Model checking temporisé
Algorithmes efficaces et complexité

(Mémoire d'habilitation à diriger des recherches)

François Laroussinie

Laboratoire Spécification et Vérification
ENS de Cachan & CNRS UMR 8643
email : fl@lsv.ens-cachan.fr

Table des matières

Chapitre 1

Introduction

Mes travaux de recherche portent sur la vérification par model checking et la logique temporelle. Aujourd'hui, une des principales limites du model checking est l'explosion combinatoire de l'ensemble des états : le modèle représentant un système complexe est souvent gigantesque. Cette complexité a suscité le développement de structures de données efficaces comme les BDD, ou d'heuristiques particulières, "à la volée" ou symboliques... L'analyse de complexité permet, elle, de mesurer finement ces difficultés et de comparer les modèles.

La vérification temporisée, où l'on s'intéresse aux contraintes quantitatives sur l'écoulement du temps, pose des problèmes de complexité supplémentaires. Plusieurs de mes travaux abordent ces questions et forment un ensemble cohérent, "à la recherche de modèles temporisés et efficaces". C'est ce fil que j'ai suivi pour la rédaction de ce mémoire d'habilitation à diriger des recherches.

La question que nous abordons ici est donc le coût induit par la prise en considération de contraintes temps réel dans le model checking temporisé. Pour cela, nous considérons plusieurs types de modèle allant des simples structures de Kripke avec durées entières jusqu'aux automates temporisés. A chaque fois, on cherche à isoler des variantes ou sous-classes de ces modèles afin de trouver des algorithmes efficaces pour la vérification.

Le mémoire est organisé comme suit :

Chapitre 2 : Dans cette partie, on définit les *systèmes de transitions temporisés* (STT) qui permettront de définir les sémantiques des différents modèles de ce mémoire. Ensuite on rappelle les définitions des logiques temporelles temporisées et on donne leur sémantique sur les STT. Le chapitre se termine par des rappels de résultats classiques sur la com-

plexité du model checking non temporisé qui seront utilisés comme points de comparaison pour évaluer ceux du cas temporisé. Je mentionne aussi quelques contributions personnelles dans ce domaine.

Chapitre 3 : Cette partie porte sur les extensions des structures de Kripke avec des durées entières. Plusieurs sémantiques sont proposées (en terme de STT) et différents résultats de complexité pour les logiques temporelles temporisées sont présentés. Une extension probabiliste est aussi rapidement exposée.

Chapitre 4 : Ce chapitre porte sur les automates temporisés (le domaine de temps est $\mathbb{R}_+$). Nous en rappelons la sémantique, ainsi que des constructions (par ex. le graphe des régions) et des résultats de complexité classiques. Ensuite nous exposons des résultats pour les logiques modales temporisées, les automates temporisés à une ou deux horloges et les compositions parallèles. Le model checking compositionnel est aussi présenté à la fin du chapitre.

L'annexe A contient un survol de mes autres travaux en dehors de la problématique de la complexité de la vérification temporisée.

J'ai mené ces recherches avec d'autres collègues. Pour le chapitre 3, j'ai travaillé avec Nicolas Markey (LSV), Philippe Schnoebelen (LSV), Jeremy Sproston (Univ. de Turin) et Mathieu Turuani (LORIA) ([LST03, LMS02a, LMS05, LS05]).

Les recherches présentées dans le chapitre 4 ont été réalisées en collaboration avec Luca Aceto (Univ. d'Aalborg), Franck Cassez (IRCCyN), Kim G. Larsen (Univ. d'Aalborg), Nicolas Markey et Philippe Schnoebelen ([AL02, LMS04, LS00b, LL95, LL98, CL00]).

Le chapitre 2 et l'annexe A contiennent des résultats obtenus avec Houda Bel Mokadem (LSV), Béatrice Bérard (LAMSADE), Patricia Bouyer (LSV), Franck Cassez (IRCCyN), Stéphane Demri (LSV), Kim G. Larsen (Univ. d'Aalborg), Nicolas Markey et Philippe Schnoebelen ([LMS01, LLW95, LS00a, LMS02b, DLS02, LS97, BBBL05, BCL05]).

Dans ce document, mes contributions sont signalées par un encadrement comme celui-ci.

Chapitre 2

Systèmes de transitions, logiques temporelles et model checking

2.1 Introduction

L'objectif de cette partie est d'introduire les *systèmes de transitions temporisés* (STT) qui nous serviront comme modèle de base pour définir les sémantiques des systèmes temporisés. Dans un deuxième temps, nous présentons les différentes logiques temporelles et modales pour énoncer des propriétés temps-réel ainsi que plusieurs équivalences observationnelles. Enfin, nous rappelons une série de résultats sur la complexité du model checking non temporisé.

2.2 Systèmes de transitions temporisés

On suppose donnés un domaine de temps $\mathbb{T}$ (qui sera par la suite $\mathbb{N}$, $\mathbb{Q}_+$ ou $\mathbb{R}_+$), un alphabet Σ, un symbole particulier $\delta \notin \Sigma$ pour étiqueter les transitions de temps et un ensemble de propositions atomiques Prop pour les états. Un système de transitions temporisé (STT) est un quadruplet $\mathcal{T} = \langle S, s_{\text{init}}, \rightarrow, l \rangle$ où

- S est un ensemble (éventuellement infini) d'états,
- s_{init} est l'état initial,
- $\rightarrow \subseteq S \times \mathbb{T} \times (\Sigma \cup \{\delta\}) \times S$ est une relation de transition avec durée et étiquette,
- $l : S \rightarrow 2^{\mathsf{Prop}}$ étiquette chaque état par des propositions atomiques.

On note les transitions $(s_1, t, a, s_2) \in \rightarrow$ par $s_1 \xrightarrow{t,a} s_2$. Une telle transition contient donc une durée t et une action a. Le symbole δ est utilisé pour les *(pures) transitions de temps*, — on les notes plus simplement $s \xrightarrow{t} s'$. On parlera aussi de *(pures) transitions d'action* lorsque le temps associé t est nul — on les note $s \xrightarrow{a} s'$.

Une exécution d'un STT est une séquence infinie de transitions $\pi = s_0 \xrightarrow{t_0,a_0} s_1 \xrightarrow{t_1,a_1} s_2 \ldots$ On note $\mathsf{Exec}(s)$ les exécutions issues de s. Lorsque $\mathbb{T}$ est discret, une telle exécution contient les états s_0, s_1, ... Lorsque $\mathbb{T}$ est dense, une transition de temps de durée t décrit une évolution continue (une trajectoire). Dans ce cas, une exécution contiendra les états s_0, s_1, ... mais également les états situés à une durée $0 \leq t \leq t_i$ à partir de s_i (par une transition $s_i \xrightarrow{t}$). Cet ensemble d'états est infini dès qu'il existe une durée t_i strictement positive.

Nous supposons que les transitions de temps sont déterministes : $s \xrightarrow{t} s'$ et $s \xrightarrow{t} s''$ impliquent $s' = s''$. Lorsque le temps est dense, nous aurons d'autres propriétés sur les exécutions (additivité temporelle et variabilité finie).

Que l'on considère un domaine de temps dense ou discret, nous retrouvons les notions classiques de préfixe, suffixe et sous-exécutions pour les exécutions de $\mathcal{T}$. À tout préfixe fini σ de π (on le note $\sigma \in \mathsf{Pref}(\pi)$ ou $\pi = \sigma \cdot \pi'$ avec π' un suffixe de π), on peut associer un mot fini de Σ^* et une *durée*, notée $\mathsf{Time}(\sigma)$, définie comme la somme des durées des transitions de σ. Enfin, on notera $s <_\rho s'$ le fait qu'un état s précède strictement s' le long de ρ, c'est-à-dire qu'il existe une sous-exécution σ de ρ, non réduite à s, menant de s à s', on le notera aussi $s \xmapsto{\sigma} s'$.

Propriétés des exécutions. En général, on impose des conditions particulières sur les exécutions d'un STT. On peut, par exemple, exiger que le *temps diverge*, *i.e.* que pour toute durée d, il existe un préfixe fini σ de π tel que $\mathsf{Time}(\sigma) \geq d$. On parle aussi d'exécutions *non-Zénon*. Cela nécessite que la relation de transition soit totale. Il peut aussi être intéressant d'imposer que toute exécution contienne un nombre infini de transitions étiquetées par un symbole différent de δ (*i.e.* des exécutions sans un suffixe contenant exclusivement des transitions de temps). Dans le reste du document nous supposerons donc que la définition des exécutions intègre des contraintes de ce type, nous les préciserons pour chacun des différents modèles étudiés. Notons que cela ne change pas les résultats de complexité pour le model checking. Nous renvoyons notamment à [HNSY94] pour une discussion précise sur les propriétés des exécutions des systèmes temps-réel (pour le temps

dense) et leur importance du point de vue sémantique.

STT à petit pas. On appelle *STT à petit pas* les STT dont les durées associées aux transitions appartiennent à l'ensemble $\{0, 1\}$. Cette classe nous intéresse pour caractériser certains modèles temporisés dans le cadre du temps discret. Nous verrons dans le chapitre 3 qu'elle donne lieu à une série de propriétés particulières pour le model checking.

Composition parallèle. Lorsqu'on modélise un système complexe, il est important de disposer de mécanismes de composition : on définit des composants, on les met en parallèle, on les synchronise, etc. De nombreuses solutions existent, la plus utilisée est sans doute la composition parallèle avec une table de synchronisation. On peut définir ce genre de construction pour les STT, l'idée est alors d'exiger en plus que les transitions synchronisées aient la même durées (on suppose toujours un temps global). Nous verrons ces différents aspects pour les deux familles de modèles présentées dans ce document.

2.3 Énoncer des propriétés temps-réel

Étant donné un système temporisé dont le comportement est défini sous la forme d'un STT, nous souhaitons disposer d'un langage de spécification pour énoncer des propriétés temps-réel. Par exemple, on peut vouloir énoncer la propriété suivante :

"L'alarme se déclenche *au plus 3 secondes après* l'apparition d'un problème" (2.1)

Pour cela on peut utiliser des extensions des formalismes de spécification classiques ; l'objectif de cette section est de présenter plusieurs approches possibles. Nous étudions notamment le cas des logiques temporelles (ou modales) quantitatives ou temporisées. Nous évoquons aussi les équivalences comportementales (simulation ou bisimulation).

Accessibilité. Avant de considérer ces différents problèmes, nous rappelons le problème de l'accessibilité. C'est le problème de vérification le plus connu et le plus utilisé en pratique, tous les model checkers permettent de le traiter. Il s'agit, étant donné un — ou une composition parallèle de — système(s) temporisé(s) S et un état de contrôle q, de décider s'il existe une

exécution de S menant à une configuration où l'état de contrôle courant est q. À la place d'un état q, on peut aussi considérer un ensemble d'états de contrôle. L'accessibilité ne concerne donc pas à proprement parler une propriété quantitative mais il est parfois possible de modifier simplement le modèle afin de ramener la vérification d'une propriété temporisée simple (par exemple l'accessibilité en temps borné par une constante) à un problème d'accessibilité.

Selon que l'on utilise des logiques temporelles ou des logiques modales, on s'intéresse soit aux propositions atomiques étiquetant les états (en oubliant les étiquettes des transitions), soit uniquement à l'étiquetage des transitions.

