varphi$ signifie qu'**il existe** un chemin le long duquel φ est **toujours vraie** après une durée d de la configuration courante telle que $d \sim c$.

2.2.2.2 Sémantique de TCTL

Nous donnons maintenant la sémantique formelle des formules de TCTL. Ces formules sont évaluées sur une configuration s d'un système de transitions temporisé (on note $s \models \varphi$).

Définition 5 (Sémantique de TCTL)
Soient $\mathcal{T} = (S, s_{init}, \rightarrow, L)$ un système de transitions temporisé et s une configuration de $\mathcal{T}$. Soient φ et ψ deux formules de TCTL*. La sémantique de* TCTL *est définie de manière inductive par les règles suivantes :*

$$\begin{array}{lll} s \models_{\mathcal{T}} \neg\varphi & \textit{si et seulement si} & s \not\models_{\mathcal{T}} \varphi \\ s \models_{\mathcal{T}} \varphi \wedge \psi & \textit{si et seulement si} & s \models_{\mathcal{T}} \varphi \textit{ et } s \models_{\mathcal{T}} \psi \\ s \models_{\mathcal{T}} \mathsf{E}\varphi\mathsf{U}_{\sim c}\psi & \textit{si et seulement si} & \exists\, \rho \in Exec(s) \textit{ t.q } \rho \models_{\mathcal{T}} \varphi\mathsf{U}_{\sim c}\psi \\ s \models_{\mathcal{T}} \mathsf{A}\varphi\mathsf{U}_{\sim c}\psi & \textit{si et seulement si} & \forall\, \rho \in Exec(s),\ \rho \models_{\mathcal{T}} \varphi\mathsf{U}_{\sim c}\psi \end{array}$$

avec

$$\rho \models_{\mathcal{T}} \varphi\mathsf{U}_{\sim c}\psi \quad \textit{si et seulement si} \quad \begin{array}{l} \exists\, p \in \rho \textit{ t.q. } \mathsf{Time}(\rho^{\leq p}) \sim c \ \wedge\ s_p \models_{\mathcal{T}} \psi \\ \wedge\, \forall p' <_{\rho} p,\ s_{p'} \models_{\mathcal{T}} \varphi \end{array}$$

Dans la suite, on note simplement $s \models \varphi$, s'il n'y a pas d'ambiguité sur le système $\mathcal{T}$.
On dit que $\mathcal{T} \models \varphi$ si et seulement si $s_{init} \models \varphi$.

Exemple 1
La propriété 2.1 s'énonce en TCTL *par la formule suivante :*

$$AG(\textit{ voiture } \Rightarrow AF_{\leq 5} \textit{ barrière ouverte})$$

2.2.3 Expressivité

Dans ce paragraphe, nous définissons la notion d'équivalence entre formules, que nous utilisons pour comparer l'expressivité des logiques.

Définition 6 (Équivalence entre formules)
Soient φ et ψ deux formules. Nous disons que φ et ψ sont équivalentes *pour une classe de modèle $\mathcal{M}$, noté $\varphi \overset{\mathcal{M}}{\equiv} \psi$, si pour tout modèle M de $\mathcal{M}$, M vérifie φ si et seulement si M vérifie ψ.*

Nous noterons parfois deux formules *équivalentes* par simplement $\varphi \equiv \psi$, s'il n'y a pas d'ambiguité sur les modèles.

Définition 7 (Expressivité de deux logiques)
Soient $\mathcal{L}$ et $\mathcal{L}'$ deux logiques évaluées sur les mêmes modèles.

- *la logique $\mathcal{L}'$ est dite* **aussi expressive que** *la logique $\mathcal{L}$, noté $\mathcal{L} \preceq \mathcal{L}'$, si pour toute formule φ de $\mathcal{L}$, il existe φ' de $\mathcal{L}'$, tel que $\varphi \equiv \varphi'$.*
- *la logique $\mathcal{L}'$ est dite* **strictement plus expressive que** *la logique $\mathcal{L}$, on note $\mathcal{L} \prec \mathcal{L}'$, si et seulement si $\mathcal{L} \preceq \mathcal{L}'$ et $\mathcal{L}' \not\preceq \mathcal{L}$.*

2.3 Les automates temporisés

Les *automates temporisés*, introduits par Alur et Dill dans [AD94], constituent un modèle classique dont la sémantique s'exprime en terme de système de transitions temporisé. Ils sont formés d'une structure de contrôle finie et manipulent un ensemble fini de variables réelles appelées *horloges* pour spécifier les contraintes de temps.

Horloges : Nous considérons un ensemble X de variables réelles appelées horloges. Une *valuation* des horloges est une application $v : X \to \mathbb{R}_+$ qui prend ses valeurs dans le domaine du temps $\mathbb{R}_+$. Nous notons $\mathbb{R}_+^X$ l'ensemble de toutes les valuations. Pour toute valuation v, si $d \in \mathbb{R}_+$, nous notons $v + d$ la valuation définie par $(v + d)(x) = v(x) + d$ pour toute horloge $x \in X$ et si $r \subseteq X$ nous désignons par $v[r \leftarrow 0]$ la valuation obtenue à partir de v en donnant aux horloges de r la valeur zéro et à une horloge $x \in X \setminus r$ la valeur $v(x)$.

Contraintes d'horloges : Étant donné un ensemble d'horloges X, l'ensemble des *contraintes* d'horloges $C(X)$ est l'ensemble engendré par la grammaire :

$$g ::= x \sim c \mid g \wedge g \mid \neg g$$

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

Définition 8 (Automate temporisé [AD94])
Un automate temporisé *est un 7-uplet* $\mathcal{A} = \langle X, Q, q_{init}, \Sigma, \to_{\mathcal{A}}, \mathsf{Inv}, l \rangle$ *où :*

- X *est un ensemble fini d'horloges,*
- Q *est un ensemble fini d'*états de contrôle*,*
- q_{init} *est l'état initial,*
- Σ *est un alphabet qui désigne un ensemble fini d'actions,*
- $\to_{\mathcal{A}} \subseteq Q \times C(X) \times \Sigma \times 2^X \times Q$ *est un ensemble fini de transitions d'action : pour* $(q, g, a, r, q') \in \to_{\mathcal{A}}$ *nous écrivons* $q \xrightarrow{g,a,r}_{\mathcal{A}} q'$*. La contrainte d'horloges associée* g *est appelée* garde*,* r *est l'ensemble d'horloges à remettre à zéro,* a *est une lettre de l'alphabet* Σ*.*
- $\mathsf{Inv} : Q \to C(X)$ *est une application qui associe à chaque état de contrôle une conjonction de contraintes d'horloges de la forme* $x \sim c$ *où* $\sim \in \{<, \leq\}$*, appelée* invariant*.*
- $l : Q \to 2^{\mathsf{AP}}$ *est une application qui étiquette chaque état de contrôle par un ensemble* $l(q)$ *de propositions atomiques.*

Une configuration (ou état) d'un automate temporisé est un couple (q, v), où q est un état de contrôle et v une valuation d'horloges. La configuration initiale d'un automate temporisé est (q_{init}, v_0) où $v_0(x) = 0$ pour toute horloge x. La notation $v \models g$, où $v \in \mathbb{R}_+^X$ et $g \in C(X)$, signifie que la valuation v vérifie la garde g.

Exemple 2
L'automate temporisé de la figure 2.2 décrit une barrière qui s'ouvre (en moins de 5 *secondes) à l'approche d'une voiture et reste ouverte pendant* 120 *secondes avant de se refermer. L'automate part de l'état initial* **fermé***. Lorsqu'une voiture s'approche de la barrière (signal* **app ?***), il passe dans l'état* **ouverture** *pour arriver à l'état* **ouvert** *(en remettant l'horloge* x *à zéro) après un délai inférieur ou égal à* 5 *secondes. Dans cet état, le temps s'écoule et l'approche d'autres voitures (***app ?***) remet à chaque fois l'horloge* x *à zéro. Lorsque* x *vaut* 120*, la barrière commence à se refermer (l'automate va dans l'état* **fermeture***). Elle se rabat après* 5 *secondes (l'automate revient à l'état* **fermé***), à moins qu'une autre voiture ne s'approche.*

2.3.1 Sémantique d'un automate temporisé

La sémantique d'un automate temporisé est définie par un système de transitions temporisé (STT). À partir d'une configuration donnée de l'automate temporisé, deux types de transitions sont possibles : une transition d'action si la valuation courante vérifie la garde, ou une

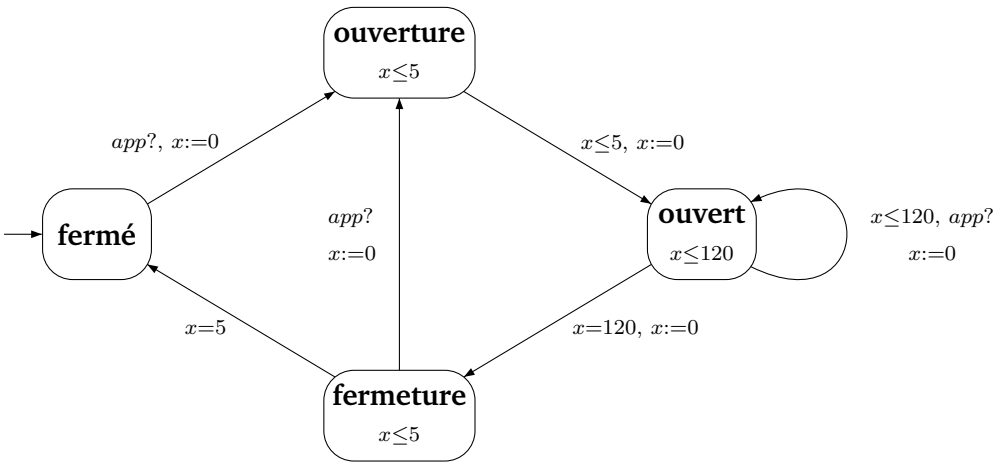


FIG. 2.2 – Exemple d'un automate temporisé modélisant une barrière

transition de durée qui consiste à rester dans le même état de contrôle et à incrémenter les valeurs des horloges uniformément en respectant l'invariant.

Définition 9 (STT associé à un automate temporisé)
Le STT associé à un automate temporisé $\mathcal{A} = \langle X, Q, q_{init}, \Sigma, \rightarrow_{\mathcal{A}}, \mathsf{Inv}, l \rangle$ *est le quadruplet* $\mathcal{T}_{\mathcal{A}} = (S, s_{init}, \rightarrow, L)$ *où :*

- $S = \{(q, v) \mid q \in Q$ *et* $v \in \mathbb{R}_+^X$ *avec* $v \models \mathsf{Inv}(q)\}$ *et* $s_{init} = (q_{init}, v_0)$.
- $\rightarrow \subseteq S \times (\Sigma \times \mathbb{R}_+) \times S$ *contient deux types de transitions :*
  - *des transitions de durée, notées* $(q, v) \xrightarrow{d} (q, v + d)$, *si* $v + d' \models \mathsf{Inv}(q)$ *pour tout* $d' \leq d$,
  - *des transitions d'action, notées* $(q, v) \xrightarrow{a} (q', v')$, *si* $\exists\ q \xrightarrow{g,a,r}_{\mathcal{A}} q'$ *tel que* $v \models g$, $v' = v[r \leftarrow 0]$ *et* $v' \models \mathsf{Inv}(q')$.
- $L \colon S \rightarrow 2^{\mathsf{AP}}$ *est une application qui étiquette chaque configuration* (q, v) *avec le sous-ensemble* $l(q)$ *de* AP .

On appelle exécution d'un automate temporisé $\mathcal{A}$, une exécution du système de transitions temporisé $\mathcal{T}_{\mathcal{A}}$.

2.4 Problème d'accessibilité et graphe des régions

Dans cette section, nous évoquons brièvement le *problème d'accessibilité*. Ce problème est décidable pour la classe des automates temporisés [AD94]. Nous présentons la technique principale de cette preuve de décidabilité.

Le problème d'accessibilité : Il consiste à décider si un état de contrôle est atteignable. Le fait que ce problème est décidable permet, par exemple, de vérifier l'absence ou la présence

de certains comportements qui conduisent à des états critiques.

Théorème 1 ([AD94])
Le problème d'accessibilité pour la classe des automates temporisés est PSPACE-complet.

La preuve de décidabilité pour ce problème est fondée sur une abstraction finie de l'ensemble des valuations d'horloges $\mathbb{R}_+^X$. Cette abstraction, appelée *graphe des régions*, est obtenue en construisant une partition (finie) de l'ensemble $\mathbb{R}_+^X$ des valuations, respectant les contraintes d'horloges, et compatible avec la progression du temps et les remises à zéro.

Définition 10 (Équivalence de valuations [AD94])
Soient X un ensemble d'horloges et K un entier. Deux valuations v et v' sont équivalentes, *noté $v \cong_{X,K} v'$, si les deux conditions ci-dessous sont vérifiées :*
(1) Pour toute horloge $x \in X$, nous avons :
soit $(\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor)$, soit $(v(x) > K \wedge v'(x) > K)$
(2) Pour tout couple $(x, y) \in X^2$ tel que $v(x) \leq K$ et $v(y) \leq K$, nous avons :
- $\mathsf{frac}(v(x)) \leq \mathsf{frac}(v(y)) \Leftrightarrow \mathsf{frac}(v'(x)) \leq \mathsf{frac}(v'(y))$
- $\mathsf{frac}(v(x)) = 0 \Leftrightarrow \mathsf{frac}(v'(x)) = 0$

La relation $\cong_{X,K}$ est une relation d'équivalence. Dans ce document, parfois l'ensemble X et la constante K sont omis, s'il n'y a pas d'ambiguité.
On appelle région une classe d'équivalence de $\mathbb{R}_+^X/\cong$. Il est facile de voir qu'il y a un nombre fini de régions.

Définition 11 (Région successeur)
Soit une région $R \in \mathbb{R}_+^X/\cong_{X,K}$, nous définissons la région successeur *de R, notée $succ(R)$, par la région distincte de R, si elle existe, telle que les conditions ci-dessous sont vérifiées :*

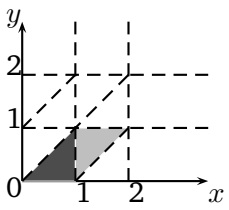
$$\forall\ v \in R,\ \exists\ d \in \mathbb{R}_+ \text{ tel que } [v + d \in succ(R)\ \wedge\ \forall\ 0 \leq d' < d,\ v + d' \in R \cup succ(R)]$$

Définition 12 (Région frontière)
Une région $R \in \mathbb{R}_+^X/\cong_{X,K}$ est dite région frontière *si et seulement si pour tout réel $d > 0$ et pour toute valuation v de R, $v + d$ n'appartient pas à R.*

L'opération de remise à zéro s'applique aussi à une région, en prenant pour $R[r \to 0]$ la classe d'équivalence de $v[r \to 0]$ pour un v quelconque de R.

Soit $\mathcal{A} = \langle X, Q, q_{\text{init}}, \Sigma, \to_{\mathcal{A}}, \mathsf{Inv}, l \rangle$ un automate temporisé et K la plus grande constante apparaissant dans les contraintes d'horloges de $\mathcal{A}$. La relation $\cong_{X,K}$ vérifie les propriétés suivantes :
(a) deux valuations équivalentes ne sont pas distinguées par les gardes de l'automate $\mathcal{A}$:
Pour toute garde g de $\mathcal{A}$, $v \cong_{X,K} v' \Rightarrow (v \models g \Leftrightarrow v' \models g)$
(b) elle est compatible avec l'écoulement du temps :
$v \cong_{X,K} v' \Rightarrow \forall d \in \mathbb{R}_+,\ \exists d' \in \mathbb{R}_+$ tel que $v + d \cong_{X,K} v' + d'$.
(c) elle est aussi compatible avec les remises à zéro :
$v \cong_{X,K} v' \Rightarrow \forall r \in X, v[r \leftarrow 0] \cong_{X,K} v'[r \leftarrow 0]$.

Exemple 3
Considérons un automate avec deux horloges x et y et la constante K égale à 2. l'ensemble des régions associées à cet automate peut être décrit par la figure ci-contre.
La région en gris foncé correspond aux valuations vérifiant les contraintes suivantes :
$0 < x < 1 \wedge 0 < y < 1 \wedge \mathsf{frac}(y) < \mathsf{frac}(x)$
La région successeur de cette région grisée est la région en gris clair.



Le graphe des régions : Nous sommes maintenant en mesure de définir le graphe des régions associé à un automate temporisé $\mathcal{A} = \langle X, Q, q_{\text{init}}, \Sigma, \rightarrow_{\mathcal{A}}, \mathsf{Inv}, l \rangle$.

Définition 13 (Graphe des régions)
Le graphe des régions *associé à $\mathcal{A}$ est le graphe fini $G_{\mathcal{A}} = (V, E)$ tel que :*
- $V = \{(q, R) \mid q \in Q_A\ ,\ R \in \mathbb{R}_+^X/\cong$ *et* $R \models \mathsf{Inv}(q)\}$
- *E est l'ensemble formé de deux types d'arcs :*
  - *un arc $\twoheadrightarrow$ représente le passage du temps :*
    $(q, R) \twoheadrightarrow (q, \mathsf{Succ}(R))$
  - *un arc $\xrightarrow{a}\!\!\!\!\!\twoheadrightarrow$ où $a \in \Sigma$ représente une action :*
    $(q, R) \xrightarrow{a}\!\!\!\!\!\twoheadrightarrow (q', R')$ *s'il existe* $q \xrightarrow{g,a,r} q'$ *avec* $R \models g$ *et* $R' = R[r \leftarrow 0]$.

Notons que pour toute exécution ρ de $\mathcal{T}_{\mathcal{A}}$

$$\rho = (q_0, v_0) \xrightarrow{d_0}\xrightarrow{a_0} (q_1, v_1) \xrightarrow{d_1}\xrightarrow{a_1} (q_2, v_2) \cdots$$

il existe un chemin $\bar{\rho}$ du graphe des régions $G_{\mathcal{A}}$

$$\bar{\rho} = (q_0, R_0) \twoheadrightarrow (q_0, \gamma_1^0) \cdots \twoheadrightarrow (q_0, \gamma_{n_0}^0) \xrightarrow{a_0}\!\!\!\!\!\twoheadrightarrow (q_1, R_1) \twoheadrightarrow (q_1, \gamma_1^1) \cdots \xrightarrow{a_1}\!\!\!\!\!\twoheadrightarrow (q_2, R_2) \cdots$$

tel que $v_i \in R_i$, $\mathsf{Succ}(R_i) = \gamma_1^i$ et $\mathsf{Succ}(\gamma_j^i) = \gamma_{j+1}^i$ où $i \geq 0$ et $1 \leq j < n_i$. Et inversement, pour tout chemin $\bar{\rho}$ du graphe des régions $G_{\mathcal{A}}$, il existe une infinité de chemins ρ de $\mathcal{T}_{\mathcal{A}}$.

À partir de cette correspondance des chemins entre $G_{\mathcal{A}}$ et $\mathcal{T}_{\mathcal{A}}$, on déduit le résultat classique suivant :

Théorème 2 ([AD94])
Un état de contrôle q est accessible dans un automate temporisé $\mathcal{A}$ si et seulement si un état de la forme (q, R) est accessible dans $G_{\mathcal{A}}$.

En conclusion, ce résultat permet d'obtenir clairement la décidabilité du problème de l'accessibilité d'un état de contrôle dans un automate temporisé.

Complexité. Une région peut être codée en espace polynomial [ACD93] : une région peut se représenter sous la forme d'un triplet (α, β, γ) où α associe à chaque horloge de X un intervalle dans $\{[0,0],]0,1[, [1,1], \ldots,]K-1, K[, [K,K],]K, \infty[\}$, β est une permutation sur

X donnant l'ordre des parties fractionnaires des valeurs d'horloges et γ est un tableau booléen utilisé pour indiquer les égalités des parties fractionnaires dans β. Par exemple, la région $(1 < x < 2) \wedge (0 < y < 1) \wedge (0 < z < 1) \wedge (0 < \mathsf{frac}(y) < \mathsf{frac}(x) = \mathsf{frac}(z) < 1)$ est représentée par le triplet $\big(]1;2[,]0;1[,]0;1[), \{y, x, z\}, \{0, 0, 1\}\big)$. Cela permet de borner le nombre de régions par $(2 \cdot K + 2)^{|X|} \cdot |X|! \cdot 2^{|X|}$. Le graphe des régions est donc exponentiel dans le nombre d'horloges et dans le codage binaire de la constante maximale K. Néanmoins, l'appartenance du problème à PSPACE-facile s'obtient en considérant un algorithme à la volée non déterministe : on stocke à chaque étape la configuration courante (état de contrôle et région), on choisit de manière non déterministe une configuration, on vérifie que cette nouvelle configuration est un successeur de la précédente et on teste si elle est finale, etc. D'après le théorème de Savitch [Sav70], cet algorithme non déterministe peut être transformé en un algorithme déterministe en espace polynomial. Le côté Pspace-difficile vient de la possibilité de coder le comportement d'une machine de turing bornée en espace linéaire sur un mot w sous la forme d'un automate temporisé.

Le long d'un chemin divergent de $\mathcal{T}_\mathcal{A}$, le temps progresse sans être borné. Ainsi, le long d'un tel chemin, toute horloge x est remise à zéro infiniment souvent, ou bien sa valeur progresse sans être bornée. Par conséquent, le chemin du graphe des régions $G_\mathcal{A}$ qui correspond à un chemin divergent de $\mathcal{A}$ traverse, pour toute horloge x, infiniment souvent les régions R avec $R(x) > K$ ou $R(x) = 0$. De tels chemins dans $G_\mathcal{A}$ sont appelés des chemins équitables. Dans toute la suite, nous considérons seulement des chemins équitables.

Définition 14 (Chemins équitables)
Soit F l'ensemble de contraintes défini par $F = \{F_x \mid x \in X\}$ où $F_x = \{(q, R) \mid R(x) = 0 \vee R(x) > K\}$. Un chemin du graphe des régions $G_\mathcal{A}$ est dit équitable *s'il traverse infiniment souvent pour toute horloge x les régions F_x associées à F.*

Remarque 1
Dans les chapitres 4 et 5, l'étiquetage d'une transition par une lettre de l'alphabet Σ n'intervient pas. Nous considérons alors une variante des automates temporisés où cet étiquetage est omis. Ainsi, nous notons les transitions d'action d'un tel automate temporisé $\mathcal{A}$ par $\xrightarrow{g,r}_\mathcal{A}$, de $\mathcal{T}_\mathcal{A}$ par $\rightarrow_a$ et de $G_\mathcal{A}$ par $\twoheadrightarrow_a$, où a est le symbole unique désignant une action quelconque.

2.5 Model-checking de TCTL

Dans cette section, nous rappelons les principes du model-checking de la logique TCTL. Étant donnés un automate temporisé $\mathcal{A}$ et une formule φ de TCTL, le problème du model-checking consiste à décider si $\mathcal{A}$ satisfait φ. Là encore, la difficulté de ce problème vient du fait que le système de transitions temporisé décrivant la sémantique d'un automate temporisé est infini (comme pour l'accessibilité). Un algorithme [ACD93] a été proposé par Alur, Courcoubetis et Dill : il consiste à réduire ce problème à celui du model-checking de la logique temporelle CTL sur une variante du graphe des régions.
On peut parler de model-checking "symbolique" parce que, plutôt que d'analyser le graphe infini de configurations (STT), on analyse le graphe fini de configurations *symboliques* (q, R), où q est un état de contrôle et R une région.

En effet, les configurations équivalentes du système de transitions temporisé vérifient les mêmes formules de TCTL. Plus formellement Alur, Courcoubetis et Dill ont montré le résultat suivant :

Lemme 1 ([ACD93])
Soient $\mathcal{A}$ un automate temporisé, q un état de contrôle de $\mathcal{A}$ et v, v' deux valuations telles que $v \cong v'$. Alors, pour toute formule φ de la logique TCTL *:*
$(q, v) \models \varphi \Leftrightarrow (q, v') \models \varphi$.

Ainsi, avec l'écoulement du temps, les configurations changent mais la satisfaction d'une formule de TCTL ne change que lors du changement de régions. L'idée de l'algorithme [ACD93] est de construire un graphe des régions avec une horloge supplémentaire afin de mesurer le délai séparant deux configurations le long d'une exécution. Ensuite, étiqueter les états de ce graphe par des formules de TCTL, en appliquant les techniques classiques de model-checking pour le cas discret.

2.5.1 Algorithme d'étiquetage

Pour vérifier si une configuration donnée s de $\mathcal{A}$ satisfait une formule φ de TCTL, on introduit une horloge z externe à l'automate $\mathcal{A}$ (c'est-à-dire elle n'intervient pas dans les gardes de l'automate). Cette horloge n'est jamais remise à zéro et elle va servir à traiter les contraintes $\sim c$ présentes dans la formule φ. Si X est l'ensemble des horloges de $\mathcal{A}$, on note $X^* = X \cup \{z\}$. Soit K la plus grande constante apparaissant dans les contraintes d'horloges de $\mathcal{A}$ et dans la formule φ. On note $\cong^*$ la relation d'équivalence $\cong_{X^*,K}$ et on appelle région étendue une classe d'équivalence de $\mathbb{R}_+^{X^*}/\cong^*$.

À toute contrainte $\sim c$ apparaissant dans φ, on associe une proposition atomique $p_{\sim c}$ telle qu'une région étendue R vérifie $p_{\sim c}$ si et seulement si $R \models z \sim c$. On considère aussi une proposition atomique particulière p_f telle que R vérifie p_f si et seulement si R est une région frontière.

On construit maintenant le graphe des régions relatif à la formule φ, avec X^* comme ensemble d'horloges, K comme constante maximale et $\cong^*_{X^*,K}$ comme relation d'équivalence.

Une fois ce graphe des régions étendues construit, on commence par étiqueter ses configurations (q, R) par les propositions atomiques $p_{\sim c}$ et p_f. Ensuite, comme pour CTL, on les étiquette de manière inductive par les sous-formules de φ : on commence par les sous-formules de longueur 1 (propositions atomiques), puis les sous-formules de longueur 2 et ainsi de suite.
Par exemple, pour étiqueter les configurations (q, R) du graphe des régions étendues qui satisfont la formule $\mathsf{E}\varphi\mathsf{U}_{\sim c}\psi$, on commence par l'étiquetage de φ et ψ. Une fois cet étiquetage terminé, on sait à cette étape de l'algorithme, pour toute configuration (q, R) si elle satisfait (ou non) φ et/ou ψ. Il suffit donc de vérifier s'il existe un chemin dans le graphe des régions étendues partant de $(q, R[z \leftarrow 0])$, le long duquel φ est toujours vraie jusqu'à une configuration $s_n = (q_n, R_n)$ qui satisfait ψ. De plus, $R_n \models z \sim c$: comme z n'est jamais remise à zéro, le temps écoulé à partir de $(q, R[z \leftarrow 0])$ pour atteindre ψ satisfait bien $\sim c$. Il suffit donc de vérifier si la formule suivante (de CTL) est vraie en $(q, R[z \leftarrow 0])$:

$$\Phi = \mathsf{E}\varphi\mathsf{U}(p_{\sim c} \wedge (p_f \vee After_a \vee \varphi) \wedge \psi)$$

où $After_a$ est vrai dans une configuration s le long d'un chemin si et seulement si la dernière transition avant s est une transition d'action. Le prédicat $After_a$ peut être codé par une proposition atomique moyennant l'ajout d'une horloge qui est remise à zéro à chaque transition d'action.
La formule Φ signifie qu'il existe une exécution ρ dans $\mathcal{A}$ le long duquel φ est toujours vraie jusqu'à une configuration (q_n, v_n) qui vérifie $p_{\sim c} \wedge \psi$, de plus (q_n, v_n) doit vérifier φ à moins de vérifier $p_f \vee After_a$. En effet, si (q_n, R_n) ne vérifie pas $p_f \vee After_a$, alors il existe une valuation v' de R_n telle que (q_n, v') est avant (q_n, v_n) le long de ρ. Dans ce cas, (q_n, v') doit satisfaire φ ($(q_n, R_n) \models p_f \vee After_a \vee \varphi$).

Complexité. Le côté PSPACE-facile vient d'un algorithme de model-checking en espace polynomial proposé par Alur, Courcoubetis et Dill [ACD93] : plutôt que de construire le graphe des régions, puis d'appliquer l'algorithme de model-checking classique, ils ont défini un algorithme à **la volée** où au plus $\mathcal{O}(|\varphi|)$ régions sont stockées en même temps. Le côté PSPACE-difficile vient directement du problème de l'accessibilité sur les automates temporisés [AD94]. Le model-checking de TCTL est alors PSPACE-complet.

En conclusion, nous avons le résultat fondamental suivant :

Théorème 3 ([ACD93])
Le model-checking de la logique temporisée TCTL *est PSPACE-complet pour les automates temporisés.*

2.5.2 Travaux existants

Comme nous l'avons dit précédemment, les automates temporisés ont été définis par Alur et Dill. La première définition est apparue dans [AD90], ensuite ils ont publié une version longue de leur travail [AD94]. Depuis cette classe de modèle a été largement étudiée. Par exemple, il a été montré dans [HKWT95] que le nombre d'horloges augmente le pouvoir d'expression des automates temporisés. D'autre part, un algorithme a été proposé dans [DY96] pour minimiser le nombre d'horloges d'un automate donné.
D'autres travaux ont étudié des sous-classes intéressantes des automates temporisés. Par exemple, le problème universel qui est indécidable pour le modèle général, a été montré décidable pour la classe des **automates temporisés à deux sens, bornés ("two-way bounded timed automata")** dans [AH92], et pour la classe **des automates événement-horloge ("event-clock automata")** dans [AFH94] (le problème universel consiste à décider si le langage reconnu par un automate donné est égal à $(\Sigma \times \mathbb{R}_+)^\infty$).
De nombreuses variantes du modèle original ont été proposées. Une extension des automates temporisés avec **des actions silencieuses** (ε-transitions) a été étudiée dans [BGP96, BDGP98], il a été prouvé que l'ajout des actions silencieuses augmente strictement le pouvoir d'expression. Une variante du modèle d'Alur et Dill avec l'ajout des **contraintes d'horloges du type ($x + y \sim c$)** a été étudiée dans [Duf97, BD00], il a été prouvé que ces modèles sont indécidables pour tester le vide du langage accepté, avec au moins quatre horloges. Dans [AHV93], il a été montré que les modèles avec des **paramètres** dans les contraintes d'horloges sont aussi indécidables pour le test du vide. Une autre extension des automates temporisés avec des mises à jour des horloges a été étudiée dans [BDFP00a, BDFP00b].

2.6 Outils de vérification

Les algorithmes de model-checking sont implémentés dans des logiciels. Une importante activité de recherche concerne ces outils et vise à augmenter leur efficacité et à repousser leurs limites. Il existe plusieurs *model-checkers* temporisés : HyTech [HHWT95], Uppaal [LPY97] [1], Kronos [DOTY96], etc. Kronos est un model-checker pour TCTL. Uppaal permet la vérification d'un fragment de CTL tandis que HyTech réalise une analyse d'accessibilité sur les systèmes hybrides, un modèle qui étend les automates temporisés avec des horloges de pentes variables, des contraintes linéaires sur les horloges et des mises à jour plus riches. Nous présentons brièvement l'outil Uppaal ci-dessous.

Uppaal permet d'analyser des systèmes formés d'une collection d'automates temporisés qui communiquent par des variables entières partagées et des synchronisations par messages. Les automates temporisés d'Uppaal (figure **??**) sont une variante des automates temporisés d'Alur et Dill. Leurs états de contrôle peuvent être étiquetés par deux étiquettes particulières :

– L'étiquette *u* pour *Urgent* : un tel état est équivalent au fait de remettre à zéro une horloge x avant d'arriver dans cet état et d'ajouter la garde $x \leq 0$ sur les transitions d'action sortant de cet état. L'automate doit donc le quitter immédiatement. Les délais dans les états urgents sont toujours égalent à zéro.
– L'étiquette *c* pour *Committed* : cette notion est plus restrictive que la notion d'état urgent. L'automate doit quitter un état marqué par "*c*" immédiatement après son arrivée en empruntant forcément une des transitions sortant (s'il y en a plusieurs) d'un état étiqueté par *c*.

En plus des horloges, les automates d'Uppaal manipulent des variables entières bornées. Les transitions de ces automates sont étiquetées par des gardes, des remises à zéro, des mises à jour de certaines variables entières et des étiquettes de synchronisation $m?$ et $m!$ où m est un canal. Notons qu'il y a deux possibilités de synchronisation de composants par messages :

– Les canaux binaires : un émetteur envoie le signal $m!$ et un unique récepteur se synchronise avec lui par le signal complémentaire $m?$. Les transitions de ces deux composants étiquetées par $m!$ et $m?$ sont franchies en même temps. Si, à partir d'une configuration, plusieurs couples de transitions peuvent être synchronisées, le choix est non déterministe. Ce mode de synchronisation peut conduire à des blocages si un automate envoie un signal $m!$ mais aucun composant ne peut le recevoir (par exemple si la garde de la transition étiquetée par $m?$ est fausse).
– Les canaux multiples (broadcast) : les transitions étiquetées par $m?$ et $m!$ sont synchronisées, mais cette fois avec un émetteur et un nombre arbitraire de récepteurs, qui peut être égal à zéro. Ainsi, une synchronisation de ce type entre plusieurs composants ne conduit jamais à un blocage.

[1] http ://www.uppaal.com

Première partie

Modélisation des automates programmables industriels

Chapitre 3

Les automates programmables industriels

Les *automates programmables industriels (API)* sont des mémoires programmables pour contrôler les systèmes automatisés. Ces appareils de contrôle/commande largement répandus dans l'industrie, sont apparus aux États-Unis vers 1969 et en France vers 1971. Ils sont couramment utilisés dans des systèmes critiques (avions, ascenseurs, trains,...), aussi leur validation est primordiale. L'objectif de la première partie de ce chapitre est de décrire les particularités des APIs ainsi que leurs langages de programmation. Nous rappelons ensuite les résultats principaux les concernant. Dans la deuxième partie, nous présentons une étude de cas concernant la *plate-forme MSS* (Mecatronic Standard System) du groupe Bosch. Ce système, qui comporte à la fois des aspects temps-réel et multi-tâches, est un cas intéressant, pour lequel nous avons pu proposer un modèle conduisant à de bonnes performances de vérification.

3.1 Description des Automates Programmables Industriels

Un API (en anglais *PLC* pour *Programmable Logic Controller*) est un ordinateur simplifié : il reçoit des données en *entrée*, celles-ci sont ensuite traitées par un *programme* et les résultats obtenus forment des *sorties*. Par exemple, un API peut maintenir le niveau d'eau dans un réservoir entre deux niveaux donnés, en ouvrant et en fermant des vannes électriques. Un mécanisme plus complexe pourrait impliquer une balance sous le réservoir (comme entrée) et un contrôleur d'écoulement (comme sortie) permettant à l'eau de couler avec un débit commandé. Une application industrielle typique pourrait commander plusieurs réservoirs intégrés dans un processus tels que chaque réservoir doit satisfaire une variété de conditions comme : n'être ni trop plein ni trop vide, avoir le pH dans une certaine fourchette, etc. Un API est donc un instrument de calcul et de commande qui est relié physiquement par une interface d'entrée à des *capteurs* et par une interface de sortie à des *actionneurs* (figure 3.1). Il contient une *unité centrale (CPU)* qui gère l'ensemble du processus, elle contient le processeur, les mémoires vives et des mémoires mortes pour une taille débutant à 40 Koctets. Elle est programmable par le biais d'une liaison spécifique et d'un logiciel adapté.
Un programme contient donc des variables spéciales permettant de représenter les entrées et les sorties. Ces variables, comme nous le précisons à la section suivante, sont mises à jour par l'API selon une procédure précise.

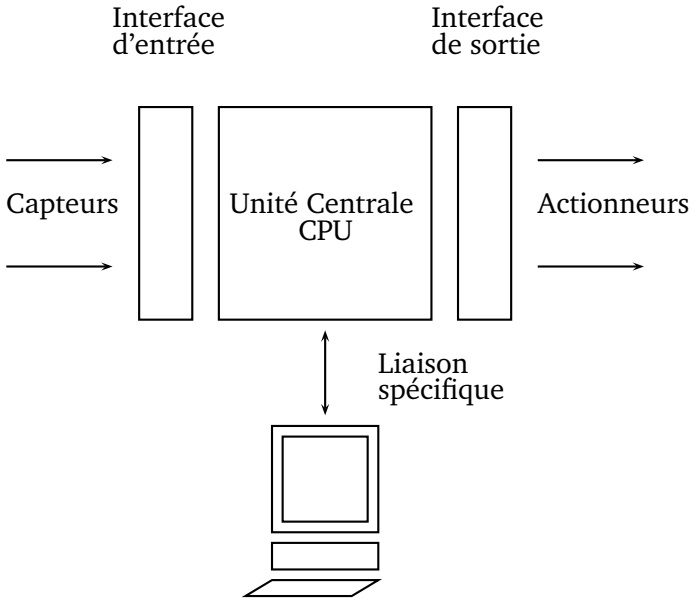


FIG. 3.1 – Automate programmable industriel

3.2 Comportement des Automates Programmables Industriels

Le moniteur d'exécution d'un API peut être composé de plusieurs sous-programmes appelés *tâches*. Une tâche est un ensemble d'opérations programmées pour s'exécuter successivement, puis s'arrêter jusqu'au prochain lancement. Dans un automate programmable industriel, une tâche est :
– ou bien *cyclique* : la tâche est immédiatement relancée après sa fin,
– ou bien *périodique* : la tâche est relancée toutes les T unités de temps,
– ou bien *événementielle* : la tâche est lancée à chaque fois qu'un événement prédéfini se produit.

L'exécution d'une tâche est un cycle composé de trois phases (figure 3.2) :
– *l'acquisition des entrées :* les variables d'entrées sont accessibles en lecture seule. Pendant cette première phase, leurs valeurs sont lues et ensuite stockées dans la mémoire de l'API,
– *le traitement interne :* c'est une phase d'exécution du programme et de calcul des valeurs de sorties à partir des valeurs stockées en mémoire dans la phase précédente, les résultats des calculs sont ensuite à leur tour stockés en mémoire,
– *l'affectation des sorties :* les variables de sorties sont accessibles en écriture seule. Pendant cette phase, leurs valeurs sont mises à jour à partir des valeurs calculées dans la phase de traitement interne.

Les APIs peuvent être programmés selon deux modes différents :
– mode *mono-tâche* : le moniteur d'exécution comporte une unique tâche cyclique, appelée *tâche maître*.
– mode *multi-tâches* : le moniteur d'exécution comporte plusieurs tâches dont l'ordonnance-

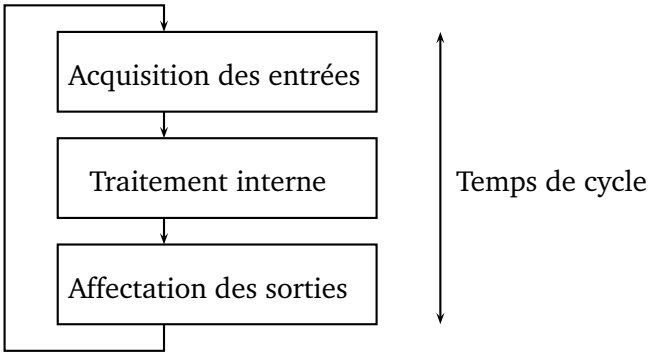


FIG. 3.2 – Un cycle de la tâche maître

ment est réalisé en fonction de leurs priorités (figure 3.3). A tout moment, une seule tâche est active et chaque tâche possède son propre cycle d'acquisition des entrées, traitement interne et affectation des sorties. Les tâches possibles sont :

– *la tâche maître* : elle est unique et cyclique.
– *les tâches rapides* : elles sont optionnelles et périodiques. Elles peuvent lire un nombre limité d'entrées, mais modifier toutes les variables internes.
– *les tâches événementielles* : elles sont optionnelles. Elles ont accès à un nombre limité d'entrées, de variables internes et de sorties.

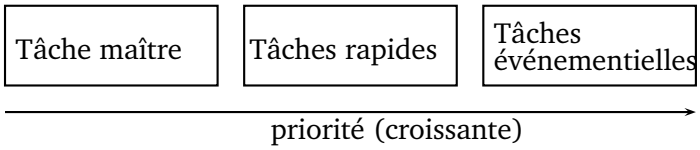


FIG. 3.3 – Priorités des tâches d'un API multi-tâches

Le temps de cycle t_c de chaque tâche est variable et doit toujours rester entre une borne minimale t_{min} et une borne maximale t_{max} paramétrées dans l'API. La borne maximale appelée *chien de garde* permet de détecter les boucles infinies qui conduiraient à ne jamais finir la phase de traitement interne. Si une telle boucle est détectée, l'exécution du programme est systématiquement interrompue et les systèmes connectés à l'API sont arrêtés dans un état sûr, en attendant une intervention humaine pour remettre l'API en marche. Le temps de cycle t_c est petit (quelques dizaines de millisecondes) relativement aux durées de l'environnement. Par exemple, pour actionner un moteur par un API mono-tâche, plusieurs cycles de la tâche maître s'exécutent successivement.

Le temps de réaction t_r d'une sortie aux changements des valeurs des entrées est variable. Ce temps t_r dépend de la méthode de programmation d'un API. Dans le cas de la programmation mono-tâche, si le changement de la valeur d'une entrée (par exemple l'entrée e_1 de la figure 3.4) se produit pendant la phase d'*acquisition des entrées* (de la tâche maître), alors l'effet de ce changement sur le système contrôlé est détecté à la fin du **cycle courant**, après l'émission de la sortie correspondante (sortie s_1 de la figure 3.4). Si au contraire, ce changement se produit plus tard (par exemple l'entrée e_2), alors il est lu par le programme de

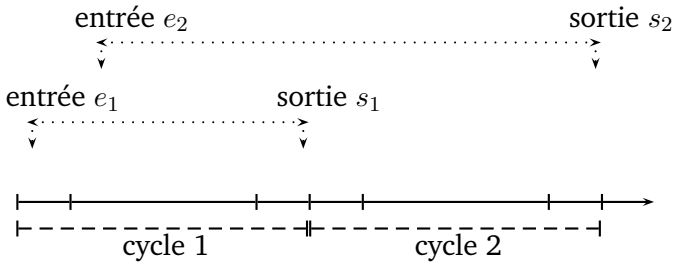


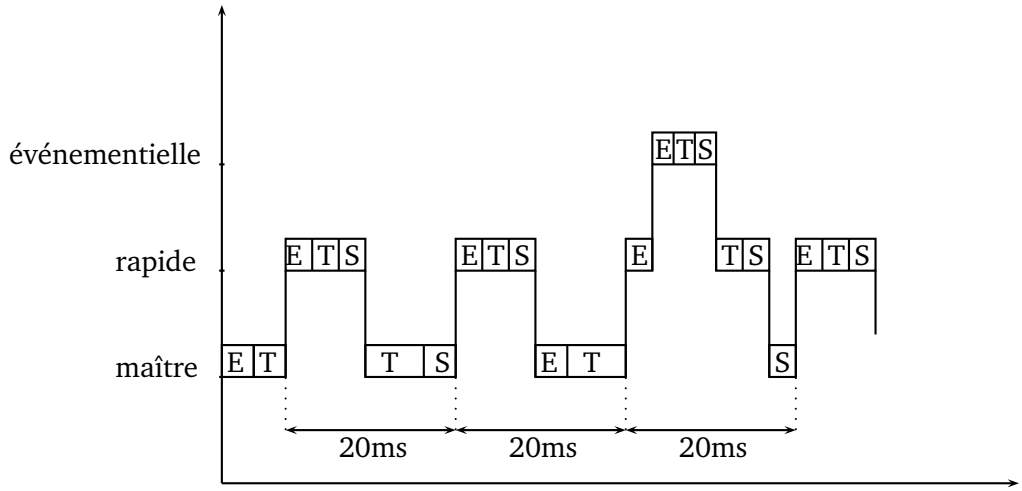
FIG. 3.4 – Illustration du temps de réaction dans un API mono-tâche

contrôle pendant la phase d'*acquisition des entrées* (de la tâche maître) du **prochain cycle** et détecté par le système contrôlé à la fin de celui-ci (sortie e_2).
Dans la programmation multi-tâches, l'exécution cyclique de la tâche maître peut être interrompue pour prendre en compte un éventuel changement de la valeur d'une entrée pendant **le cycle courant** (via une tâche rapide ou événementielle). Ainsi la méthode de programmation d'un API peut augmenter ou réduire le temps de réaction t_r : sa valeur est dans $[t_c, 2.t_c]$ si la programmation est mono-tâche et dans $[t_{min}, t_c]$ sinon, où t_{min} est la borne minimale de temps d'exécution du programme de contrôle (elle dépend du matériel de l'API) et t_c est le temps du cycle de la tâche maître.

Exemple 4
le schéma 3.5 illustre l'ordonnancement des tâches dans un API qui comporte une tâche maître, une tâche rapide de période 20 ms et une tâche événementielle.