2.3.1 Logiques temporelles quantitatives

Pour intégrer des contraintes quantitatives dans les logiques temporelles, on peut soit ajouter des contraintes aux modalités classiques, soit ajouter des horloges — des *horloges de formule* — et des opérateurs pour les manipuler.

Modalités avec contraintes quantitatives. L'idée est de compléter l'opérateur temporel Until (U) avec une contrainte de la forme $\sim c$ avec $\sim \in \{=, <, \leq, \geq, >\}$ et $c \in \mathbb{N}$. La formule $\varphi \mathsf{U}_{<c} \psi$ est vraie pour une exécution ρ ssi il existe un état s, situé à moins de c unités de temps de l'état initial, et vérifiant ψ et tel que tous les états précédents le long de ρ vérifient ϕ. Il est aussi possible d'associer un intervalle $[a; b]$ à un opérateur Until. Cette approche pour étendre les logiques temporelles est assez classique [KVdR83, Koy90]. Pour *CTL*, elle a été proposée pour le temps discret [EMSS92] et pour le temps dense [ACD93]. Formellement on définit la logique *TCTL* de la manière suivante :

Définition 1 (Syntaxe de *TCTL* [ACD93, EMSS92]) *Les formules de TCTL sont décrites par la grammaire suivante :*

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

avec $\sim \in \{<, \leq, =, \geq, >\}$, $c \in \mathbb{N}$ *et* $\forall i$, $P_i \in \mathsf{Prop}$.

La taille d'une formule est définie de manière classique en supposant que les constantes numériques sont codées en binaire. Ainsi nous avons $|\mathsf{E}\varphi \mathsf{U}_{\sim c}\, \psi| = |\varphi| + |\psi| + \lceil \log(c+1) \rceil$.

On définit aussi les abréviations courantes suivantes : $\top, \bot, \varphi \vee \psi, \varphi \Rightarrow \psi, \ldots$ Nous utilisons la modalité U pour $\mathsf{U}_{\geq 0}$. Enfin, les contraintes "$\sim c$" peuvent aussi s'utiliser dans les abréviations de *CTL*, nous avons donc :

– $\mathsf{EF}_{\sim c}\,\varphi$ ($\stackrel{\text{def}}{=} \mathsf{E}\top\mathsf{U}_{\sim c}\,\varphi$) pour exprimer l'accessibilité de φ dans un délai vérifiant $\sim c$.
– $\mathsf{AF}_{\sim c}\,\varphi$ ($\stackrel{\text{def}}{=} \mathsf{A}\top\mathsf{U}_{\sim c}\,\varphi$) pour énoncer l'inévitabilité de φ dans un délai vérifiant $\sim c$.
– $\mathsf{EG}_{\sim c}$ le dual de $\mathsf{AF}_{\sim c}$.
– $\mathsf{AG}_{\sim c}$ le dual de $\mathsf{EF}_{\sim c}$.

La propriété 2.1 énoncée ci-dessus s'écrit alors :

$$\mathsf{AG}\Big(\ \mathtt{problème}\ \Rightarrow\ \mathsf{AF}_{\leq 3}\,\mathtt{alarme}\ \Big)$$

La sémantique des formules de *TCTL* est définie sur un état d'un système de transition temporisé :

Définition 2 (Sémantique de *TCTL***)** *Les clauses suivantes définissent la valeur de vérité des formules de TCTL sur un état s d'un STT $\mathcal{T} = \langle S, s_{init}, \rightarrow, l\rangle$, noté $s \models \varphi$. Le cas des opérateurs booléens est omis.*

$$\begin{array}{ll} s \models \mathsf{E}\varphi\mathsf{U}_{\sim c}\,\psi & \textit{ssi}\ \ \exists\,\rho \in \mathsf{Exec}(s)\ \textit{avec}\ \rho = \sigma\cdot\rho'\ \textit{et}\ s \stackrel{\sigma}{\longmapsto} s'\ \textit{t.q.} \\ & \mathsf{Time}(\sigma) \sim c\,,\ s' \models \psi\ \textit{et}\ \forall\, s'' <_\rho s',\ s'' \models \varphi \\ s \models \mathsf{A}\varphi\mathsf{U}_{\sim c}\,\psi & \textit{ssi}\ \ \forall \rho \in \mathsf{Exec}(s), \exists \sigma \in \mathsf{Pref}(\rho)\ \textit{t.q.}\ s \stackrel{\sigma}{\longmapsto} s', \\ & \mathsf{Time}(\sigma) \sim c,\ s' \models \psi\ \textit{et}\ \forall\, s'' <_\rho s',\ s'' \models \varphi \end{array}$$

On peut noter que l'opérateur Next (EX) de *CTL* n'est pas présent dans cette définition de *TCTL* : en effet la notion de successeur immédiat n'est pas définie lorsque l'on considère un modèle de temps dense. On peut néanmoins ajouter cet opérateur lorsque l'on s'intéresse au temps discret.

Autres logiques temporelles quantitatives. On peut aussi étendre les logiques de temps linéaire et obtenir *TLTL* (notée $L(\mathsf{U}_{\sim c})$ en suivant les notations de [Eme90] pour les logiques temporelles classiques) ou $TLTL_{\leq,\geq}$. La propriété 2.1 s'écrit alors "$\mathsf{G}(\mathtt{problème} \Rightarrow \mathsf{F}_{\leq 3}\ \mathtt{alarme})$". Parmi les logiques temporelles temporisées de temps linéaire, on peut notamment citer *MTL* [Koy90, AH93] qui associe des intervalles à la modalité U (et qui contient parfois l'opérateur du passé Since), ainsi que *MITL* [AFH96] une restriction de *MTL* où les intervalles ne sont pas réduits à une unique valeur (*i.e.* la modalité $\mathsf{U}_{=c}$ n'est pas permise).

Horloges de formule. Une autre méthode pour intégrer des aspects quantitatifs dans les logiques temporelles consiste à ajouter des horloges de formules [AH94] (on note H l'ensemble de ces horloges) qui augmentent

de manière synchrone avec le temps, un opérateur de remise à zéro ($\underline{\texttt{in}}$) et des contraintes simples $x \sim c$ ou $x - y \sim c$ avec $x, y \in H$. La remise à zéro suivie, plus tard, d'une contrainte $x \sim c$ permet ainsi de mesurer le délai séparant deux états du système. Formellement, on définit $TCTL_h$ par :

Définition 3 (Syntaxe de $TCTL_h$ [Alu91]) *Les formules de $TCTL_h$ sont décrites par la grammaire suivante :*

$$\begin{array}{rcl} \varphi, \psi & ::= & P_1 \mid P_2 \mid \ldots \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi \\ & & \mid x \ \underline{\texttt{in}} \ \varphi \mid x \sim c \mid x - y \sim c \end{array}$$

avec $\sim \in \{<, \leq, =, \geq, >\}$, $c \in \mathbb{N}$, $x, y \in H$ *et* $P_i \in \mathit{Prop}$

La sémantique des formules de $TCTL_h$ est définie sur un état d'un STT et une valuation v ($v : H \to \mathbb{T}$) pour les horloges de H. Étant donnée une valuation v et $d \in \mathbb{T}$, on note $v + d$ la valuation qui associe la valeur $v(x) + d$ à chaque hologe $x \in H$ et $v[y \leftarrow 0]$ désigne la valuation qui associe à y la valeur 0 et laisse les autres horloges inchangées par rapport à v.

Définition 4 (Sémantique de $TCTL_h$) *Les clauses suivantes définissent la valeur de vérité des formules de $TCTL_h$ sur un état s d'un STT $\mathcal{T} = \langle S, s_{init}, \to, l\rangle$ et une valuation $v : H \to \mathbb{T}$, noté $s, v \models \varphi$:*

$$\begin{array}{ll} s, v \models x \sim c & \textit{ssi } v(x) \sim c \\ s, v \models x - y \sim c & \textit{ssi } v(x) - v(y) \sim c \\ s, v \models x \ \underline{\texttt{in}} \ \varphi & \textit{ssi } s, v[x \leftarrow 0] \models \varphi \\ s, v \models \mathsf{E}\varphi\mathsf{U}\psi & \textit{ssi } \exists\, \rho \in \mathsf{Exec}(s) \textit{ avec } \rho = \sigma \cdot \rho' \textit{ et } s \xmapsto{\sigma} s' \textit{ t.q.} \\ & s', v + \mathsf{Time}(\sigma) \models \psi \textit{ et } \forall\, s'' <_\rho s', \textit{ t.q. } \rho = \sigma' \cdot \rho'' \textit{ et} \\ & s \xmapsto{\sigma'} s'', \textit{ on a } s'', v + \mathsf{Time}(\sigma') \models \varphi \\ s, v \models \mathsf{A}\varphi\mathsf{U}\psi & \textit{ssi } \forall\, \rho \in \mathsf{Exec}(s)\ \exists \sigma \in \mathsf{Pref}(\rho) \textit{ t.q. } s \xmapsto{\sigma} s', \\ & s', v + \mathsf{Time}(\sigma) \models \psi \textit{ et } \forall\, s'' <_\rho s', \textit{ t.q. } s \xmapsto{\sigma'} s'', \\ & s'', v + \mathsf{Time}(\sigma') \models \varphi \end{array}$$

La propriété 2.1 s'écrit comme suit :

$$\mathsf{AG}\Big(\texttt{problème} \Rightarrow \Big(x \ \underline{\texttt{in}} \ \Big(\mathsf{AF}(x \leq 3 \wedge \texttt{alarme})\Big)\Big)\Big)$$

L'opérateur $\underline{\texttt{in}}$ remet l'horloge x à zéro lorsque l'on rencontre un état vérifiant `problème`, et il suffit donc de vérifier que $x \leq 3$ lorsqu'on rencontre un

état vérifiant `alarme` pour s'assurer que le délai séparant ces deux positions est bien inférieur à 3.

Clairement, l'utilisation des horloges de formules permet d'exprimer tous les opérateurs de *TCTL*. On a l'équivalence suivante lorsque φ et ψ sont des formules de *TCTL* [1] :

$$\mathsf{E}\varphi\mathsf{U}_{\sim c}\,\psi \;\equiv\; x \;\underline{\texttt{in}}\; \mathsf{E}\varphi\mathsf{U}(\psi \wedge x \sim c)$$

La logique $TCTL_h$ permet d'exprimer des propriétés très fines et était réputée être plus expressive que *TCTL* dans le temps dense [ACD93], ce résultat a été récemment prouvé [BCM05] : L'argument repose sur le fait que la formule suivante n'a pas d'équivalent en *TCTL* :

$$x \;\underline{\texttt{in}}\; \mathsf{EF}\Big(P_1 \wedge x < 1 \wedge \mathsf{EG}(x < 1 \Rightarrow \neg P_2)\Big)$$

Cette formule énonce le fait qu'il est possible d'atteindre un état vérifiant P_1 en moins de 1 unité de temps, à partir duquel il y a une exécution où il n'y a pas d'état vérifiant P_2 avant que x ne valle 1.

On peut aussi utiliser des *freeze variables* au lieu des horloges : une telle variable peut être instanciée avec la date associée à l'état courant, puis on peut comparer la différence entre ces variables et un entier. Ces deux approches (*freeze variables* ou horloges de formule) sont équivalentes.

L'ajout d'horloges peut aussi se faire dans des logiques de temps linéaire. Il existe notamment la logique *TPTL* [AH94] qui contient aussi des opérateurs du passé. En conservant nos notations, on peut définir $TLTL_h$.

Nous renvoyons à [AH92, ACD93, AH93, HNSY94] pour une présentation détaillée de plusieurs logiques temporelles quantitatives.

2.3.2 Logiques modales temporisées

Les logiques modales peuvent aussi être étendues pour exprimer des propriétés temps-réel. Dans ce cadre, nous nous intéressons aux étiquettes des transitions plutôt qu'aux propositions atomiques sur ces états [2].

Comme dans le cadre classique, par exemple dans la logique d'Hennessy et Milner [HM85], nous allons définir des modalités sur les transitions du STT. On distingue la quantification existentielle, de la forme $\langle a \rangle\, \varphi$ ("il est

[1] φ et ψ peuvent aussi être des formules de $TCTL_h$ si elles ne contiennent pas d'occurrence de l'horloge x hors de la portée d'un opérateur $\underline{\texttt{in}}$.

[2] Ajouter des propositions atomiques dans ces logiques ne changerait aucun résultat mentionné dans ce document.

possible de faire une transition a puis de vérifier φ"), et la quantification universelle, $[a]\ \varphi$ ("après toute transition étiquetée par a, φ est vérifié"). Nous utilisons aussi des opérateurs de point fixe pour énoncer des propriétés portant sur des comportements non bornés [Lar90]. Nous renvoyons à l'ouvrage de C. Stirling [Sti01] pour une présentation complète de ces logiques modales dans le cas non temporisé.

Les aspects quantitatifs, pour mesurer les délais séparant les actions du système étudié, sont traités à l'aide d'horloges de formule comme dans le cas de $TCTL_h$. Nous obtenons ainsi la logique modale $L_{\mu,\nu}$:

Définition 5 (Syntaxe de $L_{\mu,\nu}$) *Soient H un ensemble fini d'horloges de formules et* Id *un ensemble de variables de point fixe. Les formules de $L_{\mu,\nu}$ sont décrites par la grammaire suivante :*

$$\begin{array}{rcl} \varphi, \psi & ::= & \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle \alpha \rangle\, \varphi \mid [\alpha]\, \varphi \mid \mathtt{max}(X, \varphi) \mid \mathtt{min}(X, \varphi) \mid X \mid \\ & & x \sim c \mid x - y \sim c \mid x \ \underline{\mathtt{in}}\ \varphi \end{array}$$

où $\alpha \in \Sigma \cup \{\delta\}$, $x, y \in H$ et $X \in$ Id*. On se restreint aux formules* closes *où toute occurrence d'une variable $X \in$* Id *apparaît dans la portée d'un opérateur de point fixe* $\mathtt{min}(X, \varphi)$ *ou* $\mathtt{max}(X, \varphi)$*.*

Les formules $\top$, $\bot$ ou $g \Rightarrow \varphi$ (si g est une combinaison booléenne de contraintes d'horloges) font clairement partie de $L_{\mu,\nu}$.

Les formules de $L_{\mu,\nu}$ sont interprétées sur des *états étendus* (s, v) d'un système de transitions temporisé $T = \langle S, s_{\text{init}}, \rightarrow, l \rangle$: s est un état de S et v est une valuation pour H.

Comme pour le cas non temporisé, les opérateurs de point fixe permettent d'énoncer des propriétés faisant intervenir un nombre arbitrairement grand de transitions, ce qui est le cas de la plupart des propriétés qui nous intéressent. Un état étendu vérifie $\mathtt{max}(X, \varphi)$ (resp. $\mathtt{min}(X, \varphi)$) ssi il appartient à la plus grande (resp. la plus petite) solution de l'équation $X = \varphi$ sur le treillis complet des ensembles de configurations étendues. Chaque modalité de $L_{\mu,\nu}$ peut être vue comme un opérateur sur un ensemble d'états, l'absence de négation dans la définition de $L_{\mu,\nu}$ permet de garantir leur monotonicité et assure l'existence des points fixes.

Pour définir formellement la sémantique de $L_{\mu,\nu}$, nous utilisons des fonctions associant aux variables de Id des ensembles d'états étendus de S, une telle fonction est appelée un *environnement*. Soit ε un environnement et W

$$
\begin{array}{rcl}
[\![g]\!]_\varepsilon & \stackrel{\text{def}}{=} & \{(s,v) \mid v \models g\} \\
[\![\langle\alpha\rangle\,\varphi]\!]_\varepsilon & \stackrel{\text{def}}{=} & \{(s,v) \mid \exists\, s'.\ s \xrightarrow{t,\alpha} s' \text{ et } (s', v+t) \in [\![\varphi]\!]_\varepsilon\} \\
[\![[\delta]\,\varphi]\!]_\varepsilon & \stackrel{\text{def}}{=} & \{(s,v) \mid \forall\, s'.\ s \xrightarrow{t,\alpha} s' \ \Rightarrow\ (s', v+t) \in [\![\varphi]\!]_\varepsilon\} \\
[\![H' \ \underline{\texttt{in}}\ \varphi]\!]_\varepsilon & \stackrel{\text{def}}{=} & \{(s,v) \mid (s, v[H' \leftarrow 0]) \in [\![\varphi]\!]_\varepsilon\} \\
[\![X]\!]_\varepsilon & \stackrel{\text{def}}{=} & \varepsilon(X) \\
[\![\texttt{min}(X,\varphi)]\!]_\varepsilon & \stackrel{\text{def}}{=} & \bigcap\{W \mid [\![\varphi]\!]_{\varepsilon[X\mapsto W]} \subseteq W\} \\
[\![\texttt{max}(X,\varphi)]\!]_\varepsilon & \stackrel{\text{def}}{=} & \bigcup\{W \mid W \subseteq [\![\varphi]\!]_{\varepsilon[X\mapsto W]}\}\ .
\end{array}
$$

TAB. 2.1 – Sémantique de $L_{\mu,\nu}$

un ensemble d'états étendus de T, nous notons $\varepsilon[W/X]$ l'environnement qui associe à X l'ensemble W et équivaut à ε pour toute autre variable de Id. Le tableau 2.1 définit la sémantique des principales modalités de $L_{\mu,\nu}$: on associe à une formule (éventuellement non close) φ et un environnement ε l'ensemble (noté $[\![\varphi]\!]_\varepsilon$) des états vérifiant φ sous la condition que toute variable X de Id est vérifiée par les états de $\varepsilon(X)$. Clairement si φ est une formule close, alors $[\![\varphi]\!]_\varepsilon$ ne dépend pas de ε.

La logique $L_{\mu,\nu}$ permet d'exprimer des propriétés très fines sur le comportement des systèmes grâce à un contrôle précis des actions le long des exécutions. Bien sûr une contrepartie de cette expressivité réside dans le caractère moins naturel et moins lisible des formules obtenues. Notons aussi que le point de vue local imposé par les modalités de base fait qu'il n'est pas possible d'énoncer toutes les propriétés de *TCTL* (voir [HMP94] pour une discussion précise sur ces questions).

Voici quelques exemples de formules de $L_{\mu,\nu}$:

– La formule $\mathsf{AG}\varphi$ de *CTL* ("toujours φ") s'exprimera par :

$$\texttt{max}\Big(X, \varphi \wedge \bigwedge_{a\in\Sigma} [a]\,X \wedge [\delta]\,X\Big)$$

i.e. elle correspond les états vérifiant φ et à partir desquels toute transition d'action et toute transition de temps conduit à un état de X.

– La formule $\mathsf{EF}_{<10}\ \psi$ s'exprimera, lorsqu'on ne fait pas d'hypothèse particulière sur les exécutions, avec la formule suivante :

$$z\ \underline{\texttt{in}}\ \texttt{min}\Big(X, (\psi \wedge z < 10) \vee \big(\bigvee_{a\in\Sigma} \langle a\rangle\ X \vee \langle\delta\rangle\ X\big)\Big)$$

Autres logiques temporisées avec point fixe. Dans la littérature, il existe d'autres logiques modales temporisées que l'on peut définir comme des fragments de $L_{\mu,\nu}$:

- L_ν est une logique introduite dans [LLW95], elle correspond à $L_{\mu,\nu}$ privée de l'opérateur de plus petit point fixe. Dans [LLW95], nous avons montré qu'étant donné un automate temporisé A, il est possible de construire une formule φ_A caractérisant exactement le comportement de A vis-à-vis de la bisimulation forte temporisée (voir section 2.3.3), *i.e.* que tout automate vérifiant φ_A est fortement bisimilaire à A. Un semi-algorithme pour décider la satisfaisabilité des formules de L_ν y est présenté. Dans [LL95, LL98] nous avons proposé un algorithme de model checking compositionnel pour L_ν (voir section 4.7).
- Trois logiques L_s ("Logic for Safety"), *SBLL* (Safety and Bounded Liveness Logic") and $L_{\forall S}$ ont été définies (voir [LPY95, ABL98, ABBL98]) pour exprimer des propriétés de sureté et de vivacité bornée. On peut les voir comme des fragments de $L_{\mu,\nu}$. Trois restrictions sont alors faites : la disjonction est limitée au cas où l'un des deux termes est une contrainte d'horloges (*i.e.* à des formules du type "$g \vee \varphi$" avec $g \in \mathcal{C}(H)$), il n'y a pas de modalité $\langle\delta\rangle$, ni de modalité $\langle a\rangle$ sauf lorsqu'elle est suivie de $\top$. Une caractéristique de ces logiques est qu'il est possible, à partir de toute formule φ, de construire un *automate de test* B_φ tel que la vérification $A \models \varphi$ se ramène à un problème d'accessibilité dans la composition parallèle $(A|B_\varphi)$ [ABBL03].
- Dans [HMP94], un μ-calcul temporisé, T_μ a été défini pour exprimer des propriétés sur les automates temporisés. Une différence par rapport à $L_{\mu,\nu}$ est le fait qu'il contienne un opérateur de succession ($\triangleright$) au lieu des quantificateurs $\langle\alpha\rangle$ et $[\alpha]$. Cet opérateur est très puissant et permet d'exprimer de nombreuses propriétés. Cet article contient de plus une discussion sur la sémantique de ces logiques sur les systèmes temporisés (par exemple sur les exécutions non-Zénon), et sur la comparaison d'expressivité avec *TCTL*.

2.3.3 Bisimulation, simulation

Les équivalences comportementales permettent de comparer des systèmes temporisés. Ici nous nous contentons de mentionner le cas de la bisimulation forte temporisée. Deux états s_1 et s_2 sont fortement bisimilaires, $s_1 \approx s_2$, ssi :

- $l(s_1) = l(s_2)$
- pour toute transition $s_1 \stackrel{a,t}{\rightarrow} s'_1$, il existe une transition $s_2 \stackrel{a,t}{\rightarrow} s'_2$ avec $s'_1 \approx s'_2$.
- pour toute transition $s_2 \stackrel{a,t}{\rightarrow} s'_2$, il existe une transition $s_1 \stackrel{a,t}{\rightarrow} s'_1$ avec $s'_1 \approx s'_2$.