3.3 Programmation des APIs

La programmation des automates programmables industriels s'effectue à l'aide de langages spécifiés qui appartiennent en général à trois grandes familles :
– *langage machine :* c'est un langage en binaire, interprété par le microprocesseur d'un ordinateur.
– *Grafcet :* il s'agit d'un langage graphique inspiré des réseaux de Petri, bien adapté aux systèmes à évolution séquentielle. Un programme *Grafcet* décrit un procédé comme une suite d'étapes, reliées entre elles par des transitions. À chaque transition est associée une réceptivité. Celle-ci est une condition logique qui doit être vraie pour franchir la transition et passer à l'étape suivante. Des actions sont associées aux étapes du programme.
Le format graphique d'un programme *Grafcet* (figure 3.6) est le suivant :
  – une étape est représentée par un carré qui a un numéro identificateur. Une étape active est désignée par un point au-dessous du numéro. Les actions associées sont indiquées dans un rectangle relié à la partie droite du carré. Une étape initiale est représentée par un carré doublé.
  – une liaison orientée est représentée par une ligne, parcourue par défaut de haut en bas ou de gauche à droite. Dans le cas contraire, on utilise des flèches.



E : acquisition des entrées
T : traitement interne
S : affectation des sorties

FIG. 3.5 – Ordonnancement de tâches dans un API Schneider

– une transition entre deux étapes est représentée par une barre perpendiculaire aux liaisons orientées qui relient ces étapes. Les réceptivités associées sont inscrites à droite de la barre de transition.

Le *Grafcet* est utilisé aussi bien pour la spécification que pour la programmation.

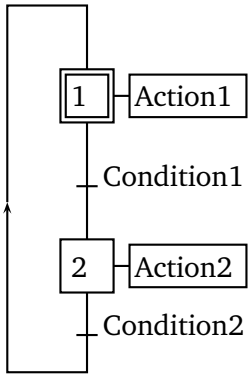
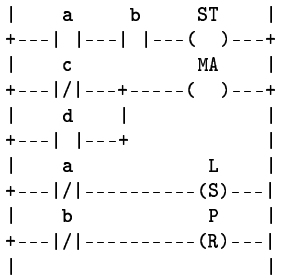


FIG. 3.6 – Programme Grafcet

– *Ladder :* c'est une représentation graphique d'équations booléennes sous une forme analogue à celle des schémas électriques. Le Ladder (une échelle en anglais) est basé sur le principe d'une alimentation en tension représentée par deux traits verticaux (*barres d'alimentation*) reliés horizontalement par des échelons qui contiennent :
  – des *contacts* qui permettent de lire la valeur d'une variable booléenne,

– des *bobines* qui permettent d'écrire la valeur d'une variable booléenne,
– des *blocs fonctionnels* qui permettent de réaliser des opérations plus complexes que la lecture ou l'écriture de variables (paragraphe 3.3.2).

Chaque échelon est associé à une valeur booléenne. L'évaluation de chaque échelon se fait de gauche à droite et le programme se lit de haut en bas. Au bout de chaque échelon figure une *bobine* représentant l'affectation du résultat. Par exemple, le programme de la figure 3.7 réalise les opérations décrites à droite.




```
ST := a and b;
MA := not(c) or d;
IF (a=0) THEN L:=1;
IF (b=0) THEN P:=0;
```

FIG. 3.7 – Programme Ladder

Les constructeurs des APIs proposent pour leurs programmations un ou plusieurs langages qui appartiennent aux familles décrites précédemment. Cependant, bien que les langages d'une même famille se ressemblent, ils ne sont pas nécessairement compatibles, ce qui rend très délicat le transfert du programme de contrôle d'un API à un autre.

3.3.1 La norme IEC 61131-3

Cette norme décrite dans [IEC93, Lew98], résulte de la volonté des constructeurs et des utilisateurs de normaliser ces langages dans le but de faciliter la réutilisation logicielle. Son apport est donc considérable. Cependant, la sémantique des constructions proposées n'est pas définie de façon précise : certaines définitions sont ambiguës ou manquantes. La résolution de ces ambiguïtés justifie certains travaux de recherches, par exemple [Tou97, Can01]. La norme IEC 61131-3 a défini cinq langages pour la programmation des APIs.

– Le langage *SFC* (Sequential Function Chart) appartient à la famille des langages Grafcet. Les actions dans les étapes sont décrites avec les langages *ST*, *IL*, *LD* ou *FBD*.
– Le langage *LD* (Ladder Diagram) appartient à la famille des langages Ladder.
– Le langage *ST* (Structured Text) est un langage textuel de haut niveau (type Pascal) dédié à la description de procédures complexes, difficilement modélisables avec les langages graphiques. C'est le langage par défaut pour la programmation des actions et des réceptivités du langage SFC.
– Le langage *IL* (Instruction List) appartient à la famille des langages machine.
– Le langage *FBD* (Function Block Diagram) est un langage graphique qui permet de réaliser des équations complexes à partir de fonctions élémentaires représentées par des blocs fonctionnels. Les variables d'entrées et de sorties sont connectées aux blocs par des *arcs de liaison*. Une sortie d'un bloc peut être connectée sur une entrée d'un autre bloc (figure 3.8).

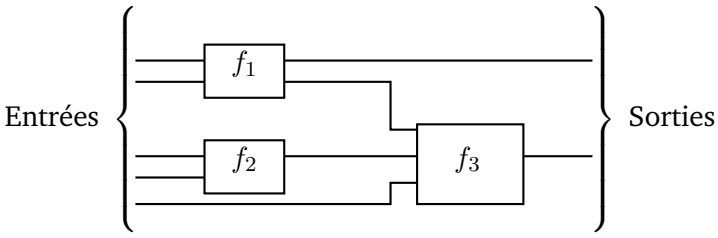


FIG. 3.8 – Programme FBD

3.3.2 Les blocs fonctionnels

L'utilisation des blocs fonctionnels dans la programmation des APIs est un mécanisme très utile, formalisé par la norme IEC 61131-3. Un bloc fonctionnel est représenté par un rectangle, a un nombre prédéfini de points de connexion en entrée (à gauche) et en sortie (à droite). C'est un programme qui réalise une fonction élémentaire entre ses entrées et ses sorties, caractérisée par un type. Il est utilisé d'une manière globale en faisant abstraction de son code interne. Un bloc fonctionnel possède en plus de ses entrées et de ses sorties, des variables internes. Son programme est écrit dans un des langages de la norme. Celle-ci n'exige pas de connecter toutes les entrées et les sorties des blocs fonctionnels au programme global.

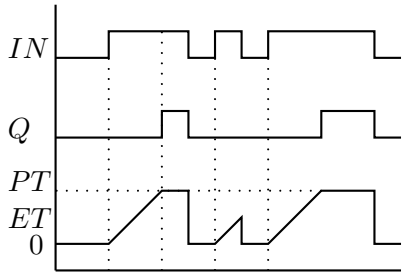


FIG. 3.9 – Chronogramme d'un bloc TON

Les temporisateurs. Ce sont des blocs fonctionnels dont la fonction fait intervenir le temps de façon explicite. Nous décrivons dans ce paragraphe le temporisateur *d'enclenchement (Timer ON-delay (TON))*. Il possède :
– une entrée : une variable booléenne *IN*, qui permet de lancer et d'annuler la temporisation,
– une variable interne *PT* (Preset Time) qui spécifie la durée de la temporisation,
– deux sorties : une variable booléenne *Q* qui vaut 1 si le délai de temporisation a été atteint et une variable *ET* (Elapsed Time) qui indique le temps écoulé depuis le lancement de la temporisation.

Le comportement d'un bloc *TON* peut être décrit par le chronogramme de la figure 3.9. La temporisation est enclenchée lorsque *IN* passe de 0 à 1. La sortie *ET* augmente alors jusqu'à atteindre la valeur de *PT*. À ce moment, la sortie Q est mise à 1. Quand *IN* passe à 0, les sorties *ET* et *Q* sont immédiatement mises à 0. Si la temporisation est abandonnée avant terme

(c'est-à-dire si *IN* passe de 1 à 0 avant que le délai *PT* ne soit atteint), alors *ET* est immédiatement remise à 0.

3.4 État de l'art : vérification des APIs

Dans ce paragraphe nous citons différentes approches pour la vérification formelle des APIs. Des inventaires de ces travaux existent dans [LCRRL99, Fre00c, Mad00]. Ces travaux appartiennent à deux familles complémentaires : la conception de programmes sûrs et la vérification de programmes existants.

Conception de programmes sûrs. Dans le cas des langages de la norme IEC 61131-3, des modèles ont été développés pour l'extraction automatique de programmes corrects. Par exemple, un modèle à base de réseaux de Petri a été proposé dans [Fre98]. Une extension temporisée de ce modèle a été définie dans [Fre00b] avec une traduction vers le langage *SFC*. Une traduction vers les langages *LD* et *IL* a été introduite dans [Fre00a]. Dans [EFP94] un langage de programmation a été défini (c'est un sous ensemble de *ST*) pour la génération de programmes sûrs (pour les APIs) à partir d'une spécification.
D'autre travaux ont proposé des modèles temporisés. Par exemple, dans [Hen97], les auteurs proposent un formalisme implémentable pour l'extraction automatique de programmes sûrs, à base de machines à états temporisées, appelées *PLC-automates*. Ils donnent aussi une traduction de ce formalisme vers du code *ST*. L'approche de conception de programmes sûrs nécessite de la part des constructeurs, un changement majeur de leurs techniques de programmation.

Vérification de programmes existants. Dans ce cadre, les auteurs de [CDP+00] ont introduit une sémantique à base de systèmes de transitions pour vérifier un sous-ensemble du langage *IL*. Dans [HM98b, HM98a], une autre sémantique à base de réseaux de Petri a été proposée pour la vérification d'un autre fragment de ce langage. Dans [Moo94], il a été défini un modèle à base de systèmes de transitions pour les programmes en *Ladder*. Une sémantique à base d'automates temporisés a été proposée par Mader et Wupper dans [MW99]. Le modèle permet de modéliser explicitement la structure de boucle du programme avec une borne minimale et une borne maximale (figure 3.10). Le modèle possède une seule horloge qui est remise à zéro avant le début de chaque cycle. Un invariant $x \leq \varepsilon_2$ est associé à chaque état de contrôle pour représenter la borne maximale et la garde $x \geq \varepsilon_1$ associée à la dernière transition représente la borne minimale.
Les auteurs proposent aussi un modèle pour les temporisateurs qui est mis en parallèle avec le modèle du programme de contrôle. La synchronisation entre ces deux composants est faite par variables booléennes et par messages et elle requiert trois canaux de synchronisation pour chaque temporisateur. Une suite à ce travail est donnée dans [Wil99]. L'auteur a développé un outil pour traduire le programme de contrôle de l'API en un automate temporisé prêt à être implémenté dans l'outil Uppaal.
Un modèle à base de systèmes hybrides pour les programmes SFC a été introduit dans [Huu03].

L'étude proposée dans le reste de ce chapitre, concerne la deuxième approche de la vérification formelle des APIs.

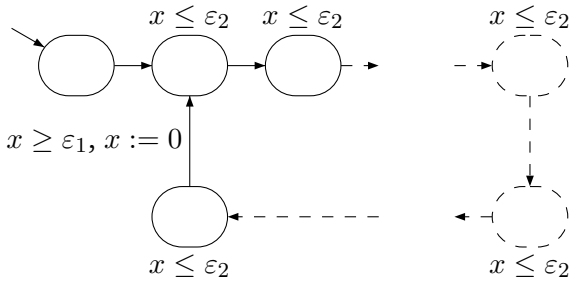


FIG. 3.10 – Le modèle de Mader-Wupper

3.5 Étude de cas : la plate-forme MSS (Mechatronic Standard System)

Cette section est consacrée à l'étude des aspects temporisés et multi-tâches d'un automate programmable industriel. En utilisant la *plate-forme MSS* du groupe Bosch [1], nous proposons une modélisation du système contrôlé ainsi que du programme de contrôle écrit en *Ladder Diagram*. Cette modélisation repose sur une forte hypothèse d'atomicité pour l'ensemble des instructions du programme et sur une simplification du modèle des temporisateurs par rapport à ce qui a été fait dans la littérature précédemment [MW99]. Nous utilisons l'outil Uppaal pour illustrer les performances de cette modélisation par la vérification de propriétés temporisées du système.

3.5.1 Description de la plate-forme MSS

Présentation. La plate-forme MSS (figure 3.11) permet de trier des *pignons* (des petites pièces cylindriques) selon le matériau dont ils sont formés et de leur ajouter ou de leur retirer un *palier* (une petite pièce ronde).
Cette plate-forme est composée de quatre postes consécutifs :

– *poste 1 :* Les *pignons* sont placés manuellement sur des palettes, et sont ensuite transportés et expédiés individuellement par un cylindre pneumatique au poste 2.
– *poste 2 :* Les *pignons* sont transportés par un *chariot*, piloté par une ceinture linéaire bidirectionnelle (figure 3.12). Le chariot s'arrête d'abord sur une *position de test*, où la présence/absence d'un *palier* est détectée. Ce test est effectué par un *vérin* et un *capteur de fin de course*. Le vérin commence à descendre aussitôt que le chariot est en position de test : si le vérin est détecté par le capteur de fin de course alors le palier est absent sinon le palier est présent. Ensuite, le chariot transporte le pignon pour être examiné par 3 capteurs (respectivement inductif, capacitif et optique) pour déterminer son matériau (respectivement acier, cuivre ou plastique noir). L'information récoltée est ensuite transmise au poste 3. Finalement, le chariot transporte le pignon à ce troisième poste.
– *poste 3 :* L'information obtenue précédemment est traitée à cette étape : les pignons sont emportés par le chariot pour une phase d'extraction ou d'insertion de palier.
– *poste 4 :* Les pignons sont déchargés.

L'étude de cas décrite dans cette section concerne le poste 2 de la plate forme MSS.

[1] http//www.boschrexroth.com/country_units/europe/germany/sub_websites/brs_germany/de/didactic/lehrsysteme/mechatronik/mechatronik_standard_system_mss/index.jsp

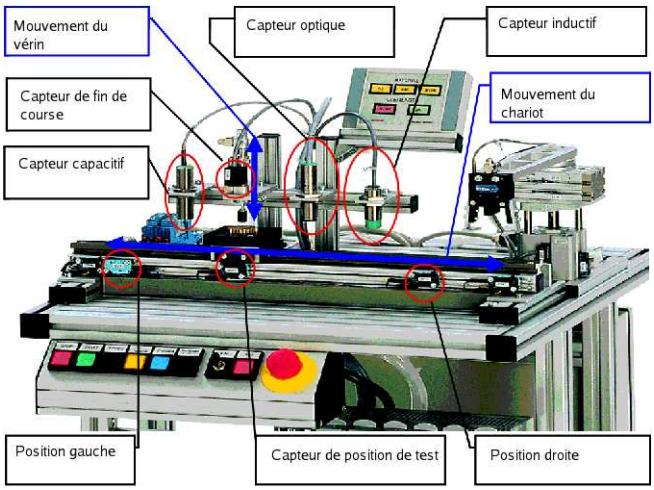


FIG. 3.11 – La plate-forme MSS

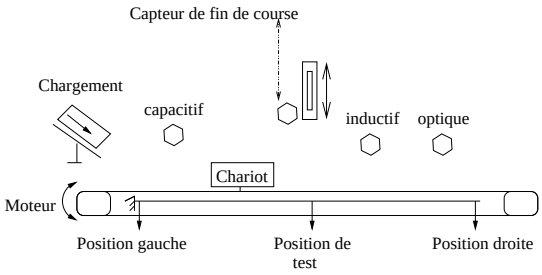


FIG. 3.12 – un schéma de la station 2

Dysfonctionnement du test de vérin. L'arrêt du chariot à la position de test doit se faire avec une précision de 1 mm, permettant au vérin de pénétrer dans le pignon, si le palier est absent. Un dysfonctionnement peut surgir si cette précision n'est pas respectée. Étant donné que le chariot a une grande vitesse de déplacement (200mm/s), la variation du temps de réaction de la sortie correspondant à l'arrêt du chariot (t_r^{stop}) doit donc être inférieure à 5ms (le chariot parcourt 1 mm en 5ms). Une solution pour réduire le temps de réaction serait la programmation multi-tâches. Cette étude de cas a pour but de vérifier qu'un programme de contrôle multi-tâches pour la plate-forme, avec une tâche maître et une tâche événementielle, permet d'obtenir que $t_r^{stop} < 5ms$.
Propriété à vérifier. L'API de la station 2 doit assurer que le chariot s'arrête en moins de 5 ms à la position de test. Cette propriété est notée P dans la suite.

3.5.2 Modélisation de la station 2 de la plate-forme MSS

Le but de cette étude est de valider l'interaction entre le programme de contrôle et le système contrôlé. Nous avons donc besoin de modéliser l'environnement (le chariot, le vérin, etc). Les modèles que nous construisons sont des abstractions du comportement réel de la plate-forme et de l'environnement, qui omettent les aspects non pertinents pour la propriété à vérifier. Cette étape est indispensable pour réduire la taille du modèle et obtenir ainsi une performance acceptable en termes de consommation mémoire et de temps d'exécution. Cependant, le modèle doit conserver tous les comportements qui valident où invalident la propriété. Nous expliquerons en détail les abstractions de chaque composant au moment de décrire le modèle choisi pour le représenter.

3.5.2.1 Structure globale du modèle.

Le modèle complet (figure 3.13) du poste 2 est obtenu par composition des modèles des différents composants (de l'environnement et du programme de contrôle). Un composant

est modélisé par un automate temporisé non déterministe. La communication entre ces modèles se fait via des canaux de synchronisation et des variables partagées. La communication fonctionne de la manière suivante : Le modèle de l'environnement initialise les valeurs des variables d'entrées de l'API. Ces valeurs sont ensuite utilisées par le programme de contrôle pour calculer les valeurs des variables de sorties de l'API. Selon ces nouvelles valeurs, des ordres sont envoyés à l'environnement via des canaux de synchronisation. Un canal particulier (*postest*) sert à activer la tâche événementielle dont le rôle est d'arrêter le chariot.

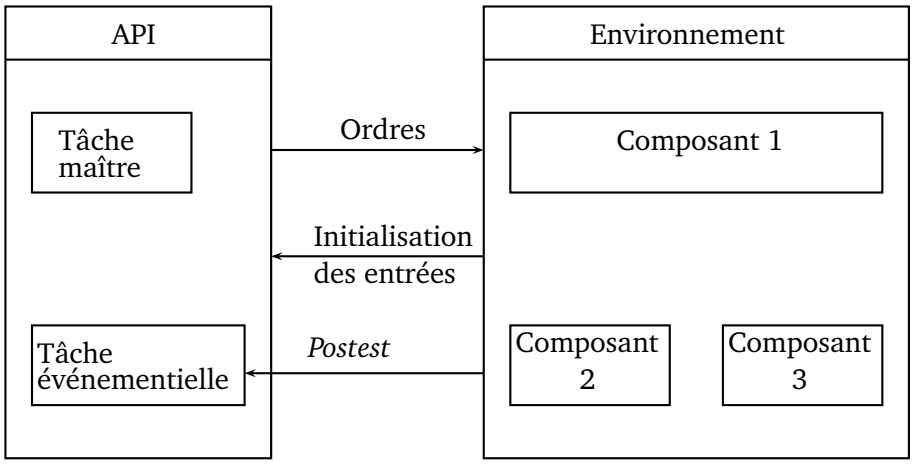


FIG. 3.13 – Structure globale du modèle

3.5.2.2 Programme de contrôle

Le programme de contrôle a été spécifié en langage *SFC* (figure 3.14) à partir du cahier des charges, avant d'être programmé en langage *Ladder*. Il comporte plusieurs temporisateurs de type *TON*. Le comportement cyclique du programme *Ladder* est modélisé par une structure de boucle.

Modèle de la tâche maître. La tâche maître est modélisée par un automate temporisé ayant la forme d'une boucle (figure 3.15), avec une horloge x_cycle qui mesure le temps de cycle dont la durée minimale est 1 et la durée maximale est 10. Cet automate est composé de quatre étapes principales :

1. initialisation des variables d'entrées dont les valeurs permettent d'évaluer les variables correspondant aux réceptivités du programme *SFC*,
2. calcul des valeurs des autres variables du programme : les variables des étapes du programme SFC (x_i représente l'étape i) et les variables de sorties,
3. transmission des sorties : des ordres sous forme de messages (*down_jack !, stop !*, etc.) sont envoyés à l'environnement via des canaux de synchronisation,

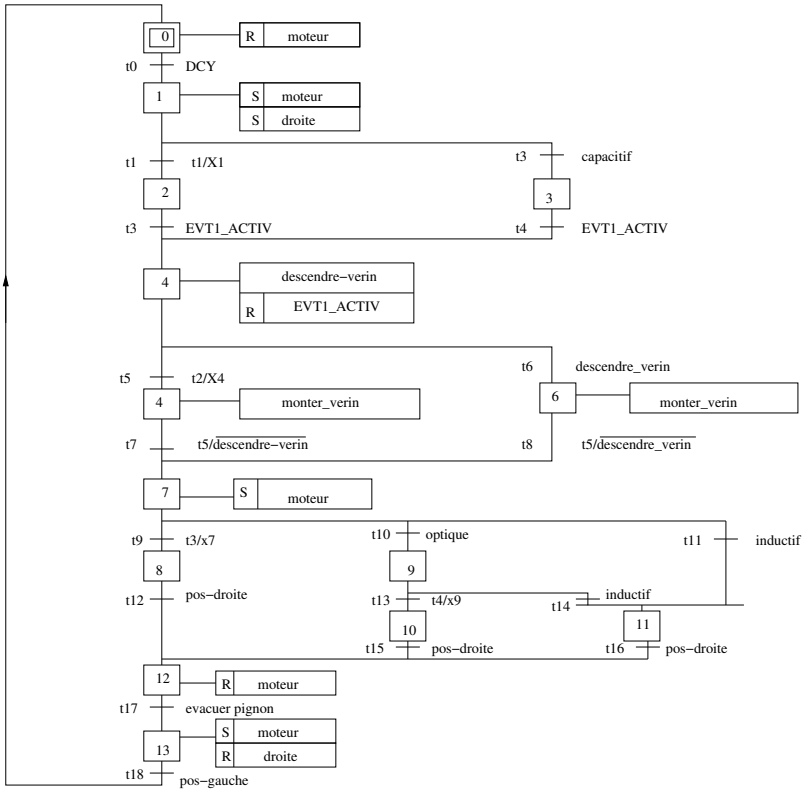


FIG. 3.14 – Programme Grafcet de la station 2

4. remise à zéro de l'horloge modélisant la durée du cycle.

Ce modèle fait abstraction du temps entre les états de ces étapes. Le temps ne peut pas s'écouler dans les états étiquetés par *c* (pour *committed*) mais seulement dans les trois états qui séparent les phases du modèle, pour représenter la durée de leur exécution. Nous expliquerons plus tard dans le paragraphe de la page 44, consacré à la vérification avec l'outil Uppaal, pourquoi cette hypothèse d'atomicité est correcte. En effet, si P est fausse sur le modèle simplifié, elle est fausse aussi sur le modèle initial.

Modèle des temporisateurs. Le programme de contrôle de la plate-forme MSS comporte six temporisateurs, sous forme de blocs *TON*. Chaque bloc *TON* est modélisé par un automate temporisé (figure 3.16), avec deux variables discrètes *ine* (entrée) et *Qe* (sortie), une horloge x pour mesurer le temps écoulé depuis le déclenchement de la temporisation et trois états : *idle*, où le temporisateur est inactif, *running* où le déclenchement a eu lieu, *Timeout* lorsque le délai est atteint. Les variables *ine* et *Qe* sont des variables partagées avec le modèle de la tâche maître. La variable *pte* associée à chaque temporisateur est définie dans la partie déclaration de Uppaal. Les variables *ET* des temporisateurs ne sont pas utilisées par le programme *LD* et donc ne figurent pas explicitement dans les modèles des temporisateurs. Les six blocs *TON* communiquent avec le modèle de la tâche maître via un seul canal *TON* de type broadcast (canal multiple). À chaque début de cycle, la tâche maître initialise les six variables *ine* des blocs. Ensuite un message de synchronisation *TON!* est envoyé, obligeant ainsi les modèles

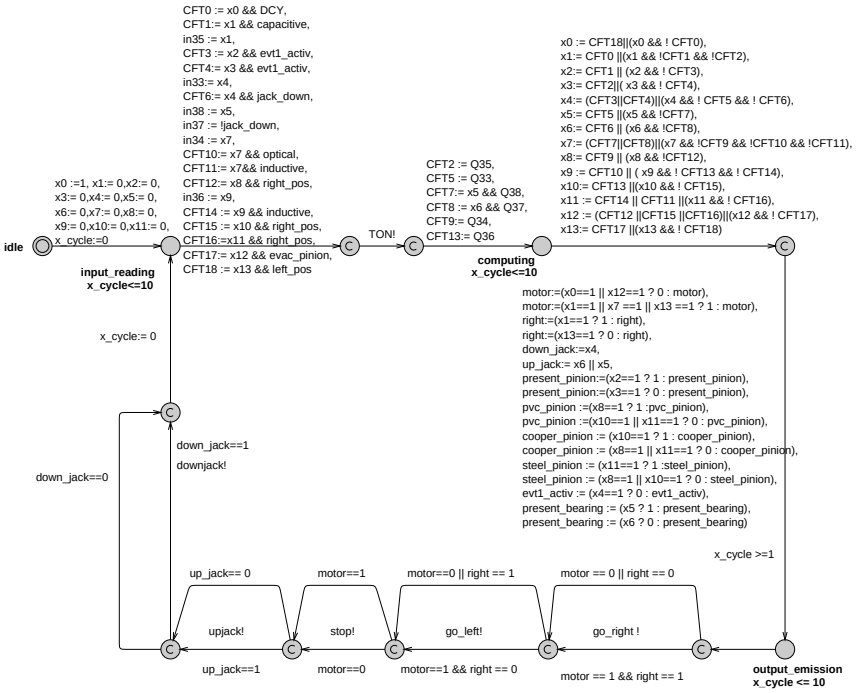


FIG. 3.15 – Modèle de la tâche maître

des temporisateurs à évoluer selon les nouvelles valeurs des variables *ine* : initialement le bloc *TON* est dans l'état *idle*, ensuite il passe à l'état *running* quand il reçoit le message *TON ?* et que sa variable *ine* est égale à 1. Quand la temporisation est atteinte (x a pour valeur la constante *pte*) l'automate va dans l'état *Timeout*. La transition qui revient de *running* à *idle* représente l'abandon de la temporisation. Un temporisateur qui reçoit un message *TON ?* dans l'état *idle* avec la valeur de *ine* égale à 0 reste dans l'état *idle* sans bloquer l'évolution des autres automates grâce à la sémantique des canaux de type multiple.
L'utilisation d'un seul canal de type multiple pour synchroniser les modèles des temporisateurs avec la tâche maître constitue une amélioration par rapport à la modélisation de Mader-Wupper [MW99]. Celle-ci utilise un canal de synchronisation binaire pour chaque temporisateur, ce qui produit un modèle de taille plus grande et un risque de blocage si un automate ne peut pas recevoir un message de synchronisation : une telle situation peut se produire si la garde associée au message n'est pas vérifiée.

Modèle de la tâche événementielle. Cette tâche est représentée par l'automate temporisé de la figure 3.17. Elle doit être déclenchée dès que le chariot est détecté par le capteur placé avant la position de test. La détection est modélisée par la réception du message *postest ?*, envoyé par le chariot dès que ce dernier atteint la position du capteur (figures 3.17, 3.18). La réception de ce message déclenche l'évaluation de l'expression algébrique ($x_2 \vee x_3$) du programme *Ladder* de la tâche événementielle (en effet, le programme *SFC* doit être soit dans l'étape x_2 ou l'étape x_3 pour pouvoir descendre le vérin, si la réceptivité EVT1_ACTIV est mise à 1). Si ce test est satisfait ($x_2 \vee x_3$ vaut 1), les variables *moteur* et *EVT1_ACTIV* sont mises respectivement à 0 et 1. Par conséquent, les ordres *stop !* et *downjack !* sont envoyés (par le

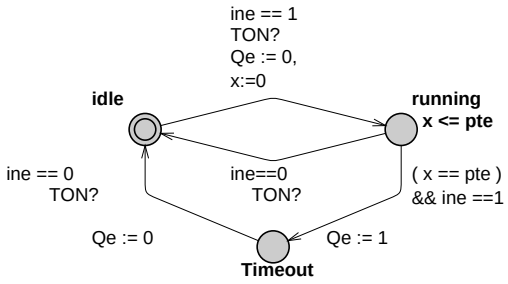
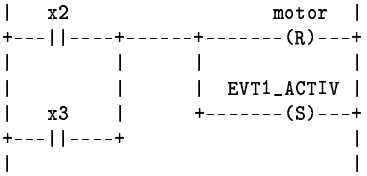


FIG. 3.16 – Modèle d'un temporisateur *TON*

programme de contrôle) pour arrêter le chariot et descendre le vérin. On suppose que la durée d'exécution de la tâche événementielle est nulle.



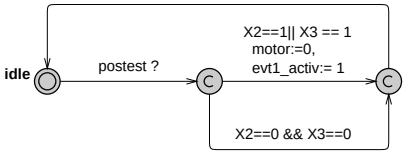


FIG. 3.17 – Le programme *Ladder* et le modèle Uppaal de la tâche événementielle

3.5.2.3 Système contrôlé : environnement

La modélisation de l'environnement permet d'accélérer le processus de vérification, en réduisant l'espace des états accessibles. En effet, si les valeurs des entrées sont générées par un processus non déterministe, toutes les combinaisons possibles des valeurs des entrées seront considérées, même les combinaisons impossibles dans le système réel.
Certains composants de l'environnement ont des comportements discrets (par exemple, les capteurs), alors que les mouvements du chariot et du vérin sont continus (hybrides). Une modélisation avec un outil hybride (par exemple HyTech) serait certainement plus appropriée pour de tels composants. Cependant, le processus de vérification serait plus coûteux et sans garantie de terminaison. Dans le but d'obtenir des performances raisonnables de vérification, nous discrétisons le mouvement physique de ces composants (chariot et vérin). Les horloges sont les seules variables continues dans nos modèles. Cette abstraction est aussi imposée par la sémantique de l'outil Uppaal que nous utilisons car les seules variables continues qui y sont autorisées sont les horloges.

Modèle du chariot. Le chariot est le composant principal de l'environnement car l'évolution des autres composants dépend fortement de sa position. Il communique avec tous les autres modèles par messages ou variables partagées. Le modèle du chariot est un automate temporisé avec une horloge x_c, qui constitue une abstraction discrète du mouvement du chariot. Cette abstraction modélise seulement les positions pertinentes pour la propriété P, c'est-à-dire

les positions stables où le chariot peut s'arrêter ou activer un capteur. Ces positions correspondent aux 6 états : *capteur-inductif*, *capteur-capacitif*, *capteur-optique*, *capteur-test*, *gauche*, *droite*. Entre ces positions, nous modélisons le mouvement du chariot par un seul état avec un invariant qui représente la durée nécessaire pour se déplacer entre deux positions. Par exemple, le temps que met le chariot pour parcourir la distance entre la position *gauche* et la position *capteur-capacitif* est dans l'intervalle $[490ms, 500ms]$.
Ce modèle ne permet donc pas de représenter un éventuel changement de direction du mouvement du chariot entre deux positions stables données. Notons qu'un tel comportement du chariot est aberrant et non intéressant, ce qui justifie cette restriction dans le modèle.
L'automate du chariot envoie des messages de synchronisation aux différents capteurs (induc !, postest !, etc), il modifie aussi certaines variables d'entrées du programme de contrôle. Il est représenté dans la figure 3.18.

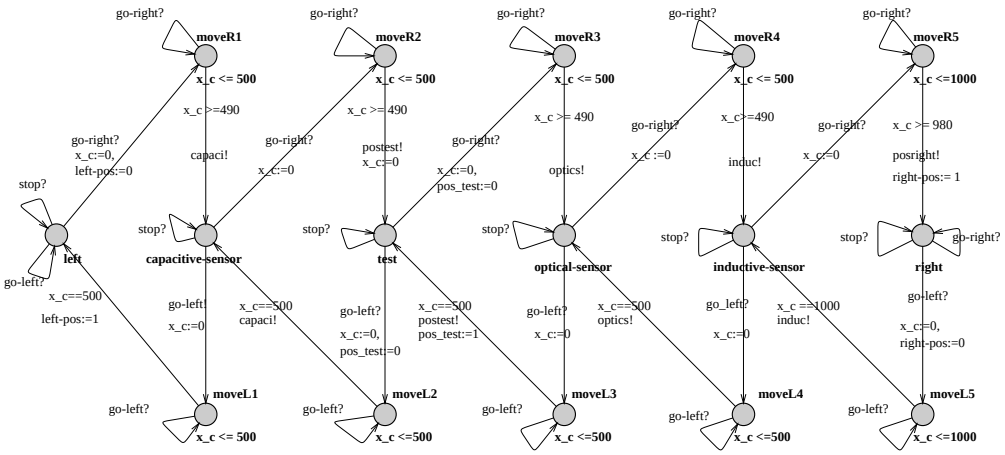


FIG. 3.18 – Modèle du chariot

Modèle du vérin. Le vérin a un mouvement vertical descendant jusqu'à une position limite qui correspond à l'absence de palier dans le pignon (voir figures 3.11, 3.12). Cette position limite est détectée dans un temps fixé par un *capteur de fin de course*. Le vérin commence sa descente dès que le chariot s'arrête à la position de test. Si la réponse du *capteur de fin de course* est positive alors le palier est absent sinon le palier existe. Le modèle du vérin (figure 3.19) dépend donc des caractéristiques des pignons. Ces caractéristiques sont modélisées par la variable *ob* (tableau 3.1). Celle-ci code les six types possibles de pignons et elle est initialisée par un processus non déterministe (figure 3.21). La valeur 0 de *ob* correspond à l'absence de pignon c'est-à-dire à un chariot vide. Initialement, l'automate modélisant le vérin est dans l'état *top* et va dans l'état *go_down* dès qu'il reçoit le message *downjack ?* du programme de contrôle. Il reste dans cet état si le pignon contient un palier, cette condition est représentée dans le modèle par $ob == 1 \wedge ob == 5 \wedge ob == 3$, sinon il arrive à l'état *limiting_position*.

La variable *ob*	Signification
0	absence de pignon
1	acier et présence d'un palier
2	acier et absence d'un palier
3	cuivre et présence d'un palier
4	cuivre et absence d'un palier
5	plastique noir et présence d'un palier
6	plastique noir et absence d'un palier

TAB. 3.1 – Les significations des variables *ob*

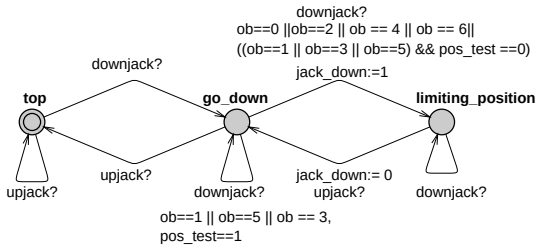


FIG. 3.19 – Modèle du vérin

Modèles des capteurs. Les capteurs capacitif, optique et inductif sont modélisés par des automates temporisés synchronisés avec l'automate du chariot. Les modèles sont représentés dans la figure 3.20 et ils ont tous des fonctionnements similaires. Le chariot envoie des messages de synchronisation aux capteurs quand il arrive à leur position respective (par exemple *optics ?* pour le capteur optique). Les modèles des capteurs modifient ainsi les valeurs des variables correspondantes selon la nature du pignon transporté par le chariot. Ces valeurs sont ensuite utilisées par le programme de contrôle.

Modèle de l'environnement extérieur. Les opérations de chargement et de déchargement des pignons ont lieu respectivement à la position gauche et à la position droite du poste 2. Ces opérations ne sont pas contrôlées par le programme de contrôle. Ce dernier attend leur terminaison qui est communiquée par les variables *DCY* pour le chargement et *evac_pinion* pour le déchargement. L'automate de la figure 3.21 modélise ce fonctionnement en choisissant d'une manière non-déterministe un type de pignon (représenté par le choix d'une valeur pour *ob*), et en modifiant les valeurs de *DCY* et *evac_pinion*.

3.5.2.4 Vérification avec l'outil Uppaal

La propriété P est testée grâce à un automate additionnel qui joue le rôle d'un observateur externe au modèle décrit précédemment. Il calcule le délai séparant l'arrivée du chariot à la position du capteur de test et son arrêt physique. L'automate que nous ajoutons est décrit dans la figure 3.22. Initialement, il est dans l'état *idle* avec la valeur de l'horloge X égale à 0. Lorsqu'il reçoit le message *postest ?* (envoyé par le chariot dès son arrivée à la position

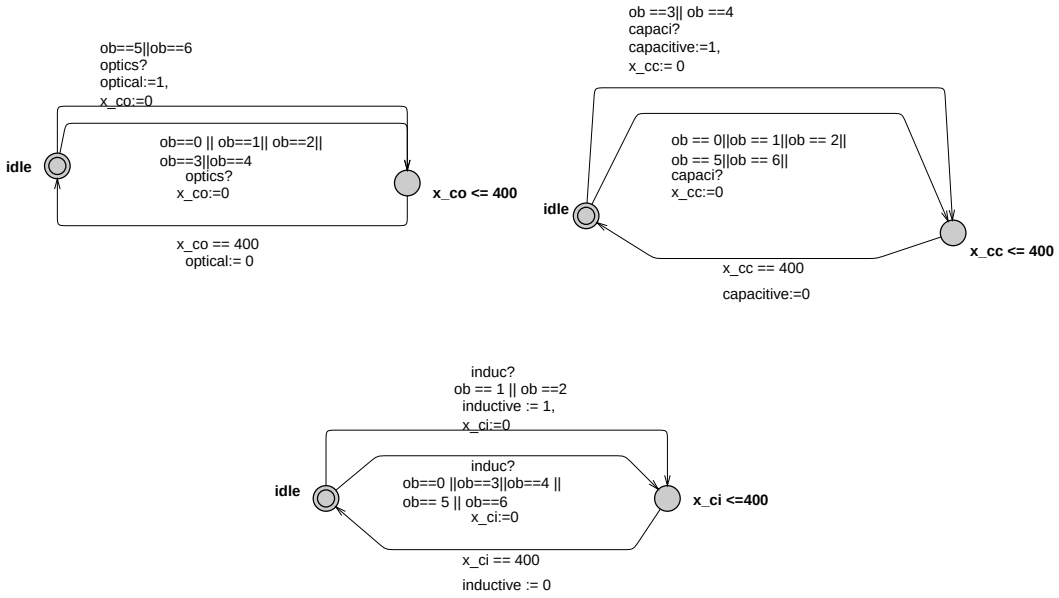


FIG. 3.20 – Les modèles des capteurs

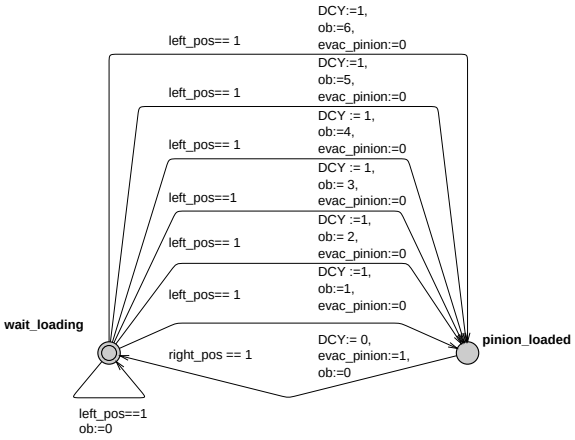


FIG. 3.21 – Modèle de l'environnement extérieur

du capteur de test), il va dans l'état *obs* en mettant X à zéro. Dans cet état la valeur de X augmente. Quand le programme de contrôle émet le message *stop !* (sur un canal multiple), le message est reçu par le chariot et par l'observateur qui va dans l'état *stop*. Donc la valeur de l'horloge X à l'arrivée dans l'état *stop* de l'observateur correspond au temps de réaction t_r^{stop} nécessaire à l'arrêt du chariot. L'automate de l'observateur facilite l'expression de la propriété à vérifier. Nous exprimons en fait la négation de cette propriété par une formule d'accessibilité C_1 (tableau 3.2), dans la syntaxe Uppaal : E<> ($obs.stop$ and $X > 5$), l'opérateur E<> signifie "il existe un chemin tel que dans le futur..." et $obs.stop$ désigne l'état *stop* de l'observateur. Cette formule signifie qu'il est possible d'atteindre l'état *stop* de l'observateur avec la

valeur de l'horloge X supérieure à $5ms$.

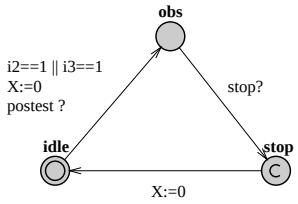


FIG. 3.22 – L'observateur.

Nous avons modélisé les deux situations suivantes : (1) la tâche événementielle est inactive (programmation mono-tâche) et (2) la tâche événementielle est active (programmation multi-tâches). Nous avons aussi réalisé un modèle comme celui de [MW99] pour la plate-forme MSS avec le programme multi-tâches. Notons que la propriété P est évidemment vérifiée si la tâche événementielle peut elle-même transmettre instantanément l'ordre d'arrêt. Cependant, cette manière de programmer est difficile à mettre en oeuvre dans l'API de la plate-forme MSS.

propriété	résultat	temps	mémoire
avec tâche événementielle			
C1 :E<> obs.stop and $X > 5$	oui	15 s	30 Mo
C2 :E<> obs.stop and $X \leq 5$	oui	15 s	30 Mo
C3 :E<> obs.stop and $X > 10$	non	22 s	61 Mo
sans tâche événementielle			
C5 :E<> obs.stop and $X \geq 10$	oui	16 s	30 Mo
C6 :E<> obs.stop and $X > 20$	non	22 s	70 Mo
C7 :E<> obs.stop and $X < 10$	non	22 s	69 Mo
modèle de Mader-Wupper			
C8 :E<> obs.stop and $X > 5$	-	-	-

TAB. 3.2 – Résultats des expériences pour le poste 2

Nous avons implémenté ces modèles dans l'outil Uppaal. Le modèle global comporte environ 30.10^6 configurations. Les propriétés $C_1, \ldots, C_8$ sont des variantes de la propriété cherchée, dans lesquelles apparaissent plusieurs contraintes possibles sur la valeur de X. Le tableau 3.2 récapitule les résultats des processus de vérification.
Cette étude a été réalisée sur une machine linux, pentium4, 2.4 GHz et 3 Go RAM. Notons d'abord que dans le cas du modèle de Mader-Wupper, nous avons été confrontés à des pro-

blèmes de mémoire et de temps de calculs très importants. Nous avons arrêté le processus de vérification après 29 heures d'exécution. Le tableau 3.2 compare aussi les temps de réaction entre la programmation mono-tâche et la programmation multi-tâches. Les résultats confirment ce qui a été expliqué dans le paragraphe de la page 31. En effet, les propriétés C_5, C_6 et C_7 montrent que le chariot s'arrête dans un délai compris entre t_c et $2t_c$ après son arrivée au capteur de test. La propriété C_3 prouve que le chariot s'arrête dans un délai inférieur à t_c. On conclut de cette étude que la programmation multi-tâches réduit bien le temps de réaction global mais qu'elle n'est pas suffisante pour garantir la propriété P.
Ces automates (implémentés dans Uppaal) sont des modèles simplifiés des programmes réels. Ils abstraient l'écoulement du temps dans certains états du programme de contrôle. Si C_1 (la négation de P) est vraie dans le modèle abstrait alors elle est vraie aussi dans le programme réel. En effet, si le temps s'écoule dans les états intermédiaires, la valeur de l'horloge X peut être plus grande mais jamais plus petite. Cependant, si C_1 est fausse dans le modèle abstrait, elle ne l'est pas forcément dans le programme réel car du temps peut s'écouler juste avant l'émission du message *stop ?* par le programme du contrôle. On peut conclure (tableau 3.2) que la propriété P n'est pas satisfaite dans la plate forme MSS : la précision de 1mm concernant l'arrêt du chariot n'est pas toujours respectée par le programme de contrôle.

3.6 Conclusion

Dans ce travail, nous avons présenté une sémantique formelle à base d'automates temporisés pour une sous-classe de programmes Ladder comportant des blocs TON. Nous l'avons utilisée pour modéliser le programme de l'API qui contrôle le poste 2 de la plate-forme MSS. Nous avons aussi modélisé les autres composants physiques de ce système réel. En utilisant l'outil Uppaal, nous avons prouvé que la programmation multi-tâches réduit effectivement le temps de réaction du chariot à l'émission d'un ordre d'arrêt provenant du programme de l'API. Cependant, ce n'est pas suffisant pour garantir la précision d'arrêt de 1 mm décrite dans le cahier de charges.

Pour des raisons d'efficacité, la modélisation simplifiée de la plate-forme MSS que nous proposons utilise seulement des horloges et des variables bornées, ce qui permet l'utilisation de l'outil Uppaal pour l'implémentation. D'autre part les modules de simulation et d'interface graphique de cet outil ont été très utiles lors de la phase de modélisation. Ils ont permis une étape intermédiaire de test du modèle.