Une autre notion de bisimulation est utilisée dans le cas des automates temporisés, il s'agit de la bisimulation de temps abstrait (notée $\approx_{ta}$), où il n'est plus requis que les durées d'attentes soient les mêmes pour les deux composants ; il s'agit d'une bisimulation plus faible que $\approx$.

2.4 Complexité du model checking non-temporisé

Dans cette section, nous rappelons brièvement (et partiellement) les résultats de complexité pour le model checking non temporisé. Cela nous permet d'introduire quelques techniques classiques et sert de point de comparaison dans les chapitres suivants. Bien sûr ces résultats portent sur la vérification de propriétés non temporisées exprimées sous la forme de questions d'accessibilité, de formules de logiques temporelles ou de μ-calcul propositionnel.

Nous commençons par ceux portant sur un modèle décrit sous la forme d'une seule structure de Kripke $\mathcal{S}$ (on parle aussi de *systèmes plats*), puis nous considérons le cas du model checking des *systèmes non-plats* où le système à vérifier est décrit, en général, par une composition parallèle de structures de Kripke.

Nous distinguons la complexité générale (en fonction de $|\mathcal{S}|$ et $|\Phi|$), la complexité en programme (en fonction de $|\mathcal{S}|$ uniquement, $|\Phi|$ est alors supposée être constante) et la complexité en formule (en fonction de $|\Phi|$ uniquement, $|\mathcal{S}|$ est supposée être constante).

2.4.1 Model checking sur une structure de Kripke

Problèmes d'accessibilité. L'accessibilité d'un état de contrôle dans une structure de Kripke est un problème NLOGSPACE-complet (voir par

exemple [Pap94]). Ce résultat vaut aussi pour le test de vacuité du langage d'un automate de Büchi [VW94].

Logiques temporelles de temps linéaire. Le model checking pour la logique *LTL* (*i.e.* $L(\mathsf{X}, \mathsf{U})$ pour laquelle les formules sont construites à partir des opérateurs X et U) est PSPACE-complet [SC85]. En fait, cette complexité provient essentiellement de la formule à vérifier : la complexité en programme est NLOGSPACE-complet tandis que la complexité en formule est PSPACE-complet ; les algorithmes courants sont en temps $O(|\mathcal{S}| \cdot 2^{|\Phi|})$. Le model checking de *LTL* est un problème très étudié car *LTL* est, avec *CTL*, la logique temporelle la plus connue et la plus utilisée dans le domaine de la vérification. Les algorithmes de model checking procèdent de la manière suivante : d'abord ils construisent un automate $\mathcal{A}_{\neg\Phi}$ reconnaissant les modèles de $\neg\Phi$, puis il reste à tester le vide du langage associé à $\mathcal{S} \cap \mathcal{A}_{\neg\Phi}$: s'il est vide, le système $\mathcal{S}$ est correct, sinon il ne vérifie pas Φ (*i.e.* il existe des exécutions vérifiant $\neg\Phi$). Le problème central est de construire $\mathcal{A}_{\neg\Phi}$, on peut distinguer deux approches : soit on prend un automate de Büchi de taille exponentielle en $|\Phi|$ (en fait chaque état correspond à un sous-ensemble possible de sous-formules de Φ) [VW94], soit on construit un automate de Büchi *alternant* de taille linéaire en $|\Phi|$ (chaque état correspond à une sous-formule de Φ) [Var95]. Ces deux approches fournissent un algorithme optimal, *i.e.* polynomial en espace. De nombreux travaux ont porté sur la construction de $\mathcal{A}_{\neg\Phi}$ afin d'améliorer en pratique cette phase cruciale de l'algorihme (par exemple [GO01]).

L'ajout des opérateurs du passé (S pour "Since" et X^{-1} pour "Previous") ne change pas la complexité du model checking : vérifier une formule de *LTL*+Passé, *i.e.* $L(\mathsf{U}, \mathsf{X}, \mathsf{S}, \mathsf{X}^{-1})$, sur une structure de Kripke est un problème PSPACE-complet [SC85]. Des études fines de la complexité de nombreuses logiques du temps linéaire figurent aussi dans [DS02, Mar04].

Logiques temporelles de temps arborescent. La logique *CTL* a été très étudiée et il y eut très tôt des algorithmes simples de model checking (basés sur des algorithmes de graphes) pour cette logique [CES83, QS83]. Le problème se résout en temps $O(|\Phi| \cdot |\mathcal{S}|)$. L'algorithme procède en étiquetant les états de $\mathcal{S}$ par les sous-formules de Φ qui y sont vérifiées. Pour chaque opérateur de *CTL* ($\wedge$, $\neg$, EX, $\mathsf{E\_U\_}$ et $\mathsf{A\_U\_}$), on construit une procédure d'étiquetage en supposant que les sous-formules ont déjà été traitées. Cet algorithme est très simple à mettre en application et on peut facilement l'étendre pour d'autres logiques. Par exemple, pour construire un algorithme

pour la logique *ECTL* qui s'obtient à partir de *CTL* en ajoutant la modalité $\mathsf{E}\overset{\infty}{\mathsf{F}}$ ($\overset{\infty}{\mathsf{F}}\ P$ exprime l'existence, infiniment souvent sur un chemin, d'états vérifiant P), il suffit d'ajouter une procédure pour le nouvel opérateur.
La complexité en programme du model checking de *CTL* est NLOGSPACE-complet [KVW00] et celle en formule est LOGSPACE [Sch01].
Le model checking de la logique CTL^* (qui étend *LTL* et *CTL*) est un problème PSPACE-complet [EH86], il est directement lié à celui de *LTL*. Nous renvoyons aux travaux d'Emerson (par exemple [Eme90]) pour des synthèses complètes sur ces logiques temporelles.

μ-calcul. La complexité exacte du μ-calcul est un problème ouvert, on sait néanmoins que ce problème appartient à UP∩co-UP [Jur98]. La complexité de l'algorithme dépend fortement du nombre d'alternations de points fixes dans Φ. Par exemple le premier algorithme proposé [EL86] est en temps $O((|\Phi| \cdot |\mathcal{S}|)^{d+1})$ où d est le nombre d'alternations. On considère souvent le μ-calcul sans alternation [EL86] qui permet par exemple d'exprimer toutes les formules de *CTL* ou de caractériser la bisimulation, sa complexité est en $O(|\Phi| \cdot |\mathcal{S}|)$. Nous renvoyons à [EJS01, KVW00] pour une présentation de la complexité du model checking pour le μ-calcul.

Enfin, le problème du test de la bisimulation est un problème P-complet pour deux structures de Kripke. En fait toute relation comprise entre l'inclusion de traces et la bisimulation est P-difficile [Saw03].

Deux contributions. Dans le cadre de la complexité du model checking non temporisé, nous signalons deux contributions obtenues ces dernières années.

– La première porte sur le model checking de *FCTL* et de CTL^+ : *FCTL* est une extension de *CTL* avec une condition d'équité Ψ définie par une combinaison booléenne de formules $\overset{\infty}{\mathsf{F}}\ P$ (la sémantique des quantificateurs sur les chemins E ou A porte alors sur les chemins vérifiant la condition Ψ). La logique CTL^+ étend *CTL* en autorisant des formules de la forme $\mathsf{E}(P_1 \mathsf{U} P_2 \wedge P_1' \mathsf{U} P_2')$ où des opérateurs booléens peuvent être utilisés entre les quantificateurs de chemin et les modalités Until. Nous avons montré que le model checking pour ces deux logiques est Δ_2^p-complet [LMS01]. La classe Δ_2^p regroupe les problèmes que l'on résout en temps polynomial avec une machine de Turing ayant accès à un oracle NP. Ces deux problèmes étaient les premiers exemples de model checking Δ_2^p-complet. Nous reviendrons sur cette classe de complexité dans le chapitre 3.

– La seconde contribution porte sur le model checking de $LTL + \text{Passé} + \mathsf{N}$ (*i.e.* *NLTL*) où N est l'opérateur Now [LS95, LS00a] permettant d'"oublier" le passé d'une exécution (les opérateurs du passé contenus dans une formule $\mathsf{N}\varphi$ ne peuvent plus faire référence à des états situés dans le passé de l'état courant). Dans [LMS02b], nous avons montré que le model checking de *NLTL* était EXPSPACE-complet. L'opérateur Now induit donc un saut de complexité dans ce cas.

2.4.2 Model checking des systèmes non plats

Les résultats précédents concernent le model checking où le système à vérifier est donné sous la forme d'une simple structure de Kripke, mais en pratique on décrit souvent ces systèmes sous la forme d'une composition parallèle de processus communicants, ou d'un automate manipulant des variables booléennes (comme, par exemple, dans SMV [McM93]), des variables entières bornées, ou encore de réseaux de Petri bornés. Les nombreuses variantes reposent essentiellement sur l'idée de la composition de structures finies. La taille d'un tel système est alors la somme des tailles de ses composants et le coût de la composition (par le produit synchronisé) n'est donc pas intégré dans ce calcul. Dans ce cadre, on parle de model checking pour les programmes parallèles, concurrents ou non plats, ou parfois de model checking symbolique [3]. Un problème est modélisé par les différents composants $A_1, \ldots A_n$ et une propriété Φ, et on cherche à décider si $A_1 | \ldots | A_n \models \Phi$. La complexité est alors plus élevée que dans le cas d'une simple structure de Kripke, ce phénomène est appelé l'*explosion (combinatoire) du nombre d'états*. Ce problème est très important, ces limites théoriques sont effectivement ressenties en pratique, elles demandent la mise en place d'heuristiques particulières pour tenter de contourner ces difficultés pour des systèmes donnés.

L'accessibilité d'un état de contrôle et le model checking de *CTL*, *LTL* et CTL^* sont des problèmes PSPACE-complets pour les compositions parallèles de structures de Kripke [KVW00]. Le model checking du μ-calcul (avec ou sans alternation) devient un problème EXPTIME-complet [KVW00]. Enfin, verifier que deux compositions parallèles sont (bi)similaires est aussi EXPTIME-complet [JM96, HKV02].

[3] Ce terme peut prêter à confusion dans la mesure où il peut aussi désigner la technique de vérification basée sur des représentations symboliques des ensembles d'états.

Deux contributions. Dans le cadre de la complexité du model checking pour des systèmes concurrents, nous mentionnons les deux contributions suivantes :

- Dans [LS00b], nous montrons que la complexité de toute relation comprise entre la simulation et la bisimulation est EXPTIME-dure. Nous abordons aussi d'autres relations portant sur les traces (voir [Gla90] pour une présentation de ce domaine) et donnons des conditions suffisantes pour qu'elles soient EXPSPACE-dures.
- Dans [DLS02], nous analysons la complexité du model checking pour les systèmes concurrents selon les critères de la complexité paramétrée (voir par exemple [DF99]). Cette approche permet d'avoir une vue plus fine sur la complexité des problèmes, puisqu'on peut ainsi distinguer des problèmes qui appartiennent à la même classe de complexité dans l'approche classique. Cela peut expliquer pourquoi certains problèmes NP-complets admettent en pratique des algorithmes plus efficaces que d'autres. Nous avons montré que les problèmes de model checking étaient durs même dans le cadre de la complexité paramétrée. Ce résultat renforce donc les résultats précédents et montre combien l'explosion du nombre d'états est un problème fondamental.