Malgré la taille importante du système et le nombre de variables manipulées par le programme, nous avons obtenu des résultats raisonnables pour le temps de vérification (moins de 30 s). Alors que nous étions obligés d'interrompre la vérification de la même formule avec le modèle de Mader-Wupper. Cette performance est due à deux raisons principales :

1. l'hypothèse d'atomicité : nous abstrayons l'écoulement du temps à l'intérieur des quatre étapes principales du programme de contrôle. Le temps s'écoule seulement entre ces étapes.
2. le modèle simplifié des blocs TON : nous utilisons un seul canal de type multiple pour synchroniser tous les blocs TON avec le programme de contrôle, au lieu de trois canaux de type binaire comme dans le modèle de Mader-Wupper.

Dans ce travail, nous avons été confrontés à des problèmes de vérification pratique, ainsi qu'au problème de l'explosion combinatoire que nous avons partiellement résolu dans ce cas par des abstractions de certains comportements du système réel et du programme *Ladder*.

La suite de ce travail est consacrée à l'étude d'une extension temporisée de la logique TCTL, destinée à répondre à des questions pratiques liées à la modélisation des APIs : l'abstraction d'états transitoires.

Deuxième partie

Spécifications et algorithmes

Chapitre 4

Abstraction des états transitoires

À partir des problèmes pratiques posés par la modélisation des APIs, nous avons isolé une question théorique : celle de la vérification de propriétés temporisées “partout sauf sur un sous-ensemble d'états de mesure bornée (éventuellement nulle)”. En effet, la modélisation des programmes de contrôle des automates programmables industriels produit naturellement des automates temporisés comportant des états intermédiaires qu'il n'est pas important de vérifier pour la spécification globale. C'est le cas par exemple d'états où le modèle reste une durée “négligeable” par rapport aux temps d'exécution. Afin d'abstraire de tels états transitoires lors de la spécification de propriétés, nous avons défini une logique temporisée qui étend TCTL. Pour cette logique, nous proposons deux sémantiques : une sémantique locale et une sémantique globale. Ce chapitre est consacré à la présentation de ces sémantiques : nous y étudions les questions d'expressivité et nous montrons que le model-checking de la seconde sémantique est indécidable.

4.1 Motivations

Le modèle d'un programme manipulant des variables est souvent obtenu en définissant les états de contrôle comme des n-uplets de valeurs de ces variables. Les transitions sont alors des mises à jour de ces variables avec des gardes. Lorsqu'un programme utilise des variables discrètes (entières ou booléennes) toute modification de la valeur d'une variable correspond à un changement d'état de contrôle. Plus précisément, la valeur change au moment de quitter un état pour arriver dans un autre. Ainsi, on peut obtenir des erreurs de vérification dues uniquement à cette modélisation : une variable peut avoir deux valeurs différentes dans un même instant. Nous illustrons cette ambiguïté par un exemple qui représente un API dont le programme est écrit en langage SFC (figure 4.1).
Cet API contrôle un dispositif réel de sûreté [1]. Il est utilisé dans l'industrie pour des machines dangereuses afin de s'assurer que durant leur fonctionnement les deux mains de l'ouvrier sont en dehors de la partie zone de danger. Pour cela, l'API contrôle le démarrage de la machine en imposant que les deux boutons soient simultanément (en moins de 0.5 seconde) enfoncés : dès qu'un bouton est relâché, la machine doit s'arrêter.
Ce dispositif peut être modélisé par trois automates temporisés (figure 4.2), qui communiquent par deux variables booléennes partagées *L* (pour le bouton gauche) et *R* (pour le

[1] http ://www.ab.com/en/epub/catalogs/3377539/5866177/4444281/4444297/4444640/

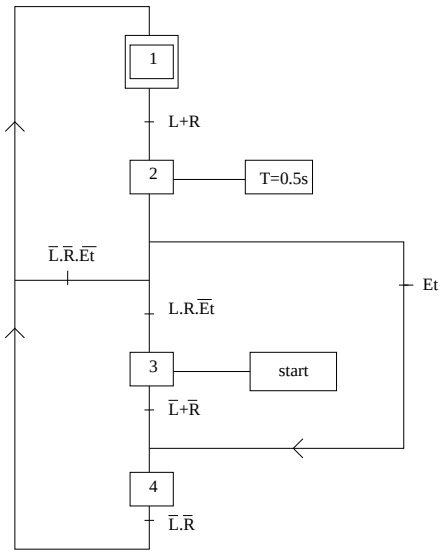


FIG. 4.1 – le programme SFC pour le dispositif du presse boutons

bouton droit). L'ensemble des valeurs possibles de ces variables est modélisé par deux automates non déterministes. Notons d'abord que les transitions du modèle du programme de contrôle sont urgentes, c'est-à-dire qu'elles sont franchies dès que leur garde est vérifiée. Ce modèle a la même structure que le programme SFC. Il possède une horloge t pour mesurer la temporisation (0.5 seconde). À partir de l'état *init*, le programme de contrôle passe dans l'état S_1. Dès qu'un bouton est pressé (la garde $L \vee R$ est vraie), l'automate déclenche la temporisation ($t := 0$) et passe dans l'état S_2. Si le deuxième bouton est pressé avant la fin de la temporisation ($L \wedge R \wedge t < 0.5$), l'automate va dans l'état *running* et la machine démarre en mettant le signal s à 1 ($s := 1$). Sinon, deux cas peuvent se produire : soit le premier bouton est relâché ($!L \wedge !R \wedge t < 0.5$), soit la temporisation est atteinte ($t \geq 0.5$). Dans ces deux cas, le processus doit être repris, le programme de contrôle revient dans l'état S_1 et la variable s reste égale à 0.

Considérons maintenant la propriété de sûreté suivante :

`"si la machine est en marche alors les deux boutons sont enfoncés"`

Cette propriété se traduit dans notre modèle par : `si` $s = 1$ `alors` $L = 1$ `et` $R = 1$. Un programme de contrôle correct de ce dispositif doit assurer que cette formule est vérifiée à tout instant. Or, dans le modèle de la figure 4.2, cette formule n'est pas toujours satisfaite. Il existe un instant dans l'état *running* du modèle du programme de contrôle, où $s = 1$ `et` $(L = 0$ `ou` $R = 0)$. Cet instant correspond au moment où on relâche un des deux boutons. En effet, après qu'une des deux valeurs L ou R est remise à zéro, l'automate se trouve au **même moment**, pendant une durée nulle, dans l'état *running* où $s = 1$ et dans l'état suivant où $s = 0$. Ce phénomène est dû au fait que le changement de valeur de L ou R entraîne un changement de la sortie s (en temps nul) mais cette séquence de mises à jour fait apparaître des états "artificiels" dans le modèle. Par conséquent, la propriété P n'est pas toujours vraie dans le modèle mais elle est "presque toujours" vraie c'est-à-dire qu'elle peut être fausse pour

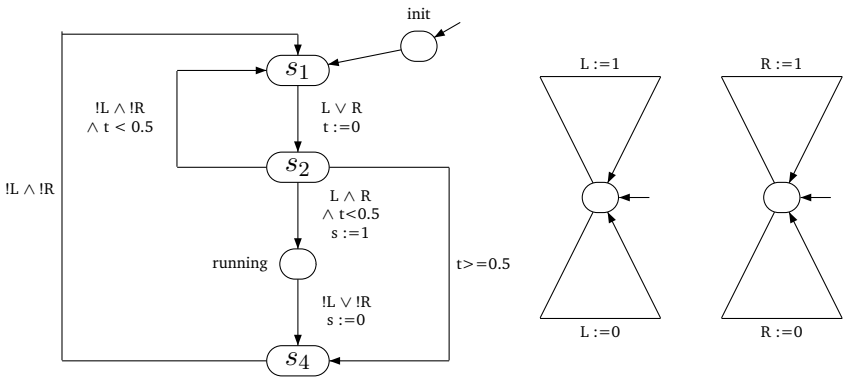


FIG. 4.2 – Modèle pour le programme de contrôle et les deux boutons

une durée nulle. Ainsi, ces états transitoires peuvent produire des erreurs de vérification et le but de ce chapitre est d'introduire un formalisme de spécification adapté pour éviter de telles erreurs.

Il est parfois possible de modifier le modèle pour résoudre ce type de problème. Dans l'exemple, on peut remplacer les variables L et R par des canaux de synchronisation. Il faut alors modifier le modèle du programme de contrôle pour synchroniser ses transitions avec celles des modèles des deux boutons, correspondant à des mises à jour des deux variables. Cependant, cette manière de faire n'est pas automatique et suppose de pouvoir isoler à l'avance les états qui sont la cause des ambiguïtés. Ces états ne correspondent pas toujours aux instants de franchissement des transitions. Une formule peut devenir fausse pour une durée nulle au milieu d'un état de contrôle, nous en verrons plus loin un exemple (page 57). De plus, un modèle comme celui de la figure 4.2 est une traduction fidèle du programme SFC ce qui ne serait pas le cas en remplaçant les variables par des canaux de synchronisation. Le fait que le modèle reflète étroitement le programme est un avantage dans le cas des APIs puisque, généralement, le programme SFC est utilisé seulement pour la spécification et sera traduit plus tard de manière automatique en un autre langage pour l'implémentation.

Une deuxième possibilité pour éviter ces erreurs de vérification est de modifier la formule, en exigeant que celle-ci soit vraie seulement dans les états non ambigus. C'est ce qui a été fait par exemple dans le travail de vérification du protocole ABR [BFKM03] avec l'outil HyTech. Cependant, cette manière de faire requiert aussi une bonne connaissance préalable du système.

Une autre façon plus générale de traiter ce problème, est de restreindre la sémantique des automates temporisés, en associant au plus une configuration à chaque instant. Cependant, ce type de solution n'est guère satisfaisant. En pratique, il est souvent utile de pouvoir exécuter plusieurs actions de manière instantanée, c'est-à-dire sans délai entre ces actions. Cette hypothèse d'atomicité donne des modèles plus simples et réduit considérablement le temps de vérification : c'était le cas de l'exemple de la plate forme MSS étudié au chapitre 2. La propriété d'atomicité est aussi nécessaire pour simuler la synchronisation n-aire quand seule-

ment la synchronisation binaire est possible.

Pour certaines propriétés, il est possible d'ajouter un automate observateur. Par exemple, pour vérifier qu'une proposition atomique p est vraie sauf pendant des durées nulles, l'observateur ira dans un état erreur s'il reste plus de zéro unité de temps dans l'état étiqueté par $\neg p$. Mais, cette méthode ne peut s'appliquer qu'aux propriétés de sûreté [ABBL03].

Dans certains cas, l'abstraction des configurations de durée nulle n'est pas suffisante, par exemple pour les systèmes avec un temps de stabilisation non nul. Sur de tels systèmes, il est souvent utile de pouvoir vérifier une propriété seulement pendant les phases stables et d'abstraire les intervalles de temps où le signal d'entrée peut changer et donner un signal de sortie erroné.
Dans ce chapitre, nous proposons une solution au problème évoqué ci-dessus qui consiste à étendre la logique TCTL avec de nouvelles modalités U^k avec $k \in \mathbb{N}$. Ces modalités permettent d'ignorer les événements qui ne se produisent pas de manière continue pendant au moins k unités de temps. Par exemple, la formule AF$^2_{\leq 100}$ *alarme* exprime que, pour n'importe quelle exécution, la proposition atomique *alarme* se déclenche avant 100 unités de temps et durera pendant au moins 2 unités de temps. Nous pouvons aussi exprimer avec cette logique le fait qu'un événement "significatif" a précède un événement "significatif" b où un événement est significatif s'il dure (de manière continue) pendant au moins k unités de temps. Par exemple la formule A *demande* P^3 *réponse* indique que, le long de toute exécution, si l'événement *réponse* a duré plus de 3 u.t alors une demande a été faite dans le passé et a duré pendant au moins 3 u.t. Dans un premier temps, nous allons définir une nouvelle logique TCTL$^{\Delta}$ et donner la sémantique de ces opérateurs. Ensuite, nous prouvons des équivalences entre différentes formules de cette logique et nous étudions l'expressivité de ces opérateurs. À la fin du chapitre, nous considérons une sémantique *globale* permettant d'exprimer que la durée totale -le long d'une exécution- où une formule est fausse est bornée par un entier k avec $k \in \mathbb{N}$. Nous montrons que le model-checking de notre logique munie de cette sémantique, qui est plus naturelle que la précédente, est indécidable.

4.2 Définition de la logique pour abstraire des états transitoires : TCTL$^{\Delta}$

Nous étendons dans ce paragraphe la syntaxe de la logique TCTL pour exprimer des propriétés sur des états stables i.e durant au moins k unités de temps. On note TCTL$^{\Delta}$ cette nouvelle logique. Elle est obtenue par l'ajout des modalités E_ U^k _ et A_ U^k _, avec $k \in \mathbb{N}$.

Définition 15 (La logique TCTL$^{\Delta}$)
La logique TCTL$^{\Delta}$ *est définie par la syntaxe suivante :*

$$\varphi, \psi ::= P_1 \mid P_2 \mid \ldots \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{E}\varphi\mathsf{U}_{\sim c}\psi \mid \mathsf{A}\varphi\mathsf{U}_{\sim c}\psi \mid \mathsf{E}\varphi\mathsf{U}^k_{\sim c}\psi \mid \mathsf{A}\varphi\mathsf{U}^k_{\sim c}\psi$$

où P_i sont des éléments de l'ensemble AP*, $\sim$ appartient à l'ensemble $\{<, >, \leq, \geq, =\}$ et c, k sont des éléments de $\mathbb{N}$.*

La modalité EφU$^k_{\sim c}\psi$ signifie "il est possible d'atteindre un intervalle suffisamment long ($> k$) où ψ est vraie, autour d'une position située à une distance $\sim c$ et, avant cette position, φ

est partout vraie sauf le long de sous-chemins de durées négligeables ($\leq k$)". La modalité $\mathsf{A}\varphi\mathsf{U}^k_{\sim c}\psi$ signifie "le long de tout chemin, ψ dure suffisamment longtemps ($> k$) autour d'une position à distance $\sim c$ et, avant cette position, φ est partout vraie sauf le long de sous-chemins de durées négligeables ($\leq k$)".

Nous formalisons la notion de durée pour les sous-chemins d'une exécution :

Définition 16 (La mesure $\hat{\mu}$)
Soient $\mathcal{T}$ un système de transitions temporisé, ρ une exécution de $\mathcal{T}$ et P un sous-ensemble de positions de ρ. La mesure de P notée $\hat{\mu}(P)$, est définie par $\mu(\{Time(\rho^{\leq p}) \mid p \in P\})$, où μ est la mesure de Lebesgue sur l'ensemble des réels, sous réserve que l'ensemble $\{Time(\rho^{\leq p}) \mid p \in P\}$ soit mesurable.

Notons que la mesure d'un sous-chemin $p \overset{\sigma}{\mapsto} p'$ est égale à $\hat{\mu}(\sigma) = \mathsf{Time}(\rho^{\leq p'}) - \mathsf{Time}(\rho^{\leq p})$.

4.3 Sémantique de la logique TCTL$^{\Delta}$

Nous donnons maintenant la sémantique des formules de TCTL$^{\Delta}$. Ces formules sont évaluées dans un état s d'un système de transitions temporisé $\mathcal{T}$.
Par extension, on écrit $\sigma \models \varphi$, où σ est un sous-chemin, lorsqu'on a $s_p \models \varphi$, pour toute position p de σ.

Définition 17
Les clauses suivantes définissent d'une manière inductive, quand un état s de $\mathcal{T}$ vérifie une formule de TCTL$^{\Delta}$*. La sémantique des opérateurs booléens, de* E_U$_{\sim c}$_ *et de* A_U$_{\sim c}$_ *est la même que celle de* TCTL*.*

$$
\begin{array}{lll}
s \models \mathsf{E}\varphi\mathsf{U}_{\sim c}\psi & \textit{si et seulement si} & \exists\, \rho \in \textit{Exec}(s)\ \textit{t.q}\ \rho \models \varphi\mathsf{U}_{\sim c}\psi \\
s \models \mathsf{A}\varphi\mathsf{U}_{\sim c}\psi & \textit{si et seulement si} & \forall\, \rho \in \textit{Exec}(s),\ \rho \models \varphi\mathsf{U}_{\sim c}\psi \\
s \models \mathsf{E}\varphi\mathsf{U}^k_{\sim c}\psi & \textit{si et seulement si} & \exists\, \rho \in \textit{Exec}(s)\ \textit{t.q}\ \rho \models \varphi\mathsf{U}^k_{\sim c}\psi \\
s \models \mathsf{A}\varphi\mathsf{U}^k_{\sim c}\psi & \textit{si et seulement si} & \forall\, \rho \in \textit{Exec}(s),\ \rho \models \varphi\mathsf{U}^k_{\sim c}\psi \\
\textit{avec} & & \\
\rho \models \varphi\mathsf{U}_{\sim c}\psi & \textit{si et seulement si} & \exists\, p \in \rho\ \textit{t.q}\ \mathsf{Time}(\rho^{\leq p}) \sim c \wedge s_p \models \psi \wedge \forall p' <_\rho p,\ s_{p'} \models \varphi \\
\rho \models \varphi\mathsf{U}^k_{\sim c}\psi & \textit{si et seulement si} & \exists\, \sigma \in SC(\rho)\ \textit{t.q} \\
 & & \big(\exists\, p \in \sigma \mid \mathsf{Time}(\rho^{\leq p}) \sim c\big) \wedge \hat{\mu}(\sigma) > k \wedge \sigma \models \psi \\
 & & \wedge \forall\, \sigma' \in SC(\rho)\ \textit{t.q}\ \sigma' <_\rho p,\ \sigma' \models \neg\varphi \Rightarrow \hat{\mu}(\sigma') \leq k
\end{array}
$$

Notons que nous requérons que ψ soit vraie tout le long d'un sous-chemin (de mesure $> k$), autour d'une position $\sim c$, (au lieu d'une unique position comme pour la sémantique classique de TCTL) afin que le "témoin" de ψ ne correspond pas à une séquence d'états transitoires. De plus, nous verrons plus loin (section 4.3.1 et 4.3.2) que cette condition sur ψ est très utile quand on a besoin de faire la négation des modalités U^k, par exemple pour définir les opérateurs $\mathsf{EG}^k_{\sim c}$, $\mathsf{AG}^k_{\sim c}$ et P^k.

Notons aussi que dans le cas où k vaut 0, la sémantique de U^0 est équivalente à la définition ci-dessous :

$$\rho \models \varphi \mathsf{U}^0_{\sim c} \psi \quad \text{si et seulement si} \quad \begin{array}{l} \exists\, \sigma \in SC(\rho) \text{ t.q} \\ \big(\exists\, p \in \sigma \mid \mathsf{Time}(\rho^{\leq p}) \sim c\big) \wedge \hat{\mu}(\sigma) > 0 \wedge \sigma \models \psi \\ \text{et } \hat{\mu}\{p' <_\rho p \mid s_{p'} \models \neg\varphi\} = 0 \end{array}$$

4.3.1 Définition d'autres opérateurs

Soit φ une formule de TCTL$^{\Delta}$. Nous définissons ici d'autres modalités à partir des modalités de base de TCTL$^{\Delta}$:

– $\mathsf{EF}^k_{\sim c}\,\varphi \stackrel{def}{=} \mathsf{E}(\top \mathsf{U}^k_{\sim c}\,\varphi)$
On a donc :

$$s \models \mathsf{EF}^k_{\sim c}\,\varphi \;\Leftrightarrow\; \begin{array}{l} \exists\, \rho \in \text{Exec}(s),\ \exists\, \sigma \in SC(\rho) \text{ tel que} \\ \big(\exists\, p \in \sigma \text{ t.q } \mathsf{Time}(\rho^{\leq p}) \sim c\big) \wedge \hat{\mu}(\sigma) > k \wedge \sigma \models \varphi \end{array}$$

Ainsi, la formule $\mathsf{EF}^k_{\sim c}\varphi$ signifie qu'il est possible que φ dure suffisamment longtemps ($> k$) autour d'une position $\sim c$.

– $\mathsf{AF}^k_{\sim c}\,\varphi \stackrel{def}{=} \mathsf{A}(\top \mathsf{U}^k_{\sim c}\,\varphi)$
On a donc :

$$s \models \mathsf{AF}^k_{\sim c}\,\varphi \;\Leftrightarrow\; \begin{array}{l} \forall\, \rho \in \text{Exec}(s),\ \exists\, \sigma \in SC(\rho) \text{ tel que} \\ \big(\exists\, p \in \sigma \text{ t.q } \mathsf{Time}(\rho^{\leq p}) \sim c\big) \wedge \hat{\mu}(\sigma) > k \wedge \sigma \models \varphi \end{array}$$

Ainsi, la formule $\mathsf{AF}^k_{\sim c}\varphi$ signifie qu'il est inévitable que φ dure suffisamment longtemps ($> k$) autour d'une position $\sim c$.

– $\mathsf{EG}^k_{\sim c}\,\varphi \stackrel{def}{=} \neg\mathsf{AF}^k_{\sim c}\neg\varphi$

$$s \models \mathsf{EG}^k_{\sim c}\,\varphi \;\Leftrightarrow\; \begin{array}{l} \exists\, \rho \in \text{Exec}(s) \mid \forall\, \sigma \in SC(\rho) : \\ \Big(\hat{\mu}(\sigma) > k \;\Rightarrow\; \big(\forall\, p \in \sigma,\ \mathsf{Time}(\rho^{\leq p}) \not\sim c\big) \vee \big(\exists p \in \sigma \text{ t.q } s_p \models \varphi\big)\Big) \end{array}$$

Ainsi, la formule $\mathsf{EG}^k_{\sim c}\varphi$ signifie qu'il existe un chemin le long duquel φ est vraie partout –sauf pour des durées $\leq k$– autour de toutes les positions de ce chemin vérifiant $\sim c$.

– $\mathsf{AG}^k_{\sim c}\,\varphi \stackrel{def}{=} \neg\mathsf{EF}^k_{\sim c}\neg\varphi$

$$s \models \mathsf{AG}^k_{\sim c}\,\varphi \;\Leftrightarrow\; \begin{array}{l} \forall\, \rho \in \text{Exec}(s) \mid \forall\, \sigma \in SC(\rho) : \\ \Big(\hat{\mu}(\sigma) > k \;\Rightarrow\; \big(\forall\, p \in \sigma,\ \mathsf{Time}(\rho^{\leq p}) \not\sim c\big) \vee \big(\exists p \in \sigma \text{ t.q } s_p \models \varphi\big)\Big) \end{array}$$

Enfin, la formule $\mathsf{AG}^k_{\sim c}\varphi$ indique que φ est vraie partout –sauf pour des durées $\leq k$– autour de toutes les positions vérifiant $\sim c$.

Exemple 5
Le modèle de la figure 4.2 vérifie alors la formule ci-dessous de TCTL$^{\Delta}$ *:*

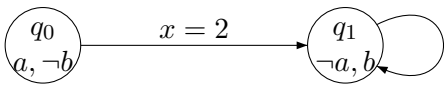
$$\mathsf{AG}^0(\texttt{si } s = 1 \texttt{ alors } (L = 1 \texttt{ et } R = 1))$$

Notation. La logique TCTL^k avec $k \in \mathbb{N}$, est la restriction de la logique TCTL$^\Delta$ obtenue en fixant le paramètre k pour toutes les modalités. La logique TCTL^a [BBBL05] (a pour "almost") est le fragment de TCTL^0 où seulement les opérateurs U^0 sont autorisés (on exclut les modalités U classiques), c'est-à-dire que TCTL^a est définie par la grammaire suivante :

$$\varphi, \psi ::= P_1 \mid P_2 \mid \ldots \mid \neg\varphi \;\mid\; \varphi \wedge \psi \;\mid\; \mathsf{E}\varphi\mathsf{U}^0_{\sim c}\psi \;\mid\; \mathsf{A}\varphi\mathsf{U}^0_{\sim c}\psi$$

Ainsi, dans la sémantique de TCTL^a, nous demandons que sur les chemins considérés, φ soit presque toujours satisfaite, c'est-à-dire que l'ensemble des positions où φ est fausse est un ensemble de mesure nulle. Par exemple, la formule AG$^0\varphi$ signifie que φ est presque toujours vraie.

Notons aussi que les positions où φ devient fausse ne correspondent pas seulement aux transitions discrètes. En effet, considérons l'exemple de l'automate temporisé ci-dessous où a et b sont deux propositions atomiques.



Considérons la formule $\varphi = \mathsf{E}a\mathsf{U}^0_{=1}b$ et une exécution ρ partant de $(q_0, 0)$. La seule position le long de ρ qui satisfait φ correspond à la configuration $(q_0, 1)$. La formule $\psi \stackrel{def}{=} \neg\varphi$ est donc presque toujours vraie le long de ρ. Ainsi $(q_0, 0) \models \mathsf{AG}^0(\psi)$ alors que $(q_0, 0) \not\models \mathsf{AG}(\psi)$. On remarque donc que le changement de valeur de vérité de ψ ne correspond pas à un instant de franchissement d'une transition discrète.

4.3.2 Discussion

Cette sémantique est délicate à manier car certaines formules de la logique TCTL$^\Delta$ sont contre-intuitives : c'est le cas par exemple de AG$^k\varphi$. Cette formule exprime qu'il n'est pas possible d'avoir $\neg\varphi$ pendant suffisamment longtemps –plus de k u.t– le long de toutes les exécutions. Mais une exécution où $\neg\varphi$ est presque toujours vraie sauf ponctuellement toutes les k u.t, satisfait aussi G$^k\varphi$. Malgré ces inconvénients, la sémantique TCTL$^\Delta$ est motivée par l'utilité de la négation des modalités U^k. Par exemple pour définir l'opérateur de précédence évoqué dans la page 54 : AφP$^k\psi \stackrel{def}{=} \neg\mathsf{E}(\neg\varphi)\mathsf{U}^k\psi$.

$$\begin{aligned} s \models \mathsf{A}\varphi\mathsf{P}^k\psi \;\Leftrightarrow\; & \neg\Big(\exists\,\rho \in \text{Exec}(s),\ \exists\,\sigma \in SC(\rho) \text{ tel que } \hat{\mu}(\sigma) > k \ \wedge\ \sigma \models \psi \\ & \wedge\, \exists\, p \in \sigma \text{ tel que } \forall\,\sigma' \in SC(\rho),\ \sigma' <_\rho p,\ \sigma' \models \varphi \ \Rightarrow\ \hat{\mu}(\sigma') \leq k\Big) \\ \Leftrightarrow\; & \forall\,\rho \in \text{Exec}(s) \mid \forall\,\sigma \in SC(\rho) \text{ tel que } \hat{\mu}(\sigma) > k, \\ & \sigma \not\models \psi \ \vee\ \Big(\forall\, p \in \sigma,\ \exists\,\sigma' \in SC(\rho),\ \sigma' <_\rho p \text{ tel que } \hat{\mu}(\sigma') > k \text{ et } \sigma' \models \varphi\Big) \end{aligned}$$

La formule AφP$^k\psi$ signifie donc que le long de toute exécution, si ψ dure assez longtemps alors φ a aussi duré assez longtemps dans le passé.

4.4 Équivalences de formules

Nous prouvons dans cette section des équivalences pour les formules de TCTL^{Δ}. Ces équivalences sont vérifiées pour tout état s d'un système de transitions temporisé $\mathcal{T}_{\mathcal{A}}$ associé à un automate temporisé.

$$\mathsf{A}\varphi\mathsf{U}^0\psi \overset{\mathcal{T}_{\mathcal{A}}}{\equiv} \mathsf{AF}^0\psi \wedge \neg(\mathsf{E}\neg\psi\mathsf{U}^0(\neg\psi \wedge \neg\varphi)) \tag{4.1}$$

$$\mathsf{A}\varphi\mathsf{U}^0_{<c}\psi \overset{\mathcal{T}_{\mathcal{A}}}{\equiv} \mathsf{AF}^0_{<c}\psi \wedge \neg(\mathsf{E}\neg\psi\mathsf{U}^0(\neg\psi \wedge \neg\varphi)) \tag{4.2}$$

$$\mathsf{A}\varphi\mathsf{U}^0_{\leq c}\psi \overset{\mathcal{T}_{\mathcal{A}}}{\equiv} \mathsf{AF}^0_{\leq c}\psi \wedge \neg(\mathsf{E}\neg\psi\mathsf{U}^0(\neg\psi \wedge \neg\varphi)) \tag{4.3}$$

$$\mathsf{A}\varphi\mathsf{U}^0_{>c}\psi \overset{\mathcal{T}_{\mathcal{A}}}{\equiv} \mathsf{AG}_{\leq c}(\mathsf{A}(\varphi\mathsf{U}^0_{>0}\psi)) \tag{4.4}$$

$$\mathsf{A}\varphi\mathsf{U}^0_{\geq c}\psi \overset{\mathcal{T}_{\mathcal{A}}}{\equiv} \mathsf{AG}_{<c}(\mathsf{A}(\varphi\mathsf{U}^0_{>0}\psi)) \quad \text{si } c > 0 \tag{4.5}$$

$$\mathsf{A}\varphi\mathsf{U}^k_{=c}\psi \overset{\mathcal{T}_{\mathcal{A}}}{\equiv} \mathsf{AF}^k_{=c}\psi \wedge \mathsf{AG}^k_{<c-k}\varphi \quad \text{si } c > k \tag{4.6}$$

$$\mathsf{A}\varphi\mathsf{U}^k_{=c}\psi \overset{\mathcal{T}_{\mathcal{A}}}{\equiv} \mathsf{AF}^k_{=c}\psi \quad \text{si } c \leq k \tag{4.7}$$

Démonstration:

(4.1) $\Rightarrow$ Supposons que $s \models \mathsf{A}\varphi\mathsf{U}^0\psi$ et soit $\rho \in \mathrm{Exec}(s)$. Nous avons $\rho \models \varphi\mathsf{U}^0\psi$. Il existe donc un sous-chemin σ de ρ tel que $\hat{\mu}(\sigma) > 0$, $\sigma \models \psi$, et il existe une position $p \in \sigma$ telle que $\hat{\mu}(\{p' \mid p' <_\rho p \wedge s_{p'} \not\models \varphi\}) = 0$. Nous avons donc $\rho \models \mathsf{F}^0\psi$ et $s \models \mathsf{AF}^0\psi$.
Si on suppose maintenant que $\rho \models \neg\psi\mathsf{U}^0(\neg\psi \wedge \neg\varphi)$, il existe alors un sous-chemin σ' le long de ρ tel que $\hat{\mu}(\sigma') > 0$ et $\sigma' \models \neg\psi \wedge \neg\varphi$. Donc σ' précède strictement σ, puisque $\sigma \models \psi$, $\hat{\mu}(\sigma) > 0$ et $\rho \models \neg\psi\mathsf{U}^0(\neg\psi \wedge \neg\varphi)$. Mais ceci contredit le fait que $\hat{\mu}(\{p' \mid p' <_\rho p \wedge s_{p'} \not\models \varphi\}) = 0$. Par conséquent, nous avons prouvé que $\rho \models \neg(\neg\psi\mathsf{U}^0(\neg\psi \wedge \neg\varphi))$ et $s \models \neg(\mathsf{E}\neg\psi\mathsf{U}^0(\neg\psi \wedge \neg\varphi))$.

$\Leftarrow$ Réciproquement, supposons que $s \models \mathsf{AF}^0\psi \wedge \neg(\mathsf{E}\neg\psi\mathsf{U}^0(\neg\psi \wedge \neg\varphi))$. Soit $\rho \in \mathrm{Exec}(s)$, on a $\rho \models \mathsf{F}^0\psi$. On en déduit l'existence d'un sous-chemin σ de ρ tel que $\hat{\mu}(\sigma) > 0$ et $\sigma \models \psi$. Considérons alors σ le premier sous-chemin qui possède ces propriétés.
De plus, on a $\rho \not\models (\neg\psi)\mathsf{U}^0(\neg\psi \wedge \neg\varphi)$. Ceci signifie que pour tout sous-chemin σ' avec $\hat{\mu}(\sigma') > 0$, nous avons soit (1) $\sigma' \not\models \neg\psi \wedge \neg\varphi$ (i.e. σ' contient une position qui satisfait $\varphi \vee \psi$), soit (2) $\sigma' \models \neg\psi \wedge \neg\varphi$ et pour toute position p dans σ', nous avons $\hat{\mu}(\{p' <_\rho p \mid s_{p'} \models \psi\}) > 0$.
Considérons maintenant un sous-chemin σ' avant σ de mesure non nulle. Alors σ' ne satisfait pas la condition (2) à cause du choix de σ. Par conséquent, σ' contient au moins une position vérifiant $\varphi \vee \psi$, et toujours par le choix de σ, σ' contient au moins une position vérifiant φ. On conclut ainsi que l'ensemble des positions avant σ où φ est fausse est de mesure nulle. Donc $s \models \mathsf{A}\varphi\mathsf{U}^0\psi$.

La même preuve peut être adaptée pour démontrer l'équivalence 4.2 (respectivement 4.3) : la seule différence est qu'il faut prendre en compte la contrainte $< c$ (resp $\leq c$).

(4.4) $\Rightarrow$ Supposons que $s \models \mathsf{A}\varphi\mathsf{U}^0_{>c}\psi$. Soit $\rho \in \mathrm{Exec}(s)$, nous avons alors $\rho \models \varphi\mathsf{U}^0_{>c}\psi$. Considérons une position p dans ρ telle que $\mathsf{Time}(\rho^{\leq p}) \leq c$, et $\rho' \in \mathrm{Exec}(s_p)$. Alors $\rho^{\leq p} \cdot \rho'$ est une exécution de s. Ainsi elle satisfait $\varphi\mathsf{U}^0_{>c}\psi$. Cependant, $\mathsf{Time}(\rho^{\leq p}) \leq c$ implique que $\rho' \models \varphi\mathsf{U}^0_{>0}\psi$, donc $\rho \models \mathsf{G}_{\leq c}(\mathsf{A}\varphi\mathsf{U}^0_{>0}\psi)$.

$\Leftarrow$ Réciproquement, supposons que $s \models \mathsf{AG}_{\leq c}(\mathsf{A}(\varphi \mathsf{U}^0_{>0} \psi))$. Considérons $\rho \in Exec(s)$ et p une position de ρ telle que $\mathsf{Time}(\rho^{\leq p}) = c$, alors $\rho^{\geq p} \models \varphi \mathsf{U}^0_{>0} \psi$. Donc il existe un sous-chemin σ de ρ tel que $\hat{\mu}(\sigma) > 0$, $\sigma \models \psi$, et une position p' de σ avec $\mathsf{Time}(\rho^{\leq p'}) > c$ telle que $\hat{\mu}(\{p <_\rho p'' <_\rho p' \mid s_{p''} \models \neg\varphi\}) = 0$. Considérons maintenant un sous-chemin σ' de mesure non nulle qui précède p et supposons que $\sigma' \models \neg\varphi$. Dans ce cas, on peut trouver au moins une position dans σ' telle que le suffixe de ρ issu de cette position ne vérifie pas $\varphi \mathsf{U}^0_{>0} \psi$. Ceci contredit le fait que $\rho \models \mathsf{G}_{\leq c}(\mathsf{A}\varphi \mathsf{U}^0_{>0} \psi)$. Donc σ' contient au moins une position qui satisfait φ. Par conséquent l'ensemble des positions avant p' où φ est fausse est de mesure nulle. Ainsi, $s \models \mathsf{A}\varphi \mathsf{U}^0_{>c} \psi$.

(4.5) $\Rightarrow$ Supposons que $s \models \mathsf{A}\varphi \mathsf{U}^0_{\geq c} \psi$. Soit $\rho \in \mathrm{Exec}(s)$, donc $\rho \models \varphi \mathsf{U}^0_{\geq c} \psi$. Considérons une position p dans ρ telle que $\mathsf{Time}(\rho^{\leq p}) < c$, et $\rho' \in \mathrm{Exec}(s_p)$. Alors $\rho^{\leq p} \cdot \rho'$ est une exécution de s, qui satisfait $\varphi \mathsf{U}^0_{\geq c} \psi$. Cependant, $\mathsf{Time}(\rho^{\leq p}) < c$ implique $\rho' \models \varphi \mathsf{U}^0_{>0} \psi$, donc $\rho \models \mathsf{G}_{<c}(\mathsf{A}\varphi \mathsf{U}^0_{>0} \psi)$.

$\Leftarrow$ Supposons maintenant que $s \models \mathsf{AG}_{<c}(\mathsf{A}(\varphi \mathsf{U}^0_{>0} \psi))$. Considérons $\rho \in Exec(s)$. Puisque $\rho \models \mathsf{G}_{<c}(\mathsf{A}\varphi \mathsf{U}^0_{>0} \psi)$, il existe une position p de ρ telle que le long de $\rho^{\geq p}$, il existe un sous-chemin σ de ρ tel que $\hat{\mu}(\sigma) > 0$, $\sigma \models \psi$, et une position p' de σ avec $\mathsf{Time}(\rho^{\leq p'}) \geq c$ telle que $\hat{\mu}(\{p <_\rho p'' <_\rho p' \mid s_{p''} \models \neg\varphi\}) = 0$. Supposons maintenant qu'il existe un sous-chemin σ' qui précède p avec $\hat{\mu}(\sigma') > 0$ tel que $\sigma' \models \neg\varphi$. On peut alors trouver au moins une position dans σ' telle que $\mathsf{A}(\varphi \mathsf{U}^0_{>0} \psi)$ soit fausse. Par conséquent $s \models \mathsf{A}\varphi \mathsf{U}^0_{\geq c} \psi$.

(4.6) $\Leftarrow$ Supposons que $s \models \mathsf{AF}^k_{=c} \psi \wedge \mathsf{AG}^k_{<c-k} \varphi$. Soit $\rho \in \mathrm{Exec}(s)$, $\rho \models \mathsf{F}^k_{=c} \psi$. Donc il existe le long de ρ, un sous-chemin σ de mesure strictement supérieure à k où ψ est vraie, qui contient une position p telle que $\mathsf{Time}(\rho^{\leq p}) = c$. Considérons maintenant un sous-chemin σ' avant la position p tel que $\sigma' \models \neg\varphi$. Si $\hat{\mu}(\sigma') > k$, alors il contient une position p' telle que $\mathsf{Time}(\rho^{\leq p'}) < c - k$. Donc il contient aussi une position vérifiant φ. On conclut alors que $s \models \mathsf{A}\varphi \mathsf{U}^k_{=c} \psi$.

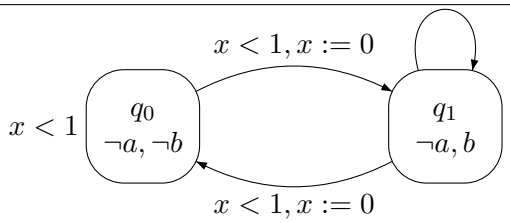
$\Rightarrow$ Supposons maintenant que $s \models \mathsf{A}\varphi \mathsf{U}^k_{=c} \psi$. Soit $\rho \in Exec(s)$, donc $\rho \models \varphi \mathsf{U}^k_{=c} \psi$. Il existe alors un sous-chemin σ de ρ et une position p de σ telle que $\hat{\mu}(\sigma) > k$, $\sigma \models \psi$, $\mathsf{Time}(\rho^{\leq p}) = c$, et pour tout sous-chemin σ' avant p, qui ne satisfait pas φ, nous avons $\hat{\mu}(\sigma') \leq k$. Ceci implique directement que $\rho \models \mathsf{F}^k_{=c} \psi$. D'un autre côté, Soit σ' un sous-chemin de longueur strictement supérieure à k qui contient une position p' de ρ telle que $\mathsf{Time}(\rho^{\leq p'}) < c - k$. Si $\sigma' \models \neg\varphi$, ceci donne l'existence d'un sous-chemin avant p de longueur strictement supérieure à k où $\neg\varphi$ est vraie. Ceci contredit le fait que $\rho \models \varphi \mathsf{U}^k_{=c} \psi$. Donc σ' contient une position vérifiant φ. Par conséquent, $\rho \models \mathsf{G}^k_{<c-k} \varphi$.

L'équivalence 4.7 est immédiate.

□

Contre-exemples dans le cas $k > 0$

L'équivalence 4.1 n'est pas satisfaite dans le cas général : considérons l'automate ci-dessous où a et b sont des propositions atomiques :



Considérons l'exécution ρ issue de $(q_0, 0)$:

$$(q_0, 0) \xrightarrow{0.7} (q_0, 0.7) \rightarrow_a (q_1, 0) \xrightarrow{0.7} (q_1, 0) \rightarrow_a (q_0, 0) \xrightarrow{0.6} (q_0, 0.6) \rightarrow_a (q_1, 0) \overset{\rho_1}{\mapsto}$$

où le suffixe ρ_1 boucle dans l'état q_1.

Alors $\rho \models \mathsf{F}^1 b$ et $\rho \models \neg\big(\neg b \mathsf{U}^1(\neg a \wedge \neg b)\big)$ tandis que $\rho \not\models a\mathsf{U}^1 b$ (figure 4.3), donc :

$$\mathsf{A}\varphi\mathsf{U}^1\psi \not\equiv \mathsf{AF}^1\psi \wedge \neg(\mathsf{E}\neg\psi\mathsf{U}^1(\neg\psi \wedge \neg\varphi))$$

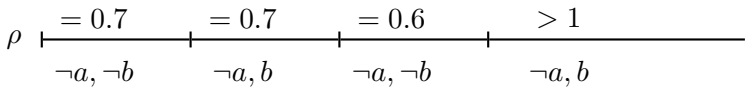
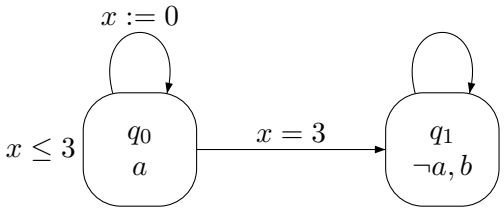


FIG. 4.3 – $(q_0, 0) \xrightarrow{0.7} (q_0, 0.7) \rightarrow_a (q_1, 0) \xrightarrow{0.7} (q_1, 0) \rightarrow_a (q_0, 0) \xrightarrow{0.6} (q_0, 0.6) \rightarrow_a (q_1, 0) \overset{\rho_1}{\mapsto}$

L'équivalence 4.4 n'est pas satisfaite pour $k > 0$: considérons l'automate ci-dessous où a et b sont des propositions atomiques :



Considérons l'exécution ρ partant de $(q_0, 0)$:

$$(q_0, 0) \xrightarrow{3} (q_0, 3) \rightarrow_a (q_1, 3) \overset{\rho_1}{\mapsto}$$

où le suffixe ρ_1 boucle dans l'état q_1.

Dans ce cas, $\rho \models \mathsf{G}_{\leq 5}(\mathsf{A}a\mathsf{U}^1_{>0}b)$ alors que $\rho \not\models a\mathsf{U}^1_{>5}b$ (figure 4.4), donc :

$$\mathsf{A}\varphi\mathsf{U}^1_{>5}\psi \not\equiv \mathsf{AG}_{\leq 5}(\mathsf{A}(\varphi\mathsf{U}^1_{>0}\psi))$$

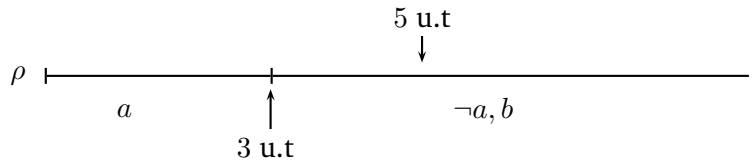


FIG. 4.4 – $(q_0, 0) \xrightarrow{3} (q_0, 3) \rightarrow_a (q_1, 3) \overset{\rho_1}{\mapsto}$

4.5 Expressivité des modalités U^k

Nous allons voir dans cette section que les modalités utilisant U^k apportent de l'expressivité à la logique TCTL. Nous montrons que U^0 ne peut pas s'exprimer avec les opérateurs de la logique TCTL et que, inversement, il ne peut pas exprimer les opérateurs de TCTL. Nous prouvons aussi que les opérateurs U^k et U^{k+1} n'ont pas le même pouvoir expressif. Introduisons d'abord quelques notations.

Notation 1
• *On notera $|\varphi|$ la taille d'une formule φ définie par le nombre total de propositions atomiques, d'opérateurs logiques et de modalités nécessaires à écrire cette formule.*
• *Pour deux configurations s et s', on écrira :*
– $s \equiv^k_{\mathcal{L}} s'$ *si et seulement si pour toute formule φ de $\mathcal{L}$ avec $|\varphi| \leq k$, on a $s \models \varphi \Leftrightarrow s' \models \varphi$.*
– $s \equiv_{\mathcal{L}} s'$ *si et seulement si $s \equiv^k_{\mathcal{L}} s'$ pour tout $k \geq 1$.*

4.5.1 $\mathsf{TCTL}^a \prec \mathsf{TCTL}^0$

Nous montrons d'abord que la modalité U ne peut pas s'exprimer avec la modalité U^0. On rappelle que TCTL^a est le fragment de TCTL^0 où seulement les opérateurs U^0 sont autorisés.

Théorème 4
La logique TCTL^0 *est strictement plus expressive que la logique* TCTL^a.

Démonstration: Considérons la formule de TCTL suivante $\Psi = \mathsf{E}a\mathsf{U}b$. Nous allons montrer qu'il n'existe aucune formule de TCTL^a qui soit équivalente à Ψ. Considérons pour cela l'automate A de la Figure 4.5.
Pour cet automate, $(q_0, t) \models \mathsf{E}a\mathsf{U}b$ alors que $(q'_0, t) \not\models \mathsf{E}a\mathsf{U}b$. De plus, il est facile de voir que (q_0, t) et (q'_0, t) satisfont les mêmes formules de TCTL^a, et donc $(q_0, 0) \equiv_{\mathsf{TCTL}^a} (q'_0, 0)$. Ceci est une conséquence du fait que pour toute exécution ρ de $(q_0, 0)$, il existe une exécution ρ' de $(q'_0, 0)$ qui lui est "quasiment identique". En effet, la seule différence entre ρ et ρ' est l'état r'_0, or ce dernier doit être quitté immédiatement à cause de la garde $x = 0$. Donc $\hat{\mu}(\mathbb{P}(\rho') \setminus \mathbb{P}(\rho)) = 0$. Cette différence n'a pas d'effet sur la valeur de vérité des formules de la logique TCTL^a.

□

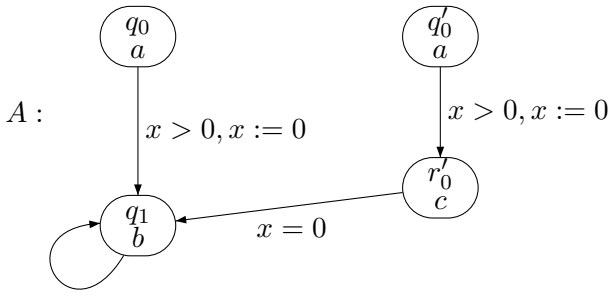


FIG. 4.5 – $(q_0, 0) \models \mathsf{E}(a\mathsf{U}b)$, $(q'_0, 0) \not\models \mathsf{E}(a\mathsf{U}b)$, mais $(q_0, 0) \equiv_{\mathsf{TCTL}^a} (q'_0, 0)$.

4.5.2 TCTL $\prec$ TCTL^0

Nous montrons dans ce paragraphe que la modalité U^0 ne peut pas s'exprimer avec la modalité U. La preuve repose sur les techniques classiques utilisées pour des logiques temporelles [Eme91, EH86]. Cependant, dans le cas temporisé, les preuves sont un peu plus techniques.

Nous considérons les deux familles d'automates temporisés M_i et N_i, pour $i \geq 1$, décrites dans la figure 4.6. Notons que n'importe quelle configuration de la forme $(q_j, t), (q'_j, t), (r_j, t), (r'_j, t)$ où $j \geq 1$, apparaît au plus une fois le long de toutes les exécutions de M_i ou N_i.

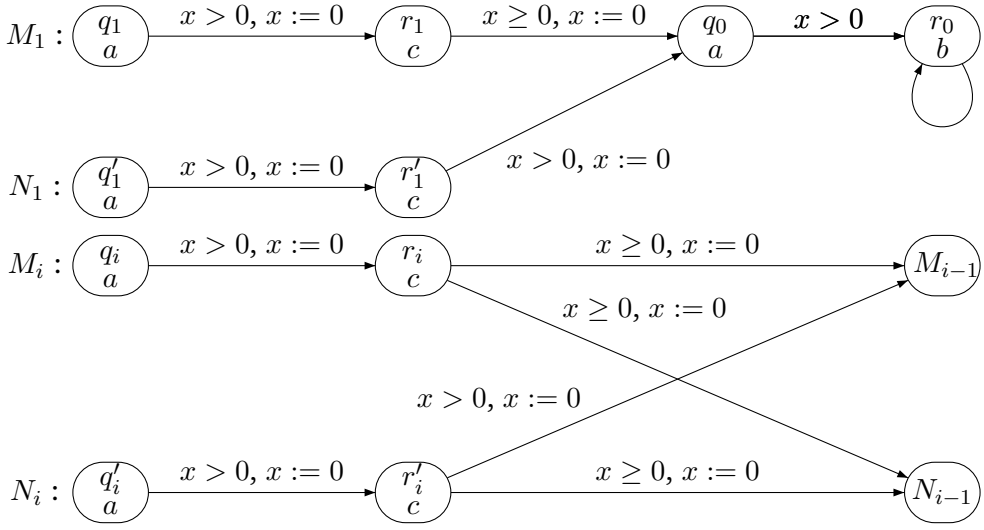


FIG. 4.6 – Les automates M_i et N_i, $i = 1, 2, \ldots$

Soit Ψ la formule de TCTL^0, $\mathsf{E}a\mathsf{U}^0b$, alors :

$$(q_i, 0) \models \Psi \text{ et } (q'_i, 0) \not\models \Psi \text{ pour tout } i \geq 1 \tag{4.8}$$

En effet, dans l'automate M_i où $i \geq 1$, il existe une exécution à partir de $(q_i, 0)$ telle qu'avant d'arriver à l'état r_0 (l'unique état étiqueté par b), les délais dans les états qui ne vérifient pas a sont nuls, alors que dans N_i, il faut rester un délai strictement supérieur à zéro dans un état

étiqueté par $\neg a$ avant d'arriver à r_0.

Nous montrons maintenant qu'il n'existe aucune formule de TCTL équivalente à Ψ. Ceci est une conséquence de la proposition suivante :

Proposition 1
Soient M_i et N_i les deux familles d'automates de la figure 4.6. Alors $\forall k \geq 1$, $\forall i \geq k$ et $\forall t \in \mathbb{R}_+$:

$$(q_i, t) \equiv^k_{\mathsf{TCTL}} (q'_i, t) \quad \textit{et} \quad (r_i, t) \equiv^k_{\mathsf{TCTL}} (r'_i, t)$$

Cette proposition exprime que pour tout $i \geq 1$, les automates M_i et N_i satisfont les mêmes formules de TCTL de taille inférieure ou égale à i. On en déduit directement le résultat d'expressivité suivant :

Théorème 5
La logique TCTL^0 *est strictement plus expressive que la logique* TCTL*.*

Démonstration: Supposons qu'il existe une formule Φ de TCTL équivalente à la formule Ψ. D'après 4.8, $(q_i, 0) \models \Phi$ et $(q'_i, 0) \not\models \Phi$ pour tout $i \geq 1$. Cependant, ceci contredit la proposition 1 d'après laquelle $(q_i, 0) \equiv^{|\Phi|}_{\mathsf{TCTL}} (q'_i, 0)$ pour tout $i \geq |\Phi|$.

□

La preuve de la proposition 1 utilise plusieurs propriétés des automates M_i et N_i, que nous allons maintenant présenter.

Remarque 2
– *Soit ρ une exécution de M_i partant de (q_i, t) pour $i \geq 2$ et $t \in \mathbb{R}_+$. Cette exécution est caractérisée par le paramètre δ_0 qui est le temps écoulé dans l'état q_i, par le paramètre δ_1 qui est le temps écoulé dans l'état r_i et par le suffixe ρ_1 qui est soit dans N_{i-1} soit dans M_{i-1}. Donc ρ s'écrit sous la forme suivante :*

$$(q_i, t) \xrightarrow{\delta_0} (q_i, \delta_0 + t) \rightarrow_a (r_i, 0) \xrightarrow{\delta_1} (r_i, \delta_1) \overset{\rho_1}{\mapsto}$$

Si ρ_1 est dans N_{i-1} alors ρ a un correspondant dans N_i, noté $f_{N_i}(\rho)$ et défini par :

$$(q'_i, t) \xrightarrow{\delta_0} (q'_i, \delta_0 + t) \rightarrow_a (r'_i, 0) \xrightarrow{\delta_1} (r'_i, \delta_1) \overset{\rho_1}{\mapsto}$$

Si ρ_1 est dans M_{i-1} et $\delta_1 > 0$, on peut définir de la même manière le correspondant de ρ dans N_i. Notons que dans le cas où $\delta_1 = 0$, le correspondant $f_{N_i}(\rho)$ n'existe pas.

– *La même correspondance peut être établie pour chaque exécution de M_i à partir d'une configuration (r_i, t). Ces exécutions sont caractérisées par le paramètre δ_1 et le suffixe ρ_1.*

– *Pour chaque exécution ρ dans N_i, on peut définir un correspondant dans M_i noté $f_{M_i}(\rho)$ comme ci-dessus.*

Étant données les gardes et les remises à zéro des familles d'automates M_i et N_i, nous avons la propriété suivante :

Propriété 1
Soit $j \geq 0$, on a :

$$(r_j, 0) \equiv_{\mathsf{TCTL}} (r_j, t) \qquad \forall t > 0 \tag{4.9}$$
$$(\ell, t) \equiv_{\mathsf{TCTL}} (\ell, t') \qquad \forall t, t' > 0,\ \forall \ell \in \{q_j, r_j, q'_j, r'_j\} \tag{4.10}$$

Démonstration: Nous prouvons l'équivalence 4.9 par induction sur n la taille des formules de TCTL. Si $n = 1$, nous avons évidemment $(r_j, 0) \equiv^1_{\mathsf{TCTL}} (r_j, t)$. Supposons maintenant que $n > 1$ et que $(r_j, 0) \equiv^{n-1}_{\mathsf{TCTL}} (r_j, t)$. Pour toute exécution ρ à partir de $(r_j, 0)$,

$$\rho : (r_j, 0) \xrightarrow{\delta_0} (r_j, 0 + \delta_0) \rightarrow_a (q_{j-1}, 0) \overset{\rho_1}{\mapsto}$$

on peut construire (vues les contraintes de M_i) l'exécution ρ' issue de (r_j, t) suivante :

$$\rho' : (r_j, t) \xrightarrow{\delta_0} (r_j, t + \delta_0) \rightarrow_a (q_{j-1}, 0) \overset{\rho_1}{\mapsto}$$

En appliquant l'hypothèse d'induction sur le préfixe de ρ et ρ', on obtient que ρ et ρ' satisfont les mêmes formules de TCTL de taille n. Donc l'équivalence 4.9 est satisfaite.

L'équivalence 4.10 se prouve d'une manière similaire.

□

Nous prouvons maintenant la proposition 1.

Démonstration: [de la proposition 1] Nous prouvons le résultat par induction sur la taille n des formules.

Pour les formules de taille $n = 1$, les équivalences sont vérifiées puisque les états q_i et q'_i (respectivement r_i et r'_i) sont étiquetés par les mêmes propositions atomiques.

Considérons maintenant une formule Ψ de taille $n > 1$ et supposons que les équivalences de la proposition 1 sont vraies pour toutes les formules de taille strictement inférieure à n. Nous procédons par induction structurelle sur Ψ :

Si Ψ est une négation d'une sous-formule ou une conjonction de deux sous-formules, alors l'hypothèse d'induction, appliquée aux sous-formules, donne le résultat.

Formule $\mathsf{E\_U}_{\sim c}\mathsf{\_}$**.** Si $\Psi = \mathsf{E}\varphi_1\mathsf{U}_{\sim c}\varphi_2$, d'après la remarque 2, si ρ est une exécution dans N_i, alors l'exécution correspondante $f_{M_i}(\rho)$ existe. En appliquant l'hypothèse d'induction, on aura :

$$\rho \models \varphi_1\mathsf{U}_{\sim c}\varphi_2 \iff f_{M_i}(\rho) \models \varphi_1\mathsf{U}_{\sim c}\varphi_2$$

On en déduit alors que :

$$\begin{cases} (q'_i, t) \models \mathsf{E}(\varphi_1\mathsf{U}_{\sim c}\varphi_2) & \Longrightarrow \quad (q_i, t) \models \mathsf{E}(\varphi_1\mathsf{U}_{\sim c}\varphi_2) \\ (r'_i, t) \models \mathsf{E}(\varphi_1\mathsf{U}_{\sim c}\varphi_2) & \Longrightarrow \quad (r_i, t) \models \mathsf{E}(\varphi_1\mathsf{U}_{\sim c}\varphi_2) \end{cases}$$

Montrons maintenant les implications inverses :

1. Si $(q_i, t) \models \mathsf{E}(\varphi_1 \mathsf{U}_{\sim c} \varphi_2)$, il existe alors une exécution ρ dans M_i de (q_i, t) vérifiant $\varphi_1 \mathsf{U}_{\sim c} \varphi_2$. Si $f_{N_i}(\rho)$ existe (le temps écoulé dans l'état r_i est non nul), alors l'hypothèse d'induction appliquée à $f_{N_i}(\rho)$ donne le résultat.
   Sinon, $f_{N_i}(\rho)$ n'existe pas (le temps écoulé dans l'état r_i est nul et le suffixe $\overset{\rho_1}{\mapsto}$ est dans M_{i-1}). Dans ce cas nous allons construire une exécution ρ' de M_i qui vérifie aussi $\varphi_1 \mathsf{U}_{\sim c} \varphi_2$ telle que $f_{N_i}(\rho')$ existe et nous montrons ensuite que $f_{N_i}(\rho') \models \varphi_1 \mathsf{U}_{\sim c} \varphi_2$.

   On note (ℓ, v) la configuration associée à la position p le long de ρ qui satisfait φ_2 telle que $\mathsf{Time}(\rho^{\leq p}) \sim c$, et toutes les positions précèdent p satisfont φ_1. Nous distinguons maintenant deux sous-cas :

   – Si (ℓ, v) précède $(q_{i-1}, 0)$ (ou $(\ell, v) = (q_{i-1}, 0)$), on construit alors directement dans N_i une exécution qui a un préfixe similaire que ρ jusqu'à la configuration (ℓ, v). Par hypothèse d'induction, cette exécution dans N_i vérifie $\varphi_1 \mathsf{U}_{\sim c} \varphi_2$. Par conséquent, $(q'_i, t) \models \mathsf{E}(\varphi_1 \mathsf{U}_{\sim c} \varphi_2)$.

   – Sinon, nous modifions légèrement l'exécution ρ (pour obtenir une exécution ρ' qui possède un correspondant dans N_i) comme suit :
   ρ' est obtenue à partir de ρ en augmentant le temps passé dans l'état r_i. Ce délai ajouté dans r_i doit être suffisamment petit pour que les deux exécutions soient identiques après l'état r_{i-1}. C'est-à-dire que le temps total accumulé dans les états r_i et q_{i-1} le long de la nouvelle exécution ρ' doit être égal au temps total écoulé dans l'état q_{i-1} le long de ρ. Ceci est illustré par la figure 4.7.

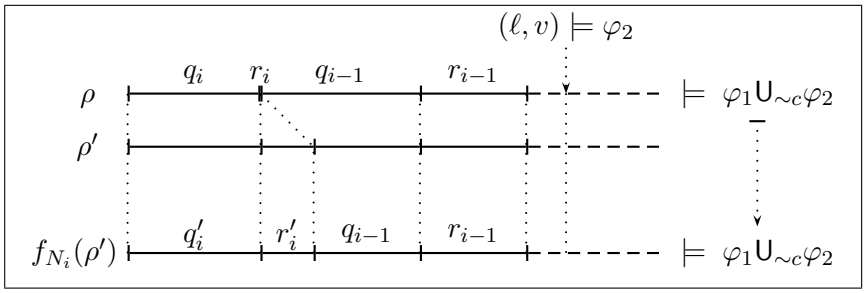


FIG. 4.7 – Construction pour le cas $(q_{i-1}, 0) <_\rho (\ell, v)$

   La nouvelle exécution ρ' vérifie $\varphi_1 \mathsf{U}_{\sim c} \varphi_2$: la configuration (ℓ, v) existe le long de ρ', satisfait aussi φ_2 et la durée du préfixe jusqu'à cette configuration vérifie aussi $\sim c$ (cette durée n'a pas changé). De plus, les configurations ajoutées le long de ρ' avant (ℓ, v) (dans l'état r_i) vérifient toutes φ_1 (par l' équivalence 4.9). En appliquant l'hypothèse d'induction, $f_{N_i}(\rho')$ vérifie aussi $\varphi_1 \mathsf{U}_{\sim c} \varphi_2$. Par conséquent, $(q'_i, t) \models \mathsf{E}(\varphi_1 \mathsf{U}_{\sim c} \varphi_2)$.

2. On adapte facilement la preuve et la construction précédentes pour prouver que si $(r_i, t) \models \mathsf{E}\varphi_1\mathsf{U}_{\sim c}\varphi_2$ alors $(r'_i, t) \models \mathsf{E}\varphi_1\mathsf{U}_{\sim c}\varphi_2$.

Formule $\mathsf{A\_U}_{\sim c}\mathsf{\_}$. Si $\Psi = \mathsf{A}\varphi_1\mathsf{U}_{\sim c}\varphi_2$, d'après la remarque 2, si ρ est une exécution dans N_i, alors l'exécution correspondante $f_{M_i}(\rho)$ existe. En appliquant l'hypothèse d'induction, on déduit que :

$$\begin{cases} (q_i, t) \models \mathsf{A}\varphi_1\mathsf{U}_{\sim c}\varphi_2 & \Longrightarrow \quad (q'_i, t) \models \mathsf{A}\varphi_1\mathsf{U}_{\sim c}\varphi_2 \\ (r_i, t) \models \mathsf{A}\varphi_1\mathsf{U}_{\sim c}\varphi_2 & \Longrightarrow \quad (r'_i, t) \models \mathsf{A}\varphi_1\mathsf{U}_{\sim c}\varphi_2 \end{cases}$$

Pour les implications réciproques, nous distinguons plusieurs sous-cas selon la nature de la relation $\sim$:

1. Supposons que $\Psi = \mathsf{A}\varphi_1\mathsf{U}_{\prec c}\varphi_2$ avec $\prec \in \{<, \leq\}$ et $c > 0$. Soit un état $\ell \in \{q_i, r_i, q'_i, r'_i\}$, Nous avons :
   - si $t > 0$, $(\ell, t) \models \mathsf{A}\varphi_1\mathsf{U}_{\prec c}\varphi_2$ si et seulement si $(\ell, t) \models \varphi_2$. En effet, il suffit de considérer une exécution à partir de (ℓ, t) qui reste au moins c unités de temps dans l'état ℓ. Il existe donc un délai $d \prec c$, tel que $(\ell, t + d)$ vérifie φ_2, par l'équivalence 4.10, on a que $(\ell, t) \models \varphi_2$.

   - $(\ell, 0) \models \mathsf{A}(\varphi_1\mathsf{U}_{\prec c}\varphi_2)$ si et seulement si $(\ell, 0) \models \varphi_1$ et $(\ell, t) \models \varphi_2$ pour tout $t > 0$ (ou $(\ell, 0) \models \varphi_2$). En effet, il suffit de considérer une exécution à partir de $(\ell, 0)$ qui reste au moins c unités de temps dans l'état ℓ. Il existe donc un délai $d \prec c$, tel que (ℓ, d) vérifie φ_2 et toutes les configurations avant satisfont φ_1. Si $d = 0$, on a $(\ell, 0) \models \varphi_2$. Sinon $d > 0$, donc $(\ell, 0) \models \varphi_1$, et par l'équivalence 4.10, $(\ell, t) \models \varphi_2$ pour tout $t > 0$.

   En appliquant l'hypothèse d'induction, dans ces deux cas, sur les sous-formules φ_1 et φ_2, on obtient :
   si $(\ell', t) \models \mathsf{A}(\varphi_1\mathsf{U}_{\prec c}\varphi_2)$ alors $(\ell, t) \models \mathsf{A}(\varphi_1\mathsf{U}_{\prec c}\varphi_2)$ avec $\ell \in \{q_i, r_i\}$.

2. Supposons que $\Psi = \mathsf{A}(\varphi_1\mathsf{U}_{=c}\varphi_2)$, avec $c > 0$. En utilisant l'équivalence 4.10, on a pour tout $\ell \in \{q_i, r_i, q'_i, r'_i\}$:
   - pour tout $t > 0$, $(\ell, t) \models \mathsf{A}(\varphi_1\mathsf{U}_{=c}\varphi_2)$ si et seulement si $(\ell, t) \models \varphi_1 \wedge \varphi_2$.
   - $(\ell, 0) \models \mathsf{A}(\varphi_1\mathsf{U}_{=c}\varphi_2)$ si et seulement si $(\ell, 0) \models \varphi_1$ et chaque configuration atteignable à partir de $(\ell, 0)$ vérifie $\varphi_1 \wedge \varphi_2$.

   Dans ces deux cas, l'hypothèse d'induction appliquée sur les sous-formules φ_1 et φ_2 donne que :
   si $(\ell', t) \models \mathsf{A}(\varphi_1\mathsf{U}_{=c}\varphi_2)$ alors $(\ell, t) \models \mathsf{A}(\varphi_1\mathsf{U}_{=c}\varphi_2)$ avec $\ell \in \{q_i, r_i\}$.

3. Si $\Psi = \mathsf{A}(\varphi_1\mathsf{U}_{=0}\varphi_2)$, alors Ψ est équivalente à φ_2 sur les configurations des automates M_i et N_i. Donc l'hypothèse d'induction donne le résultat.

4. Supposons maintenant que $\Psi = \mathsf{A}(\varphi_1\mathsf{U}_{\geq c}\varphi_2)$ et que $(q'_i, t) \models \mathsf{A}(\varphi_1\mathsf{U}_{\geq c}\varphi_2)$. Nous allons prouver que $(q_i, t) \models \mathsf{A}(\varphi_1\mathsf{U}_{\geq c}\varphi_2)$. Soit ρ une exécution dans M_i partant de (q_i, t), si

$f_{N_i}(\rho)$ existe alors le résultat est évident par hypothèse d'induction, sinon (le délai dans r_i est 0), nous allons construire dans N_i une exécution à partir de (q_i', t) qui est "équivalente" à ρ. Nous distinguons alors deux sous-cas selon le délai δ dans l'état q_i le long de ρ :

– Si $\delta < c$, nous modifions ρ pour obtenir ρ' en restant un temps non nul dans l'état r_i tel que (1) le temps total dans q_i et r_i le long de la nouvelle exécution ρ' reste inférieur strictement à c et (2) le temps total accumulé dans les états r_i et q_{i-1} le long de ρ' doit être égal au temps écoulé dans l'état q_{i-1} le long de ρ. Ceci est illustré dans la figure 4.8. Puis, nous considérons le correspondant $f_{N_i}(\rho')$ dans N_i (il est bien défini). Par l'hypothèse d'induction, toutes les positions correspondantes de ρ' et $f_{N_i}(\rho')$ ont la même valeur de vérité pour les formules φ_1 et φ_2, or $(q_i', t) \models \mathsf{A}(\varphi_1 \mathsf{U}_{\geq c} \varphi_2)$ donc $f_{N_i}(\rho') \models \varphi_1 \mathsf{U}_{\geq c} \varphi_2$ et par conséquent $\rho' \models \varphi_1 \mathsf{U}_{\geq c} \varphi_2$. En particulier, φ_1 est vérifiée dans toutes les configurations (ℓ, t) avec $\ell \in \{q_i, r_i\}$ et $t \geq 0$. De plus, la formule φ_2 est vérifiée dans une position le long de ρ' (après l'état r_i) et par construction, φ_2 est vérifiée dans la même position le long de ρ. On conclut que $\rho \models \varphi_1 \mathsf{U}_{\geq c} \varphi_2$.

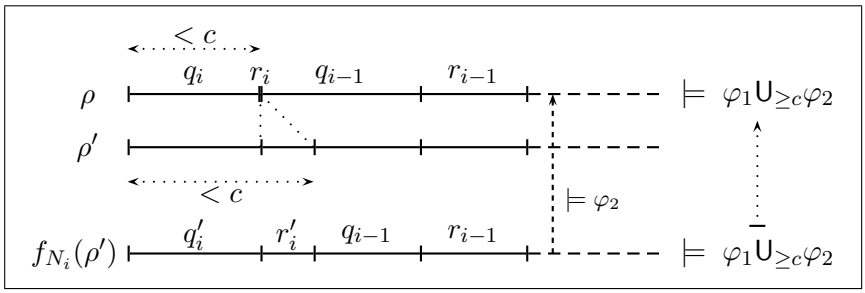


FIG. 4.8 – Construction pour le cas $\geq$ et $\delta < c$

– Si, pour $\delta \geq c$, nous modifions l'exécution ρ pour obtenir ρ' en restant un temps non nul dans l'état r_i comme dans le cas précédent (figure 4.9). Nous considérons ensuite l'exécution $f_{N_i}(\rho')$ dans N_i. Par hypothèse, cette exécution vérifie $\varphi_1 \mathsf{U}_{\geq c} \varphi_2$. Nous avons donc plusieurs sous-cas :

  i La sous-formule φ_2 est vérifiée dans une configuration $(q_i', t+d)$ avec $d \geq c$. Dans ce cas, φ_2 est aussi vérifiée dans $(q_i, t+d)$ par hypothèse d'induction et φ_1 est vraie dans chaque $(q_i, t+d')$ où $d' < d$ (en utilisant aussi l'hypothèse d'induction), donc $\rho \models \varphi_1 \mathsf{U}_{\geq c} \varphi_2$.

  ii La formule φ_2 est vérifiée dans une configuration (r_i', d) où $d \geq 0$, par conséquent φ_2 est aussi vraie dans (r_i, d) (par hypothèse d'induction), donc $(r_i, 0) \models \varphi_2$ (par l'équivalence 4.9), donc $\rho \models \varphi_1 \mathsf{U}_{\geq c} \varphi_2$ (par hypothèse d'induction sur les sous-formules).

  iii la formule φ_2 est vérifiée dans une autre configuration (ℓ, d), elle le sera aussi le long de ρ, donc $\rho \models \varphi_1 \mathsf{U}_{\geq c} \varphi_2$ (en appliquant toujours l'hypothèse d'induction sur les sous-formules).

Nous concluons alors pour ces trois cas que $(q_i, t) \models \mathsf{A} \varphi_1 \mathsf{U}_{\geq c} \varphi_2$.

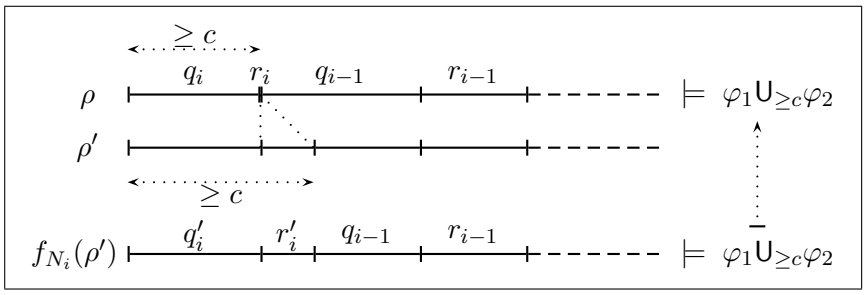


FIG. 4.9 – Construction pour le cas $\geq$ et $\delta \geq c$

La même preuve peut être adaptée pour prouver que $(r'_i, t) \models \mathsf{A}\varphi_1 \mathsf{U}_{\geq c} \varphi_2$ implique $(r_i, t) \models \mathsf{A}\varphi_1 \mathsf{U}_{\geq c} \varphi_2$.

5. Supposons maintenant que $\Psi = \mathsf{A}(\varphi_1 \mathsf{U}_{>c} \varphi_2)$. Notons δ le délai dans l'état q_i, nous distinguons deux sous-cas :
   – Si $\delta = 0$ et $c = 0$, nous modifions ρ pour obtenir ρ' en restant un temps non nul dans l'état r_{i-1} que nous diminuons du délai écoulé dans q_{i-1} (ce dernier étant strictement positif). L'exécution ρ' ainsi construite possède un correspondant ρ'_{N_i} dans N_i qui un préfixe similaire jusqu'à la configuration $(q_{i-2}, 0)$ et un suffixe identique à partir $(q_{i-2}, 0)$. Cette construction est expliquée dans la figure 4.10. Comme $\rho'_{N_i} \models \varphi_1 \mathsf{U}_{>0} \varphi_2$, en appliquant l'hypothèse d'induction et les équivalences 4.9 et 4.10, on obtient que $\rho \models \varphi_1 \mathsf{U}_{>0} \varphi_2$.

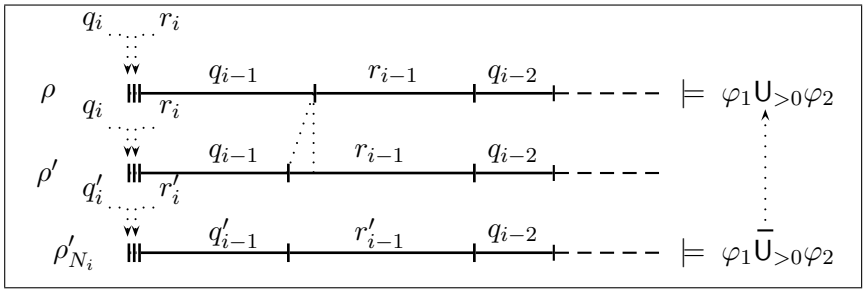


FIG. 4.10 – Construction pour le cas > 0 et $\delta = 0$

   – Sinon $\delta \neq 0$ ou $c \neq 0$, ce cas est presque identique au cas 4. Si $\delta < c$ ou $\delta > c$, les mêmes constructions du cas 4 nous donnerons le résultat. Sinon $\delta = c$, nous construisons à partir de ρ une exécution ρ' qui reste un délai non nul dans r_i que nous diminuons du délai écoulé dans q_i (celui-ci étant strictement positif) et nous considérons l'exécution $f_{N_i}(\rho')$ qui satisfait $\varphi_1 \mathsf{U}_{>c} \varphi_2$. Par hypothèse d'induction, on déduit que $\rho' \models \varphi_1 \mathsf{U}_{>c} \varphi_2$, puis que $\rho \models \varphi_1 \mathsf{U}_{>c} \varphi_2$ (en utilisant les équivalences (4.9) et (4.10)). La différence avec les précédentes constructions est que le temps écoulé dans q_i est diminué pour que le temps total dans q_i et r_i (le long de ρ') reste égal exactement à c u.t (figure 4.11).

Pour prouver que si $(r'_i, t) \models \mathsf{A}\varphi_1 \mathsf{U}_{>c} \varphi_2$ alors $(r_i, t) \models \mathsf{A}\varphi_1 \mathsf{U}_{>c} \varphi_2$, on adapte facilement la construction de la figure 4.8.

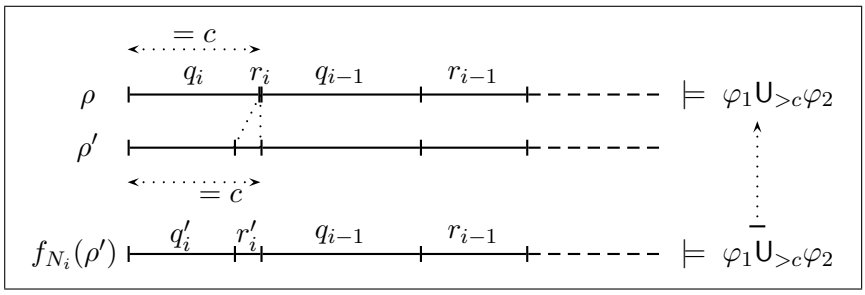


FIG. 4.11 – Construction pour le cas $>$ et $\delta = c$

Ceci termine la preuve de la proposition 1.

□

4.5.3 U^k ne peut pas s'exprimer avec U^{k+1}

L'idée de la preuve est similaire au cas précédent, cependant elle est techniquement plus difficile à obtenir. Notons $\mathsf{TCTL}^k_{\setminus \mathsf{U}}$ le fragment de TCTL^k où les opérateurs $\mathsf{U}_{\sim c}$ ne sont pas autorisés. Nous allons construire parallèlement (1) deux familles d'automates temporisés telles qu'à partir d'un rang donné, elles sont indistinguables par les formules de $\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}$ et (2) une formule de $\mathsf{TCTL}^k_{\setminus \mathsf{U}}$ qui va pourtant les distinguer.

Considérons les familles $(M_i)_{i\geq 1}$ et $(N_i)_{i\geq 1}$ de la figure 4.12. Notons que n'importe quelle configuration de la forme $(q_j, t), (q'_j, t), (r_j, t), (r'_j, t)$ où $j \geq 1$, existe au plus une seule fois le long de toutes les exécutions de M_i et N_i. Soit Ψ la formule de $\mathsf{TCTL}^k_{\setminus \mathsf{U}}$, $\mathsf{E}a\mathsf{U}^k_{>0}b$, alors :

$$(q_i, 0) \models \Psi \text{ et } (q'_i, 0) \not\models \Psi \text{ pour tout } i \geq 1 \tag{4.11}$$

En effet, dans l'automate M_i où $i \geq 1$, il existe une exécution à partir de $(q_i, 0)$ telle qu'avant d'arriver à l'état r_0 (l'unique état étiqueté par b), les délais dans les états qui ne vérifient pas a sont inférieurs ou égaux à k (en particulier, étant données les gardes de M_i, ces délais sont exactement égaux à k unités de temps), alors que dans N_i, il faut rester un délai strictement plus grand que k dans un état étiqueté par $\neg a$ avant d'arriver à r_0.

Nous allons montrer maintenant qu'il n'existe aucune formule de $\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}$ équivalente à Ψ. Ceci est une conséquence de la proposition suivante :

Proposition 2
Pour les deux familles M_i et N_i de la figure 4.12, $\forall n \geq 1$, $\forall i \geq n$ et $\forall t \in \mathbb{R}_+$, nous avons :

$$(q_i, t) \equiv^n_{\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}} (q'_i, t) \quad \textit{et} \quad (r_i, t) \equiv^n_{\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}} (r'_i, t)$$

Cette proposition dit que pour tout $i \geq 1$, les automates M_i et N_i satisfont les mêmes formules de $\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}$ de taille inférieure ou égale à i. On en déduit directement le résultat d'expressivité suivant :

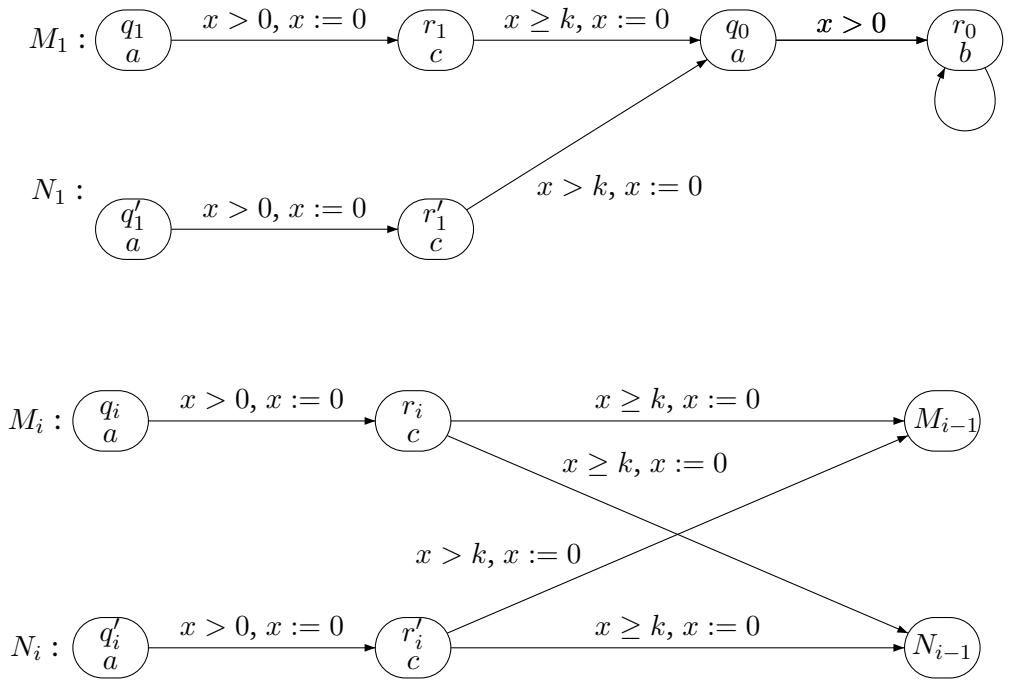


FIG. 4.12 – Les automates M_i et N_i, $i = 1, 2, \ldots$

Théorème 6
La logique $\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}$ *n'est pas aussi expressive que la logique* $\mathsf{TCTL}^{k}_{\setminus \mathsf{U}}$.

Démonstration: Supposons qu'il existe une formule Φ de $\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}$ équivalente à la formule Ψ. D'après 4.11, on aurait $(q_i, 0) \models \Phi$ et $(q'_i, 0) \not\models \Phi$ pour n'importe quel $i \geq 1$. Cependant, ceci contredit la proposition 2 d'après laquelle $(q_i, 0) \equiv^{|\Phi|}_{\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}} (q'_i, 0)$ pour n'importe quel $i \geq |\Phi|$.

□

Il nous reste maintenant à prouver la proposition 2. Celle-ci résulte de plusieurs propriétés des automates M_i et N_i, que nous décrivons dans la remarque et la propriété qui suivent :

Remarque 3
– *Soit* ρ *une exécution de* M_i *partant de* (q_i, t) *avec* $i \geq 2$ *et* $t \in \mathbb{R}_+$. *Cette exécution est caractérisée par le paramètre* δ_0 *qui est le temps écoulé dans l'état* q_i, *par le paramètre* δ_1 *qui est le temps écoulé dans l'état* r_i *et par le suffixe* ρ_1 *qui est soit dans* N_{i-1} *soit dans* M_{i-1}. *Donc* ρ *s'écrit sous la forme suivante :*

$$(q_i, t) \xrightarrow{\delta_0} (q_i, t + \delta_0) \rightarrow_a (r_i, 0) \xrightarrow{\delta_1} (r_i, \delta_1) \overset{\rho_1}{\mapsto}$$

Si ρ_1 *est dans* N_{i-1} *alors* ρ *a un correspondant dans* N_i, *noté* $f_{N_i}(\rho)$ *et défini par :*

$$(q'_i, t) \xrightarrow{\delta_0} (q'_i, t+\delta_0) \rightarrow_a (r'_i, 0) \xrightarrow{\delta_1} (r'_i, \delta_1) \overset{\rho_1}{\mapsto}$$

Si ρ_1 est dans M_{i-1} et $\delta_1 > k$, on peut définir de la même manière un correspondant de ρ dans N_i. Mais dans le cas où $\delta_1 = k$, le correspondant $f_{N_i}(\rho)$ n'est pas défini.

– *La même correspondance peut être établie pour chaque exécution dans M_i à partir d'une configuration (r_i, t). Ces exécutions sont caractérisées seulement par le paramètre δ_1 et le suffixe ρ_1.*

– *Pour chaque exécution ρ dans N_i, on peut définir un correspondant dans M_i noté $f_{M_i}(\rho)$ comme ci-dessus.*

Étant données les gardes et les remises à zéro des familles d'automates M_i et N_i, nous avons la propriété suivante :

Propriété 2
Pour tout $j \geq 0$, on a :

$$(r_j, k) \equiv_{\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}} (r_j, t) \qquad \forall t > k \tag{4.12}$$

$$(\ell, t) \equiv_{\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}} (\ell, t') \qquad \forall t, t' > 0,\ \forall \ell \in \{q_j, q'_j\} \tag{4.13}$$

$$(s, t) \equiv_{\mathsf{TCTL}^{k+1}_{\setminus \mathsf{U}}} (s, t') \qquad \forall t, t' > k,\ \forall s \in \{r_j, r'_j\} \tag{4.14}$$

Cette propriété peut être facilement démontrée de manière similaire à la propriété 1.

Nous donnons finalement la preuve de la proposition 2.

Démonstration: [de la proposition 2]
Nous prouvons le résultat par induction sur la taille n des formules.

Pour les formules de taille $n = 1$, les équivalences sont vérifiées puisque les états q_i et q'_i (respectivement r_i et r'_i) sont étiquetés par les mêmes propositions atomiques.

Considérons une formule Ψ de taille supérieure strictement à 1 et supposons que les équivalences de la proposition 2 sont vraies pour toutes les formules de taille inférieure strictement à n. Nous procédons maintenant par induction structurelle sur Ψ :

Si Ψ est une négation d'une sous-formule ou une conjonction de deux sous-formules, alors l'hypothèse d'induction, appliquée aux sous-formules, donne le résultat.

Formule $\mathsf{E\_U}^{k+1}_{\sim c}\mathsf{\_}$. Si $\Psi = \mathsf{E}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2)$, d'après la remarque 3, si ρ est une exécution dans N_i, alors l'exécution correspondante $f_{M_i}(\rho)$ existe, par hypothèse d'induction :

$$\rho \models \varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2 \Leftrightarrow f_{M_i}(\rho) \models \varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$$

On en déduit alors que :

$$\begin{cases} (q'_i,t) \models \mathsf{E}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2) & \Longrightarrow & (q_i,t) \models \mathsf{E}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2) \\ (r'_i,t) \models \mathsf{E}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2) & \Longrightarrow & (r_i,t) \models \mathsf{E}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2) \end{cases}$$

Montrons maintenant les implications inverses :

– Supposons que $(q_i,t) \models \mathsf{E}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2)$, il existe alors dans M_i une exécution ρ telle que $\rho \models \varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$. Si $f_{N_i}(\rho)$ existe alors l'hypothèse d'induction appliquée à $f_{N_i}(\rho)$ nous donne que $(q'_i,t) \models \mathsf{E}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2)$.

Si $f_{N_i}(\rho)$ n'est pas défini, le temps écoulé le long de ρ dans l'état r_i est égal à k unités de temps et le suffixe $\overset{\rho_1}{\mapsto}$ est dans M_{i-1}. L'exécution ρ a donc la forme suivante :

$$(q_i,t) \overset{\delta_0}{\to} (q_i, t+\delta_0) \to_a (r_i,0) \overset{k}{\to} (r_i,k) \to_a (q_{i-1},0) \overset{\rho_1}{\mapsto}$$

Notons I le sous-chemin de ρ de longueur strictement supérieure à $k+1$ qui contient une configuration (l,v) tel que I satisfait φ_2, la durée du préfixe de ρ jusqu'à (l,v) satisfait $\sim c$ et tous les sous-chemins avant (l,v) où la formule φ_1 est fausse sont de longueur inférieure ou égale à $k+1$.

Supposons que I est avant $(q_{i-1},0)$ (ou atteint $(q_{i-1},0)$), on construit alors dans N_i une exécution qui a le même préfixe que ρ qui se termine par I, par hypothèse d'induction, cette exécution vérifie $\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$ et par conséquent on a $(q'_i,t) \models \mathsf{E}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2)$.

Sinon, nous construisons dans M_i une exécution ρ' telle que $f_{N_i}(\rho')$ existe. Pour cela, nous augmentons légèrement le temps passé dans l'état r_i, de sorte que ce délai ajouté dans r_i soit suffisamment petit pour que ρ' vérifie aussi $\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$. Pour cela, nous distinguons plusieurs sous-cas :
■ (l,v) est avant (r_i,k) (ou $(l,v) = (r_i,k)$). C'est-à-dire que le sous-chemin I commence avant la configuration (r_i,k) mais se termine après la configuration $(q_{i-1},0)$. Pour construire ρ', nous allons ajouter le long de ρ un petit délai dans l'état r_i que nous diminuons du temps écoulé dans l'état q_{i-1}(figure 4.13). Ainsi la somme du délai ajouté dans l'état r_i et du délai passé dans l'état q_{i-1} le long de ρ' est égale au temps passé dans l'état q_{i-1} le long de ρ.
Comme le long de ρ, la formule φ_2 est vraie dans (r_i,k), alors elle l'est aussi dans toutes les configurations de la forme $(r_i,k+d)$ le long de ρ' avec $d \geq 0$ (en appliquant la propriété 4.12). L'ajout d'un délai dans l'état r_i préserve donc le fait que φ_2 est vraie pendant un délai strictement supérieur à $k+1$ unités de temps. Notons que la contrainte $\sim c$ reste vérifiée dans la configuration (l,v) de ρ' (et par conséquent le long de $f_{N_i}(\rho')$ aussi) : en effet, la durée du préfixe jusqu'à (l,v) est inchangée. Donc l'exécution $f_{N_i}(\rho')$ est un "témoin" pour $(q'_i,t) \models \mathsf{E}\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$.