Chapitre 3

Structures de Kripke avec durées

3.1 Introduction

Dans ce chapitre, nous nous intéressons exclusivement à des modèles de temps discret. Notre objectif est de proposer des modèles intégrant du temps quantitatif pour lesquels la vérification peut se faire de manière efficace, nous cherchons donc des modèles avec une temporisation plus simple que celle utilisée dans les automates temporisés.

La solution la plus simple pour intégrer les aspects quantitatifs sur les délais séparant les actions du système, est d'utiliser une structure de Kripke classique et de considérer que chaque transition du système prend exactement une unité de temps. Atteindre un état en moins de 10 u.t. revient à atteindre cet état en moins de 10 transitions. Cette approche naturelle a été utilisée par exemple dans [EMSS92, CCM$^+$94, CCG00].

Une autre solution consiste à utiliser des évènements pour mesurer le temps. Par exemple, une proposition *tick* peut étiqueter certains états pour marquer la progression du temps, le nombre de *tick* le long d'un chemin correspond alors à sa durée. Cela revient à avoir des transitions de durée zéro ou une unité de temps. C'est le choix fait dans [CTM$^+$99, LST00] ou encore dans [GHKK05] où ce point de vue est utilisé pour s'intéresser aux propriétés des langages temporisés.

Ces tentatives permettent effectivement d'associer une dimension temporisée au modèle mais elles restent assez limitées. Par exemple, la représentation d'une transition de durée 100 nécessiterait un modèle avec 100 transitions ! La modification de l'unité de temps entraînerait une explosion

(ou une compression) exponentielle de la taille du système. De plus, dans cette approche on mélange intimement la partie contrôle du système (les transitions) et le mécanisme de temporisation, cela n'est pas toujours naturel, ni souhaitable. Par exemple, il peut être utile d'extraire la partie contrôle sans aucune temporisation ; cela représente une abstraction particulière du modèle initial.

Nous allons généraliser les démarches décrites ci-dessus en introduisant les *structures de Kripke avec durées (SKD)* où chaque transition se voit associer un intervalle correspondant aux durées possibles (entières) qu'elle peut prendre.

Nous nous intéressons à la complexité des différents problèmes de model checking, et plus particulièrement au coût induit par les contraintes temporelles. Cela nous a conduit à ne pas considérer le cas des compositions parallèles : nous verrons que la complexité du model checking des systèmes non-plats sans aspects quantitatifs est la même que la composition parallèle d'automates temporisés (cf. le chapitre 4) !

Enfin le cas des automates temporisés sur le domaine de temps $\mathbb{N}$ n'est pas non plus considéré ici car en général, il ne permet pas d'algorithmes plus efficaces que le temps dense (nous évoquerons ces questions dans le chapitre 4). Notons néanmoins que du point de vue de l'expressivité, considérer $\mathbb{N}$ comme domaine de temps change beaucoup puisqu'un automate temporisé avec une seule horloge peut reconnaître tout langage temporisé reconnu par un automate avec n horloges [HKWT95].

3.2 Structure de Kripke avec durées

Le domaine $\mathbb{T}$ est l'ensemble des entiers naturels $\mathbb{N}$. Nous définissons les structures de Kripke avec durées puis nous considérerons plusieurs sémantiques possibles en terme de STT.

Notation. On note $\mathcal{I}_{\mathbb{N}}$ l'ensemble des intervalles sur $\mathbb{N}$. Un intervalle ρ de $\mathcal{I}_{\mathbb{N}}$ est soit fini (*i.e.* de la forme $[n, m]$) soit ouvert à droite et infini (de la forme $[n, \infty[$).

Définition 6 *Une structure de Kripke avec durées est un quadruplet $\mathcal{S} = \langle Q, q_{init}, R, l\rangle$ où Q est un ensemble d'états, q_{init} est l'état initial, $R \subseteq Q \times \mathcal{I} \times Q$ est une relation de transition avec durée et $l : Q \to 2^{\mathsf{Prop}}$ étiquette chaque état avec un sous-ensemble de propositions atomiques.*

La taille $|\mathcal{S}|$ de $\mathcal{S}$ est $|Q|+\Sigma_{t\in R}|t|$ où la taille d'une transition $(q,[a;b),q')$ est $1+\lceil\log(a+1)\rceil+\lceil\log(b+1)\rceil$ avec la convention $\lceil\log(\infty)\rceil=1$.

La figure 3.1 décrit une structure de Kripke avec durées modélisant l'activité d'un chercheur. On peut prendre le jour comme unité de temps et pour simplifier le nom des états fait ici office de proposition atomique.

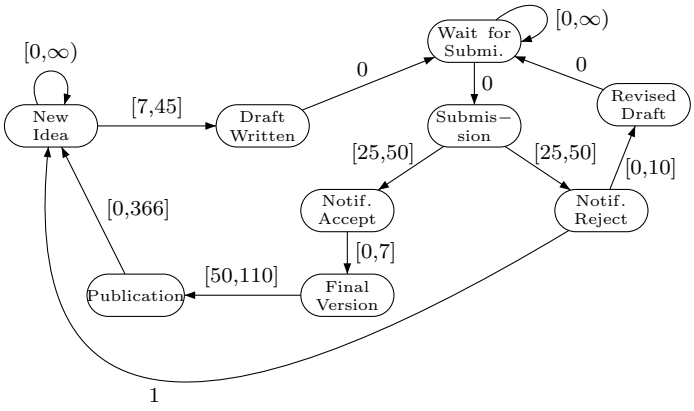

FIG. 3.1 – Une SKD modélisant l'activité d'un chercheur

Sémantique. À partir de la définition syntaxique des SKD, il est possible de définir plusieurs sémantiques à base de systèmes de transitions temporisés. En effet, une transition de q à q' avec une durée d peut s'interpréter au moins de trois manières :

- Passer de q à q' prend t unités de temps sans qu'il n'y ait d'états intermédiaires lors du franchissement de la transition. Si l'on se trouve à l'instant θ dans l'état q, on se retrouve à l'instant $\theta+t$ en q' et les instants $\theta+1,\ldots,\theta+t-1$ ne sont pas représentés : ils n'existent pas. Nous parlerons de *sémantique de saut (s)*. C'est la sémantique utilisée pour les Timed Transition Graph (TTG) dans [CC95].
  Selon cette sémantique, on peut voir les durées plutôt comme des coûts : on associe à chaque transition un certain coût et cette notion est étendue aux exécutions (ensembles de transitions). On peut ainsi parler d'automates à coûts pour la sémantique de saut. En suivant cette idée, on pourrait aussi associer aux états un coût et les considérer lors de l'évaluation du coût des exécutions, cela ne changerait pas les résultats présentés dans ce document.
- Passer de q à q' prend t unités de temps et le système traverse $t-1$ états intermédiaires *entre ces deux états* : le système est alors engagé le long de cette transition. On supposera que les états intermédiaires sont

étiquetés par les mêmes propositions atomiques que l'état de départ q. Nous parlerons de *sémantique continue avec états intermédiaires (cei).*

– Passer de q à q' est possible après avoir attendu $t-1$ unités de temps dans l'état q. Cette sémantique rejoint celle des automates temporisés que nous verrons ultérieurement. Nous parlerons de *sémantique continue avec attente (ca).*

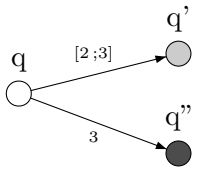

FIG. 3.2 – Exemple de structure de Kripke avec durées.

Considérons la SKD S de la figure 3.2, la figure 3.3 illustre les trois sémantiques possibles présentées ci-dessus pour S.

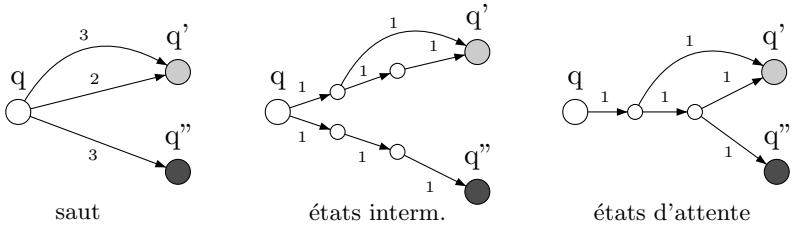

FIG. 3.3 – Les trois sémantiques des SKD.

Nous allons maintenant définir formellement ces différentes sémantiques en terme de systèmes de transitions temporisés. Dans ce chapitre, nous nous intéresserons au model checking où la propriété à vérifier est énoncée avec une logique temporelle, et donc seul l'étiquetage des états par des propositions atomiques nous intéresse. On suppose que Σ est restreint à $\{a\}$: Le symbole δ marque les transitions de temps (pour les sémantiques continues) et une unique action a est utilisée pour les autres transitions.

La sémantique de saut. Une structure de Kripke avec durées $\mathcal{S}$ définit alors un STT $T_s(\mathcal{S}) = \langle S, s_{\text{init}}, \rightarrow, l \rangle$ avec :

– $S = Q$ et $s_{\text{init}} = q_{\text{init}}$.

– $s_1 \xrightarrow{t,a} s_2$ ssi il existe $(s_1, \rho, s_2) \in R$ et $t \in \rho$.

– les fonctions d'étiquetage de $\mathcal{S}$ et $T_s(\mathcal{S})$ coïncident.

En fait, la différence entre $\mathcal{S}$ et $T_s(\mathcal{S})$ réside uniquement dans le remplacement des transitions avec intervalles ρ par des transitions avec une durée fixée. Le nombre de transitions peut être infini s'il existe des intervalles ouverts, mais le nombre d'états est fini. Nous n'imposons pas de conditions particulières pour la notion d'exécution : $\mathsf{Exec}(s)$ contient tout chemin infini de $T_s(\mathcal{S})$ issu de s.

La sémantique continue avec états intermédiaires. Étant donnée une transition (q, ρ, q') de $\mathcal{S}$, on note $\delta_{\max}(q \xrightarrow{\rho} q')$ la durée maximale que peut prendre cette transition (*i.e.* u si $\rho = [l; u]$ et ∞ si ρ est infini). Selon la sémantique avec états intermédiaires, $\mathcal{S}$ définit un STT $T_{cei}(\mathcal{S}) = \langle S, s_{\text{init}}, \rightarrow, l \rangle$ avec :

– $S = Q \cup \{(q \xrightarrow{\rho} q', i) \mid (q, \rho, q') \in R \ \wedge \ 1 \leq i < \delta_{\max}(q \xrightarrow{\rho} q')\}$ et $s_{\text{init}} = q_{\text{init}}$.
– La relation de transition $\rightarrow$ est définie par :
  – $q \xrightarrow{0,a} q'$ si $\exists (q, \rho, q') \in R$ et $0 \in \rho$,
  – $q \xrightarrow{1,a} q'$ si $\exists (q, \rho, q') \in R$ et $1 \in \rho$,
  – $q \xrightarrow{1,a} (q \xrightarrow{\rho} q', 1)$ if $1 < \delta_{\max}(q \xrightarrow{\rho} q')$,
  – $(q \xrightarrow{\rho} q', i) \xrightarrow{1,\delta} (q \xrightarrow{\rho} q', i+1)$ si $i+1 < \delta_{\max}(q \xrightarrow{\rho} q')$,
  – $(q \xrightarrow{\rho} q', i) \xrightarrow{1,\delta} q'$ si $i+1 \in \rho$.
– Les états q de T_{cei} sont étiquetés comme dans $\mathcal{S}$ et ceux de la forme $(q \xrightarrow{\rho} q', i)$ sont étiquetés par $l(q)$.