■ (l,v) est après (r_i,k). Dans ce cas nous traitons des sous-cas séparément suivant que (r_i,k) satisfait ou non la sous-formule φ_1 :

– $(r_i,k) \models \varphi_1$. Comme précédemment nous allons modifier légèrement ρ pour construire une exécution ρ' dans M_i qui possède un correspondant dans N_i. Nous ajoutons donc un

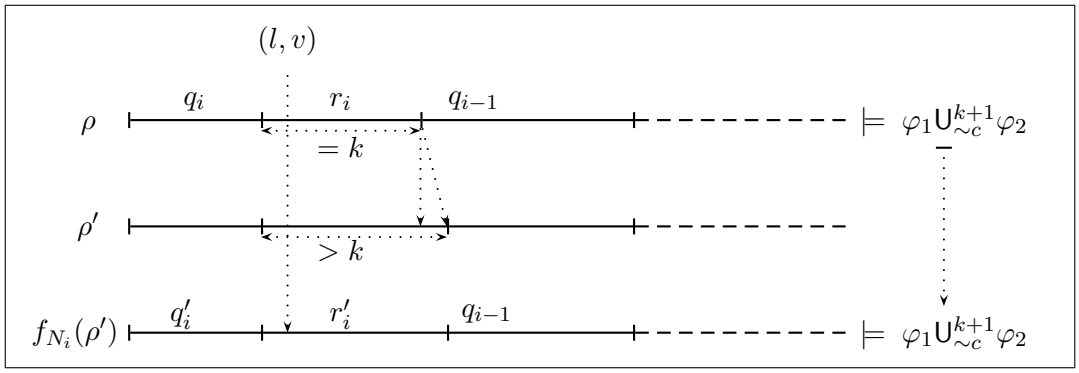


FIG. 4.13 – Construction pour le cas où (l, v) est avant (r_i, k)

délai dans l'état r_i que nous diminuons du temps passé dans q_{i-1}. Or, nous voulons que la nouvelle exécution ρ' satisfasse aussi $\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$, c'est pourquoi le délai diminué du temps écoulé dans q_{i-1} doit être suffisamment petit pour ne pas trop réduire la longueur du sous-chemin I qui doit rester supérieure strictement à $k + 1$ (Ceci peut se produire si I commence juste après (r_i, k)). Donc, pour obtenir ρ', nous ajoutons un délai ε dans l'état r_i tel que les deux conditions suivantes soient satisfaites (figure 4.14) :

Cond 1. la somme du délai ajouté dans l'état r_i et du délai dans l'état q_{i-1} le long de ρ' est égale au temps passé dans q_{i-1} le long de ρ.

Cond 2. si I commence par une position de la forme (q_{i-1}, t) avec $t \geq 0$ (par la propriété 4.13, I contient toutes les configurations (q_{i-1}, t) avec $t > 0$), alors le délai ε doit être suffisamment petit pour que la longueur de I diminuée de ε reste supérieure strictement à $k + 1$ (i.e. $\hat{\mu}(I) - \varepsilon > k + 1$).

Notons qu'il est toujours possible de choisir un tel délai ε car $\hat{\mu}(I) > k + 1$.

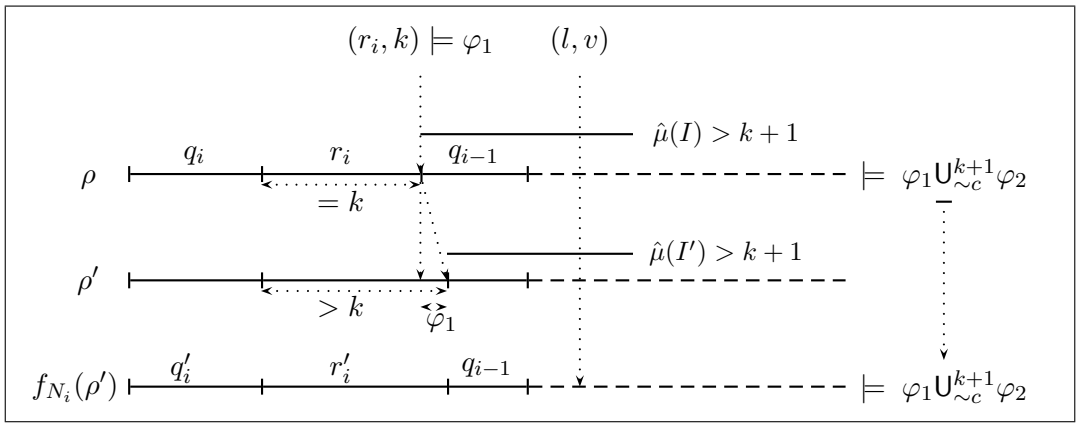


FIG. 4.14 – Construction pour le cas où $(r_i, k) \models \varphi_1$

Les configurations ajoutées le long de ρ' vérifient φ_1 par la propriété 4.12, donc $\rho' \models \varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$. Par conséquent, $f_{N_i}(\rho') \models \varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$ (en appliquant l'hypothèse de récurrence sur φ_1 et φ_2).

– $(r_i, k) \models \neg\varphi_1$. Considérons $\sigma_{(\neg\varphi_1)}$ le sous-chemin maximal de ρ qui contient (r_i, k) et qui vérifie $\neg\varphi_1$.

• Si la longueur de $\sigma_{(\neg\varphi_1)}$ est inférieure strictement à $k+1$ ($\hat{\mu}(\sigma_{(\neg\varphi_1)}) < k+1$), nous pouvons adapter facilement la construction précédente (figure 4.14) : nous ajoutons un délai ε dans r_i que nous diminuons du temps écoulé dans q_{i-1}. Cependant, les configurations ajoutées par ce délai vérifient $\neg\varphi_1$ (propriété 4.12), donc ε doit être suffisamment petit pour que la longueur de $\sigma_{(\neg\varphi_1)}$ augmentée de ε reste toujours inférieure ou égale à $k+1$. Ainsi pour obtenir ρ', le délai ajouté ε doit vérifier en plus des conditions *Cond 1* et *Cond 2* le fait que $\hat{\mu}(\sigma_{(\neg\varphi_1)}) + \varepsilon \leq k+1$ (ε est bien défini car $\hat{\mu}(\sigma_{(\neg\varphi_1)}) < k+1$).

• Si la longueur de $\sigma_{(\neg\varphi_1)}$ est exactement égale à $k+1$ ($\hat{\mu}(\sigma_{(\neg\varphi_1)}) = k+1$). Dans ce cas, l'ajout du temps dans r_i risque de changer la valeur de vérité de la formule $\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$ (en effet, on peut obtenir –à cause des nouvelles configurations dans r_i– le long de ρ', un sous-chemin avant (l, v) de longueur strictement supérieure à $k+1$ où φ_1 est toujours fausse –propriété 4.12–). C'est pour cela que nous distinguons encore des sous-cas :

Soit $(q_{i-1}, 0) \models \varphi_1$: Comme la longueur de $\sigma_{(\neg\varphi_1)}$ est égale à $k+1$ et le délai dans r_i le long de ρ est exactement de k unités de temps, $\sigma_{(\neg\varphi_1)}$ contient alors au moins une position de la forme (q_i, t) avec $t > 0$. Ceci implique (en utilisant la propriété 4.13) que toutes les positions (q_i, t) avec $t > 0$ vérifient $\neg\varphi_1$. Donc, pour construire l'exécution ρ' dans M_i nous augmentons le temps écoulé dans l'état r_i que nous diminuons de δ_0 (dans ce cas δ_0, le temps passé dans q_i, est nécessairement strictement positif). Ainsi, la somme le long de ρ' du délai ajouté dans r_i et du délai écoulé dans q_i est égale à δ_0 le long de ρ (figure 4.15).

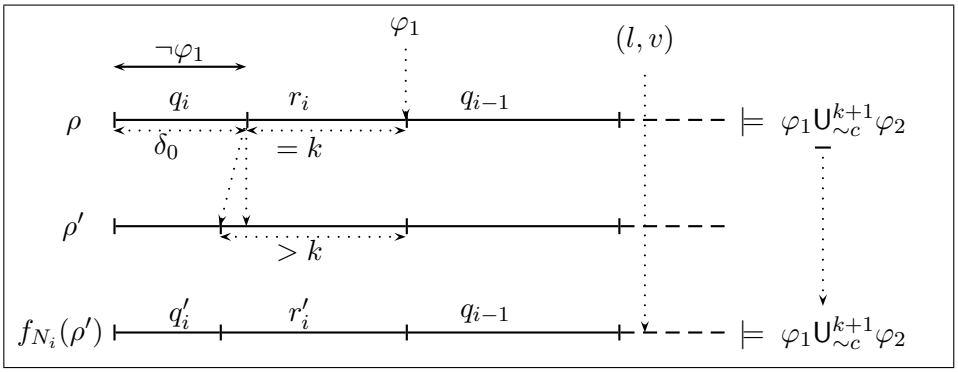


FIG. 4.15 – Construction pour le cas avec $(q_{i-1}, 0) \models \varphi_1$

Notons que la durée du préfixe jusqu'à la position (l, v) n'a pas changé, donc $\rho' \models \varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$. Par hypothèse d'induction, on a $f_{N_i}(\rho') \models \varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$.

Soit $(q_{i-1}, 0) \models \neg\varphi_1$: on distingue deux sous-cas :

1. s'il existe une position (q_{i-1}, t) avec $t > 0$ qui satisfait $\neg\varphi_1$, alors toutes les positions de ρ de la forme (q_{i-1}, t) avec $t > 0$ satisfont $\neg\varphi_1$ (par la propriété 4.13) : ceci signifie que $\sigma_{(\neg\varphi_1)}$ contient alors toutes les positions de ρ dans l'état q_{i-1}. Dans

ce cas, nous pouvons alors ajouter un petit délai dans r_i que nous diminuons du délai dans q_{i-1} (δ_2) sans augmenter la longueur du sous-chemin maximal le long de ρ' qui contient (r_i, k) et qui vérifie $\neg\varphi_1$. Donc en ajoutant un petit délai dans r_i qui vérifie les conditions *Cond 1* et *Cond 2*, on construit une exécution ρ' telle que $f_{N_i}(\rho') \models \varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$ (par hypothèse d'induction).

2. sinon, toutes les positions (q_{i-1}, t) avec $t > 0$ satisfont φ_1. Dans ce cas, $\sigma_{(\neg\varphi_1)}$ contient au moins une position de la forme (q_i, t) avec $t > 0$. Donc, pour construire l'exécution ρ' dans M_i nous augmentons le temps écoulé dans l'état r_i que nous diminuons de δ_0 (le temps passé dans q_i) –même construction que la figure 4.15–. Donc $f_{N_i}(\rho')$ est un témoin pour $(q'_i, t) \models \mathsf{E}\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$.

– On adapte facilement les preuves et les constructions précédentes pour démontrer que :

$$(r_i, t) \models \mathsf{E}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2) \implies (r'_i, t) \models \mathsf{E}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2)$$

Formule $\mathsf{A\_U}^{k+1}_{\sim c}\mathsf{\_}$**.** Supposons maintenant que $\Psi = \mathsf{A}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2)$. D'après la remarque 2, si ρ est une exécution dans N_i, alors l'exécution correspondante $f_{M_i}(\rho)$ existe. En appliquant l'hypothèse d'induction, on déduit :

$$(q_i, t) \models \mathsf{A}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2) \implies (q'_i, t) \models \mathsf{A}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2) \quad (4.15)$$

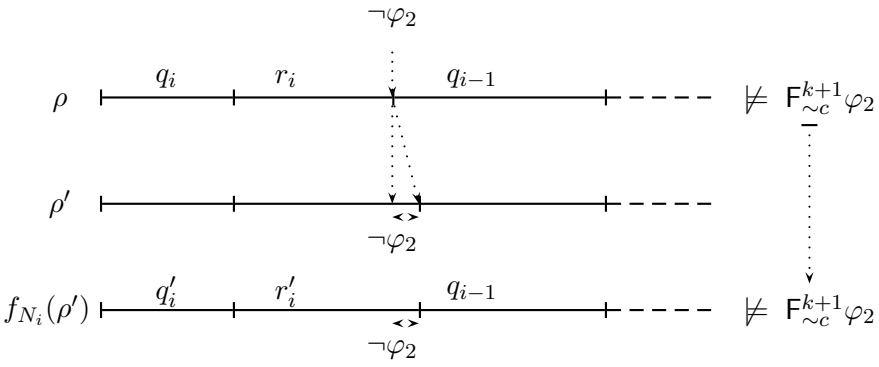
$$(r_i, t) \models \mathsf{A}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2) \implies (r'_i, t) \models \mathsf{A}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2) \quad (4.16)$$

Avant de prouver les réciproques, nous énonçons la proposition ci-dessous :

Proposition 3
Si $(q_i, t) \not\models \mathsf{AF}^{k+1}_{\sim c} \varphi_2$ *alors* $(q'_i, t) \not\models \mathsf{AF}^{k+1}_{\sim c} \varphi_2$

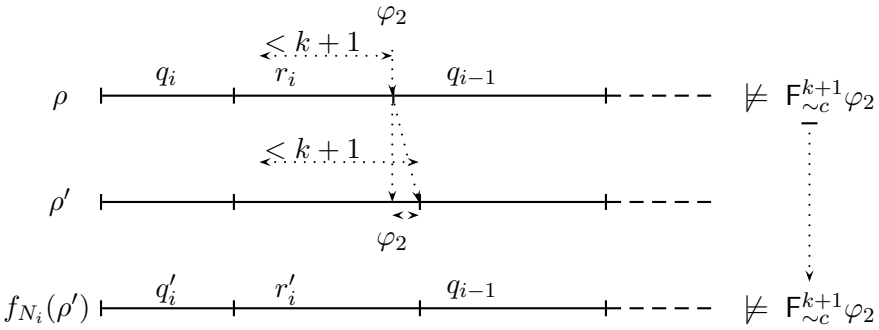
Démonstration: Supposons que $(q_i, t) \not\models \mathsf{AF}^{k+1}_{\sim c} \varphi_2$. Soit donc ρ l'exécution de M_i partant de (q_i, t) telle que $\rho \not\models \mathsf{F}^{k+1}_{\sim c} \varphi_2$. Si ρ possède un correspondant dans N_i alors $f_{N_i}(\rho) \not\models \mathsf{F}^{k+1}_{\sim c} \varphi_2$ (par hypothèse d'induction). Sinon le délai dans r_i est égal à k unités de temps et le suffixe $\overset{\rho_1}{\mapsto}$ est dans M_{i-1}. Dans ce cas, nous modifions ρ en augmentant le délai dans r_i :

– Si $(r_i, k) \models \neg\varphi_2$, on construit ρ' en ajoutant un petit délai dans r_i que nous diminuons du temps passé dans q_{i-1}.



S'il existe le long de $f_{N_i}(\rho')$ un sous-chemin σ de longueur strictement supérieure à $k+1$ qui contient une position p telle que $\sigma \models \varphi_2$ et $\mathsf{Time}(\rho^{\leq p}) \sim c$, alors par hypothèse d'induction un tel sous-chemin existerait le long de ρ, ce qui donne une contradiction. Donc l'exécution $f_{N_i}(\rho')$ ne satisfait pas $\mathsf{F}^{k+1}_{\sim c}\varphi_2$.

– Si $(r_i, k) \models \varphi_2$. Soit $\sigma_{(\varphi_2)}$ le sous-chemin maximal contenant (r_i, k) qui satisfait φ_2.
Si $\hat{\mu}(\sigma_{(\varphi_2)}) < k+1$, on ajoute alors un petit délai ε dans r_i que nous diminuons du délai dans q_{i-1} tel que la valeur $\hat{\mu}(\sigma_{(\varphi_2)}) + \varepsilon$ reste inférieure à $k+1$. Donc $f_{N_i}(\rho') \not\models \mathsf{F}^{k+1}_{\sim c}\varphi_2$: en effet, si $f_{N_i}(\rho')$ contient un sous-chemin σ "témoin" de $\mathsf{F}^{k+1}_{\sim c}\varphi_2$ (c'est-à-dire de longueur $> k+1$ qui contient une position p telle que $\sigma \models \varphi_2$ et $\mathsf{Time}(\rho^{\leq p}) \sim c$), par hypothèse d'induction et la propriété 4.12 un tel sous-chemin existe le long de ρ.



Sinon $\hat{\mu}(\sigma) = k+1$, on distingue alors deux sous-cas :

1. Si $(q_{i-1}, t) \models \varphi_2$ avec $t \geq 0$, on construit ρ' en ajoutant un délai dans r_i que nous diminuons du délai dans q_{i-1}.
2. Si $(q_{i-1}, t) \models \neg\varphi_2$ avec $t > 0$, comme $\hat{\mu}(\sigma) = k+1$ et que le délai dans r_i est égal à k, on a : $(q_i, t) \models \varphi_2$ avec $t > 0$. Dans ce cas, pour obtenir ρ', nous ajoutons un délai dans r_i que nous diminuons du temps passé dans q_i (c'est-à-dire δ_0).

Dans ces deux cas, on montre facilement en appliquant l'hypothèse d'induction et la propriété 4.12 que $f_{N_i}(\rho') \not\models \mathsf{F}^{k+1}_{\sim c}\varphi_2$

□

Nous allons maintenant démontrer les réciproques des implications 4.15 et 4.16. Pour cela nous prouvons leur contraposée :

1. Supposons que $(q_i, t) \not\models \mathsf{A}(\varphi_1 \mathsf{U}^{k+1}_{\sim c}\varphi_2)$. Donc il existe dans M_i une exécution partant de (q_i, t) qui ne satisfait pas $\varphi_1 \mathsf{U}^{k+1}_{\sim c}\varphi_2$. Soit ρ une telle exécution. Si $f_{N_i}(\rho)$ existe, alors $(q'_i, t) \not\models \mathsf{A}(\varphi_1 \mathsf{U}^{k+1}_{\sim c}\varphi_2)$. Sinon le délai dans r_i est égal à k unités de temps et le suffixe de ρ est dans M_{i-1}. Comme $\rho \not\models \varphi_1 \mathsf{U}^{k+1}_{\sim c}\varphi_2$, on a soit $\rho \not\models \mathsf{F}^{k+1}_{\sim c}\varphi_2$, soit pour tout sous-chemin I le long de ρ de longueur strictement supérieure à $k+1$, contenant une position p telle que $\mathsf{Time}(\rho^{\leq p}) \sim c$ et $I \models \varphi_2$, il existe un sous-chemin σ avant p tel que $\hat{\mu}(\sigma) > k+1$ et $\sigma \models \neg\varphi_1$. Nous traitons ces deux cas séparément :

   ■ Si $\rho \not\models \mathsf{F}^{k+1}_{\sim c}\varphi_2$, alors $(q_i, t) \not\models \mathsf{AF}^{k+1}_{\sim c}\varphi_2$ et en appliquant la proposition 3, $(q'_i, t) \not\models \mathsf{AF}^{k+1}_{\sim c}\varphi_2$. Il existe alors dans N_i une exécution partant de (q'_i, t) qui ne sa-

tisfait pas $\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$.

■ Sinon, on considère I, le premier sous-chemin de ρ de longueur $> k+1$, contenant une configuration (l, v) telle que la longueur du préfixe de ρ jusqu'à cette configuration satisfait $\sim c$ ($\mathsf{Time}(\rho^{\leq (l,v)}) \sim c$) et $I \models \varphi_2$. Il existe alors un sous-chemin σ avant (l, v) tel que $\hat{\mu}(\sigma) > k+1$ et $\sigma \models \neg\varphi_1$.
Soit (l, v) est avant (r_i, k) (ou $(l, v) = (q_{i-1}, 0)$). Donc il suffit de considérer dans N_i l'exécution qui a le même préfixe que ρ jusqu'à (l, v) (resp. jusqu'à (r_i, k)). En appliquant l'hypothèse d'induction, cette exécution ne satisfait pas $\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$.
Soit (l, v) est après $(q_{i-1}, 0)$. Dans ce cas, nous modifions légèrement l'exécution ρ pour construire une exécution ρ' dans M_i telle que $f_{N_i}(\rho')$ ne satisfait pas $\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$. Nous distinguons plusieurs sous-cas :

• Soit $(r_i, k) \models \neg\varphi_2$:

(a) Si le sous-chemin σ est avant (r_i, k) (c'est-à-dire σ commence et se termine avant (r_i, k)). On construit alors ρ' en augmentant le temps dans r_i par un petit délai que nous diminuons du temps passé dans q_{i-1} (figure 4.16). En appliquant l'hypothèse d'induction, il existe alors le long de $f_{N_i}(\rho')$, un sous-chemin σ' tel que $\hat{\mu}(\sigma') > k+1$ et $\sigma' \models \neg\varphi_1$ (σ' est le "correspondant" de σ) avant que φ_2 dure "assez longtemps" (c'est-à-dire avant le premier sous-chemin témoins de $\mathsf{F}^{k+1}_{\sim c}\ \varphi_2$: de longueur strictement supérieure à $k+1$, qui vérifie φ_2 et contenant une position p telle que $\mathsf{Time}(f_{N_i}(\rho')^{\leq p}) \sim c$).

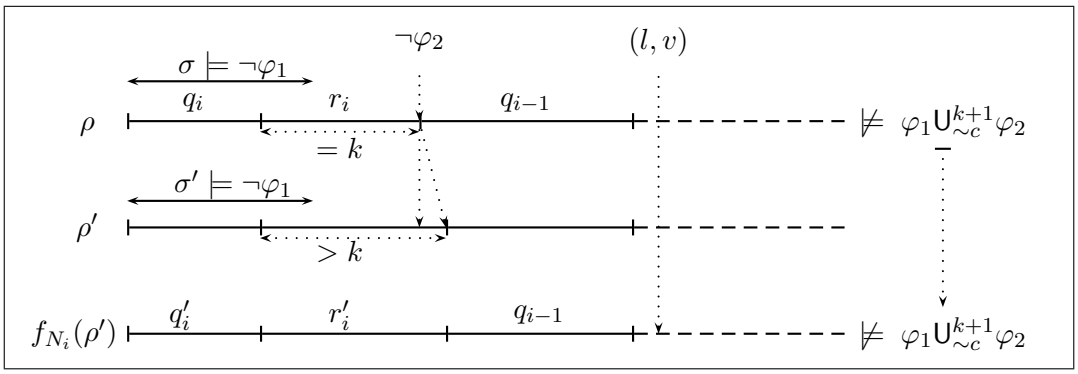


FIG. 4.16 – Construction pour le cas (a)

(b) Si le sous-chemin σ commence avant (r_i, k) mais se termine après (r_i, k). ρ' est alors obtenue à partir de ρ en ajoutons un petit délai dans r_i que nous diminuons du temps écoulé dans q_{i-1} (figure 4.17). En appliquant l'hypothèse d'induction et la propriété 4.12, il existe alors le long de $f_{N_i}(\rho')$, un sous-chemin σ' de longueur supérieure à $k+1$ tel que $\sigma' \models \neg\varphi_1$ avant le premier sous-chemin témoin de $\mathsf{F}^{k+1}_{\sim c}\ \varphi_2$ (σ' est la "projection" de σ sur $f_{N_i}(\rho')$).

(c) Si le sous-chemin σ commence après (r_i, k). On construit ρ' en ajoutons un délai ε dans r_i que nous diminuons du temps écoulé dans q_{i-1}. Le délai ε est défini comme

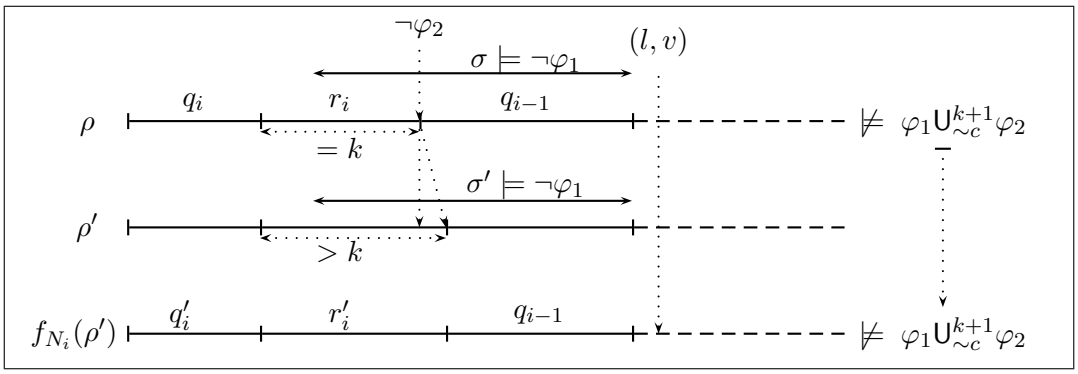


FIG. 4.17 – Construction pour le cas (b)

suit : si σ commence par une position (q_{i-1}, t) avec $t \in \mathbb{R}_+$ (par la propriété 4.13, σ contient toutes les positions de la forme (q_{i-1}, t) avec $t > 0$), alors ε doit être suffisamment petit pour que la longueur de σ diminuée de ε reste strictement supérieure à $k+1$, c'est-à-dire $\varepsilon < \hat{\mu}(\sigma) - k + 1$ (figure 4.18).
En appliquant l'hypothèse d'induction, il existe alors le long de $f_{N_i}(\rho')$, un sous-chemin σ' de longueur supérieure à $k+1$ tel que $\sigma' \models \neg\varphi_1$ avant le premier sous-chemin témoin de $\mathsf{F}^{k+1}_{\sim c}\ \varphi_2$.

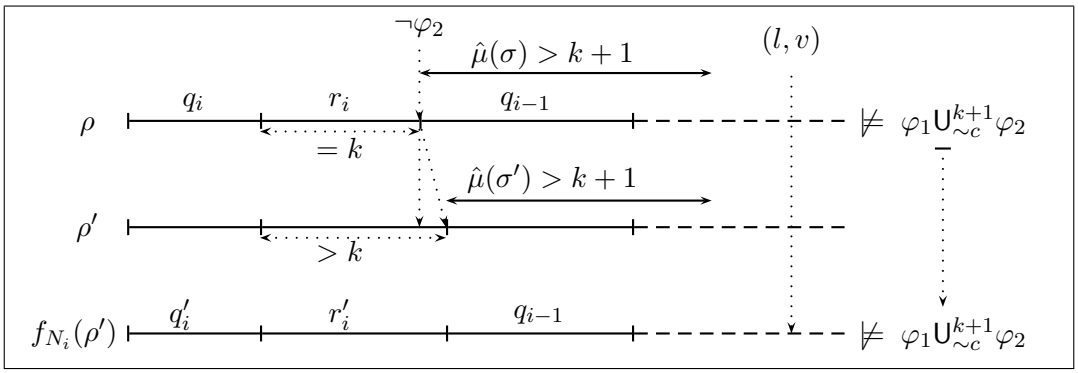


FIG. 4.18 – Construction pour le cas (c)

• Soit $(r_i, k) \models \varphi_2$ et $(r_i, k) \in I$: ce cas est similaire au précédent.

• Soit $(r_i, k) \models \varphi_2$ et $(r_i, k) \notin I$: considérons σ_{φ_2} le sous-chemin maximal qui contient (r_i, k) et qui satisfait φ_2.
⋆ Si $\hat{\mu}(\sigma_{\varphi_2}) < k+1$. On traite les mêmes sous-cas que précédemment et on les adapte ainsi : pour obtenir une exécution dans N_i qui ne satisfait pas $\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2$, à chaque sous-cas –(a), (b) et (c)– le délai ε ajouté dans r_i doit vérifier en plus $\hat{\mu}(\sigma_{\varphi_2}) + \varepsilon < k+1$. Cette condition signifie que ε est suffisamment petit pour ne pas engendrer le long de $f_{N_i}(\rho')$ un sous-chemin témoin de $\mathsf{F}^{k+1}_{\sim c}\ \varphi_2$ avant le sous-chemin "correspondant" à I.
⋆ Sinon $\hat{\mu}(\sigma_{\varphi_2}) = k+1$, nous distinguons encore des sous-cas :
– Soit $(q_{i-1}, 0) \models \neg\varphi_2$: dans ce cas, nous ne pouvons pas augmenter le délai dans r_i

et le diminuer du temps passé dans q_{i-1}, car on risque de créer le long de $f_{N_i}(\rho')$ un sous-chemin témoin de $\mathsf{F}^{k+1}_{\sim c}\ \varphi_2$, avant le correspondant de I. Comme $\hat{\mu}(\sigma_{\varphi_2}) = k+1$ et que le délai dans r_i est égal à k, les configurations (q_i, t) avec $t > 0$ vérifient φ_2. Le choix de ε pour construire ρ' dépend de l'emplacement de σ :

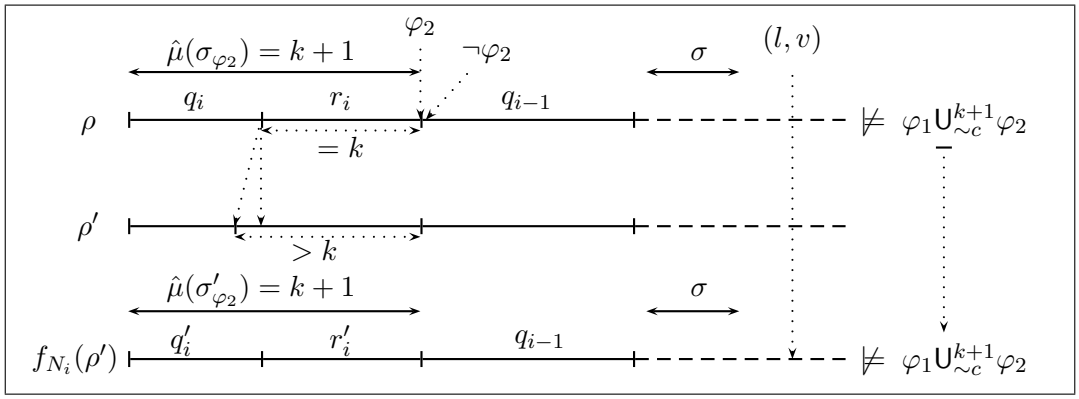


FIG. 4.19 – Construction pour le cas (i)

i Si σ se situe après (r_i, k) le long de ρ (ou commence avant (r_i, k) mais se termine après (r_i, k)). Pour obtenir ρ', nous ajoutons du temps dans r_i que nous diminuons du délai dans q_i (δ_0). Ainsi la somme du délai ajouté dans r_i et du délai dans q_i le long de ρ' est égale au temps écoulé dans q_i le long de ρ (figure 4.19). En appliquant l'hypothèse d'induction (respectivement la propriété 4.12), le correspondant de I le long de $f_{N_i}(\rho')$ est toujours le premier sous-chemin témoin de $\mathsf{F}^{k+1}_{\sim c}\ \varphi_2$ et il existe avant lui un sous-chemin σ' tel que $\hat{\mu}(\sigma') > k+1$ et $\sigma' \models \neg\varphi_1$ (σ' est le correspondant de σ).

ii Sinon σ se situe avant (r_i, k) le long de ρ (c'est-à-dire il commence et se termine avant (r_i, k)). Comme $\hat{\mu}(\sigma) > k+1$ et le délai dans r_i est égal à k, donc σ contient toutes les positions (q_i, t) avec $t > 0$. Dans ce cas, on ajoute un délai ε dans r_i que nous diminuons du temps écoulé dans q_i tel que la longueur de σ diminuée de ε reste aussi supérieure à $k+1$ ($\hat{\mu}(\sigma) - \varepsilon > k+1$) (figure 4.20).
En appliquant l'hypothèse d'induction, le correspondant de I le long de $f_{N_i}(\rho')$ est le premier sous-chemin témoin de $\mathsf{F}^{k+1}_{\sim c}\ \varphi_2$ et il existe avant lui un sous-chemin σ' tel que $\hat{\mu}(\sigma') > k+1$ et $\sigma' \models \neg\varphi_1$ (σ' est le correspondant de σ).

– Soit $(q_{i-1}, 0) \models \varphi_2$: nous avons ou bien $(q_{i-1}, t) \models \neg\varphi_2$ avec $t > 0$, donc –par la propriété 4.13– toutes les configurations (q_{i-1}, t) avec $t > 0$ vérifient $\neg\varphi_2$. Nous ajoutons alors du temps dans r_i que nous diminuons de q_i. Ce cas se traite de la même manière que le cas précédent ($(q_{i-1}, 0) \models \neg\varphi_2$).
Ou bien $(q_{i-1}, t) \models \varphi_2$ avec $t > 0$, nous augmentons le temps dans r_i par un petit délai qui sera diminué du temps écoulé dans q_{i-1}. Pour cela nous distinguons les mêmes sous-cas (a), (b) et (c) que nous traitons exactement de la même manière.

2. On adapte les constructions précédentes pour montrer la contraposée de l'implication $(r'_i, t) \models \mathsf{A}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2) \implies (r_i, t) \models \mathsf{A}(\varphi_1 \mathsf{U}^{k+1}_{\sim c} \varphi_2)$

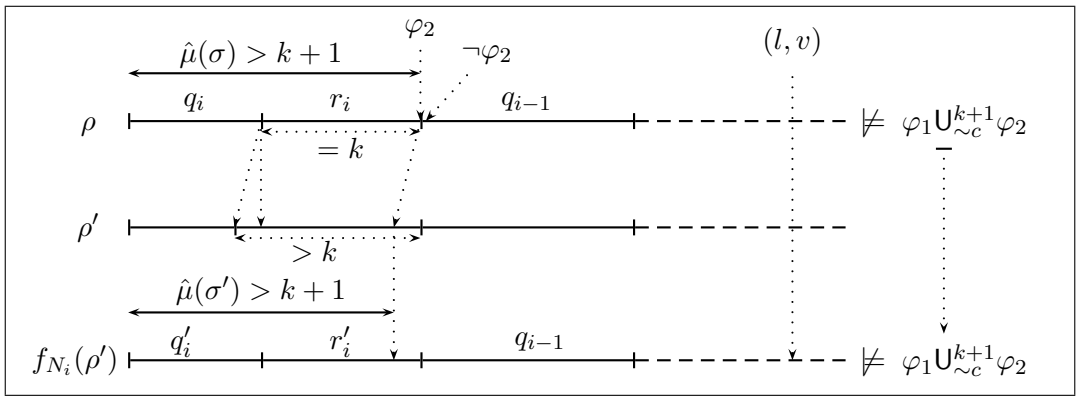


FIG. 4.20 – Construction pour le cas (ii)

Nous avons maintenons terminé la preuve de la proposition 2.

□

Nous pouvons étendre la proposition 2 à la logique TCTL^{k+1}, les preuves pour les formules $\mathsf{E\_U}_{\sim c}\mathsf{\_}$ et $\mathsf{A\_U}_{\sim c}\mathsf{\_}$ sont identiques aux preuves de la proposition 1 :

Proposition 4
Soient M_i et N_i les deux familles de la figure 4.12, $\forall n \geq 1$, $\forall i \geq n$ and $\forall t \in \mathbb{R}_+$, nous avons :

$$(q_i, t) \equiv^n_{\mathsf{TCTL}^{k+1}} (q'_i, t) \qquad (r_i, t) \equiv^n_{\mathsf{TCTL}^{k+1}} (r'_i, t)$$

4.6 Résultat d'indécidabilité pour la sémantique globale

Dans cette section, nous proposons une deuxième sémantique pour les opérateurs U^k et nous notons $\mathsf{TCTL}^{\Delta}_{\Sigma}$ la logique ainsi obtenue. La différence entre les deux sémantiques TCTL^{Δ} et $\mathsf{TCTL}^{\Delta}_{\Sigma}$ est la suivante : pour la logique TCTL^{Δ}, les durées des intervalles où la propriété est fausse doivent être bornées séparément, alors que pour la sémantique globale ($\mathsf{TCTL}^{\Delta}_{\Sigma}$), nous requérons que la somme des durées de tous les intervalles où la propriété est fausse soit bornée par une constante. La syntaxe de la logique $\mathsf{TCTL}^{\Delta}_{\Sigma}$ est la même que celle de TCTL^{Δ}, mais la sémantique de l'opérateur U^k est différente et définie ci-dessous :

$$\begin{array}{ll} \rho \models \varphi \mathsf{U}^k_{\sim c} \psi & \text{si et seulement si, } \exists\, \sigma \in SC(\rho),\ \exists p \in \sigma \text{ tel que} \\ & \mathsf{Time}(\rho^{\leq p}) \sim c \ \wedge\ \hat{\mu}(\sigma) > k \ \wedge\ \forall p' \in \sigma\ s_{p'} \models \psi \\ & \text{et } \ \hat{\mu}(\{p' \mid p' <_\rho p \ \wedge\ s_{p'} \not\models \varphi\}) \leq k \end{array}$$

Exemple 6
Considérons l'exemple de fuite du brûleur à gaz ("leaking gas burner"), utilisé souvent pour illustrer des problèmes de vérification avec les automates hybrides. Ce système est représenté par l'automate à deux états de la figure 4.21. L'état q_0 (respectivement q_1) étiqueté par la proposition atomique F (respectivement $\neg F$) représente la situation où le brûleur à gaz fuit (respectivement ne fuit pas). On suppose qu'il fuit dans la configuration initiale. Les fuites sont détectées et arrêtées après au plus une seconde et le brûleur à gaz doit garantir que le délai entre deux fuites consécutives est égal à au moins 30 secondes. La propriété usuelle

à vérifier dans le cahier de charges est la suivante : est-ce qu'après un temps écoulé d'au moins 60 secondes, le brûleur à gaz a fui pendant au plus 20% du temps total écoulé ? Cette propriété peut s'exprimer dans la logique $\mathsf{TCTL}^{\Delta}_{\Sigma}$ *par la formule :* $\mathsf{AG}(\mathsf{A}(\neg F)\mathsf{U}^{12}_{=60}\top)$*. Cette formule signifie que le long de tout intervalle de durée au moins 60 secondes, le brûleur à gaz a fui pendant moins de 12 secondes.*

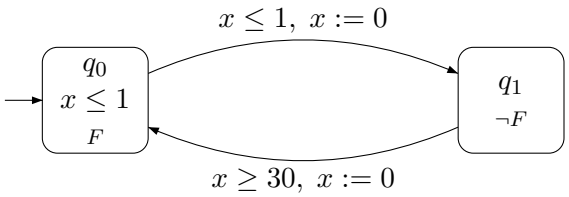


FIG. 4.21 – Le brûleur à gaz

4.6.1 Contexte général

La sémantique de $\mathsf{TCTL}^{\Delta}_{\Sigma}$ est étroitement liée aux problèmes de vérification incluant des variables hybrides. Il existe plusieurs travaux autour de ces problèmes depuis l'introduction du *Duration Calculus* (DC) dans [CRH91]. Il a été montré dans [ACH97] que le problème de l'accessibilité avec durée bornée est décidable. Cet article présente un algorithme PSPACE pour décider s'il existe une exécution (dans un automate temporisé) qui atteint un état donné avec une "durée" dans un intervalle donné I. Dans [KPJS99], il a été montré que l'accessibilité pour une sous-classe des automates hybrides CSHC (Constant Slope Hybrid Systems), formée des automates hybrides où les pentes des variables sont constantes, est indécidable. Les mêmes auteurs définissent une sous-classe des CSHC formée des graphes d'intégration (Integration graphs) où les contraintes sur les boucles ne contiennent pas de variables de pente différente de 1. Ils montrent que l'accessibilité est décidable pour cette classe [KPJS99]. Les automates temporisés étendus avec une fonction de coût ont été définis dans [ALTP01, BFH$^+$01] pour modéliser les problèmes de consommation de ressources dans les systèmes temporisés. Le problème d'accessibilité avec un coût optimal (optimal reachability) pour les automates à coût a été prouvé décidable dans ces deux travaux. La logique WCTL est définie dans [BBR04, BBM06] comme une extension de la logique TCTL, intégrant des contraintes de "coûts" sur les modalités. Cette logique permet d'exprimer par exemple qu'il est possible d'atteindre un état avec un coût borné par une constante. Le model-checking de WCTL est indécidable [BBR04, BBM06]. Les deux logiques WCTL et $\mathsf{TCTL}^{\Delta}_{\Sigma}$ n'expriment pas le même genre de propriétés mais elles sont toutes les deux des logiques "hybrides" : le coût est une variable dynamique, dont la pente varie selon les états, et dans $\mathsf{TCTL}^{\Delta}_{\Sigma}$, la somme des durées où une formule est fausse, le long d'un chemin donné, peut être vue comme la valeur d'un chronomètre (stopwatch). Notons qu'en adaptant la technique proposée dans [ACH97], on peut obtenir un algorithme pour vérifier les formules $\mathsf{E}P_1\mathsf{U}^k P_2$ et $\mathsf{E}P_1\mathsf{U}^k_{\leq c}P_2$, où $P_1 \text{et} P_2$ sont des propositions atomiques. Néanmoins ceci ne se généralise pas et nous avons le résultat d'indécidabilité suivant :

Théorème 7
Le model-checking de la logique $\mathsf{TCTL}^{\Delta}_{\Sigma}$ *sur les automates temporisés est indécidable.*

La preuve de ce théorème est donnée plus loin dans la section 4.6.3.

4.6.2 La machine à deux compteurs

Une machine à deux compteurs (ou machine de Minsky) est un ensemble fini d'instructions. Ces instructions sont étiquetées et permettent d'**incrémenter** ou de **décrémenter** deux compteurs c_1 et c_2. En notant les étiquettes ℓ_i, ℓ_j, ℓ_k, ..., les instructions sont alors de deux types :
– une instruction d'incrémentation du compteur $x \in \{c_1, c_2\}$:

$$\ell_i : x := x + 1;\ \text{goto } \ell_j$$

– une instruction de décrémentation avec test à zéro du compteur $x \in \{c_1, c_2\}$:

$$\begin{array}{lll} \ell_i : & \text{si } x > 0 & \text{alors } x := x - 1 \quad \text{goto } \ell_j \\ & \text{sinon} & \text{goto } \ell_k \end{array}$$

La machine commence une exécution par une instruction initiale définie par les deux compteurs initialisés à zéro ($c_1 = c_2 = 0$) et s'arrête lorsqu'elle arrive à une instruction spéciale étiquetée par *HALT*. Le problème de l'arrêt pour la machine à deux compteurs consiste à décider si la machine atteint l'instruction étiquetée par *HALT*. Rappelons le résultat bien connu :

Théorème 8 ([Min67])
Le problème de l'arrêt pour les machines à deux compteurs est indécidable.

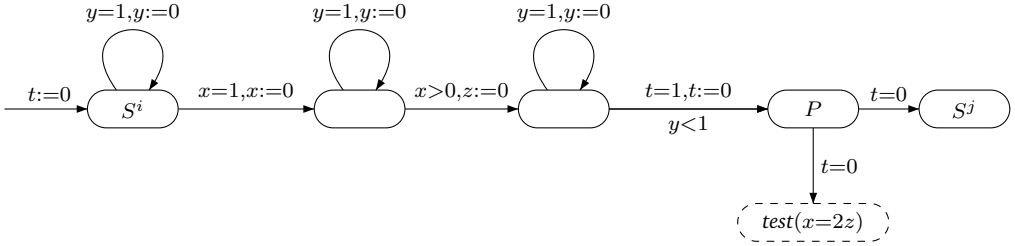
Une preuve de ce théorème peut être trouvée dans [HU79]. Elle consiste à réduire le problème de l'arrêt d'une machine de Turing au problème de l'arrêt d'une machine à deux compteurs.

4.6.3 L'indécidabilité de $\mathsf{TCTL}^{\Delta}_{\Sigma}$

Nous prouvons le théorème 7 par réduction du problème de l'arrêt d'une machine à deux compteurs au problème du model-checking d'une formule de la logique $\mathsf{TCTL}^{\Delta}_{\Sigma}$ sur un automate temporisé. Soit $\mathcal{M}$ une machine à deux compteurs, nous allons construire un automate temporisé $\mathcal{A}_{\mathcal{M}}$ avec un état initial q_{init} tel que $\mathcal{A}_{\mathcal{M}}$ simule les instructions de la machine $\mathcal{M}$. Nous définissons une formule φ de $\mathsf{TCTL}^{\Delta}_{\Sigma}$ telle que la machine $\mathcal{M}$ s'arrête si et seulement si $q_{\text{init}} \models \varphi$.
Nous utilisons quatre horloges. Trois d'entre elles (x, y et z) vont coder alternativement les deux compteurs de la machine $\mathcal{M}$, comme suit : les valeurs de c_1 et c_2 seront stockées respectivement dans $h_1 = 1/2^{c_1}$ et $h_2 = 1/2^{c_2}$ avec $h_1, h_2 \in \{x, y, z\}$. La quatrième horloge t sera utilisée pour simuler des "tics" d'horloge.

4.6.3.1 Incrémentation du compteur c_1.

Nous expliquons dans ce paragraphe comment coder l'instruction i qui incrémente le compteur c_1 puis va à l'instruction j. On considère le module de la figure suivante :

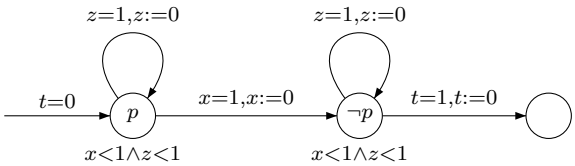


On suppose qu'au moment d'entrer dans ce module les compteurs c_1 et c_2 sont codés respectivement par les horloges x et y, c'est-à-dire que les valeurs initiales dans l'état S^i de x et y sont $x = 1/2^{c_1}$ et $y = 1/2^{c_2}$.

Étant données les contraintes d'horloges de ce module (figure précédente), les valeurs de x et y au moment d'arriver à l'état étiqueté par P (et aussi à l'état S^j) sont respectivement $1/2^{c_1}$ et $1/2^{c_2}$. En effet, avant d'arriver à l'état P, l'horloge x (respectivement l'horloge y) est remise à zéro une seule et unique fois après un délai de $(1 - 1/2^{c_1})$ (respectivement $(1 - 1/2^{c_2})$). Or le temps total écoulé entre le début du module et l'arrivée à l'état P est égal à une unité de temps.

Notons $\gamma \in [0, 1[$ la valeur de l'horloge z à l'arrivée dans l'état P. Cette valeur dépend uniquement de l'instant de franchissement de la transition étiquetée par "$x > 0, z := 0$". Le module "*test*$(x = 2z)$" que nous décrivons par la suite permet de vérifier que γ vaut la moitié de la valeur de x : nous construisons une formule de $\mathsf{TCTL}^{\Delta}_{\Sigma}$ qui est vraie dans le module "*test*$(x = 2z)$" si et seulement si $\gamma = x/2 = 1/2^{c_1+1}$. Ainsi, l'horloge z dans l'état P code correctement la valeur du compteur c_1 à la fin de l'instruction d'incrémentation (alors que la valeur du compteur c_2 est toujours correctement codée par y).