On peut distinguer trois types de transitions :

– celles issues d'un état de la forme q et *choisissant* une transition d'action (étiquetées avec a).
– celles menant d'un état intermédiaire à un nouvel état intermédiaire, il s'agit de transitions *d'attente* (étiquetées avec δ).
– celles menant à un nouvel état de contrôle, *terminant* ainsi la transition de $\mathcal{S}$ (étiquetées avec δ).

Lorsqu'il existe des transitions avec intervalles ouverts dans $\mathcal{S}$, le nombre d'états et de transitions de $T_{cei}(\mathcal{S})$ est infini. Une exécution de $T_{cei}(\mathcal{S})$ doit passer infiniment souvent par des états de Q, nous interdisons par là de rester le long d'une transition (*i.e.* dans ses états intermédiaires) pendant un temps infini : on interprète une transition $\xrightarrow{[l;\infty)}$ comme une transition de durée arbitrairement grande, mais finie. On intègre donc dans $\mathsf{Exec}(s)$ cette condition de Büchi.

Comparée à la sémantique de saut, celle-ci dispose d'une caractéristique importante : le temps progresse continûment, *i.e.* toute exécution finie de durée c peut se décomposer en un préfixe de durée $\lfloor \frac{c}{2} \rfloor$ et un suffixe de durée $\lceil \frac{c}{2} \rceil$ (cette décomposition n'est pas toujours unique).

La sémantique continue avec attente. Pour tout état q de $\mathcal{S}$, on note $\delta_{\max}(q)$ le temps d'attente maximum dans q selon les transitions de $\mathcal{S}$: si l'intervalle associé à une transition issue de q est infini, alors $\delta_{\max}(q)$ est ∞, sinon c'est la plus grande borne contenue dans ces intervalles. Selon la sémantique continue d'attente, $\mathcal{S}$ définit un STT $T_{ca}(\mathcal{S}) = \langle S, s_{\text{init}}, \rightarrow, l \rangle$ avec :

- $S = \{(q,i) \mid 0 \leq i < \delta_{\max}(q)\}$ et $s_{\text{init}} = (q_{\text{init}}, 0)$.
- La relation $\rightarrow$ est définie par :
  - $(q,0) \xrightarrow{0,a} (q',0)$ si $\exists (q,\rho,q') \in R$ et $0 \in \rho$,
  - $(q,i) \xrightarrow{1,\delta} (q,i+1)$ si $i+1 < \delta_{\max}(q)$,
  - $(q,i) \xrightarrow{1,a} (q',0)$ si $\exists (q,\rho,q') \in R$ et $i+1 \in \rho$.
- Les états (q,i) sont étiquetés par $l(q)$.

Comme pour la sémantique précédente, nous exigeons des exécutions qu'elles ne bouclent pas infiniment avec des transitions d'attente : nous imposons donc que les éléments de $\mathsf{Exec}(s)$ contiennent infiniment souvent des états de $Q \times \{0\}$.

Il est donc possible d'attendre dans un état un certain délai **puis** de choisir une transition et d'arriver alors dans un nouvel état de contrôle. Le choix de la transition est donc plus fait plus tard que dans la sémantique précédente, on est proche de la sémantique des automates temporisés (plus précisément, cela correspond à des automates temporisés sur $\mathbb{N}$ avec une horloge remise à zéro après chaque transition d'action). Là encore le temps progresse de manière continue.

Classes particulières de structures de Kripke avec durées. Le premier fragment intéressant regroupe les SKD où chaque transition a un intervalle de la forme $[0;0]$, $[0;1]$ ou $[1;1]$: c'est la classe des SKD "à petits pas", on note cette famille $\text{SKD}^{0/1}$. Elle comprend notamment les structures de Kripke où chaque transition est supposée avoir une durée 1 ou les structures de Kripke avec *tick*. Une importante propriété des $\text{SKD}^{0/1}$est que les trois sémantiques coïncident en un même système de transition temporisé et leurs caractéristiques respectives s'ajoutent. Ainsi le temps progresse de manière continue (sans saut), le nombre d'états du STT associé est $|Q|$ et donc toute exécution finie et simple (*i.e.* sans boucle) a une durée bornée par

$|Q|$. Nous verrons que ces propriétés permettent l'existence d'algorithmes efficaces pour le model checking.

On pourrait aussi considérer les structures de Kripke avec durées où les transitions ne contiennent que des intervalles réduits à des singletons. En fait, cette restriction ne change pas la complexité des problèmes de vérification.

Composition parallèle. Comme mentionné dans le chapitre précédent, il est utile de disposer d'un opérateur de composition parallèle pour modéliser des systèmes complexes. Le temps devant s'écouler à la même vitesse dans tous les composants, les transitions synchronisées dans les différents STT doivent comporter la même durée. Cela ne pose pas de problème pour les sémantiques continues : la sémantique étant définie en terme de STT à durée 0 ou 1.

Pour la sémantique de saut, la synchronisation entre deux SKD A et B est difficile à définir : supposons le cas où A puisse faire une transition de durée 3 et B une transition de durée 4 sans qu'aucune synchronisation ne soit requise : la position de $(A|B)$ après 3 ou 4 unités de temps ne se définit pas en fonction des configurations respectives de A et B. En adoptant le point de vue automate à coût, on pourrait décider d'ajouter les coûts associés aux actions, mais nous sommes alors loin d'une sémantique temporisée.

En conclusion, seules les sémantiques continues supportent la mise en parallèle, la sémantique de saut nécessite des hypothèses plus fortes pour permettre ce type de construction.

Propriétés des SKD pour *TCTL*. Lorsque le domaine de temps est $\mathbb{N}$, nous avons les équivalences [1] suivantes pour les formules de *TCTL* :

$$\begin{array}{rcl} \mathsf{E}\,\varphi\,\mathsf{U}_{<i+1}\ \psi & \equiv & \mathsf{E}\,\varphi\,\mathsf{U}_{\leq i}\ \psi \\ \mathsf{A}\,\varphi\,\mathsf{U}_{<i+1}\ \psi & \equiv & \mathsf{A}\,\varphi\,\mathsf{U}_{\leq i}\ \psi \end{array}$$

Les équivalences suivantes ne sont vraies que pour des modèles où le temps progresse continûment, c'est-à-dire lorsque le STT sur lequel les formules sont interprétées, est un STT à petits pas :

$$\begin{array}{rcl} \mathsf{E}\,\varphi\,\mathsf{U}_{\leq i+1}\ \psi & \equiv_{pp} & \mathsf{E}\,\varphi\,\mathsf{U}_{\leq 1}\left(\mathsf{E}\,\varphi\,\mathsf{U}_{\leq i}\ \psi\right) \\ \mathsf{A}\,\varphi\,\mathsf{U}_{\leq i+1}\ \psi & \equiv_{pp} & \mathsf{A}\,\varphi\,\mathsf{U}_{\leq 1}\left(\mathsf{A}\,\varphi\,\mathsf{U}_{\leq i}\ \psi\right) \end{array}$$

Elles sont donc vraies pour les SKD avec les sémantiques continues ou pour les $\mathrm{SKD}^{0/1}$. Mais elle ne sont pas vérifiées pour les SKD avec la séman-

[1] Ces équivalences ne sont plus vraies lorsque l'on considère un domaine de temps dense.

tique de saut : un chemin de durée 10 n'est pas toujours décomposable en un préfixe de longueur 1 et un suffixe de longueur 9.

3.3 Structures de Kripke avec durées 0 ou 1

Nous commençons par considérer le cas des SKD à petits pas. Nous présentons les résultats de complexité du model checking lorsque la propriété à vérifier est exprimée avec $TCTL$ ou $TCTL_h$.

3.3.1 Model checking $TCTL$

Lorsque les durées apparaissant dans les modèles sont 0 ou 1, le model checking de $TCTL$ peut se faire de manière efficace :

Théorème 1 ([LST03]) *Étant donnée $\mathcal{S} = \langle Q, q_{init}, R, l\rangle$ une structure de Kripke à durées 0 ou 1 :*
– *vérifier une propriété Φ de $TCTL$ peut se faire en temps $O(|Q|^3 \cdot |\Phi|)$.*
– *vérifier une propriété Φ de $TCTL_{\leq,\geq}$ peut se faire en temps $O(|\mathcal{S}| \cdot |\Phi|)$.*

Voici comment procède l'algorithme pour $\xi \stackrel{\text{def}}{=} \mathsf{E}\varphi\mathsf{U}_{<c}\psi$. Tout d'abord il est possible de remplacer c par $\min(c, |Q|)$. Ensuite il suffit d'étiqueter par ξ les états vérifiant $\mathsf{E}\varphi\mathsf{U}_{<i}\psi$ pour $i = 1$, puis pour $2, \ldots, c$. À l'étape i, on utilise une procédure simple de parcours des états en veillant à ne pas s'occuper des états déjà étiquetés par ξ. Cela permet donc une complexité linéaire en $\mathcal{S}$.

Pour $\xi \stackrel{\text{def}}{=} \mathsf{E}\varphi\mathsf{U}_{=c}\psi$, on ne peut plus borner la constante c. Dès lors, un calcul des $\mathsf{E}\varphi\mathsf{U}_{=i}\psi$ pour $i = 1$ jusqu'à c aurait un coût exponentiel en $|\xi|$. L'idée de l'algorithme polynomial consiste à construire un ensemble de relations R_d telles que $(q, q') \in R_d$ ssi il existe un chemin menant de q à q' en exactement d unités de temps. Le calcul des R_d se fait efficacement via les propriétés suivantes : $R_{2d} = R_d \circ R_d$ et $R_{2d+1} = R_d \circ R_d \circ R$. Le calcul de R_c nécessite donc $\lceil \log_2(c+1) \rceil$ compositions. Le coût des opérations de composition et du calcul des fermetures transitives est en $O(|Q|^3 + |R|)$. Bien sûr, dans cette approche, on tient aussi compte des étiquetages des états avec φ et ψ. Finalement on obtient donc un algorithme en $O(|Q|^3 \cdot |\Phi|)$.

Notons que ce résultat est très proche de celui de [EMSS92] pour $RTCTL$ qui correspond à $TCTL_<$, et $CRTCTL$ — équivalent à $TCTL$ — pour les structures de Kripke où chaque transition prend la durée 1.

Ce résultat — un algorithme polynomial pour $TCTL$ — est unique dans notre étude. Pour les autres modèles aussi bien dans le temps discret que

dans le temps dense, la présence des modalités avec les contraintes "$= c$" entraîne systématiquement un saut de complexité important.

3.3.2 Model checking $TCTL_h$

Lorsque l'on considère $TCTL_h$, c'est-à-dire avec des horloges dans les formules, on obtient immédiatement un saut de complexité important :

Théorème 2 ([LST03]) *Le model checking de $TCTL_h$ sur les structures de Kripke à durées 0 ou 1 est un problème PSPACE-complet.*

En fait, la complexité en formule de ce problème est déjà PSPACE-complet. Le coté PSPACE-difficile de cette preuve se fait en réduisant une instance de QBF à un problème de model checking pour $TCTL_h$ sur une structure de Kripke fixée comportant un unique état et une boucle de durée 1. La formule utilise des horloges comme des variables pour stocker la valeur des propositions atomiques de la formule de QBF.