Avant de décrire le module *test*$(x = 2z)$, nous décrivons d'abord l'automate temporisé *add*(x, z, p) par la figure suivante :



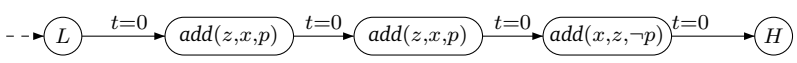
Les propriétés de cet automate sont exprimées par les deux faits suivants.

Fait 1
Si $\alpha \in [0, 1]$ et $\gamma \in [0, 1]$ sont respectivement les valeurs initiales de x et z, alors tout chemin de add(x, z, p) quittera cet automate avec ces mêmes valeurs pour x et z.

En effet, les horloges x et z sont remises à zéro une seule et unique fois quand elles atteignent la valeur 1 et le temps total passé dans le module *add*(x, z, p) est égal à une unité de temps.

Fait 2
Si $\alpha \in [0, 1]$ est la valeur initiale de x, alors le temps écoulé dans l'état étiqueté par la proposition atomique p (respectivement $\neg p$) est $(1 - \alpha)$ (respectivement α).

Nous sommes maintenant en mesure de décrire le module *test*$(x = 2z)$:



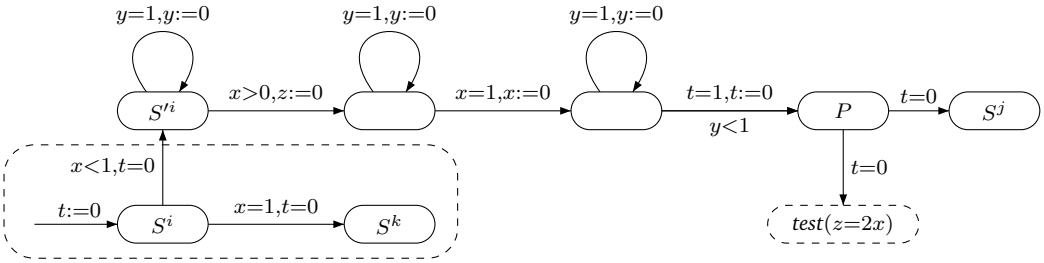
Notons que ce module possède une seule exécution qui atteint l'état étiqueté par H. Ceci est dû à la transition "$x > 0, y < 1, z := 0$" du module d'incrémentation car celle-ci impose

que les valeurs de x et z soient différentes quand on arrive à l'état P (par conséquent au module *test*$(x = 2z)$). Supposons que $\alpha \in [0,1]$ et $\gamma \in [0,1]$ soient respectivement les valeurs initiales de x et z (à l'arrivée dans l'état L). Le chemin du module *test*$(x = 2z)$ (qui arrivera à l'état étiqueté par H) va rester $2 \cdot (1-\gamma) + \alpha$ unités de temps dans les états étiquetés par la proposition atomique p et $2 \cdot \gamma + (1-\alpha)$ unités de temps dans les états étiquetés par la proposition atomique $\neg p$. Or le temps total entre les états L et H est exactement de 3 unités de temps. Par ailleurs, si la formule $\Psi = L \wedge \mathsf{E}(p\mathsf{U}^1 H) \wedge \mathsf{E}(\neg p\mathsf{U}^2 H)$ de $\mathsf{TCTL}^{\mathsf{A}}_{\Sigma}$ est satisfaite dans l'état L, alors $2 \cdot (1-\gamma) + \alpha \leq 2$ et $2 \cdot \gamma + (1-\alpha) \leq 1$, par conséquent $2 \cdot (1-\gamma) + \alpha = 2$ et $2 \cdot \gamma + (1-\alpha) = 1$, donc $\gamma = \alpha/2$.
Nous avons alors le lemme suivant :

Lemme 2
Soient $\alpha \in [0,1]$ et $\gamma \in [0,1]$ les valeurs initiales respectives de x et z dans le module test$(x = 2z)$. La formule Ψ est satisfaite dans l'état L si et seulement si $\gamma = \alpha/2$.

4.6.3.2 Décrémentation du compteur c_1.

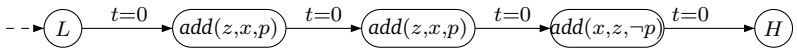
Le codage de la décrémentation est très similaire à celui de l'incrémentation. La principale différence est que le module de la décrémentation contient une partie en plus qui simule le test à zéro des compteurs. Considérons l'automate de la figure suivante :



Nous expliquons maintenant comment ce module simule l'instruction i où :

$$i: \text{ si } x > 0 \begin{cases} \text{alors} & x := x - 1; \quad \text{goto } j \\ \text{sinon} & \text{goto } k \end{cases}$$

Pour cela, nous allons décrire d'abord le module *test*$(z = 2x)$:



Ce module possède aussi une seule exécution qui atteint l'état étiqueté par H (pour les même raisons que précédemment). Supposons que $\alpha \in [0,1]$ et $\gamma \in [0,1]$ soient respectivement les valeurs initiales de x et z (à l'arrivée dans l'état L). Donc, Le chemin du module *test*$(z = 2x)$ (qui arrivera à l'état étiqueté par H) va rester $2 \cdot (1-\gamma) + \alpha$ unités de temps dans les états étiquetés par la proposition atomique p et $2 \cdot \gamma + (1-\alpha)$ unités de temps dans les états étiquetés par la proposition atomique $\neg p$. Or, le temps total entre les location L et H est exactement 3 unités de temps. Donc, si la formule $\Psi = L \wedge \mathsf{E}(p\mathsf{U}^1 H) \wedge \mathsf{E}(\neg p\mathsf{U}^2 H)$ de $\mathsf{TCTL}^{\mathsf{A}}_{\Sigma}$ est satisfaite dans l'état L, alors $\alpha = 2 \cdot \gamma$.

Lemme 3
Soient $\alpha \in [0,1]$ et $\gamma \in [0,1]$ les valeurs initiales respectives de x et z dans le module test$(z = 2x)$. La formule Ψ est satisfaite dans l'état L si et seulement si $\alpha = 2 \cdot \gamma$.

4.6.3.3 Réduction globale.

Supposons que nous avons construit, pour chaque instruction i de la machine $\mathcal{M}$, le module correspondant comme décrit précédemment avec la bonne permutation d'horloges selon le compteur qu'on incrémente ou qu'on décrémente. Nous notons les états de ces modules (à titre d'exemple) par $S^i_{\{x,y,z\}}$ pour indiquer que x (respectivement y) code le compteur c_1 (respectivement c_2) et que z code le résultat de l'instruction i. Supposons aussi que nous avons assemblé correctement ces modules pour simuler la succession des instructions de $\mathcal{M}$. Nous ajoutons ensuite un état étiqueté par S^{Halt} correspondant à l'instruction d'arrêt. L'automate global ainsi obtenu est noté $\mathcal{A}_{\mathcal{M}}$. Nous codons la configuration initiale de $\mathcal{M}$ par l'état $S^1_{\{x,y,z\}}$ et la valuation $(x,y,z,t) = (1,1,0,0)$.
Considérons maintenant la formule de $\mathsf{TCTL}^{\Delta}_{\Sigma}$ suivante :

$$\mathsf{E}\ (\xi \mathsf{U} S^{\mathsf{Halt}}) \text{ avec } \xi \stackrel{def}{=} P \Rightarrow \mathsf{E}\Big[(P \vee L)\mathsf{U}\Psi\Big]$$

Si cette formule est vérifiée dans $\mathcal{M}$ alors il existe une exécution ρ de cet automate telle que chaque état étiqueté par P satisfait $\mathsf{E}\Big[(P \vee L)\mathsf{U}\Psi\Big]$. Ceci signifie clairement que tous les modules de test attachés à ces états jouent bien leur rôle (c'est-à-dire, qu'ils permettent de vérifier les bonnes contraintes). Par conséquent, ρ simule bien une exécution de $\mathcal{M}$ qui s'arrête.

Nous obtenons donc le théorème suivant :

Théorème 9
$\mathcal{A}_{\mathcal{M}} \models \xi$ si et seulement si la machine à deux compteurs $\mathcal{M}$ s'arrête.

Ceci termine la preuve du théorème 7.

4.7 Conclusion

Cette étude a fourni une abstraction des séquences d'états de durées bornées par un seuil pendant la vérification. Cette abstraction consiste à ajouter de nouveaux opérateurs temporisés à la logique TCTL pour obtenir TCTL^{Δ}. Nous avons obtenu plusieurs résultats sur l'expressivité de cette logique. Si l'on considère une autre sémantique $\mathsf{TCTL}^{\Delta}_{\Sigma}$, le model-checking devient indécidable.
Le chapitre suivant est centré sur le model-checking de la logique TCTL^{Δ}, nous prouvons qu'il est décidable avec la même complexité que TCTL.

Chapitre 5

Model-checking

Nous présentons dans ce chapitre un algorithme de model-checking pour la logique TCTL^{Δ}. Dans un premier temps, nous introduisons une nouvelle relation d'équivalence sur les exécutions temporisées. Ensuite, nous verrons que l'ajout des modalités U^k n'augmente pas la complexité de la vérification.

5.1 Équivalence des exécutions

Le but de cette section est de montrer que la notion de région introduite par Alur et Dill dans [AD94] est correcte pour la logique TCTL^{Δ}. Plus précisément, étant donnés un automate $\mathcal{A} = \langle X, Q, q_{\text{init}}, \Sigma, \rightarrow_{\mathcal{A}}, \mathsf{Inv}, l\rangle$ et la constante maximale K associée, pour la relation d'équivalence $\cong_{X,K}$ (rappelée dans la page 20), nous prouvons le résultat suivant :

Théorème 10 (Consistence de la relation $\cong$)
Soient $\mathcal{A}$ un automate temporisé, q un état de contrôle de $\mathcal{A}$, $\Phi \in \mathsf{TCTL}^{\Delta}$ et $v, v' \in \mathbb{R}_+^X$ telles que $v \cong v'$, alors : $(q, v) \models \Phi \quad \Leftrightarrow \quad (q, v') \models \Phi$.

Pour prouver le théorème 10, nous devons montrer que, si q est un état de contrôle et si v et v' sont deux valuations équivalentes, alors pour chaque exécution ρ partant de (q, v), il existe une exécution ρ' partant de (q, v') qui lui est "quasiment identique" dans le sens suivant : $\rho \models \varphi \mathsf{U}^k_{\sim c} \psi$ si et seulement si $\rho' \models \varphi \mathsf{U}^k_{\sim c} \psi$. C'est pour cela que nous introduisons une relation d'*équivalence* sur les exécutions temporisées.

Étant donnés un automate $\mathcal{A}$ et sa constante maximale K, on considère deux exécutions $\rho = \big((q_i, v_i, t_i)\big)_{i\geq 0}$ et $\rho' = \big((q'_i, v'_i, t'_i)\big)_{i\geq 0}$. La relation $\cong$ est étendue aux paires (v_i, t_i) par :

$(v_i, t_i) \cong (v'_i, t'_i)$ si et seulement si,

(1) $v_i \cong v'_i$,

(2) $\lfloor t_i \rfloor = \lfloor t'_i \rfloor$ et $\mathsf{frac}(t_i) = 0$ si et seulement si $\mathsf{frac}(t'_i) = 0$

(3) pour toute horloge $x \in X$, (i) $\mathsf{frac}(v_i(x)) < \mathsf{frac}(t_i)$ si et seulement si $\mathsf{frac}(v'_i(x)) < \mathsf{frac}(t'_i)$ et (ii) $\mathsf{frac}(v_i(x)) = \mathsf{frac}(t_i)$ si et seulement si $\mathsf{frac}(v'_i(x)) = \mathsf{frac}(t'_i)$.

Nous définissons maintenant la relation d'*équivalence* sur les exécutions temporisées :

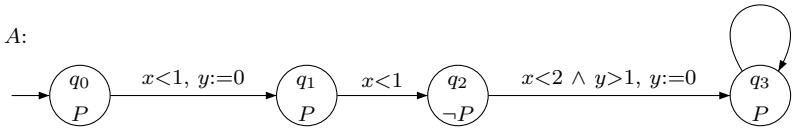
Définition 18 (Équivalence des exécutions)
Soit $\mathcal{A}$ un automate temporisé. Deux exécutions $\rho = ((q_i, v_i, t_i))_{i\geq 0}$ et $\rho' = ((q'_i, v'_i, t'_i))_{i\geq 0}$ sont équivalentes (noté $\rho \cong^* \rho'$) si

(ER a) pour tout $i \geq 0$, $q_i = q'_i$,

(ER b) pour tout $i \geq 0$, $(v_i, t_i) \cong (v'_i, t'_i)$,

(ER c) pour tout $0 \leq j < i$, (i) $\mathsf{frac}(t_j) < \mathsf{frac}(t_i)$ si et seulement si $\mathsf{frac}(t'_j) < \mathsf{frac}(t'_i)$ et (ii) $\mathsf{frac}(t_j) = \mathsf{frac}(t_i)$ si et seulement si $\mathsf{frac}(t'_j) = \mathsf{frac}(t'_i)$.

Pour prouver que les régions sont compatibles avec les formules de la logique TCTL, Alur, Courcoubetis et Dill [ACD93] utilisent une *équivalence* sur les exécutions temporisées qui requiert seulement les conditions *(ER a)* et *(ER b)*. Cette équivalence n'est pas suffisante pour la sémantique de la logique TCTL^{Δ} : en effet, considérons l'automate $\mathcal{A}$ de la figure ci-dessous où P est une proposition atomique et x et y sont deux horloges :

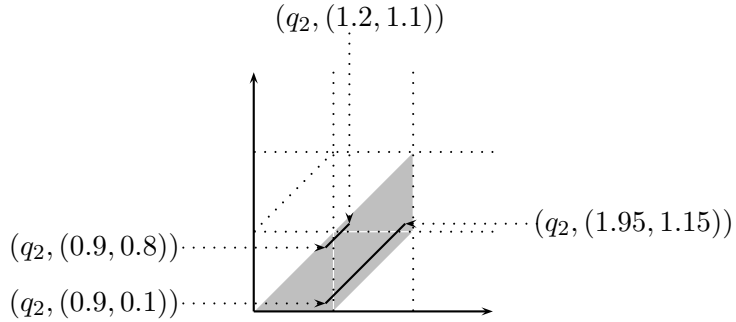


Soient ρ et ρ' les deux exécutions suivantes de $\mathcal{A}$:

$$\rho : (q_0, (0,0)) \xrightarrow{0.1}\!\!\rightarrow_a (q_1, (0.1, 0)) \xrightarrow{0.8}\!\!\rightarrow_a (q_2, (0.9, 0.8)) \xrightarrow{0.3}\!\!\rightarrow_a (q_3, (1.2, 0)) \ldots$$

$$\rho' : (q_0, (0,0)) \xrightarrow{0.8}\!\!\rightarrow_a (q_1, (0.8, 0)) \xrightarrow{0.1}\!\!\rightarrow_a (q_2, (0.9, 0.1)) \xrightarrow{1.05}\!\!\rightarrow_a (q_3, (1.95, 0)) \ldots$$

Ces exécutions satisfont les conditions *(ER a)* et *(ER b)*, donc elles sont équivalentes au sens de [ACD93], pourtant les temps passés par ρ et ρ' dans l'état q_2 étiqueté par $\neg P$ sont respectivement de 0.3 et 1.05 (voir figure suivante), par conséquent $\rho \models \mathsf{G}^1 P$ et $\rho' \not\models \mathsf{G}^1 P$.



C'est la raison pour laquelle nous avons besoin de la relation d'équivalence plus fine donnée dans la définition 18.

La proposition suivante nous garantit que, pour toute exécution partant de (q, v), il existe une exécution partant de (q, v') qui lui est équivalente.

Proposition 5
Soient $\mathcal{A}$ un automate temporisé, $q \in Q$ et $v, v' \in \mathbb{R}_+^X$ telles que $v \cong v'$. Pour toute exécution $\rho \in Exec((q, v))$, il existe une exécution $\rho' \in Exec((q, v'))$ telle que $\rho \cong^* \rho'$.

La preuve de cette proposition est donnée à la page 93. Le théorème 10 est une conséquence de ce résultat : nous montrons qu'étant données une exécution ρ de (q, v) et une formule $\Phi = \varphi \mathsf{U}^k \psi$, on peut construire une exécution ρ' de (q, v') telle que ρ et ρ' satisfont ou non Φ simultanément. Notons que nous ne montrons pas que toutes les exécutions équivalentes satisfont les mêmes formules avec les modalités U^k. Nous donnons en premier la preuve du théorème 10 :

Démonstration: La preuve repose sur une induction structurelle sur les formules de TCTL^Δ. Si Φ est une formule de TCTL, le résultat a été démontré dans [ACD93]. Supposons maintenant que $\Phi = \mathsf{E}\varphi \mathsf{U}^k_{\sim c} \psi$ et que $(q, v) \models \Phi$. Soit ρ l'exécution partant de (q, v) telle que $\rho \models \varphi \mathsf{U}^k_{\sim c} \psi$. Le long de ρ les valeurs de vérité des formules φ et ψ dépendent des régions traversées (en utilisant l'hypothèse d'induction). Si on considère maintenant l'exécution ρ' à partir de (q, v') (donnée par la proposition 5), comme ρ' franchit les mêmes régions que ρ (ceci est garanti par les conditions *(ER a)* et *(ER b)*), pour prouver le théorème il reste à montrer que le temps passé dans toute séquence de régions consécutives respectivement le long de ρ et ρ' a la même partie entière. Ainsi si $\neg\varphi$ (respectivement ψ) dure le long de ρ pendant moins (respectivement plus) de k unités de temps, c'est aussi le cas le long de ρ'.
Soit σ un sous-chemin de ρ (il correspond à une séquence de régions). On pose δ_1 la date d'arrivée dans la première région de cette séquence et δ_2 la date de départ de la dernière région de celle-ci. Étant donnée la sémantique de TCTL^Δ, on s'intéresse uniquement au sous-chemin dont les extrémités correspondent au changement de valeur de vérité d'une formule. Il suffit alors de considérer comme date d'arrivée dans une région non frontière (dite aussi région ouverte) dans le cas où l'arrivée ne se fait pas via une transition d'action, la date correspondant à la précédente région frontière. Et comme date de départ d'une région ouverte dans le cas où le départ ne se fait pas via une transition d'action, la date correspondant à la suivante région frontière. Soient δ'_1 et δ'_2 les dates correspondant le long de ρ' à δ_1 et δ_2, nous voulons alors prouver que $\lfloor \delta_2 - \delta_1 \rfloor = \lfloor \delta'_2 - \delta'_1 \rfloor$. Des conditions suffisantes pour avoir cette égalité sont :

C 1 $\lfloor \delta_i \rfloor = \lfloor \delta'_i \rfloor$ pour $i = 1, 2$,

C 2 $\mathsf{frac}(\delta_1) < \mathsf{frac}(\delta_2)$ si et seulement si $\mathsf{frac}(\delta'_1) < \mathsf{frac}(\delta'_2)$

C 3 $\mathsf{frac}(\delta_1) = \mathsf{frac}(\delta_2)$ si et seulement si $\mathsf{frac}(\delta'_1) = \mathsf{frac}(\delta'_2)$.

Ces conditions sont assurées pour ρ et ρ' (directement par la définition 18) si les dates δ_i et δ'_i où $i = 1, 2$, correspondent à des dates de transitions d'action.

Dans ce but, on construit $\bar{\mathcal{A}}$ un automate temporisé obtenu à partir de $\mathcal{A}$ en ajoutant des boucles (sans remise à zéro et sans garde) dans chaque état de contrôle. Ce nouvel automate $\bar{\mathcal{A}}$ a plus d'exécutions que l'automate $\mathcal{A}$ mais ceci n'affectera pas la valeur de vérité d'une formule $\mathsf{E\_U}^k\mathsf{\_}$ car seules des transitions d'action sans effet ont été ajoutées. Soit ρ une exécution de $\mathcal{A}$, on définit $\bar{\rho}$ comme l'exécution de $\bar{\mathcal{A}}$ qui suit ρ (i.e. elle contient les mêmes transitions que ρ) en ajoutant une transition de boucle de l'état courant avant d'arriver dans chaque nouvelle région. Il est clair que si $\rho \models \varphi \mathsf{U}^k_{\sim c} \psi$ alors $\bar{\rho} \models \varphi \mathsf{U}^k_{\sim c} \psi$. Soit $\bar{\rho'}$ l'exécution partant de (q, v') telle que $\bar{\rho} \cong^* \bar{\rho'}$. Comme les conditions *(C 1)*, *(C 2)* et *(C 3)* sont satisfaites pour les dates δ_1, δ_2, δ'_1, δ'_2 (par la définition de $\cong^*$), donc $\bar{\rho'} \models \varphi \mathsf{U}^k_{\sim c} \psi$. Par conséquent, l'exécution dans $\mathcal{A}$ qui reproduit $\bar{\rho'}$ sans les boucles satisfait $\varphi \mathsf{U}^k_{\sim c} \psi$, ainsi $(q, v') \models \Phi$.

Supposons maintenant que $\Phi = \mathsf{A}\varphi\mathsf{U}^k_{\sim c}\psi$. Nous allons montrer que $(q,v) \not\models \Phi$ si et seulement si $(q,v') \not\models \Phi$. Si $(q,v) \not\models \Phi$, il existe alors une exécution ρ de $\mathcal{A}$ telle que $\rho \not\models \varphi\mathsf{U}^k_{\sim c}\psi$. Soit $\bar{\rho}$ l'exécution dans $\bar{\mathcal{A}}$ qu'on construit à partir de ρ en ajoutant les boucles (comme expliqué ci-dessus), on a $\bar{\rho} \not\models \varphi\mathsf{U}^k_{\sim c}\psi$. Considérons $\bar{\rho}'$ une exécution de (q,v') équivalente à $\bar{\rho}$, donc $\bar{\rho}' \not\models \varphi\mathsf{U}^k_{\sim c}\psi$. Par conséquent, l'exécution dans $\mathcal{A}$ qui reproduit $\bar{\rho}'$ sans les boucles ne satisfait pas $\varphi\mathsf{U}^k_{\sim c}\psi$, ainsi $(q,v') \not\models \Phi$.

□

Nous allons maintenant établir la preuve de la proposition 5. Pour cela, nous montrons dans un premier temps qu'il existe une relation particulière comparant les parties fractionnaires des dates des transitions d'action et des valuations d'horloges le long de deux exécutions équivalentes.
On associe à une exécution $\rho = ((q_i, v_i, t_i))_{i\geq 0}$ une séquence de réels définie par $Z_i(x) = \mathsf{frac}(t_i) - \mathsf{frac}(v_i(x))$ où $i \geq 0$, pour toute horloge x avec $v_i(x) \leq K$. Cette différence a certaines propriétés par la relation $\cong^*$. Formellement on a :

Lemme 4
Soient $\rho = ((q_i, v_i, t_i))_{i\geq 0}$ et $\rho' = ((q'_i, v'_i, t'_i))_{i\geq 0}$ deux exécutions équivalentes d'un automate temporisé $\mathcal{A}$. Pour toute horloge x avec $v_i(x) \leq K$ et $v'_i(x) \leq K$ et pour tout $i \geq 0$, considérons les séquences $Z_i(x) = \mathsf{frac}(t_i) - \mathsf{frac}(v_i(x))$ et $Z'_i(x) = \mathsf{frac}(t'_i) - \mathsf{frac}(v'_i(x))$. Pour toutes les paires $(i,j) \in \mathbb{N}^2$ telles que $j \leq i$:

R 1. si $Z_i(x) \leq 0$, alors

- *a. $\mathsf{frac}(t_j) < 1 + Z_i(x)$ si et seulement si $\mathsf{frac}(t'_j) < 1 + Z'_i(x)$ et*
- *b. $\mathsf{frac}(t_j) = 1 + Z_i(x)$ si et seulement si $\mathsf{frac}(t'_j) = 1 + Z'_i(x)$*

R 2. si $Z_i(x) \geq 0$, alors

- *a. $\mathsf{frac}(t_j) < Z_i(x)$ si et seulement si $\mathsf{frac}(t'_j) < Z'_i(x)$ et*
- *b. $\mathsf{frac}(t_j) = Z_i(x)$ si et seulement si $\mathsf{frac}(t'_j) = Z'_i(x)$*

Notons que $\rho \cong^* \rho'$ implique que les valeurs $Z_i(x)$ et $Z'_i(x)$ sont soit toutes deux positives, soit toutes deux négatives, soit toutes deux égales à zéro.
Notons aussi que si $Z_i(x) = Z'_i(x) = 0$ ou si $j = i$, alors les équivalences *(R 1a)*, *(R 1b)* et *(R 2a)* sont satisfaites (en effet, les deux côtés de l'équivalence *(R 1a)* sont vrais, les deux côtés de l'équivalence *(R 2a)* sont faux et les deux côtés de l'équivalence *(R 2a)* sont faux), alors que *(R 2b)* se réduit à l'équivalence standard (i.e. l'équivalence $\cong$ sur les paires (v_i, t_i)).

Démonstration: [lemme 4] Nous décomposons cette preuve en deux parties. Dans la première, nous montrons que les relations *(R 1)* et *(R 2)* sont stables par les transitions de délai. Ceci nous permettra de conclure dans la deuxième partie par induction sur i.

Première partie. Soit ρ et ρ' deux exécutions équivalentes. Supposons qu'à la $i^{ème}$ étape, les préfixes (de ρ et ρ') jusqu'à (v_i, t_i) et (v'_i, t'_i) satisfont les relations *(R 1)* et *(R 2)*. Nous voulons montrer qu'après les transitions de délai $\xrightarrow{\delta}$ et $\xrightarrow{\delta'}$ respectivement le long de ρ et ρ', les paires $(v_i + \delta, t_i + \delta)$ et $(v'_i + \delta', t'_i + \delta')$ satisfont aussi les relations *(R 1)* et *(R 2)*. Notons que $(v_i + \delta, t_i + \delta) \cong (v'_i + \delta', t'_i + \delta')$.

Pour cela, posons

$$Z_i^{\delta}(x) = \mathsf{frac}(t_i + \delta) - \mathsf{frac}((v_i + \delta)(x)) \text{ et } Z_i'^{\delta'}(x) = \mathsf{frac}(t'_i + \delta') - \mathsf{frac}((v'_i + \delta')(x))$$

Nous avons alors à prouver que pour chaque horloge $x \in X$ telle que $(v_i + \delta)(x) \leq K$ et $(v'_i + \delta')(x) \leq K$ et pour tout $j \leq i$:

R+ 1. si $Z_i^{\delta}(x) \leq 0$, alors
   a. $\mathsf{frac}(t_j) < 1 + Z_i^{\delta}(x)$ si et seulement si $\mathsf{frac}(t'_j) < 1 + Z_i'^{\delta'}(x)$ et
   b. $\mathsf{frac}(t_j) = 1 + Z_i^{\delta}(x)$ si et seulement si $\mathsf{frac}(t'_j) = 1 + Z_i'^{\delta'}(x)$

R+ 2. si $Z_i^{\delta}(x) \geq 0$, alors
   a. $\mathsf{frac}(t_j) < Z_i^{\delta}(x)$ si et seulement si $\mathsf{frac}(t'_j) < Z_i'^{\delta'}(x)$ et
   b. $\mathsf{frac}(t_j) = Z_i^{\delta}(x)$ si et seulement si $\mathsf{frac}(t'_j) = Z_i'^{\delta'}(x)$

Soient n le nombre d'horloges et $W = (w_1, w_2, \ldots, w_{n+1})$ la séquence de réels obtenue à partir de l'ensemble $\{t_i\} \cup \{v_i(x) \mid x \in X\}$ en ordonnant les parties fractionnaires de ses éléments selon l'ordre croissant. La séquence $W' = (w'_1, w'_2, \ldots, w'_{n+1})$ est définie d'une manière similaire à partir de $\{t'_i\} \cup \{v'_i(x) \mid x \in X\}$.
Notons que si $w_j = v_i(y)$ avec $1 \leq j \leq n+1$ et $y \in X$ (respectivement $w_j = t_i$) alors $w'_j = v'_i(y)$ (respectivement $w'_j = t'_i$) (car ρ et ρ' satisfont la relation *(ER b)* à l'étape i).

Nous distinguons deux sous-cas : (1) il existe $m \in [1, n+1]$ tel que $\mathsf{frac}(w_m + \delta) = 0$, ou (2) pour tout $l \in [1, n+1]$, $\mathsf{frac}(w_l + \delta) \neq 0$.

Comme $(v_i, t_i) \cong (v'_i, t'_i)$ et $(v_i + \delta, t_i + \delta) \cong (v'_i + \delta', t'_i + \delta')$, nous avons :
– Pour le cas (1), $\exists\, \mathcal{K} \in \mathbb{N}$ tel que $\delta = \mathcal{K} - \mathsf{frac}(w_m)$, donc $\delta' = \mathcal{K} - \mathsf{frac}(w'_m)$
– Pour le cas (2),
  soit (2a) $\exists\, \mathcal{K} \in \mathbb{N}$ tel que $\delta = \mathcal{K}$, donc $\delta' = \mathcal{K}$,
  soit (2b) $\exists\, \mathcal{K} \in \mathbb{N}$, $\exists\, \varepsilon < w_1$ tel que $\delta = \mathcal{K} - \varepsilon$, donc $\delta' = \mathcal{K} - \varepsilon'$ avec $\varepsilon' < w'_1$,
  soit (2c) $\exists\, \mathcal{K} \in \mathbb{N}$, $\exists\, \varepsilon > w_{n+1}$ tel que $\delta = \mathcal{K} - \varepsilon$, donc $\delta' = \mathcal{K} - \varepsilon'$ avec $\varepsilon' > w'_{n+1}$,
  ou (2d) $\exists\, \mathcal{K} \in \mathbb{N}$, $\exists\, m \in [1, n]$ tel que $\delta = \mathcal{K} - \varepsilon$ et $\mathsf{frac}(w_m) < \varepsilon < \mathsf{frac}(w_{m+1})$, donc $\delta' = \mathcal{K} - \varepsilon'$ avec $\mathsf{frac}(w'_m) < \varepsilon' < \mathsf{frac}(w'_{m+1})$.

Les sous-cas (2a), (2b) et (2c) sont directs car ils impliquent que $Z_i^{\delta}(x) = Z_i(x)$ et $Z_i'^{\delta}(x) = Z'_i(x)$, donc *(R+ 1)* et *(R+ 2)* sont satisfaites.

Nous expliquons maintenant le cas (1). Notons d'abord que pour toute horloge x :

$$\mathsf{frac}((v_i + \delta)(x)) = \begin{cases} \mathsf{frac}(v_i(x)) - \mathsf{frac}(w_m) & \text{si } \mathsf{frac}(v_i(x)) \geq \mathsf{frac}(w_m) \\ 1 + \mathsf{frac}(v_i(x)) - \mathsf{frac}(w_m) & \text{sinon} \end{cases}$$

$$\mathsf{frac}(t_i + \delta) = \begin{cases} \mathsf{frac}(t_i) - \mathsf{frac}(w_m) & \text{si } \mathsf{frac}(t_i) \geq \mathsf{frac}(w_m) \\ 1 + \mathsf{frac}(t_i) - \mathsf{frac}(w_m) & \text{sinon} \end{cases}$$

Les valeurs $\mathsf{frac}((v'_i + \delta')(x))$ et $\mathsf{frac}(t'_i + \delta')$ sont définies d'une manière analogue. Considérons maintenant $j \leq i$. Notons que comme $(v_i, t_i) \cong (v'_i, t'_i)$, les valeurs $\mathsf{frac}(v'_i(x))$, $\mathsf{frac}(t'_i)$ et $\mathsf{frac}(w'_m)$ sont ordonnées de la même façon que leur valeurs correspondantes $\mathsf{frac}(v_i(x))$, $\mathsf{frac}(t_i)$ et $\mathsf{frac}(w_m)$. Nous avons six sous-cas possibles selon les positions respectives des valeurs $\mathsf{frac}(v_i(x))$, $\mathsf{frac}(t_i)$ et $\mathsf{frac}(w_m)$:

1. si $\mathsf{frac}(v_i(x)) < \mathsf{frac}(w_m) \leq \mathsf{frac}(t_i)$, la relation *(R 2)* est satisfaite à l'étape i (par hypothèse d'induction). De plus, on a $\mathsf{frac}((v_i+\delta)(x)) = 1 + \mathsf{frac}(v_i(x)) - \mathsf{frac}(w_m)$ et $\mathsf{frac}(t_i+\delta) = \mathsf{frac}(t_i) - \mathsf{frac}(w_m)$, donc $Z_i^\delta(x) = \mathsf{frac}(t_i+\delta) - \mathsf{frac}((v_i+\delta)(x)) = \mathsf{frac}(t_i) - \mathsf{frac}(v_i(x)) - 1$. Ceci implique que $Z_i^\delta(x) \leq 0$, nous avons donc à vérifier *(R+ 1)*. Comme $1 + Z_i^\delta(x) = \mathsf{frac}(t_i) - \mathsf{frac}(v_i(x)) = Z_i(x)$, la relation *(R+ 1)* est obtenue directement à partir de *(R 2)* à l'étape i.
2. si $\mathsf{frac}(v_i(x)) \leq \mathsf{frac}(t_i) < \mathsf{frac}(w_m)$, nous avons aussi que *(R 2)* est vraie à l'étape i par hypothèse d'induction. Dans ce cas, on a $\mathsf{frac}((v_i+\delta)(x)) = 1 + \mathsf{frac}(v_i(x)) - \mathsf{frac}(w_m)$ et $\mathsf{frac}(t_i+\delta) = 1 + \mathsf{frac}(t_i) - \mathsf{frac}(w_m)$, donc $Z_i^\delta(x) = \mathsf{frac}(t_i) - \mathsf{frac}(v_i(x)) = Z_i(x) \geq 0$. Nous avons alors à prouver *(R+ 2)* et celle-ci résulte de *(R 2)* à l'étape i.
3. si $\mathsf{frac}(t_i) \leq \mathsf{frac}(v_i(x)) < \mathsf{frac}(w_m)$, alors *(R 1)* est satisfaite à l'étape i. Comme $Z_i^\delta(x) = Z_i(x) \leq 0$, la relation *(R+ 1)* est donc satisfaite.
4. si $\mathsf{frac}(w_m) \leq \mathsf{frac}(v_i(x)) \leq \mathsf{frac}(t_i)$, alors *(R 2)* est satisfaite à l'étape i. Le fait que $Z_i^\delta(x) = Z_i(x) \geq 0$ implique directement la relation *(R+ 2)*.
5. si $\mathsf{frac}(w_m) \leq \mathsf{frac}(t_i) \leq \mathsf{frac}(v_i(x))$, alors *(R 1)* est satisfaite à l'étape i et encore $Z_i^\delta(x) = Z_i(x) \leq 0$, ce qui prouve *(R+ 1)*.
6. finalement, si $\mathsf{frac}(t_i) \leq \mathsf{frac}(w_m) \leq \mathsf{frac}(v_i(x))$, alors *(R 1)* est satisfaite à l'étape i et comme $1 + Z_i^\delta(x) = Z_i(x) \leq 0$, *(R+ 1)* est vraie.

Pour le cas (2d), notons d'abord que $\mathsf{frac}(w_m) < \mathsf{frac}(w_{m+1})$, et qu'il n'existe aucun élément de W qui se trouve strictement entre ces deux valeurs. Nous allons encore considérer des sous-cas selon les positions respectives des quatre valeurs $\mathsf{frac}(v_i(x))$, $\mathsf{frac}(t_i)$, $\mathsf{frac}(w_m)$ et $\mathsf{frac}(w_{m+1})$. Nous avons :

$$\mathsf{frac}((v_i+\delta)(x)) = \begin{cases} \mathsf{frac}(v_i(x)) + 1 - \varepsilon & \text{si } \mathsf{frac}(v_i(x)) \leq \mathsf{frac}(w_m) \\ \mathsf{frac}(v_i(x)) - \varepsilon & \text{si } \mathsf{frac}(v_i(x)) \geq \mathsf{frac}(w_{m+1}) \end{cases}$$

$$\mathsf{frac}(t_i+\delta) = \begin{cases} \mathsf{frac}(t_i) + 1 - \varepsilon & \text{si } \mathsf{frac}(t_i) \leq \mathsf{frac}(w_m) \\ \mathsf{frac}(t_i) - \varepsilon & \text{si } \mathsf{frac}(t_i) \geq \mathsf{frac}(w_{m+1}) \end{cases}$$

avec $\mathsf{frac}(w_m) < \varepsilon < \mathsf{frac}(w_{m+1})$. À partir de la remarque précédente, nous avons six sous-cas possibles pour lesquels $Z_i^\delta(x) = Z_i(x)$ ou $1 + Z_i^\delta(x) = Z_i(x)$, selon que l'ordre des parties fractionnaires correspondantes est préservé ou non après la transition de délai $\xrightarrow{\delta}$. Ces sous-cas se traitent d'une manière très similaire aux cas ci-dessus.

Deuxième partie. Nous terminons maintenant la preuve du lemme 4 par induction sur i.
Pour $i = 0$, le lemme est vrai car $t_0 = 0$ et $t'_0 = 0$. Supposons maintenant que $i > 0$ et que les relations *(R 1)* et *(R 2)* soient satisfaites jusqu'à l'étape i, nous montrons qu'elles le sont aussi à l'étape $(i+1)$.
Le cas où $j = i+1$ est trivial, nous considérons donc $j < i+1$ et une horloge x, alors :
– Soit l'horloge x a été remise à zéro pendant la $(i+1)^{ème}$ transition d'action, donc $v_{i+1}(x) = v'_{i+1}(x) = 0$, ce qui implique que $Z_{i+1}(x) = \mathsf{frac}(t_{i+1}) > 0$ et $Z'_{i+1}(x) = \mathsf{frac}(t'_{i+1}) > 0$. Comme ρ et ρ' sont équivalents, la condition *(ER c)* implique que $\mathsf{frac}(t_j) < Z_{i+1}(x)$ si et seulement si $\mathsf{frac}(t'_j) < Z'_{i+1}(x)$ et que $\mathsf{frac}(t_j) = Z_{i+1}(x)$ si et seulement si $\mathsf{frac}(t'_j) = Z'_{i+1}(x)$. Notons que ce cas ne dépend pas des valeurs des horloges.
– Sinon l'horloge x n'a pas été remise à zéro à la $(i+1)^{ème}$ transition d'action. Dans ce cas, nous pouvons conclure en utilisant la première partie de la preuve.

□

Notons que la condition *(ER c)* est nécessaire à cause des remises à zéro qui pourraient rendre faux les propriétés *(R 1)* ou *(R 2)* (d'après la deuxième partie de la preuve du lemme 4). Cette même remarque s'applique à l'exemple de la page 88 : c'est la remise à zéro de l'horloge y qui a invalidé la propriété *(R 1)*.

Nous démontrons maintenant la proposition 5.

Démonstration: [de la proposition 5] Soit ρ une exécution de $Exec((q, v))$, nous construisons l'exécution recherchée ρ' de $Exec((q, v'))$ étape par étape. Supposons que ρ' est bien construite jusqu'à l'étape i. Posons $\delta = t_{i+1} - t_i$ le délai qui sépare les deux transitions discrètes i et $(i + 1)$ de ρ. Il nous reste alors à déterminer la valeur de δ' telle que l'exécution ρ' puisse être étendue par la transition $\xrightarrow{\delta'}\rightarrow_a$, tout en préservant l'équivalence, c'est-à-dire les propriétés *(ER a)*, *(ER b)* et *(ER c)* à l'étape $(i + 1)$. Nous supposons que pour toute horloge x, $v_i(x) \leq K$ et $v'_i(x) \leq K$. Ceci n'est pas une restriction car les horloges dont les valeurs sont au delà de la constante K ne vont pas contraindre le choix de δ'. En effet, si δ' convient pour le sous-ensemble des horloges (à l'étape i) dont les valeurs sont inférieures à la plus grande constante, il convient aussi pour l'ensemble de toutes les horloges. De la même manière que dans la preuve précédente, si n est le nombre d'horloges, nous considérons la séquence $W = (w_1, w_2, \ldots, w_{n+1})$ obtenue à partir de l'ensemble $\{t_i\} \cup \{v_i(x) \mid x \in X\}$ en l'ordonnant selon l'ordre croissant des parties fractionnaires, et la séquence correspondante $W' = (w'_1, w'_2, \ldots, w'_{n+1})$ pour ρ'.

D'une manière analogue à la preuve précédente, nous avons les deux cas suivants :
– Pour le cas (1), $\exists\, \mathcal{K} \in \mathbb{N}$ tel que $\delta = \mathcal{K} - \mathsf{frac}(w_m)$, donc $\delta' = \mathcal{K} - \mathsf{frac}(w'_m)$
– Pour le cas (2), soit (2a) $\exists\, \mathcal{K} \in \mathbb{N}$ tel que $\delta = \mathcal{K}$, donc $\delta' = \mathcal{K}$,
soit (2b) $\exists\, \mathcal{K} \in \mathbb{N}$, $\exists\, \varepsilon < w_1$ tel que $\delta = \mathcal{K} - \varepsilon$, donc $\delta' = \mathcal{K} - \varepsilon'$ pour un $\varepsilon' < w'_1$,
soit (2c) $\exists\, \mathcal{K} \in \mathbb{N}$, $\exists\, \varepsilon > w_{n+1}$ tel que $\delta = \mathcal{K} - \varepsilon$, donc $\delta' = \mathcal{K} - \varepsilon'$ pour un $\varepsilon' > w'_{n+1}$,
ou (2d) $\exists\, \mathcal{K} \in \mathbb{N}$, $\exists\, m \in [1, n]$ tel que $\delta = \mathcal{K} - \varepsilon$ et $\mathsf{frac}(w_m) < \varepsilon < \mathsf{frac}(w_{m+1})$, donc $\delta' = \mathcal{K} - \varepsilon'$ pour un ε' tel que $\mathsf{frac}(w'_m) < \varepsilon' < \mathsf{frac}(w'_{m+1})$.

Rappelons que, dans tous ces cas, le choix de δ' assure la validité des conditions *(ER a)* et *(ER b)* à l'étape $(i + 1)$, si nous supposons qu'elles sont satisfaites à l'étape i ([ACD93]). Il reste alors à prouver que la condition *(ER c)* peut être maintenue elle aussi à l'étape $(i + 1)$, c'est-à-dire pour tout $j < i + 1$: $\mathsf{frac}(t_j) < \mathsf{frac}(t_{i+1})$ si et seulement si $\mathsf{frac}(t'_j) < \mathsf{frac}(t'_{i+1})$ et $\mathsf{frac}(t_j) = \mathsf{frac}(t_{i+1})$ si et seulement si $\mathsf{frac}(t'_j) = \mathsf{frac}(t'_{i+1})$. Pour cela, notons d'abord que les trois cas (2a), (2b) et (2c) sont immédiats, nous traitons maintenant les deux cas restants (1) et (2d) séparément.

Cas (1). Nous avons à prouver que la condition *(ER c)* est satisfaite à l'étape $(i + 1)$ pour la valeur fixée de δ' à $\mathcal{K} - \mathsf{frac}(w'_m)$. Nous avons la même égalité que dans la preuve du lemme 4 de la valeur de $\mathsf{frac}(t_{i+1})$:

$$\mathsf{frac}(t_{i+1}) = \begin{cases} \mathsf{frac}(t_i) - \mathsf{frac}(w_m) & \text{si } \mathsf{frac}(t_i) \geq \mathsf{frac}(w_m) \\ 1 + \mathsf{frac}(t_i) - \mathsf{frac}(w_m) & \text{sinon} \end{cases}$$

et nous définissons de manière analogue la valeur correspondante $\mathsf{frac}(t'_{i+1})$.

Nous distinguons deux sous-cas pour $w_m \in W$: soit $w_m = t_i$, soit $w_m = v_i(x)$, où x est une horloge de X. Dans le premier cas, la condition *(ER b)* implique que $w'_m = t'_i$, par conséquent $\mathsf{frac}(t_{i+1}) = \mathsf{frac}(t'_{i+1}) = 0$. La condition *(ER c)* est donc satisfaite à l'étape $i+1$. Dans le deuxième cas, comme $w'_m = v'_i(x)$, nous pouvons appliquer le lemme 4 qui donne directement la condition *(ER c)* à l'étape $i+1$.

Cas (2d). Dans ce cas, pour satisfaire la condition *(ER c)* à l'étape $(i+1)$, nous allons contraindre davantage le choix de la valeur de ε' (par rapport à [ACD93]). Nous avons :

$$\mathsf{frac}(t_{i+1}) = \begin{cases} \mathsf{frac}(t_i) - \varepsilon & \text{si } \mathsf{frac}(t_i) \geq \mathsf{frac}(w_{m+1}) \\ 1 + \mathsf{frac}(t_i) - \varepsilon & \text{si } \mathsf{frac}(t_i) \leq \mathsf{frac}(w_m) \end{cases}$$

• Supposons dans un premier temps que t_i est différente des deux valeurs w_m et w_{m+1}. Nous distinguons deux sous-cas, selon la position de $\mathsf{frac}(t_i)$ par rapport à $\mathsf{frac}(w_m)$ et $\mathsf{frac}(w_{m+1})$.