Pour montrer que le problème est dans PSPACE, il suffit de le réduire à un problème de model checking CTL (non temporisé) sur une composition parallèle de structures de Kripke. L'idée de cette réduction est la suivante. Considérons une SKD$^{0/1}S$ et une formule Φ de $TCTL_h$. Une configuration du problème est un état de contrôle et une valuation pour les N_h horloges apparaissant dans Φ. Si M est la plus grande constante entière apparaissant dans Φ, il suffit de considérer des valuations à $M+2$ valeurs : $0, 1, 2, \ldots, M$, $>M$ pour décider de la valeur de vérité des contraintes $x \sim k$. Une telle valeur peut elle-même se coder sous la forme d'une composition parallèle de $\lceil \log(M+2) \rceil$ automates à deux états. Pour les contraintes $x - y \sim k$, il suffit de considérer des booléens, *i.e.* des automates à deux états, pour coder leur valeur en fonction des remises à zéro de x et y. On note N_{cd} le nombre de contraintes diagonales distinctes présentes dans Φ. On peut donc réduire $S \models \Phi$ à un problème de model checking sur une composition parallèle de $1 + N_{cd} + N_h \times \lceil \log(M+2) \rceil$ automates (synchronisés de manière à intégrer la progression des compteurs) avec la formule Φ où les contraintes "$x \sim k$" sont vues comme des propriétés locales à vérifier sur les automates correspondant à x et où les opérateurs x `in` ... réinitialisent les automates de x.

Ce résultat est important dans la mesure où il montre combien l'ajout d'horloges dans les formules est un mécanisme puissant qui entraîne un saut de complexité même lorsque la temporisation du modèle est réduite au cas le plus simple. Nous verrons que nous obtenons la même classe de complexité lorsque l'on considère $TCTL_h$ et une composition parallèle d'automates temporisés !

3.4 SKD avec sémantique de saut

Nous nous intéressons maintenant aux structures de Kripke avec durées munies de la sémantique de saut. L'introduction de valeurs entières arbitrairement grandes rend plus difficile le model checking. Le premier résultat concerne l'accessibilité en un temps exact :

Proposition 1 (Accessibilité en temps exact [LMS02a]) *Vérifier les propriétés de la forme* $\mathsf{EF}_{=c}\, P$ *sur une structure de Kripke avec durées munie de la sémantique de saut est un problème NP-dur.*

Pour expliquer cette proposition, considérons une instance du problème SUBSET-SUM [GJ79, p. 223] : étant donnés un ensemble fini d'entiers $A = \{a_1, \ldots, a_n\}$ et un entier D, on cherche s'il existe un sous-ensemble A' de A tel que la somme des éléments de A' soit D. Un tel problème se réduit clairement à l'accessibilité, en temps D, de l'état étiqueté par P dans la SKD de la figure 3.4.

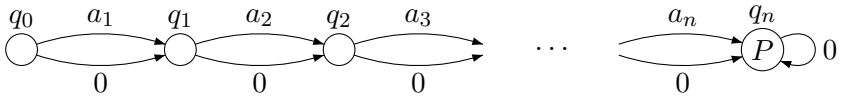

FIG. 3.4 – SKD pour coder SUBSET-SUM

Après ce premier résultat, nous considérons le model checking de $TCTL_{\leq,\geq}$ et montrons qu'il peut se traiter de manière efficace, puis nous reviendrons sur la complexité exacte de *TCTL*.

3.4.1 Model checking $TCTL_{\leq,\geq}$

Sans les contraintes exactes dans les modalités, le model checking peut se faire de manière efficace :

Théorème 3 ([LMS02a]) *Étant donnée une structure de Kripke* $\mathcal{S}$ *et une formule* Φ *de* $TCTL_{\leq,\geq}$*, décider* $T_s(\mathcal{S}) \models \Phi$ *peut se faire en temps* $O(|\mathcal{S}|^2 \cdot |\Phi|)$.

L'algorithme pour $TCTL_{\leq,\geq}$ étend l'algorithme classique de *CTL*. Notons qu'avec la sémantique de saut, il suffit d'étiqueter les états de Q : il n'y a pas d'état supplémentaire. Étant donné un étiquetage des états par φ et ψ, nous pouvons déterminer l'ensemble d'états vérifiant $\mathsf{E}\varphi\mathsf{U}_{\leq c}\psi$ et ceux vérifiant $\mathsf{E}\varphi\mathsf{U}_{\geq c}\psi$ de la manière suivante :

$\xi \stackrel{\text{def}}{=} \mathsf{E}\varphi\mathsf{U}_{\leq c}\psi$: Il suffit de considérer S' la sous-SKD composée des états vérifiant φ ou ψ et dont les durées des transitions sont limitées aux bornes inférieures des intervalles des transitions de $\mathcal{S}$: pour vérifier ξ il n'est jamais *utile* de prendre une transition d'une durée supérieure. Dès lors, tester si un état q vérifie $E\varphi\mathsf{U}_{\leq c}\psi$ consiste à vérifier qu'il existe un plus court chemin dans S' allant de q à un état vérifiant ψ de durée inférieure ou égale à c. Un simple algorithme de plus courts chemins (en $O(|Q_{S'}| \cdot | \rightarrow_{S'} |)$) fournit le résultat.

$\xi \stackrel{\text{def}}{=} \mathsf{E}\varphi\mathsf{U}_{\geq c}\psi$: Nous avons à distinguer le cas où un état vérifie ξ parce qu'il existe un chemin simple (c.-à-d. où tous les états sont différents) menant à ψ et de durée supérieure ou égale à c, et le cas où il est possible d'itérer une boucle (de durée strictement positive) vérifiant φ. Le premier cas revient à chercher des plus longs chemins dans un graphe acyclique, et le second se traite à l'aide d'une formule de *CTL* munie d'une proposition atomique étiquetant les états appartenant à une composante fortement connexe vérifiant φ et contenant au moins une transition de durée supérieure à 1.

Les autres modalités se traitent de manière similaire.

3.4.2 Model checking *TCTL*

Nous avons vu que les contraintes exactes entraînent un saut de complexité, nous pouvons maintenant mieux l'évaluer :

Théorème 4 ([LMS02a]) *Le model checking de TCTL sur les structures de Kripke avec durées munies de la sémantique de saut est un problème Δ_2^p-complet.*

Ce résultat est intéressant dans la mesure où peu de problème de model checking appartiennent à cette classe de complexité, correspondant aux problèmes que l'on résout en temps polynomial en utilisant des appels à un oracle NP (Δ_2^p est aussi noté P^{NP}).

L'appartenance à Δ_2^p repose sur l'existence d'une procédure NP pour décider si un état q vérifie $\mathsf{E}\varphi\mathsf{U}_{=c}\psi$. L'idée est d'utiliser l'image de Parikh de l'exécution candidate π : on ne retient que le nombre de fois que chaque transition apparaît dans π. Cela est suffisant pour vérifier qu'il s'agit d'une exécution bien formée (*cf.* le critère d'Euler) et que sa longueur peut être égale à c. Nous avons donc un témoin polynomial associé à π dans la mesure où le nombre de transitions est borné par $c \cdot |Q|$ (on élimine les boucles de durées nulles).

La preuve de dureté se base sur la réduction d'un problème de satisfaisabilité, SNSAT (*Sequentially Nested Satisfiability*), Δ_2^p-complet. Ce problème a été introduit dans [LMS01], nous nous contentons de le présenter ici et nous renvoyons à [LMS02a] pour la réduction de SNSAT au model checking *TCTL*. Une instance de SNSAT est définie par un ensemble de définitions de la forme :

$$\mathcal{I} = \left[\begin{array}{lcl} x_1 & := & \exists Z_1\ F_1(Z_1), \\ x_2 & := & \exists Z_2\ F_2(x_1, Z_2), \\ & \vdots & \\ x_n & := & \exists Z_n\ F_n(x_1, \ldots, x_{n-1}, Z_n) \end{array} \right]$$

où chaque F_i est une formule booléenne (par ex. une 3-CNF) sur l'ensemble de propositions $Z_i \cup \{x_1, \ldots, x_{i-1}\}$. L'instance $\mathcal{I}$ définit une unique valuation pour les x_i : x_i est vrai ssi il existe une valuation pour les variables de Z_i telle que $F_i(x_1, \ldots, x_{i-1}, Z_i)$ soit équivalente à vrai, la valeur de x_i dépend donc de celles de $x_1, \ldots, x_{i-1}$. L'instance $\mathcal{I}$ est une instance positive ssi la valeur associée à x_n est vrai. Ce problème est donc une succession de problèmes de satisfaisabilité qui doivent être résolus les uns après les autres.

Dans [MS04], un algorithme symbolique (à base de BDD) pour le model checking de *TCTL* sur des variantes de SKD a été proposé et implémenté dans nuSMV.

3.5 SKD avec sémantique continue

À la section 3.2, nous avons défini deux sémantiques continues pour les SKD. En fait, les problèmes de model checking pour la sémantique avec états intermédiaires peuvent se ramener à du model checking avec la sémantique avec états d'attente. L'idée est simplement de décomposer toute transition $q \xrightarrow{\rho} q'$ en $q \xrightarrow{1} r$ et $r \xrightarrow{\rho-1} q'$ où $\rho - 1$ désigne $[\max(0, a-1); b-1]$ si $\rho = [a; b]$: l'attente dans le nouvel état r permet de simuler les états intermédiaires de la sémantique cei. La figure 3.5 illustre cette construction sur un exemple. En fait nous verrons que ces deux sémantiques engendrent les mêmes complexités.

3.5.1 Model checking $TCTL_{\leq,\geq}$

Le model checking de $TCTL_{\leq,\geq}$ peut se faire en temps polynomial lorsqu'on considère la sémantique continue avec états d'attente :

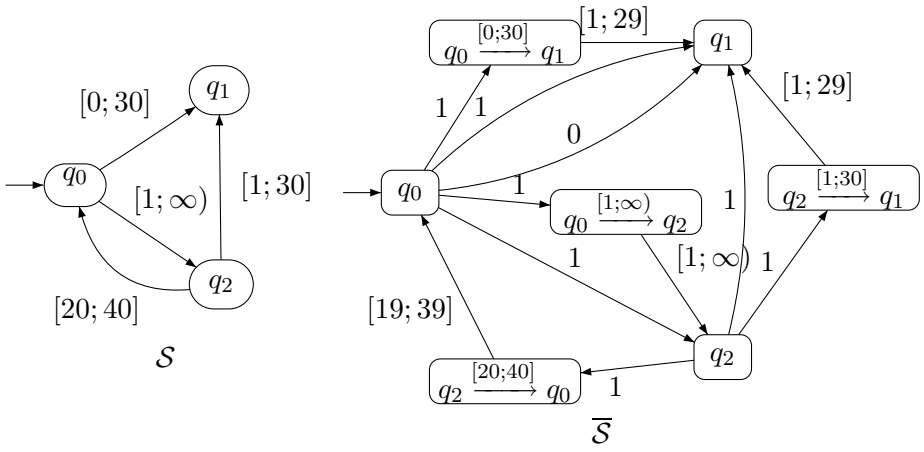

FIG. 3.5 – Réduction de la sémantique avec états intermédiaires vers celle avec états d'attente

Théorème 5 ([LMS05]) *Étant donnée une structure de Kripke $\mathcal{S}$ et une formule Φ de $TCTL_{\leq,\geq}$, décider $T_{cea}(\mathcal{S}) \models \Phi$ peut se faire en temps $O\big(|\mathcal{S}|^3 \cdot |\Phi|^3\big)$.*

Décider de la valeur de vérité de Φ sur $T_{cea}(\mathcal{S})$ nécessite de connaître la valeur de vérité des sous-formules sur tous les états de $T_{cea}(\mathcal{S})$, y compris les états d'attente de la forme (q, i). Pour ce faire, nous pouvons construire, pour tout état q et toute sous-formule φ, un ensemble $\mathsf{Sat}[q, \varphi]$ d'intervalles $\bigcup_j [\alpha_j; \beta_j[$ décrivant les positions i telles que (q, i) vérifie φ. À partir de ces ensembles, on utilise des procédures pour calculer $\mathsf{Sat}[\_, \_]$ pour les différents opérateurs de $TCTL_{\leq,\geq}$.