– si $\mathsf{frac}(t_i) \geq \mathsf{frac}(w_{m+1})$, considérons les deux intervalles :
$J =]\mathsf{frac}(t_i) - \mathsf{frac}(w_{m+1}), \mathsf{frac}(t_i) - \mathsf{frac}(w_m)[$ et
$J' =]\mathsf{frac}(t'_i) - \mathsf{frac}(w'_{m+1}), \mathsf{frac}(t'_i) - \mathsf{frac}(w'_m)[$.
Pour tout $j \leq i$, le lemme 4 assure que $\mathsf{frac}(t_j) \in J$ si et seulement si $\mathsf{frac}(t'_j) \in J'$.
Comme $\mathsf{frac}(w_m) < \varepsilon < \mathsf{frac}(w_{m+1})$, on a $\mathsf{frac}(t_{i+1}) \in J$.
Nous définissons :
$\alpha = max\{\mathsf{frac}(t_j)$ tel que $\mathsf{frac}(t_j) \in]\mathsf{frac}(t_i) - \mathsf{frac}(w_{m+1}), \mathsf{frac}(t_{i+1})]\}$,
$\beta = min\{\mathsf{frac}(t_j)$ tel que $\mathsf{frac}(t_j) \in [\mathsf{frac}(t_{i+1}), \mathsf{frac}(t_i) - \mathsf{frac}(w_m)[\}$,
et aussi :
$\alpha' = \mathsf{frac}(t'_h)$ avec $\alpha = \mathsf{frac}(t_h)$ et $\beta' = \mathsf{frac}(t'_\ell)$ avec $\beta = \mathsf{frac}(t_\ell)$.

Si les deux ensembles $\{\mathsf{frac}(t_j)$ tel que $j \leq i$ et $\mathsf{frac}(t_j) \in]\mathsf{frac}(t_i) - \mathsf{frac}(w_{m+1}), \mathsf{frac}(t_{i+1})]\}$ et $\{\mathsf{frac}(t_j)$ et $j \leq i$ tel que $\mathsf{frac}(t_j) \in [\mathsf{frac}(t_{i+1}), \mathsf{frac}(t_i) - \mathsf{frac}(w_m)[\}$ sont vides (par conséquent α et β ne sont pas définies), il faut choisir $\mathsf{frac}(t'_{i+1}) \in J$. Si seulement une des deux valeurs α ou β n'est pas définie, on choisit $\mathsf{frac}(t'_{i+1}) \in]\mathsf{frac}(t_i) - \mathsf{frac}(w_{m+1}), \beta'[$ ou $\mathsf{frac}(t'_{i+1}) \in]\alpha', \mathsf{frac}(t_i) - \mathsf{frac}(w_m)[$ selon la valeur manquante.

Sinon, nous avons clairement que $\alpha' \leq \beta'$ (car les éléments de W et W' sont ordonnés de la même manière) et que la condition *(ER c)* est satisfaite, quand on choisit :

1. $\mathsf{frac}(t'_{i+1}) \in]\alpha', \beta'[$ si $\mathsf{frac}(t_{i+1}) \neq \alpha$ et $\mathsf{frac}(t_{i+1}) \neq \beta$ (notons que cet intervalle est non vide car comme $\alpha < \beta$ alors $\alpha' < \beta'$),
2. $\mathsf{frac}(t'_{i+1}) = \alpha'$ si $\mathsf{frac}(t_{i+1}) = \alpha$,
3. $\mathsf{frac}(t'_{i+1}) = \beta'$ si $\mathsf{frac}(t_{i+1}) = \beta$.

Pour cela, comme $\mathsf{frac}(t'_{i+1}) = \mathsf{frac}(t'_i) - \varepsilon'$, il suffit de choisir dans le cas 1, ε' dans l'intervalle $I =]\mathsf{frac}(t'_i) - \beta', \mathsf{frac}(t'_i) - \alpha'[$, dans le cas 2, $\varepsilon' = \mathsf{frac}(t'_i) - \alpha'$ et dans le cas 3, $\varepsilon' = \mathsf{frac}(t'_i) - \beta'$. Ce choix de ε' est toujours possible, puisque le lemme 4, nous assure que l'intervalle $[\mathsf{frac}(t'_i) - \beta', \mathsf{frac}(t'_i) - \alpha']$ est strictement inclus dans l'intervalle du choix initial $]\mathsf{frac}(w'_m), \mathsf{frac}(w'_{m+1})[$.

Il existe donc un réel δ' qui permet de prolonger ρ' tout en préservant les conditions *(ER a)*, *(ER b)* et *(ER c)*.

– Sinon, $\mathsf{frac}(t_i) \leq \mathsf{frac}(w_m)$. Ce cas se traite d'une manière similaire au cas précédent ; la valeur de ε' doit être choisie comme suit :
  – $\varepsilon' \in]1 + \mathsf{frac}(t'_i) - \beta', 1 + \mathsf{frac}(t'_i) - \alpha'[$ si $\mathsf{frac}(t_{i+1}) \neq \alpha$ et $\mathsf{frac}(t_{i+1}) \neq \beta$ (cet intervalle est non vide puisque $\alpha' < \beta'$),
  – $\varepsilon' = 1 + \mathsf{frac}(t'_i) - \alpha'$ si $\mathsf{frac}(t_{i+1}) = \alpha$,
  – $\varepsilon' = 1 + \mathsf{frac}(t'_i) - \beta'$ si $\mathsf{frac}(t_{i+1}) = \beta$,

  avec
  $\alpha = max\{\mathsf{frac}(t_j)$ tel que $\mathsf{frac}(t_j) \in]\mathsf{frac}(t_i) + 1 - \mathsf{frac}(w_{m+1}), \mathsf{frac}(t_{i+1})]\}$,
  $\beta = min\{\mathsf{frac}(t_j)$ tel que $\mathsf{frac}(t_j) \in [\mathsf{frac}(t_{i+1}), \mathsf{frac}(t_i) + 1 - \mathsf{frac}(w_m)[\}$,
  $\alpha' = \mathsf{frac}(t'_h)$ tel que $\alpha = \mathsf{frac}(t_h)$ et $\beta' = \mathsf{frac}(t'_\ell)$ tel que $\beta = \mathsf{frac}(t_\ell)$.

• Supposons maintenant que $t_i = w_{m+1}$ ou $t_i = w_m$. Les deux situations sont similaires, nous supposons donc par exemple que $t_i = w_{m+1}$. Le raisonnement est similaire aux cas précédents avec, toutefois, une légère différence. Nous considérons :
$J =]0, \mathsf{frac}(t_i) - \mathsf{frac}(w_m)[$ et
$J' =]0, \mathsf{frac}(t'_i) - \mathsf{frac}(w'_m)[$.
En appliquant le lemme 4 et la condition *(ER b)*, nous avons encore
$\mathsf{frac}(t_j) \in J$ si et seulement si $\mathsf{frac}(t'_j) \in J'$.
Comme $\mathsf{frac}(w_m) < \epsilon < \mathsf{frac}(w_{m+1})$, on a alors $\mathsf{frac}(t_{i+1}) \in J$. Nous définissons :
$\alpha = max\{\mathsf{frac}(t_j)$ tel que $\mathsf{frac}(t_j) \in]0, \mathsf{frac}(t_{i+1})]\}$,
$\beta = min\{\mathsf{frac}(t_j)$ tel que $\mathsf{frac}(t_j) \in [\mathsf{frac}(t_{i+1}), \mathsf{frac}(t_i) - \mathsf{frac}(w_m)[\}$,
$\alpha' = \mathsf{frac}(t'_h)$ si $\alpha = \mathsf{frac}(t_h)$ et
$\beta' = \mathsf{frac}(t'_\ell)$ si $\beta = \mathsf{frac}(t_\ell)$.
Donc ε' doit être choisi comme suit :
– $\varepsilon' \in]\mathsf{frac}(t'_i) - \beta', \mathsf{frac}(t'_i) - \alpha'[$ si $\mathsf{frac}(t_{i+1}) \neq \alpha$ et $\mathsf{frac}(t_{i+1}) \neq \beta$ (intervalle non vide car $\alpha' < \beta'$),
– $\varepsilon' = \mathsf{frac}(t'_i) - \alpha'$ si $\mathsf{frac}(t_{i+1}) = \alpha$,
– $\varepsilon' = \mathsf{frac}(t'_i) - \beta'$ si $\mathsf{frac}(t_{i+1}) = \beta$.
Notons que le lemme 4 implique que $\beta < \mathsf{frac}(t'_i) - \mathsf{frac}(w'_m)$ et comme $w_{m+1} = t_i$, par la condition *(ER b)*, on a $\mathsf{frac}(t'_i) - \alpha = \mathsf{frac}(w'_{m+1}) - \alpha$, donc $[\mathsf{frac}(t'_i) - \beta', \mathsf{frac}(t'_i) - \alpha'] \subset]\mathsf{frac}(w'_m), \mathsf{frac}(w'_{m+1})[$ (qui est l'intervalle initial pour le choix de ε').

□

5.2 Algorithme de model-checking

Nous donnons dans cette section un algorithme pour décider si un automate temporisé $\mathcal{A}$ satisfait (ou non) une formule Φ de la logique TCTL^Δ.

La technique que nous présentons pour la décidabilité consiste à réduire le problème du model-checking $\mathcal{A} \models \Phi$ où $\mathcal{A} = \langle X, Q, q_{\text{init}}, \Sigma, \rightarrow_{\mathcal{A}}, \mathsf{Inv}, l\rangle$ est un automate temporisé et Φ une formule de TCTL^Δ, au problème du model-checking $\mathcal{A}' \models \Phi'$ où $\mathcal{A}'$ est une variante du graphe des régions et Φ' une formule de la logique CTL.

Nous introduisons trois nouvelles horloges $\{z, z_r, z_{\bar{l}}\}$ externes à l'automate $\mathcal{A}$. Celles-ci serviront à vérifier les contraintes temporisées des formules : z va traiter les contraintes $\sim c$ qui figurent dans l'indice des modalités $\mathsf{U}_{\sim c}$ et $\mathsf{U}^k_{\sim c}$ (comme pour le model-checking de TCTL, elle servira à mesurer le délai séparant deux configurations symboliques), l'horloge $z_{\bar{l}}$ (respectivement z_r) sera utilisée pour mesurer le délai pendant lequel la partie gauche (respectivement droite) d'une modalité $\mathsf{U}^k_{\sim c}$ est fausse (respectivement vraie). Considérons l'ensemble $X^* = X \cup \{z, z_r, z_{\bar{l}}\}$.

Soient Φ une formule de TCTL^{Δ}, M_Φ la constante maximale apparaissant dans les contraintes temporisées ($\sim c$) de Φ et K_m le plus grand entier k apparaissant dans les modalités U^k de Φ. Posons $K = \max(M_\mathcal{A}, M_\Phi + K_m)$ où $M_\mathcal{A}$ est la constante maximale apparaissant dans les contraintes de $\mathcal{A}$. On construit alors le graphe des régions $G_{\mathcal{A},\Phi} = (V, E)$ relatif à $\mathcal{A}$ et Φ avec X^* comme ensemble d'horloges et K comme constante maximale (voir page 21).

Une fois ce graphe construit, on étiquette les configurations (q, R) par la proposition atomique p_f lorsque R est une région frontière (voir page 20). De plus, on associe des propositions atomiques particulières aux horloges externes $\{z, z_r, z_{\bar{l}}\}$: on étiquette une configuration (q, R) par $(\!|y \sim a|\!)$ avec $y \in \{z, z_{\bar{l}}, z_r\}$ et $0 \leq a \leq K$ si et seulement si $R \models y \sim a$. Rappelons que nous ne considérons que les chemins équitables de $G_{\mathcal{A},\Phi}$ (page 22).

5.2.1 Algorithme d'étiquetage

On étiquette les configurations de $G_{\mathcal{A},\Phi}$ de manière inductive par les sous-formules de Φ : on commence par les sous-formules de longueur 1 (propositions atomiques), puis les sous-formules de longueur 2 et ainsi de suite. L'étiquetage par les formules avec les modalités $\mathsf{U}_{\sim c}$ étant identique à celui de TCTL, on ne considère ici que les formules avec $\mathsf{U}^k_{\sim c}$.

Soit Ψ une sous-formule de Φ de la forme $\mathsf{E}\varphi_l\mathsf{U}^k_{\sim c}\varphi_r$ ou $\mathsf{A}\varphi_l\mathsf{U}^k_{\sim c}\varphi_r$. Supposons qu'à cette étape de l'algorithme, nous savons pour toute configuration (q, R) de $G_{\mathcal{A},\Phi}$ si elle satisfait (ou non) φ_l et φ_r. Nous allons définir dans un premier temps le graphe $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ obtenu à partir de $G_{\mathcal{A},\Phi}$ en modifiant certaines transitions selon la valeur de vérité de φ_l et φ_r, de manière à ce que $z_{\bar{l}}$ et z_r représentent la délais décrits précédemment. Formellement :

Tr 1. nous remplaçons la transition $(q, R) \twoheadrightarrow (q, succ(R))$ par $(q, R) \twoheadrightarrow_a (q, R[z_{\bar{l}} \leftarrow 0])$ si $(q, R) \models \varphi_l$, $(q, succ(R)) \models \neg\varphi_l$ et $R \not\models z_{\bar{l}} = 0$ (figure 5.1).

Tr 2. nous remplaçons la transition $(q, R) \twoheadrightarrow_a (q', R')$ par $(q, R) \twoheadrightarrow_a (q', R'[z_{\bar{l}} \leftarrow 0])$ si $(q, R) \models \varphi_l$, $(q', R') \models \neg\varphi_l$.

Tr 3. nous remplaçons la transition $(q, R) \twoheadrightarrow (q, succ(R))$ par $(q, R) \twoheadrightarrow_a (q, R[z_r \leftarrow 0])$ si $(q, R) \models \neg\varphi_r$, $(q, succ(R)) \models \varphi_r$ et $R \not\models z_r = 0$ (figure 5.2).

Tr 4. nous remplaçons la transition $(q, R) \twoheadrightarrow_a (q', R')$ par $(q, R) \twoheadrightarrow_a (q', R'[z_r \leftarrow 0])$ si $(q, R) \models \neg\varphi_r$, $(q, R') \models \varphi_r$.

Les modifications *(Tr 1)* et *(Tr 3)* sont illustrées dans les figures 5.1 et 5.2 respectivement.

Dans le nouveau graphe $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$, les horloges $z_{\bar{l}}$ (respectivement z_r) mesurent le temps écoulé depuis que $\neg\varphi_l$ (respectivement φ_r) est vraie : elles sont remises à zéro quand la valeur de vérité de la formule correspondante passe de la valeur *"faux"* à la valeur *"vrai"*.

Étant donnés un chemin ρ dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ et une configuration (q, R) le long de ρ, on a alors :
$(q, R) \models \neg\varphi_l \wedge (\!|z_{\bar{l}} \leq k|\!)$ si et seulement si φ_l est fausse mais depuis peu de temps, c'est-à-dire s'il existe (le long de ρ) une région vérifiant φ_l "juste avant" (q, R) où "juste avant" signifie "au plus k unités de temps avant".
$(q, R) \models \varphi_r \wedge (\!|z_r > k|\!)$ si et seulement si φ_r est vraie depuis longtemps ; plus de k unités de temps.

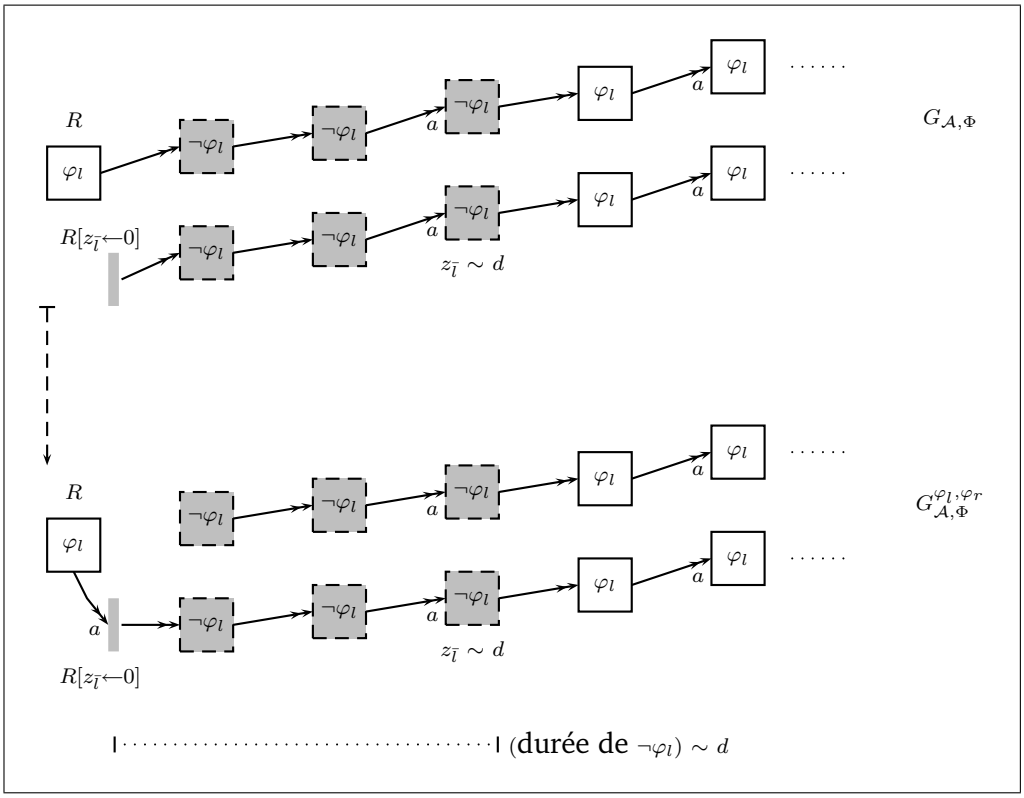


FIG. 5.1 – Construction pour le cas *(Tr 1)*

Nous définissons quelques abréviations qui seront utilisées dans la suite :

$$\overleftarrow{\varphi_l}^{\cdots} \stackrel{\text{def}}{=} \varphi_l \vee (\!|z_{\bar{l}} \leq k|\!) \qquad \overleftarrow{\varphi_r} \stackrel{\text{def}}{=} \varphi_r \wedge (\!|z_r > k|\!)$$

$$\overleftarrow{(\neg\varphi_l)} \stackrel{\text{def}}{=} \neg\varphi_l \wedge (\!|z_{\bar{l}} > k|\!) \qquad \overleftarrow{(\neg\varphi_r)}^{\cdots} \stackrel{\text{def}}{=} \neg\varphi_r \vee (\!|z_r \leq k|\!)$$

L'abréviation $\overleftarrow{\varphi_l}^{\cdots}$ signifie que ou bien φ_l est vraie ou bien elle est fausse depuis au plus k unités de temps. L'abréviation $\overleftarrow{\varphi_r}$ signifie que φ_r est vraie depuis plus de k unités de temps. Et finalement $\overleftarrow{(\neg\varphi_l)}$ signifie que $\neg\varphi_l$ dure depuis plus de k u.t.

On remarque que :

$$\begin{aligned} \neg\overleftarrow{(\neg\varphi_l)} &\equiv \neg\big(\neg\varphi_l \wedge (\!|z_{\bar{l}} > k|\!)\big) \\ &\equiv \varphi_l \vee (\!|z_{\bar{l}} \leq k|\!) \\ &\equiv \overleftarrow{\varphi_l}^{\cdots} \end{aligned} \qquad \begin{aligned} \neg\overleftarrow{(\neg\varphi_r)}^{\cdots} &\equiv \neg\big(\neg\varphi_r \vee (\!|z_r \leq k|\!)\big) \\ &\equiv \varphi_r \wedge (\!|z_r > k|\!) \\ &\equiv \overleftarrow{\varphi_r} \end{aligned}$$

Donc, on a :

$$\overset{\leftarrow}{\ddot{\varphi}} \equiv \neg\overset{\longleftarrow}{(\neg\varphi)} \text{ avec } \varphi \in \{\varphi_l, \neg\varphi_r\}$$

Par conséquent, la construction du graphe des régions $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ nous permet de décider en plus des valeurs de $\overset{\leftarrow}{\ddot{\varphi}_l}$ et $\overset{\leftarrow}{\varphi_r}$, les valeurs de $\overset{\longleftarrow}{(\neg\varphi_l)}$ et $\overset{\leftarrow\cdots\cdots}{(\neg\varphi_r)}$, grâce à l'équivalence précédente.
Nous allons maintenant distinguer plusieurs cas selon la sous-formule Ψ.

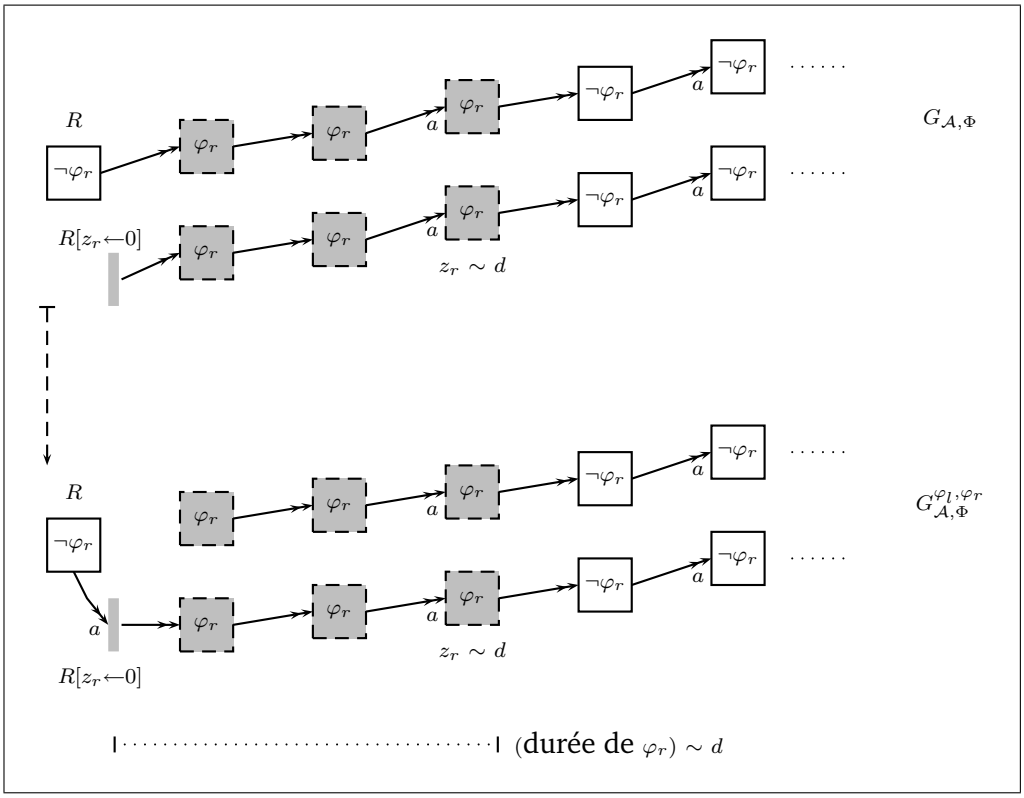


FIG. 5.2 – Construction pour le cas *(Tr 3)*

5.2.1.1 Formule $\mathsf{E}\_\mathsf{U}^k_{\sim c}\_$

Une configuration (q, R) de $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ est étiquetée par $\mathsf{E}\varphi_l\mathsf{U}^k_{\sim c}\varphi_r$ ssi $(q, R[z, z_{\overline{l}}, z_r \leftarrow 0])$ satisfait dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ la formule Ψ_1 suivante (de CTL) :

$$\Psi_1 \overset{\text{def}}{=} \mathsf{E}\ \overset{\leftarrow}{\ddot{\varphi}_l}\ \mathsf{U}\Big((\!|z \sim c|\!) \wedge (After_a \vee p_f \vee \overset{\leftarrow}{\ddot{\varphi}_l}) \wedge \mathsf{E}\ \varphi_r\ \mathsf{U}\overset{\longleftarrow}{\varphi_r}\Big)$$

Rappelons que le prédicat $After_a$, défini à la page 24, exprime que la dernière transition effectuée est une transition d'action.

Notons d'abord que Ψ_1 a la même structure que la formule utilisée pour le model-checking de TCTL :

$$\mathsf{E}\,\varphi_l\,\mathsf{U}\Big((\!|z \sim c|\!) \wedge (After_a \vee p_f \vee \varphi_l) \wedge \varphi_r\Big)$$

La formule Ψ_1 signifie qu'il existe un chemin partant de $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ qui atteint une configuration $s = (q_1, R_1)$ vérifiant
• $(\!|z \sim c|\!)$; c'est-à-dire que le temps écoulé depuis $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ jusqu'à s satisfait $\sim c$,
• $\mathsf{E}\varphi_r\,\mathsf{U}\overleftarrow{\varphi_r}$; c'est-à-dire que s se trouve dans un sous-chemin où φ_r dure plus de k unités de temps
• et soit $After_a$, p_f ou φ_l : cette condition est nécessaire pour les mêmes raisons que pour TCTL ; si $R1$ n'est pas une région frontière et si la configuration s a été atteinte via une transition de délai, la formule $\varphi_l \vee (\!|z_{\bar{l}} \leq k|\!)$ doit être satisfaite dans s.
De plus, le long de ce chemin et avant s, φ_l est soit vraie, soit fausse mais depuis moins de k unités de temps.

5.2.1.2 Formule A_U^k_

Une configuration (q, R) de $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ est étiquetée par $\mathsf{A}\varphi_l\mathsf{U}^k\varphi_r$ ssi $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ vérifie dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ la formule de la logique CTL suivante :

$$\Psi_2 = \Psi_2' \wedge \Psi_2'' \wedge \Psi_2'''$$

avec

$$\begin{array}{lcl}
\Psi_2' & \stackrel{\text{def}}{=} & \mathsf{AF}(\overleftarrow{\varphi_r}) \\
\Psi_2'' & \stackrel{\text{def}}{=} & \neg\mathsf{E}\big(\neg\,\overleftarrow{\varphi_r}\big)\mathsf{U}\Big(\overleftarrow{(\neg\varphi_l)} \wedge \neg\mathsf{A}(\varphi_r\mathsf{U}\overleftarrow{\varphi_r})\Big) \\
\Psi_2''' & \stackrel{\text{def}}{=} & \neg\mathsf{E}\big(\neg\,\overleftarrow{\varphi_r}\big)\mathsf{U}\Big(P_b \wedge \neg\varphi_r \wedge \mathsf{EX}(\neg P_b \wedge \neg\overleftarrow{(\varphi_l)})\Big)
\end{array}$$

La formule Ψ_2 signifie que :
• le long de tout chemin, inévitablement φ_r sera vraie pour au moins k unités de temps (c'est la formule $\mathsf{AF}(\overleftarrow{\varphi_r})$),
• et qu'il n'est pas possible d'avoir un chemin (à partir de $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$), où $\neg\varphi_l$ dure au moins k u.t (c'est la formule $\overleftarrow{(\neg\varphi_l)}$), à moins que, soit φ_r a duré suffisamment longtemps auparavant (au moins k u.t), soit φ_r est vraie et elle va l'être encore pendant au moins k u.t (c'est-à-dire la configuration courante se trouve dans un intervalle σ témoin de $\overleftarrow{\varphi_r}$),
• de plus si la première région de σ n'est pas une région frontière et a été atteinte par une transition de délai, elle satisfait alors $\overleftarrow{\varphi_l}$ (cette condition est donnée par Ψ_2''' pour les mêmes raisons que la modalité $E\_\mathsf{U}\_$ de TCTL).

5.2.1.3 Formule A_U$^k_{<c}$_

Pour traiter ce cas, nous avons besoin de considérer dans un premier temps la formule $\mathsf{AF}^k_{<c}\varphi_r$; plus précisément sa formule duale $\mathsf{EG}^k_{<c}$. Rappelons que la formule $\mathsf{EG}^k_{<c}\psi$ exprime qu'il existe une exécution (à partir de la configuration courante s) telle que tout sous-chemin σ de durée $\hat{\mu}(\sigma)$ strictement supérieure à k et contenant une position située à une durée inférieure strictement à c de s, doit contenir une position vérifiant ψ. Les configurations vérifiant ψ doivent donc se produire assez fréquemment "au moins toutes les k unités de temps", le

long de ρ jusqu'à $c+k$ unités de temps. Par conséquent, une configuration (q, R) est étiquetée par $\mathsf{EG}^k_{<c}\psi$ si et seulement si $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ vérifie la formule de CTL suivante :

$$\mathsf{E}\overleftarrow{\psi}\,\mathsf{U}(\!|z = c + k|\!)$$

Comme $\mathsf{AF}^k_{<c}\varphi_r = \neg\mathsf{EG}^k_{<c}\neg\varphi_r$, une configuration (q, R) est donc étiquetée par $\mathsf{AF}^k_{<c}\varphi_r$ si et seulement si la formule de CTL qui suit est satisfaite par $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$:

$$\Psi'_3 \stackrel{\text{def}}{=} \neg\mathsf{E}(\overleftarrow{\neg\varphi_r})\,\mathsf{U}\,(\!|z{=}c{+}k|\!)$$

Finalement, (q, R) est étiquetée par $\mathsf{A}\varphi_l\mathsf{U}^k_{<c}\varphi_r$ si et seulement si $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ de $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ vérifie

$$\Psi_3 = \Psi'_3 \wedge \Psi''_2 \wedge \Psi'''_2$$

Cette formule est analogue à Ψ_2, elle s'explique donc de la même façon. La seule différence est que φ_r doit se produire **avant** c unités de temps et durer pendant suffisamment longtemps "au moins k u.t".

5.2.1.4 Formule $\mathsf{A\_U}^k_{\leq c}\mathsf{\_}$

Ce cas est très similaire au précédent. Nous considérons d'abord la formule :

$$\Psi'_4 \stackrel{\text{def}}{=} \neg\mathsf{E}(\overleftarrow{\neg\varphi_r})\,\mathsf{U}\,(\!|z{>}c{+}k|\!)$$

Une configuration (q, R) est étiquetée par $\mathsf{AF}^k_{\leq c}\varphi_r$ si et seulement si $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ satisfait Ψ'_4. Cette formule est analogue à Ψ'_3 et elle prend en compte les configurations situées à une durée égale à c de la configuration initiale. Finalement, (q, R) est étiquetée par $\mathsf{A}\varphi_l\mathsf{U}^k_{\leq c}\varphi_r$ si et seulement si $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ de $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ satisfait

$$\Psi_4 = \Psi'_4 \wedge \Psi''_2 \wedge \Psi'''_2$$

5.2.1.5 Formule $\mathsf{A\_U}^k_{\geq c}\mathsf{\_}$

Une configuration (q, R) est étiquetée par $\mathsf{A}\varphi_l\mathsf{U}^k_{\geq c}\varphi_r$ si et seulement si $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ de $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ vérifie la formule (de CTL) suivante :

$$\Psi_5 \stackrel{\text{def}}{=} \mathsf{A}\overleftarrow{\varphi_l}\,\mathsf{U}\,\left(\!|z{=}c|\!\right) \wedge \mathsf{AF}(\overleftarrow{\varphi_r}) \wedge \Psi''_2 \wedge \Psi'''_2\big)$$

où Ψ''_2 et Ψ'''_2 sont les formules déjà utilisées au 5.2.1.2. La formule Ψ_5 signifie que tout chemin partant de $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ atteint une configuration s satisfaisant $(\!|z{=}c|\!)$ où :
• $\mathsf{AF}(\overleftarrow{\varphi_r})$ est vraie ; c'est-à-dire qu'inévitablement φ_r est vraie sur un sous-chemin suffisamment long (au moins k unités de temps) contenant une position située à une durée supérieure ou égale à c de la configuration initiale,
• Ψ''_2 est vraie ; c'est-à-dire qu'il n'est pas possible d'avoir un sous-chemin après la configuration s où $\neg\varphi_l$ dure longtemps "au moins k u.t.", à moins que, soit φ_r a duré suffisamment longtemps auparavant (et après s), soit φ_r est vraie et elle va l'être pendant au moins k u.t, De plus le long de ce chemin, avant s, φ_l est soit vraie, soit fausse depuis moins de k u.t.
• Ψ'''_2 est vraie ; c'est-à-dire la première région de l'intervalle témoin de $\overleftarrow{\varphi_r}$ satisfait $\overleftarrow{\varphi_l}$ à moins d'être une région frontière ou qu'elle soit atteinte par une transition d'action.

5.2.1.6 Formule $\mathsf{A\_U}^k_{>c}\mathsf{\_}$

Une configuration (q, R) est étiquetée par $\mathsf{A}\varphi_l\mathsf{U}^k_{>c}\varphi_r$ si et seulement si $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ de $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ satisfait la formule (de CTL) suivante :

$$\Psi_6 \stackrel{\text{def}}{=} \mathsf{A}\big(\llparenthesis z \leq c \rrparenthesis \wedge \overleftarrow{\varphi_l}\big) \,\mathsf{U}\, \big(\llparenthesis z > c \rrparenthesis \wedge \mathsf{AF}(\overleftarrow{\varphi_r}) \wedge \Psi_2'' \wedge \Psi_2'''\big)$$

La formule Ψ_6 a la même structure que Ψ_5 et elle s'explique de la même manière. La seule différence est que $\overleftarrow{\varphi_r}$ doit être vraie dans une position située à une durée strictement supérieure à c ($\llparenthesis z > c \rrparenthesis \wedge \mathsf{AF}(\overleftarrow{\varphi_r})$). Ainsi, le sous-chemin où φ_r dure "suffisamment longtemps" (tel que $\neg\varphi_l$ ne dure pas "trop longtemps" avant) contient nécessairement une position située strictement après c.

5.2.1.7 Formule $\mathsf{A\_U}^k_{=c}\mathsf{\_}$

Dans ce cas, nous utilisons les équivalences 4.6 et 4.7 de la page 58. Ainsi nous réduisons le problème du model-checking de $\mathsf{A}\varphi_l\mathsf{U}^k_{=c}\varphi_r$ à celui de $\mathsf{AF}^k_{=c}\varphi_r$ et $\mathsf{AG}^k_{<c-k}\varphi_l$. Pour $\mathsf{AG}^k_{\sim c}$, il suffit d'utiliser la procédure pour $\mathsf{EF}^k_{\sim c}$. Il reste alors à donner la procédure d'étiquetage de $\mathsf{AF}^k_{=c}\varphi_r$. Pour cela, nous considérons dans un premier temps sa formule duale $\mathsf{EG}^k_{=c}\psi$: cette formule exprime qu'il existe une exécution partant de s (la configuration courante) telle que tout sous-chemin σ de mesure strictement supérieure k et contenant une configuration localisée à une durée égale à c de s, doit contenir une configuration vérifiant ψ. Donc, ψ est satisfaite ou vient d'être satisfaite "récemment" (c'est-à-dire moins de k u.t avant) dans toutes les configurations localisées à une durée d de s avec $d \in [c, c+k]$. Donc, nous utilisons la formule (de CTL) suivante :

$$\mathsf{E}\llparenthesis z < c \rrparenthesis \mathsf{U}(\llparenthesis z = c \rrparenthesis \wedge \mathsf{E}\overleftarrow{\psi}\mathsf{U}\llparenthesis z > c + k \rrparenthesis$$

Par conséquent, (q, R) est étiquetée par $\mathsf{AF}^k_{=c}\varphi_r$ si et seulement si $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ satisfait dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ la formule (de CTL) qui suit :

$$\Psi_7' \stackrel{\text{def}}{=} \neg\mathsf{E}\,\llparenthesis z < c \rrparenthesis\, \mathsf{U}\, \Big(\llparenthesis z = c \rrparenthesis \wedge \mathsf{E}\,\overleftarrow{(\neg\varphi_r)}\,\mathsf{U}\,\llparenthesis z > c+k \rrparenthesis\Big)$$

Nous récapitulons cette procédure d'étiquetage dans le tableau suivant. La correction de l'algorithme sera prouvée dans la section suivante.

$\mathsf{E}\varphi_l\mathsf{U}^k_{\sim c}\varphi_r$	$\mathsf{E}\,\overleftarrow{\varphi_l}^{\shortmid}\,\mathsf{U}\Big((\!\|z\sim c\|\!) \wedge (After_a \vee p_f \vee \overleftarrow{\varphi_l}^{\shortmid}) \wedge \mathsf{E}\,\varphi_r\,\mathsf{U}\overleftarrow{\varphi_r}^{\shortmid}\Big)$
$\mathsf{A}\varphi_l\mathsf{U}^k\varphi_r$	$\mathsf{AF}(\overleftarrow{\varphi_r}^{\shortmid}) \wedge \underbrace{\neg\mathsf{E}\Big(\neg\,\overleftarrow{\varphi_r}^{\shortmid}\Big)\mathsf{U}\Big(\overleftarrow{(\neg\varphi_l)}^{\shortmid} \wedge \neg\mathsf{A}(\varphi_r\mathsf{U}\overleftarrow{\varphi_r}^{\shortmid})\Big)}_{\Psi''_2} \wedge$ $\underbrace{\neg\mathsf{E}(\neg\,\overleftarrow{\varphi_r}^{\shortmid})\mathsf{U}\Big(P_b \wedge \neg\varphi_r \wedge \mathsf{EX}(\neg P_b \wedge \neg\overleftarrow{(\varphi_l)}^{\cdots\shortmid})\Big)}_{\Psi'''_2}$
$\mathsf{A}\varphi_l\mathsf{U}^k_{<c}\varphi_r$	$\big(\neg\mathsf{E}\overleftarrow{(\neg\varphi_r)}^{\cdots\shortmid}\,\mathsf{U}\,(\!\|z=c+k\|\!)\big) \wedge \Psi''_2 \wedge \Psi'''_2$
$\mathsf{A}\varphi_l\mathsf{U}^k_{\leq c}\varphi_r$	$\big(\neg\mathsf{E}\overleftarrow{(\neg\varphi_r)}^{\cdots\shortmid}\,\mathsf{U}\,(\!\|z>c+k\|\!)\big) \wedge \Psi''_2 \wedge \Psi'''_2$
$\mathsf{A}\varphi_l\mathsf{U}^k_{\geq c}\varphi_r$	$\mathsf{A}\overleftarrow{\varphi_l}^{\shortmid}\,\mathsf{U}\,\big((\!\|z=c\|\!) \wedge \mathsf{AF}(\overleftarrow{\varphi_r}^{\shortmid}) \wedge \Psi''_2 \wedge \Psi'''_2\big)$
$\mathsf{A}\varphi_l\mathsf{U}^k_{>c}\varphi_r$	$\mathsf{A}\big((\!\|z\leq c\|\!) \wedge \overleftarrow{\varphi_l}^{\shortmid}\big)\,\mathsf{U}\,\big((\!\|z>c\|\!) \wedge \mathsf{AF}(\overleftarrow{\varphi_r}^{\shortmid}) \wedge \Psi''_2 \wedge \Psi'''_2\big)$
$\mathsf{AF}^k_{=c}\varphi_r$	$\neg\mathsf{E}\,(\!\|z<c\|\!)\,\mathsf{U}\,\Big((\!\|z=c\|\!) \wedge \mathsf{E}\,\overleftarrow{(\neg\varphi_r)}^{\cdots\shortmid}\,\mathsf{U}\,(\!\|z>c+k\|\!)\Big)$

5.3 Correction de l'algorithme d'étiquetage

Le but de cette section est de prouver que l'algorithme d'étiquetage pour le model-checking de TCTL^Δ est correct, formellement :

Lemme 5
Soient $\mathcal{A}$ un automate temporisé, Φ une formule de TCTL^Δ et Ψ une sous-formule de Φ, (q,R) est étiquetée avec Ψ dans $G_{\mathcal{A},\Phi}$ si et seulement si $(q,v) \models \Psi$ pour tout $v \in R$.

Démonstration: La preuve procède par induction structurelle sur Ψ. Les cas où $\Psi \in \mathsf{TCTL}$ ont été prouvés dans [ACD93]. Nous traitons seulement les cas où $\Psi = \mathsf{E}\varphi_l\mathsf{U}^k_{\sim c}\varphi_r$ ou $\Psi = \mathsf{A}\varphi_l\mathsf{U}^k_{\sim c}\varphi_r$.

– $\Psi = \mathsf{E}\varphi_l\mathsf{U}^k_{\sim c}\varphi_r$. Soit R une région, il faut montrer que :

$$(q, R[z, z_{\bar{l}}, z_r \leftarrow 0]) \models \mathsf{E}\,\overleftarrow{\varphi_l}^{\shortmid}\,\mathsf{U}\Big((\!\|z\sim c\|\!) \wedge (After_a \vee p_f \vee \overleftarrow{\varphi_l}^{\shortmid}) \wedge \mathsf{E}\,\varphi_r\,\mathsf{U}\overleftarrow{\varphi_r}^{\shortmid}\Big) \Leftrightarrow (q,v) \models \Psi$$

pour tout $v \in R$.

$\Leftarrow$ Supposons que pour tout $v \in R$, $(q,v) \models \Psi$. Soient ρ une exécution partant de (q,v) (avec $v \in R$) telle que $\rho \models \varphi_l\mathsf{U}^k_{\sim c}\varphi_r$, et $\bar{\rho}$ son chemin correspondant dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ partant de $(q, R[z, z^l, z^r \leftarrow 0])$. Par construction $\bar{\rho}$ atteint une région (q_n, R_n) qui satisfait $(\!\|z\sim c\|\!)$ et qui se trouve dans un sous-chemin de longueur au moins k unités de temps où φ_r est vraie, donc $(q_n, R_n) \models (\!\|z\sim c\|\!) \wedge \mathsf{E}\,\varphi_r\,\mathsf{U}\overleftarrow{\varphi_r}^{\shortmid}$. De plus, l'exécution ρ ne contient aucun sous-chemin où $\neg\varphi_l$ dure au moins k u.t (avant que φ_r ne dure assez longtemps autour d'une position située à une durée qui vérifie $\sim c$). Par conséquent $\bar{\rho}$ ne doit jamais traverser une configuration satisfaisant $\neg\varphi_l \wedge (\!\|z_{\bar{l}} > k\|\!)$. Ainsi toutes les configurations avant (q_n, R_n) vérifient $\overleftarrow{\varphi_l}^{\shortmid}$. Par ailleurs, si (q_n, R_n) n'est pas une région frontière et ne vérifie pas $After_a$, elle doit vérifier $\overleftarrow{\varphi_l}^{\shortmid}$.

$\Rightarrow$ Supposons que $(q, R[z, z_{\bar{l}}, z_r \leftarrow 0])$ vérifie dans $G^{\varphi,\psi}_{\mathcal{A},\Phi}$ la formule de CTL :

$$\Psi_1 \stackrel{\text{def}}{=} \mathsf{E}\, \overleftarrow{\varphi_l}^{\cdot|} \,\mathsf{U}\, \underbrace{\Big((\!|z \sim c|\!) \wedge (After_a \vee p_f \vee \overleftarrow{\varphi_l}^{\cdot|}) \wedge \mathsf{E}\, \varphi_r \,\mathsf{U} \overleftarrow{\varphi_r} \Big)}_{\Psi_1''}$$

Il existe donc un chemin $\bar{\rho}$ dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ qui atteint une configuration (q_n, R_n) vérifiant Ψ_1''. Soient $v \in R$ et ρ un chemin (dans $\mathcal{A}$) correspondant à $\bar{\rho}$ à partir de (q, v). Ce chemin ρ atteint donc une position p telle que $p = (q_n, v_n)$ et $v_n \in R_n$. Comme $(q_n, R_n) \models \Psi_1''$, la position p est alors située dans un sous-chemin où φ_r dure au moins k avec $\mathsf{Time}(\rho^{\leq p}) \sim c$ (car $\bar{\rho}$ commence avec l'horloge z égale à zéro). Comme toutes les configurations (le long de $\bar{\rho}$) avant (q_n, R_n) satisfont $\overleftarrow{\varphi_l}^{\cdot|}$, c'est-à-dire $\varphi_l \vee (\!|z_{\bar{l}} \leq k|\!)$, la durée de tous les sous-chemins avant p (le long de ρ) où $\neg\varphi_l$ est vraie est inférieure ou égale à k u.t, par conséquent $\rho \models \varphi_l \mathsf{U}^k_{\sim c} \varphi_r$.

– $\Psi = \mathsf{A}\varphi_l \mathsf{U}^k \varphi_r$. Soit R une région, nous avons à montrer que :

$$(q, R[z, z_{\bar{l}}, z_r \leftarrow 0]) \models \mathsf{AF}(\overleftarrow{\varphi_r}) \wedge \Psi_2'' \wedge \Psi_2''' \Leftrightarrow (q, v) \models \Psi$$

pour tout $v \in R$, où

$$\Psi_2'' = \neg\mathsf{E}\Big(\neg\, \overleftarrow{\varphi_r}\Big)\mathsf{U}\Big(\overleftarrow{(\neg\varphi_l)} \wedge \neg\mathsf{A}(\varphi_r \mathsf{U} \overleftarrow{\varphi_r})\Big)$$

$$\Psi_2''' = \neg\mathsf{E}\big(\neg\, \overleftarrow{\varphi_r}\big)\mathsf{U}\Big(P_b \wedge \neg\varphi_r \wedge \mathsf{EX}(\neg P_b \wedge \neg\overleftarrow{(\varphi_l)}^{\cdots\cdot|})\Big)$$

$\Leftarrow$ Supposons que pour toutes valuations de R, $(q, v) \models \Psi$. Soient $\bar{\rho}$ un chemin dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ partant de $(q, R[z, z_l, z_r \leftarrow 0])$, et ρ une exécution qui correspond à $\bar{\rho}$ dans $\mathcal{A}$ partant de (q, v) (avec $v \in R$). Comme $\rho \models \varphi_l \mathsf{U}^k \varphi_r$ donc $\bar{\rho}$ satisfait $\mathsf{F}(\overleftarrow{\varphi_r})$.
Supposons maintenant que $(q, R[z, z_l, z_r \leftarrow 0])$ ne satisfait pas Ψ_2''. Il existe donc $\bar{\rho}$ un chemin de $(q, R[z, z_l, z_r \leftarrow 0])$ et une configuration (q', R') avant toutes les configurations vérifiant $\overleftarrow{\varphi_r}$, tels que $(q', R') \models \overleftarrow{(\neg\varphi_l)} \wedge \neg\mathsf{A}(\varphi_r \mathsf{U} \overleftarrow{\varphi_r})$. Notons $\bar{\rho}_{|_{(q',R')}}$ le préfixe de $\bar{\rho}$ jusqu'à la position correspondante à (q', R'). Soit $\bar{\rho'}$ un chemin (dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$) à partir de (q', R') tel que $\bar{\rho'} \models \neg(\varphi_r \mathsf{U} \overleftarrow{\varphi_r})$. Considérons une exécution ρ' dans $\mathcal{A}$ partant de (q, v) et correspondant au chemin $\bar{\rho}_{|_{(q',R')}} . \bar{\rho'}$. Clairement, ρ' ne satisfait pas $\varphi_l \mathsf{U}^k \varphi_r$. En effet, comme $(q', R') \models \overleftarrow{(\neg\varphi_l)}$, il existe le long de ρ' un sous-chemin où $\neg\varphi_l$ dure au moins k u.t avant que φ_r ne dure suffisamment longtemps (c'est-à-dire strictement plus de k unités de temps). On en déduit que $(q, R[z, z_l, z_r \leftarrow 0]) \models \Psi_2''$.
Supposons maintenant que $(q, R[z, z_l, z_r \leftarrow 0])$ satisfait $\neg\Psi_2'''$. Soient $\bar{\rho}$ un chemin partant de $(q, R[z, z_l, z_r \leftarrow 0])$ qui satisfait $\mathsf{E}\big(\neg\, \overleftarrow{\varphi_r}\big)\mathsf{U}\Big(P_b \wedge \neg\varphi_r \wedge \mathsf{EX}(\neg P_b \wedge \neg\overleftarrow{(\varphi_l)}^{\cdots\cdot|})\Big)$, (q', R') la configuration qui vérifie la partie droite du U et (q'', R'') sa configuration successeur qui satisfait $\neg P_b \wedge \neg\overleftarrow{(\varphi_l)}^{\cdots\cdot|}$ (le long de $\bar{\rho}$). Soit ρ l'exécution partant de (q, v) avec $v \in R$ qui correspond à $\bar{\rho}$, elle contient alors un intervalle σ témoin de $\overleftarrow{\varphi_r}$. Le long de $\bar{\rho}$, l'intervalle σ ne peut pas être avant (q', R'), il est donc après (q'', R'') (ou commence par (q'', R'')). La transition qui va de (q', R') à (q'', R'') étant une transition de délai (en effet, la valeur de P_b passe de $\top$ à $\bot$), toute position p de σ est alors précédée par un état de la région ouverte (q'', R''). Comme (q'', R'') satisfait $\neg\overleftarrow{(\varphi_l)}^{\cdots\cdot|}$, l'exécution ρ ne satisfait pas $\varphi_l \mathsf{U}^k \varphi_r$. Ceci donne une contradiction.
$\Rightarrow$ Réciproquement, supposons $(q, R[z, z_l, z_r \leftarrow 0])$ vérifie la formule de CTL suivante :

$$\Psi_2 \stackrel{\text{def}}{=} \underbrace{\mathsf{AF}(\overleftarrow{\varphi_r^{\dashv}})}_{\Psi_2'} \wedge \underbrace{\neg\mathsf{E}\Big(\neg\overleftarrow{\varphi_r^{\dashv}}\mathsf{U}\big(\overleftarrow{(\neg\varphi_l)^{\dashv}} \wedge \neg\mathsf{A}(\varphi_r\mathsf{U}\overleftarrow{\varphi_r^{\dashv}})\big)\Big)}_{\Psi_2''} \wedge \Psi_2'''$$

Soient ρ une exécution (dans $\mathcal{A}$) qui commence à (q, v) où v est une valuation de R et $\bar{\rho}$ le chemin correspondant dans $G_{\mathcal{A},\Phi}^{\varphi_l,\varphi_r}$ partant de $(q, R[z, z_l, z_r \leftarrow 0])$. Comme Ψ_2' est satisfaite par $(q, R[z, z_l, z_r \leftarrow 0])$, il existe alors le long de ρ un sous-chemin σ où φ_r dure au moins k u.t. D'un autre côté, la formule Ψ_2'' atteste qu'il n'est pas possible que $\neg\varphi_l$ dure longtemps (plus de k unités de temps), avant le sous-chemin σ, à moins d'être en même temps dans un sous-chemin témoin de $\overleftarrow{\varphi_r^{\dashv}}$. Dans ce cas, il reste à vérifier que la première région de ce sous-chemin satisfait $\overleftarrow{\varphi_l^{\dashv}}$ sauf si c'est une région frontière ou elle a été atteinte par une transition d'action : ceci est donné par la formule Ψ_2'''.

– $\Psi = \mathsf{A}\varphi_l\mathsf{U}^k_{<c}\varphi_r$. Soit R une région, nous montrons que :

$$(q, R[z, z_{\bar{l}}, z_r \leftarrow 0]) \models \big(\neg\mathsf{E}\overleftarrow{(\neg\varphi_r)^{\dashv}}\ \mathsf{U}\ (\!|z{=}c{+}k|\!)\big) \wedge \Psi_2'' \wedge \Psi_2''' \Leftrightarrow (q, v) \models \Psi$$

pour tout $v \in R$.

$\Leftarrow$ Supposons que pour tout $v \in R$, $(q, v) \models \Psi$. Soient $\bar{\rho}$ un chemin dans $G_{\mathcal{A},\Phi}^{\varphi_l,\varphi_r}$ partant de $(q, R[z, z_l, z_r \leftarrow 0])$, et ρ une exécution correspondant à $\bar{\rho}$ dans $\mathcal{A}$ à partir de (q, v) (avec $v \in R$). Comme $\rho \models \varphi_l\mathsf{U}^k_{<c}\varphi_r$, il existe donc (le long de ρ) un sous-chemin contenant une position située strictement avant c u.t de (q, v) où φ_r dure au moins k u.t. Ceci implique que $\bar{\rho}$ contient une configuration vérifiant $\overleftarrow{\varphi_r^{\dashv}}$ avant d'atteindre une configuration située à une durée égale à $c + k$ de l'origine de $\bar{\rho}$, donc $\bar{\rho} \models \neg(\overleftarrow{(\neg\varphi_r)^{\dashv}}\ \mathsf{U}\ (\!|z = c{+}k|\!))$ car $(\neg\overleftarrow{(\neg\varphi_r)^{\dashv}} = \overleftarrow{\varphi_r^{\dashv}})$. Supposons maintenant que $(q, R[z, z_l, z_r \leftarrow 0])$ ne satisfait pas Ψ_2''. Il existe donc un chemin $\bar{\rho}$ et une configuration (q', R') (le long de $\bar{\rho}$) avant toutes les configurations vérifiant $\overleftarrow{\varphi_r^{\dashv}}$, telle que $(q', R') \models \overleftarrow{(\neg\varphi_l)^{\dashv}} \wedge \neg\mathsf{A}(\varphi_r\mathsf{U}\overleftarrow{\varphi_r^{\dashv}})$. Notons $\bar{\rho}_{|_{(q',R')}}$ le préfixe de $\bar{\rho}$ jusqu'à la position qui correspond à cette configuration (q', R'). Soit $\bar{\rho'}$ un chemin (dans $G_{\mathcal{A},\Phi}^{\varphi_l,\varphi_r}$) à partir de (q', R') tel que $\bar{\rho'} \models \neg(\varphi_r\mathsf{U}\overleftarrow{\varphi_r^{\dashv}})$. Considérons ρ' une exécution dans $\mathcal{A}$ partant de (q, v) et correspondant au chemin $\bar{\rho}_{|_{(q',R')}}.\bar{\rho'}$. Clairement, ρ' ne satisfait pas $\varphi_l\mathsf{U}^k_{<c}\varphi_r$. En effet, comme $(q', R') \models \overleftarrow{(\neg\varphi_l)^{\dashv}}$, il existe le long de ρ' un sous-chemin où $\neg\varphi_l$ dure au moins k u.t avant que φ_r ne dure suffisamment longtemps (le long d'un sous-chemin contenant une position située à une durée inférieure strictement à c u.t). supposons maintenant que $(q, R[z, z_l, z_r \leftarrow 0])$ satisfait $\neg\Psi_2'''$, pour les mêmes raisons que la précédente preuve on obtient une contradiction avec le fait que $(q, v) \models \Psi$ pour toute valuation v de R.

$\Rightarrow$ Réciproquement, si $(q, R[z, z_l, z_r \leftarrow 0])$ vérifie la formule de CTL suivante :

$$\Psi_3 \stackrel{\text{def}}{=} \big(\neg\mathsf{E}\overleftarrow{(\neg\varphi_r)^{\dashv}}\ \mathsf{U}\ (\!|z{=}c{+}k|\!)\big) \wedge \Psi_2'' \wedge \Psi_2'''$$

Soient ρ une exécution (dans $\mathcal{A}$) qui commence à (q, v) où v est une valuation de R et $\bar{\rho}$ le chemin correspondant dans $G_{\mathcal{A},\Phi}^{\varphi_l,\varphi_r}$ partant de $(q, R[z, z_l, z_r \leftarrow 0])$. Comme Ψ_3 est satisfaite par $(q, R[z, z_l, z_r \leftarrow 0])$, il existe alors une configuration de $\bar{\rho}$ satisfaisant $\overleftarrow{(\neg\varphi_r)^{\dashv}}$ avant d'atteindre une configuration située à une durée égale à $c + k$. Par conséquent, ρ contient

un sous-chemin σ de longueur strictement supérieure à k u.t où φ_r est vraie, en plus ce sous-chemin contient inévitablement une position située strictement avant c u.t. Supposons maintenant que ρ ne vérifie pas $\varphi_l \mathsf{U}^k_{<c} \varphi_r$. Ce qui signifie que avant d'atteindre (strictement) un tel sous-chemin σ, il existe un sous-chemin σ' tel que $\hat{\mu}(\sigma') > k$ et $\sigma' \models \neg\varphi_l$. Le chemin $\bar{\rho}$ contient alors une configuration (q', R') telle que $(q', R') \models \neg\varphi_l \wedge \neg(\!|z_{\bar{l}} \leq k|\!)$. Comme Ψ''_2 est satisfaite par $(q, R[z, z_l, z_r \leftarrow 0])$, donc $(q', R') \models \mathsf{A}\varphi_r \mathsf{U} \overleftarrow{\varphi_r}$, mais ceci contredit le fait que σ' précède strictement le long de ρ, tout sous-chemin σ contenant une position strictement avant c u.t où φ_r dure plus que k u.t. De plus, Ψ'''_2 donne que la première région de σ satisfait $\overleftarrow{\varphi_l}^{\cdot|}$ sauf si c'est une région frontière ou elle a été atteinte par une transition d'action. On a donc que $\rho \models \varphi_l \mathsf{U}^k_{<c} \varphi_r$.

La même preuve peut être adaptée pour montrer que pour toute région R, $(q, R[z, z_l, z_r \leftarrow 0])$ est étiquetée par $\left(\neg \mathsf{E}(\overleftarrow{\neg\varphi_r}^{\cdot|} \mathsf{U} (\!|z > c+k|\!)\right) \wedge \Psi''_2 \wedge \Psi'''_2$ si et seulement si $(q, v) \models \mathsf{A}\varphi_l \mathsf{U}^k_{\leq c} \varphi_r$ pour tout $v \in R$.

– $\Psi = \mathsf{A}\varphi_l \mathsf{U}^k_{\geq c} \varphi_r$. Soit R une région, nous avons à montrer que :

$$(q, R[z, z_{\bar{l}}, z_r \leftarrow 0]) \models \mathsf{A} \overleftarrow{\varphi_l}^{\cdot|} \mathsf{U} \left((\!|z = c|\!) \wedge \mathsf{AF}(\overleftarrow{\varphi_r}) \wedge \Psi''_2 \wedge \Psi'''_2 \right) \Leftrightarrow (q, v) \models \Psi$$

pour tout $v \in R$.

$\Leftarrow$ Supposons que pour toute valuation v de R, $(q, v) \models \Psi$. Soient $\bar{\rho}$ un chemin dans $G^{\varphi_l, \varphi_r}_{\mathcal{A}, \Phi}$ partant de $(q, R[z, z_l, z_r \leftarrow 0])$ et (q', R') une configuration de $\bar{\rho}$ tels que $(q', R') \models (\!|z = c|\!)$. Notons $\bar{\rho}_{|_{(q', R')}}$ le préfixe de $\bar{\rho}$ jusqu'à la position qui correspond à cette configuration (q', R'). Il est évident que toutes les configurations avant (q', R') vérifient $\overleftarrow{\varphi_l}^{\cdot|}$. Considérons maintenant un chemin $\bar{\rho'}$ (dans $G^{\varphi_l, \varphi_r}_{\mathcal{A}, \Phi}$) partant de (q', R'). Si $\bar{\rho'} \not\models \mathsf{F}(\overleftarrow{\varphi_r})$, il existe alors clairement une exécution dans $\mathcal{A}$ partant de (q, v) qui ne satisfait pas $\varphi_l \mathsf{U}^k_{\geq c} \varphi_r$, donc $(q', R') \models \mathsf{AF} \overleftarrow{\varphi_r}$. Par ailleurs, si $(q', R') \not\models \Psi''_2$, il existe alors un chemin $\bar{\rho'}$ partant de (q', R') et une configuration (q'', R'') (le long de $\bar{\rho'}$) avant toutes les configurations vérifiant $\overleftarrow{\varphi_r}$ telle que $\overleftarrow{(\neg\varphi_l)} \wedge \neg \mathsf{A}(\varphi_r \mathsf{U} \overleftarrow{\varphi_r})$ est vraie. Considérons donc le chemin $\bar{\rho''}$ à partir de (q'', R'') qui satisfait $\neg(\varphi_r \mathsf{U} \overleftarrow{\varphi_r})$. Clairement, l'exécution dans $\mathcal{A}$ partant de (q, v) et correspondant au chemin $\bar{\rho}_{|_{(q', R')}} \cdot \bar{\rho'}_{|_{(q'', R'')}} \cdot \bar{\rho''}$ ne satisfait pas $\varphi_l \mathsf{U}^k_{\geq c} \varphi_r$. Par conséquent, (q', R') satisfait $\mathsf{AF}(\overleftarrow{\varphi_r}) \wedge \Psi''_2$. Supposons maintenant que (q', R') satisfait $\neg\Psi'''_2$, on obtient (de la même manière que précédemment) qu'il existe une exécution à partir de (q, v) avec $v \in R$ telle que toute position du sous-chemin témoin de $\overleftarrow{\varphi_r}$ sera précédée par une position dans une région ouverte qui satisfait $\neg\overleftarrow{\varphi_l}^{\cdot|}$. Ce qui donne une contradiction

$\Rightarrow$ Réciproquement, si $(q, R[z, z_l, z_r \leftarrow 0])$ vérifie la formule CTL suivante :

$$\Psi_5 \stackrel{\text{def}}{=} \mathsf{A} \overleftarrow{\varphi_l}^{\cdot|} \mathsf{U} \left((\!|z = c|\!) \wedge \mathsf{AF}(\overleftarrow{\varphi_r}) \wedge \Psi''_2 \wedge \Psi'''_2 \right)$$

Soient ρ une exécution dans $\mathcal{A}$ partant de (q, v) avec $v \in R$ et $\bar{\rho}$ le chemin correspondant partant de $(q, R[z, z_l, z_r \leftarrow 0])$ dans $G^{\varphi_l, \varphi_r}_{\mathcal{A}, \Phi}$. Comme $\bar{\rho}$ satisfait Ψ_5, il existe donc un sous-chemin σ (le long de ρ) de longueur supérieure strictement à k, contenant une position p telle que $\mathsf{Time}(\rho^{\leq p}) \geq c$ et $\sigma \models \varphi_r$. Supposons maintenant que ρ ne satisfait pas $\varphi_l \mathsf{U}^k_{\geq c} \varphi_r$. Ceci signifie qu'avant d'atteindre une telle position p, il existe un sous-chemin σ'

de longueur supérieure strictement à k où $\neg\varphi_l$ est vraie. Donc le chemin $\bar{\rho}$ contient alors une configuration (q', R') telle que $(q', R') \models \neg\varphi_l \wedge \neg(\!|z_{\bar{l}} \leq k|\!)$. Comme Ψ_5 est satisfaite par $(q, R[z, z_l, z_r \leftarrow 0])$, (q', R') se situe donc après la durée c (strictement). Et comme Ψ''_2 est satisfaite par une configuration (q'', R'') vérifiant $(\!|z = c|\!)$ (le long de $\bar{\rho}$), on a soit (1) $(q', R') \models \mathsf{A}\varphi_r\mathsf{U}\overleftrightarrow{\varphi_r}$, soit (2) il existe une configuration entre (q'', R'') et (q', R') où $\overleftrightarrow{\varphi_r}$ est vraie. Ces deux cas donnent une contradiction avec le fait que σ' précède (strictement) toutes les positions p (situées dans un sous-chemin où φ_r dure suffisamment longtemps telle que $\mathsf{Time}(\rho^{\leq p}) \geq c$). De plus, la formule Ψ'''_2 donne que la première région du sous-chemin témoin de $\overleftrightarrow{\varphi_r}$ a les bonnes propriétés.

La même preuve peut être adaptée pour montrer que si R est une région, $(q, R[z, z_l, z_r \leftarrow 0])$ est étiquetée par $\mathsf{A}\big(\overleftarrow{\varphi_l} \wedge (\!|z \leq c|\!)\big)\ \mathsf{U}\ \big((\!|z > c|\!) \wedge \mathsf{AF}(\overleftrightarrow{\varphi_r}) \wedge \Psi''_2\big)$ si et seulement si (q, v) satisfait $\mathsf{A}\varphi_l\mathsf{U}^k_{>c}\varphi_r$ pour tout $v \in R$.

– $\Psi = \mathsf{AF}^k_{=c}\varphi_r$. Soit R une région, on montre que :

$$(q, R[z, z_{\bar{l}}, z_r \leftarrow 0]) \models \neg\mathsf{E}\,(\!|z < c|\!)\ \mathsf{U}\ \Big((\!|z = c|\!) \wedge \mathsf{E}\,\overleftarrow{(\neg\varphi_r)}\ \mathsf{U}\ (\!|z > c+k|\!)\Big) \Leftrightarrow (q, v) \models \Psi$$

pour tout $v \in R$.

$\Leftarrow$ Supposons que pour toute valuation v de R, $(q, v) \models \mathsf{AF}^k_{=c}\varphi_r$, alors $(q, v) \models \neg\mathsf{EG}^k_{=c}\neg\varphi_r$. Supposons que $(q, R[z, z_l, z_r \leftarrow 0])$ ne satisfait pas $\neg\mathsf{E}(\!|z < c|\!)\mathsf{U}\Big((\!|z = c|\!) \wedge \mathsf{E}\overleftarrow{(\neg\varphi_r)}\mathsf{U}(\!|z > c+k|\!)\Big)$. Il existe donc un chemin $\bar{\rho}$ partant de $(q, R[z, z_l, z_r \leftarrow 0])$ tel que toute configuration (q', R') qui se situe entre c et $c + k$ (c'est-à-dire elle vérifie $(\!|c \leq z|\!) \wedge (\!|z \leq c + k|\!)$) satisfait $\overleftarrow{\neg\varphi_r}$. Clairement, pour toute exécution correspondante ρ de $\mathcal{A}$, tout sous-chemin σ de longueur $\hat{\mu}(\sigma) > k$, contenant une configuration localisée à une durée égale à c, contient une configuration où $\neg\varphi_r$ est vraie : ceci contredit le fait que $(q, v) \models \mathsf{AF}^k_{=c}\varphi_r$.

$\Rightarrow$ Réciproquement, si $(q, R[z, z_l, z_r \leftarrow 0])$ vérifie dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ la formule :

$$\Psi'_5 \stackrel{\text{def}}{=} \neg\mathsf{E}(\!|z < c|\!)\mathsf{U}\Big((\!|z = c|\!) \wedge \mathsf{E}\overleftarrow{(\neg\varphi_r)}\ \mathsf{U}\ (\!|z > c + k|\!)\Big)$$

On souhaite prouver que (q, v) vérifie $\mathsf{AF}^k_{=c}\varphi_r$ avec $v \in R$, ce qui est équivalent à prouver que $(q, v) \models \neg\mathsf{EG}^k_{=c}\neg\varphi_r$. Supposons l'inverse : $(q, v) \models \mathsf{EG}^k_{=c}\neg\varphi_r$. Il existe donc une exécution ρ à partir de (q, v) telle que tout sous-chemin σ de longueur strictement supérieure à k, contenant une configuration située à c unités de temps de (q, v), doit contenir une configuration où $\neg\varphi_r$ est vraie. Soit $\bar{\rho}$ le chemin correspondant dans $G^{\varphi_l,\varphi_r}_{\mathcal{A},\Phi}$ partant de $(q, R[z, z_l, z_r \leftarrow 0])$: considérons maintenant le suffixe de $\bar{\rho}$ à partir de la première région où $(\!|z = c|\!)$ est vraie, clairement ce suffixe vérifie $\overleftarrow{\neg\varphi_r}\mathsf{U}(\!|z > c+k|\!)$. Ceci est une contradiction avec le fait que $(q, R[z, z_l, z_r \leftarrow 0]) \models \Psi'_5$.

Nous avons maintenant terminé la preuve du lemme 5 de la correction de l'algorithme.

□

Nous avons le résultat suivant :

Théorème 11 (Complexité du model-checking)
Étant donnés un automate temporisé $\mathcal{A}$ et une formule Φ de la logique TCTL$^{\Delta}$*, décider si $\mathcal{A}$ satisfait Φ est un problème PSPACE-complet.*

Complexité. Comme la logique TCTL est incluse dans la logique TCTL$^{\Delta}$, le côté PSPACE-difficile vient alors directement du model-checking de la logique TCTL. Si on construit le graphe des régions et qu'on applique l'algorithme d'étiquetage, on obtient un algorithme exponentiel. Mais, on peut éviter la construction du graphe des régions en utilisant un algorithme à la volée. Ce qui permet d'obtenir un algorithme pour le model-checking de TCTL$^{\Delta}$ en espace polynomial.

5.4 Discussion et conclusion

Dans le cas particulier de TCTL^{a}, où le paramètre k est fixé à 0, l'équivalence usuelle de Alur et Dill [ACD93] est suffisante pour monter que les régions sont compatibles avec les formules de TCTL^{a} : la condition *(ER c)* n'est pas nécessaire. De plus, on peut donner un autre algorithme, spécifiquement adapté à ce cas. Celui-ci repose aussi sur une autre construction d'une variante du graphe des régions.

Soient $\mathcal{A}$ un automate temporisé et Φ une formule de TCTL^{a}. On considère $X^* = X \cup \{z\}$ où z est une horloge externe à $\mathcal{A}$ et K la constante maximale apparaissant dans $\mathcal{A}$ et Φ. Ensuite, on construit le graphe des régions $G_{\mathcal{A},\Phi} = (V, E)$ relatif à $\mathcal{A}$ et Φ avec X^* comme ensemble d'horloges et K comme constante maximale (voir page 21).

On définit le graphe $\mathcal{G}^+_{A,\Phi}$ à partir de $G_{\mathcal{A},\Phi}$ en considérant la fermeture transitive de la relation $\twoheadrightarrow_a$: $\mathcal{G}^+_{A,\Phi} = (V, E)$ où V est le même ensemble d'états que pour $G_{\mathcal{A},\Phi}$, et $E = \twoheadrightarrow \cup \twoheadrightarrow^+_a$ où les transitions $(q, \gamma) \twoheadrightarrow^+_a (q', \gamma')$ correspondent à une séquence de transitions d'action dans $\mathcal{A}$ qui peuvent être franchissables dans un délai nul, c'est-à-dire le temps écoulé ente deux transitions d'action peut être égal à zéro (figure 5.3).

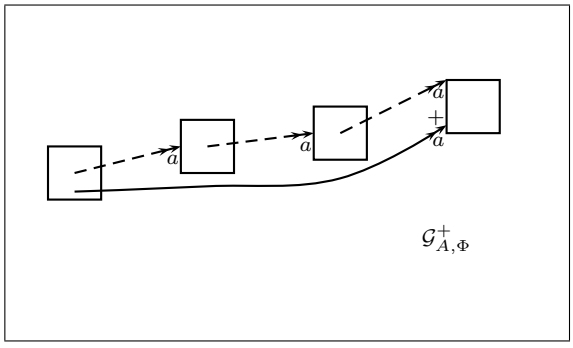


FIG. 5.3 – Le graphe $\mathcal{G}^+_{A,\Phi}$

Nous utilisons le graphe $\mathcal{G}^+_{A,\Phi}$ pour chercher s'il existe un chemin dans le graphe des régions $G_{\mathcal{A},\Phi}$ qui satisfait $\varphi \mathsf{U}^{\mathsf{a}}_{\sim \mathsf{c}} \psi$: nous allons chercher parmi les chemins de $\mathcal{G}^+_{A,\Phi}$, un chemin où "φ est toujours vraie avant ψ", ceci permet d'abstraire les configurations transitoires visitées par

la séquence de transitions d'action relative à $\twoheadrightarrow_a^+$; c'est-à-dire les configurations telles que le temps qu'on y reste est nul.
Une fois ce graphe construit, on adapte la procédure d'étiquetage pour TCTL. On étiquette les états de $G_{\mathcal{A},\Phi}$ de manière inductive par les sous-formules de Φ. Par exemple, si $\Phi = \mathsf{EG}^a_{<c}\varphi$, et si on suppose que le graphe $G_{\mathcal{A},\Phi}$ est déjà étiqueté par φ. Φ signifie qu'il existe une exécution ρ telle que tout sous-chemin σ contenant une position située à une durée inférieure strictement à c u.t, avec $\hat{\mu}(\sigma) > 0$ contient une position où φ est vraie. Donc, (q, R) est étiqueté par Φ si et seulement s'il existe un chemin dans $\mathcal{G}^+_{A,\Phi}$ partant de $(q, R[z \leftarrow 0])$ tel que tout état situé avant la durée égale à c doit vérifier φ à moins d'être une région frontière. On obtient alors la formule de CTL suivante : $\mathsf{E}^+(p_f \vee \varphi)\mathsf{U}(z = c)$.
L'opérateur E^+ signifie il existe un chemin dans $\mathcal{G}^+_{A,\Phi}$. La procédure d'étiquetage complète sur le graphe $\mathcal{G}^+_{A,\Phi}$ pour le model-checking de TCTL^a a été proposée dans [BBBL05].

En définissant une nouvelle relation d'équivalence sur les exécutions temporisées, nous avons montré que les régions introduites par Alur, Courcoubetis et Dill sont compatibles avec les formules de TCTL^Δ. Nous avons ensuite décrit un algorithme pour le problème du model-checking de cette logique. Celui-ci est fondé sur une procédure d'étiquetage d'une variante du graphe des régions. Le résultat principal de ce chapitre est que le model-checking de TCTL^Δ est décidable et qu'il est PSPACE-complet.

Conclusion et perspectives

Nous nous sommes intéressés, au cours de cette thèse, à la vérification formelle de propriétés temporisées pour les automates programmables industriels (APIs), qui sont parfois utilisés par des composants embraqués.

La vérification formelle nécessite préalablement une phase de modélisation permettant d'extraire et de formaliser les caractéristiques du système influant sur les propriétés à vérifier. Cette phase est difficile à réaliser car il n'y a pas de méthodes automatiques. De plus, un système réel est généralement composé de plusieurs modules qui communiquent et se synchronisent. Sa vérification consiste d'abord à choisir une classe de modèles pour représenter formellement ses différents composants et, ensuite, à vérifier des propriétés sur l'ensemble des modules obtenus, ce qui cause une explosion de la taille du modèle final. Ce problème constitue un obstacle majeur pour les outils de vérification. De plus, les modèles offerts par ces outils ne permettent pas de représenter toutes les caractéristiques du système réel, mais uniquement une simplification de ce système. Donc, pour mener avec succès la vérification de systèmes issus du milieu industriel, il est souvent nécessaire de trouver une abstraction correcte qui ne modifie pas la valeur de vérité des propriétés cherchées.
Pour cet aspect de la vérification, nous avons réalisé une étude de cas concernant la plate-forme *MSS (Mecatronic Standard System)* du groupe Bosch. Cette étude consiste à modéliser le processus central de la plate-forme (le *poste 2*) puis à en vérifier des propriétés temporisées. Le *poste 2* est contrôlé par un API programmé en Ladder Diagram, comportant plusieurs temporisateurs de type *TON*. Un des problèmes posé par ce système est la précision de l'arrêt d'un composant du *poste 2* (appelé le *chariot*), cette précision étant nécessaire pour garantir son bon fonctionnement.
Nous avons construit des modèles sous forme de réseaux d'automates temporisés, pour le programme de contrôle du *poste 2*, pour ses différents composants physiques et pour son environnement. Dans le but d'obtenir de bonnes performances en terme de temps de vérification, nous avons fait abstraction de certains comportements qui n'influent pas sur le résultat final. Nous avons utilisé l'outil Uppaal pour la vérification, car il est relativement bien adapté aux caractéristiques du modèle obtenu. La vérification avec Uppaal a prouvé que la précision de l'arrêt du chariot est meilleure avec un programme de contrôle multi-tâches. Cependant, ce mode de programmation reste insuffisant pour obtenir la précision d'arrêt souhaitée.

Nous nous sommes ensuite intéressés à un type particulier d'abstraction consistant à ignorer des états transitoires pendant la vérification de propriétés temporisées. Il s'agit d'états où le système ne reste qu'un temps négligeable. De tels états apparaissent naturellement dans la modélisation des automates programmables industriels. Ceci est dû aux changements instantanés des valeurs de leurs variables discrètes.

Pour traiter ce problème, nous avons étendu la logique TCTL avec de nouvelles modalités. Celles-ci permettent d'ignorer les événements qui ne se produisent pas de manière continue pendant k unités de temps (où k est un paramètre entier). Nous avons montré que cette nouvelle logique, appelée TCTL^{Δ}, est strictement plus expressive que TCTL, et que le fait d'utiliser des paramètres différents ajoute de l'expressivité. Nous avons aussi montré que le model-checking de TCTL^{Δ} est décidable. Pour cela nous avons été amenés à introduire une nouvelle relation d'équivalence sur les exécutions temporisées et un nouvel algorithme d'étiquetage.
Nous avons considéré une autre sémantique plus naturelle qui permet, par exemple, de contraindre le long d'une exécution donnée, la durée totale où une propriété est fausse. On note la logique munie de cette sémantique par $\mathsf{TCTL}^{\Delta}_{\Sigma}$. Nous avons montré que le model-checking de $\mathsf{TCTL}^{\Delta}_{\Sigma}$ est indécidable.

Perspectives de recherche

Deux directions sont envisageables pour la poursuite de ces travaux. La première concerne la modélisation des programmes des APIs, qui a donné lieu à peu d'études jusqu'à présent. Les modèles à base d'automates temporisés utilisent des horloges qui évoluent avec la même vitesse. Or le comportement des APIs est plutôt *hybride* : certaines tâches arrêtent leur exécution pour que d'autres tâches s'exécutent. Il nous semble donc utile de proposer des modèles plus appropriés aux comportements des APIs, par exemple en remplaçant les horloges par des chronomètres. La difficulté provient du fait que le modèle général est indécidable. Il serait donc intéressant de trouver des sous-classes, pertinentes pour les APIs, mais qui soient décidables.
Nous souhaitons aussi que ces modèles soient *implémentables*. Par exemple, les automates temporisés utilisent des horloges, qui évoluent dans un ensemble dense, et sont supposées "parfaites" dans le sens suivant : on peut tester leur valeur à tout moment, les remettre à zéro instantanément, etc. Ces hypothèses sont souvent trop fortes et, en pratique, le modèle (après implémentation) sera un peu différent : il faut donc s'assurer que le modèle vérifié pourra être implémenté sans perdre ses bonnes propriétés. Ces problèmes commencent à être étudiés précisément depuis quelques années (on parle de modèles robustes ou implémentables) et nous souhaitons nous y intéresser dans le cadre particulier des APIs.
Concernant toujours la modélisation des programmes des APIs, il serait intéressant d'étudier d'autres langages et d'autres blocs fonctionnels de la norme IEC 61131-3.

La deuxième direction prolonge les travaux sur la logique TCTL^{Δ} et concerne l'implémentation du model-checking de cette logique. L'efficacité de l'algorithme de model-checking de TCTL repose sur l'utilisation de structures de données bien adaptées au cadre temporisé : les DBM (Difference Bounded Matrices). Il serait donc pertinent d'examiner si ces structures peuvent être utilisées pour le model-checking de TCTL^{Δ} ou de trouver d'autres structures adaptées spécifiquement à cette logique.
Concernant la logique $\mathsf{TCTL}^{\Delta}_{\Sigma}$, qui est indécidable, nous avons toutefois identifié un fragment décidable : celui qui autorise seulement des formules telles que $\mathsf{E}P_1\mathsf{U}^{k}_{\leq c}P_2$ et $\mathsf{E}P_1\mathsf{U}^{k}P_2$ où P_1 et P_2 sont des propositions atomiques. Nous espérons trouver des fragments plus larges qui restent décidables.

Bibliographie

[ABBL03] Luca Aceto, Patricia Bouyer, Augusto Burgueño, and Kim Larsen. The Power of Reachability Testing for Timed Automata. *Theoretical Computer Science*, 300(1–3) :411–475, 2003.

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-Checking in Dense Real-Time. *Information and Computation*, 104(1) :2–34, 1993.

[ACH97] Rajeev Alur, Costas Courcoubetis, and Thomas Henzinger. Computing Accumulated Delays in Real-Time Systems. *Formal Methods in System Design*, 11(2) :137–156, 1997.

[AD90] Rajeev Alur and David Dill. Automata for Modeling Real-Time Systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[AD94] Rajeev Alur and David Dill. A Theory of Timed Automata. *Theoretical Computer Science (TCS)*, 126(2) :183–235, 1994.

[AFH94] Rajeev Alur, Limor Fix, and Thomas Henzinger. A Determinizable Class of Timed Automata. In *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.

[AH92] Rajeev Alur and Thomas Henzinger. Back to the Future : Towards a Theory of Timed Regular Languages. In *Proc. 33rd IEEE Symposium on Foundations of Computer Science (FOCS'92)*, pages 177–186. IEEE Computer Society Press, 1992.

[AHV93] Rajeev Alur, Thomas Henzinger, and Moshe Vardi. Parametric Real-Time Reasoning. In *Proc. 25th Annual ACM Symposium on the Theory of Computing (STOC'93)*, pages 592–601. ACM, 1993.

[ALTP01] Rajeev Alur, Salvatore La Torre, and George Pappas. Optimal Paths in Weighted Timed Automata. In *Proc. 4th International Workshop on Hybrid Systems : Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2001.

[BBBL05] Houda Belmokadem, Béatrice Bérard, Patricia Bouyer, and François Laroussinie. A New Modality for Almost Everywhere Propeties in Timed Automata. In *Proc. 16th International Conference on Concurrency Theory (CONCUR05)*, volume LNCS 3653, pages 110–124. Springer, 2005.

[BBBL06] Houda Bel Mokadem, Béatrice Bérard, Patricia Bouyer, and François Laroussinie. Timed Temporal Logics for Abstracting Transient States. In *Proceedings of the 4th*

International Symposium on Automated Technology for Verification and Analysis (ATVA'06), Lecture Notes in Computer Science, Beijing, ROC, Oct 2006. Springer. To appear.

[BBG+05] Houda Belmokadem, Béatrice Bérard, Vincent Gourcuff, Jean-Marc Roussel, and Olivier de Smet. Verification of a Timed Multitask System with Uppaal. In *Proc. 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'05)*, pages 347–354. IEEE Industrial Electronics Society, 2005.

[BBM06] Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved Undecidability Results on Priced Timed Automata. *Information Processing Letters*, 98(5) :188–194, Jun 2006.

[BBR04] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. Model-Checking for Weighted Timed Automata. In *Proc. Joint conference Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, volume LNCS 3253, pages 277–292. Springer, 2004.

[BD00] Béatrice Bérard and Catherine Dufourd. Timed Automata and Additive Clock Constraints. *Information Processing Letters (IPL)*, 75(1–2) :1–7, 2000.

[BDFP00a] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are Timed Automata Updatable ? In *Proc. 12th International Conference on Computer Aided Verification (CAV'2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2000.

[BDFP00b] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Expressiveness of Updatable Timed Automata. In *Proc. 25th International Symposium on Mathematical Foundations of Computer Science (MFCS'2000)*, volume 1893 of *Lecture Notes in Computer Science*, pages 232–242. Springer, 2000.

[BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the Expressive Power of Silent Transitions in Timed Automata. *Fundamenta Informaticae*, 36(2–3) :145–182, 1998.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-Cost Reachability for Priced Timed Automata. In *Proc. 4th International Workshop on Hybrid Systems : Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

[BFKM03] Béatrice Bérard, Laurent Fribourg, Francis Klay, and Jean-François Monin. A Compared Study of Two Correctness Proofs for the Standardized Algorithm of ABR Conformance. *Formal Methods in System Design*, 22(1) :59–86, 2003.

[BGP96] Béatrice Bérard, Paul Gastin, and Antoine Petit. On the Power of Non-Observable Actions in Timed Automata. In *Proc. 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS'96)*, volume 1046 of *Lecture Notes in Computer Science*, pages 257–268. Springer, 1996.

[Can01] Géraud Canet. *Vérification Automatique de programmes écrits dans les langages IL et ST de la norme IEC 61131-3*. PhD thesis, École Normale Supérieur de Cachan, Cachan, France, Dec 2001.

[CDP+00] Géraud Canet, Bruno Denis, Antoine Petit, Olivier Rossi, and Philippe Schnoebelen. Un Cadre pour la Vérification Automatique de Programmes IL. In *Actes de la 1ère Conférence Internationale Francophone d'Automatique (CIFA 2000)*, pages 693–698, Lille, France, Jul 2000.

[CES86] Edmund Clarke, Allen Emerson, and Prasad Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2) :244–263, 1986.

[CRH91] Zhou Chaochen, Anders Ravn, and Charles Hoare. A Calculus of Duration. *Information Processing Letters (IPL)*, 40(5) :269–276, 1991.

[DOTY96] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The Tool KRONOS. In *Proc. Hybrid Systems III : Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1996.

[Duf97] Catherine Dufourd. Une Extension d'un résultat d'Indécidabilité pour les Automates Temporisés. In *Proc. 9th Rencontres Francophones du Parallélisme (RenPar'97)*, 1997.

[DY96] Conrado Daws and Sergio Yovine. Reducing the Number of Clock Variables of Timed Automata. In *Proc. 17th IEEE Real-Time Systems Symposium (RTSS'96)*, pages 73–81. IEEE Computer Society Press, 1996.

[EFP94] Gottfried Egger, Andreas Fett, and Peter Pepper. Formal Specification of a Safe PLC Language and its Compiler. In Victor Maggioli, editor, *SAFECOMP'94, Proc. 13th International Conference on Computer Safety, Reliability and Security (SAFECOMP'94)*, pages 11–20, Oct 1994.

[EH86] Allen Emerson and Joseph Halpern. "Sometimes" and "not Never" Revisited : On Branching Versus Linear Time Temporal Logic. *Journal of the ACM*, 33(1) :151–178, 1986.

[Eme91] Allen Emerson. *Temporal and Modal Logic*, volume B (Formal Models and Semantics) of *Handbook of Theoretical Computer Science*, pages 995–1072. MIT Press Cambridge, 1991.

[Fre98] Litz Frey. Verification and Validation of Control Algorithms by Coupling of Interpreted Petri Nets. In *Proc. IEEE Int Conf on Systems Man and Cybernetics (SMC'98)*, pages 7–12, Oct 1998.

[Fre00a] Georg Frey. Automatic Implementation of Petri Net based Control Algorithms on PLC. In *Proc. American Control Conference (ACC'2000), Chicago*, pages 2819–2823, Jun 2000.

[Fre00b] Georg Frey. PLC Programming for Hybrid Systems via Signal Interpreted Petri Nets. In *Proc. 4th Int Conf on Automation of Mixed Processes ADPM, Dortmund, Germany*, pages 189–194, Sep 2000.

[Fre00c] Litz Frey. Formals Methods in PLC Programming. In *Proc. IEEE Int Conf on Systems Man and Cybernetics (SMC'2000)*, pages 2431–2436, 2000.

[Hen97] Dierks Henning. PLC-Automata : A New Class of Implementable Real-Time Automata. In *Transformation-Based Reactive Systems Development (ARTS'97)*, volume 1231 of *Lecture Notes in Computer Science*, pages 111–125, 1997.

[HHWT95] Thomas Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A User Guide to HYTECH. In *Proc. 1st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer, 1995.

[HKWT95] Thomas Henzinger, Peter Kopke, and Howard Wong-Toi. The Expressive Power of Clocks. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP'95)*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer, 1995.

[HM98a] Monika Heiner and Thomas Menzel. Instruction List Verification Using a Petri Net Semantics. In *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'98), 11-14, San Diego, CA*, pages 716–721, Oct 1998.

[HM98b] Monika Heiner and Thomas Menzel. A Petri Net Semantics for the PLC Language Instruction List. In *Proc. IEE Workshop on Discrete Event Systems (WODES '98), Cagliari/Italy*, pages 161–165, Aug 1998.

[HU79] John Hopcroft and Jeffrey Ullman. Introduction to Automata Theory, Languages and Computation. *Addison-Wesley*, 1979.

[Huu03] Ralf Huuck. *Software Verification for Programmable Logic Controllers*. PhD thesis, Technischen fakultät der Christian-Albrechts-Universität zu Kiel, 2003.

[IEC93] IEC (International Electrotechnical Commission). *IEC Standard 61131-3 : Programmable Controllers - Part 3*, 1993.

[KPJS99] Yonit. Kesten, Amir Pnueli, Sifakis Joseph, and Yovine Sergio. Decidable Integration Graphs. *Information and Computation*, 150(2) :209–243, 1999.

[LCRRL99] Sandrine Lampérière-Couffin, Olivier Rossi, Jean-Marc Roussel, and Jean-Jacques Lesage. Formal Validation of PLC Programs : a survey. In *European Control Conference (ECC'99), Karlsruhe, Germany, Aug-Sep*, 1999. proceedings on CD-ROM, communication 741.

[Lew98] Robert Lewis. *Programming Industrial Control Systems using the IEC 1131-3 (Revised edition)*. The Institution of Electrical Enginneers, 1998.

[LPY97] Kim Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Journal of Software Tools for Technology Transfer (STTT)*, 1(1–2) :134–152, 1997.

[LT93] N. Leveson and C. Turner. An Investigation of the Therac-25 accidents. *Computer*, pages 18–41, 1993.

[Mad00] Angelika Mader. A Classification of PLC Models and Applications. In *Proc. 5th Int. Workshop on Discrete Event Systems (WODES)*, pages 239–247. Kluwer Academic Publishers, Aug 2000.

[Min67] Marvin Minsky. Computation : Finite and Infinite Machines. *Prentice Hall International*, 1967.

[Moo94] I. Moon. Modeling Programmable Logic Controllers for Logic Verification. *IEEE Control Systems Magazine*, 1994.

[MW99] Angelika Mader and Hanno Wupper. Timed Automaton Models for Simple Programmable Logic Controllers. In *Proc. 11th Euromicro Conference on Real-Time Systems (ECRTS'99)*, pages 114–122. IEEE Comp. Soc. Press, Jun 1999.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *Proc. 18th IEEE Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *Proc. 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[Sav70] Walter Savitch. Relationships between Nondeterministic and Deterministic Space Complexities. *Computer and System Sciences*, 4(2) :177–192, 1970.

[Tou97] Konstantinos Tourlas. An Assessment of the IEC 1131 -3 Standard on Languages for Programmable Controllers. In Peter Daniel, editor, *SAFECOMP97 : the 16th International Conference on Computer Safety, Reliability and Security York, UK*, pages 210–219. Springer, 1997.

[Wil99] H. Willems. Compact Timed Automata for PLC programs. *Technical Report CSI-R9925*, 1999.

Index