Considérons par exemple $\xi \stackrel{\text{def}}{=} \mathsf{E}\varphi\mathsf{U}_{<c}\psi$. Dans ce cas, l'idée est de calculer la distance minimale menant à un état vérifiant ψ (le long d'un chemin satisfaisant φ) pour un *sous-ensemble pertinent et de taille polynomiale* de configurations (q, i). Par *pertinent*, on entend le fait que connaître ces distances minimales pour les configurations de ce sous-ensemble suffit pour en déduire facilement les ensembles $\mathsf{Sat}[q, \xi]$ pour tout q.

À partir de $\mathsf{Sat}[q, \varphi]$, $\mathsf{Sat}[q, \psi]$ et des intervalles associés aux transitions issues de q, on partitionne les positions i de l'état q en une séquence d'intervalles adjacents tels que (1) le premier intervalle est $[0; 1[$, (2) chaque intervalle est homogène pour les valeurs de vérité de φ et ψ et (3) les posi-

tions de chaque intervalle peuvent exécuter les mêmes transitions d'action dans $T_{cea}(\mathcal{S})$. On considère le plus petit découpage vérifiant les conditions ci-dessus et on note $L(q)$ la liste des intervalles associés à q.

Étant donné un intervalle $[a; b[$ de $L(q)$, nous avons la propriété suivante : si la distance minimale à un état ψ est λ pour (q, a), alors la distance minimale pour $(q, b-1)$ vaut soit λ, soit la distance minimale pour (q, b) incrémentée de 1 : tout plus court chemin depuis (q, a) commence soit par une transition d'action (et il est donc aussi possible depuis $(q, b-1)$ avec le même coût) soit par une attente jusqu'à (q, b) — il n'est jamais intéressant pour un plus court chemin de faire une transition d'action au milieu d'un intervalle.

À partir des $L(q)$, on construit une SKD restreinte aux configurations (q, a) où a est une borne gauche d'un intervalle de $L(q)$. Cette SKD contient des transitions d'action (de la SKD initiale) et des transitions de temps pour passer à l'intervalle suivant. Il suffit alors d'utiliser l'algorithme du Théorème 3 pour la sémantique de saut pour obtenir la distance minimale $d(q, a)$ séparant tout état (q, a) de ψ. Ensuite si $d(q, a) \leq c$, l'intervalle $[a; b[$ vérifie intégralement ξ. En cas contraire, et si l'état (q, b) existe et si $d(q, b) < c$, on sait que $[b - c + d(b); b[$ vérifie ξ. Il reste alors à fusionner les intervalles adjacents satisfaisant ξ.

La complexité polynomiale de l'algorithme est due à la possibilité de borner le nombre d'intervalle des $\mathsf{Sat}[q, \varphi]$ par $(|\varphi| \cdot |R^q_{\mathcal{S}}|)$ — où $|R^q_{\mathcal{S}}|$ représente le nombre de transitions issues de q dans $\mathcal{S}$ — et donc de borner le nombre de groupes d'états à considérer à chaque étape.

Nous utiliserons une technique assez proche pour le model checking des automates temporisés à une horloge qui généralisent les SKD.

3.5.2 Model checking *TCTL*

Théorème 6 ([LMS05]) *Le model checking de TCTL sur les structures de Kripke avec durées munies de la sémantique continue avec états intermédiaires ou avec états d'attente est un problème PSPACE-complet.*

Il suffit de montrer ce résultat pour la sémantique avec états intermédiaires. Cela se fait en réduisant une instance $\Phi \stackrel{\text{def}}{=} \forall p_0 \exists p_1 \ldots \exists p_{n-1} \cdot \varphi$ de QBF à un problème de model checking $T_{cei}(\mathcal{S}) \models \varphi$. La SKD $\mathcal{S}_\Phi$ est décrite par la figure 3.6. On considère qu'un chemin de q_0 à q_n dans $\mathcal{S}_\Phi$ décrit une valuation pour les p_i selon la convention suivante : si le chemin passe par r_i (resp. r'_i) la valeur associée à p_i est $\bot$ (resp. $\top$).

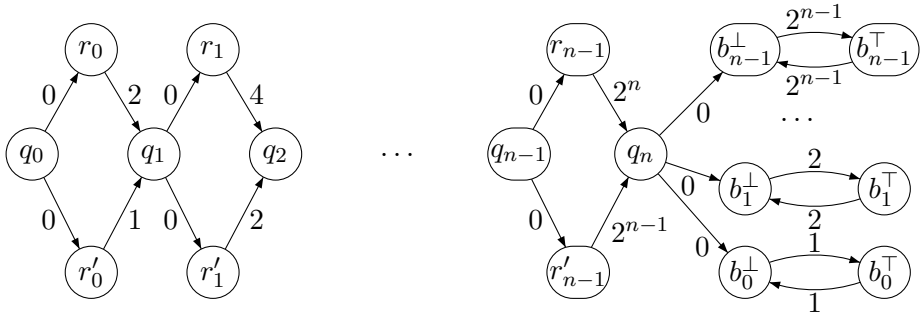

FIG. 3.6 – SKD $\mathcal{S}_\Phi$ pour la réduction de QBF.

On note S_i l'ensemble des états (de $T_{cei}(\mathcal{S})$) situés à la distance $\Sigma_{j=0}^{i-1} 2^j$ depuis q_0. Remarquons que chaque état de S_i a deux successeurs dans S_{i+1} à la distance 2^i : le premier obtenu est en passant par r_i et le second en passant par r'_i. Ainsi l'ensemble S_n contient exactement 2^n états situés entre r_{n-1} et q_n et entre r'_{n-1} et q_n et ils correspondent chacun à une unique valuation pour les p_i selon la convention décrite ci-dessus. De plus, chacun de ces états est aussi caractérisé par sa distance à l'état q_n : un état s de S_n situé à la distance d de q_n définit la valuation $2^n - 1 - d$ (*i.e.* le j^e bit de $2^n - 1 - d$ correspond à la valeur de vérité de p_j selon la valuation associée à s). Finalement on peut observer que la valeur du j^e bit du nombre $2^n - 1 - d$ vaut 1 ssi l'état s vérifie $\mathsf{EF}_{=2^n-1} b_j^{\top}$.

Dès lors, il suffit de vérifier la formule *TCTL* suivante sur la structure $\mathcal{S}$ pour obtenir une réduction du problème initial :

$$\mathsf{AF}_{=1}\, \mathsf{EF}_{=2}\, \mathsf{AF}_{=4}\, \ldots \mathsf{EF}_{=2^{n-1}} \left(\varphi[\mathsf{EF}_{=2^n-1}\, b_j^{\top} / p_j] \right)$$

3.6 Model checking *TLTL* et temps discret

Tout d'abord on peut noter que les deux sémantiques continues coïncident pour l'interprétation des formules de *TLTL*. Pour les versions temporisées de *LTL*, le model checking est au moins PSPACE-dur puisqu'il est déjà PSPACE-complet dans le cas non temporisé.

Dans le cas des $\text{SKD}^{0/1}$, le problème devient EXPSPACE-complet pour *TLTL* et reste PSPACE-complet pour $TLTL_{\leq,\geq}$:

Théorème 7 ([LMS05]) *Pour les SKD$^{0/1}$ et les SKD avec sémantique continue ou sémantique de saut, nous avons :*
– *Le model checking de TLTL et TLTL$_h$ est EXPSPACE-complet.*
– *Le model checking de TLTL$_{\leq,\geq}$ est PSPACE-complet.*

Ce résultat s'explique comme suit.

– EXPSPACE-dur : Le model checking de *TLTL* sur les SKD$^{0/1}$est EXPSPACE-difficile. Il est en effet possible de coder une exécution acceptante d'une machine de Turing $\mathcal{M} = (Q, \Sigma, q_0, q_F, T)$ sur un mot w utilisant un espace $2^{|w|}$. On note $n = |w|$. Une configuration de $\mathcal{M}$ est décrite par une séquence de 2^n états étiquetés par $\Sigma \cup (\Sigma \times Q)$. Chaque état correspondant à une des 2^n cases du ruban. La position de l'état de contrôle indique la position de la tête de lecture. L'opérateur $\mathsf{F}_{=2^n}$ permet d'accéder au contenu de la même case pour la configuration suivante et permet donc de décrire les transitions possibles. On peut ainsi écrire une formule $\Phi_{\mathcal{M},w}$ qui caractérise les exécutions acceptées par $\mathcal{M}$. Il suffit ensuite de prendre la SKD S qui engendre tous les mots sur $\Sigma \cup (\Sigma \times Q)$ avec transitions de durées 1. Alors, w est accepté par $\mathcal{M}$ ssi il existe une exécution de S qui vérifie $\Phi_{\mathcal{M},w}$ (*i.e.* ssi $S \models \neg\Phi_{\mathcal{M},w}$).
– EXPSPACE-facile : Pour montrer le coté EXPSPACE-facile on utilise le fait que la satisfaisabilité des formules de TPTL est EXPSPACE-complet [AH94]. TPTL est une logique plus expressive que *TLTL*, par exemple il est possible d'utiliser des horloges de formules. Un algorithme en EXPSPACE est donc proposé pour décider si, étant donné Φ, il existe une exécution π de la forme $\sigma_0 \xrightarrow{d_0} \sigma_1 \xrightarrow{d_1} \sigma_2 \cdots$, avec $\pi \models \Phi$ et où chaque σ_i est étiqueté par un sous-ensemble de propositions atomiques de Φ. Comme dans le cas non temporisé, on peut réduire le model checking $S \models \Phi$ à un problème de satisfaisabilité. On construit une formule Φ_S qui décrit les exécutions de S et il suffit ensuite de vérifier que $\Phi_S \Rightarrow \Phi$ est une formule valide, c'est à dire que $\Phi_S \wedge \neg\Phi$ n'est pas satisfaisable. Notons que la construction de Φ_S dépendra de la sémantique choisie.
– Enfin, il existe des algorithmes en PSPACE pour $TLTL_{\leq,\geq}$: Cela vient du résultat de [AFH96] pour la logique MITL qui contient $TLTL_{\leq,\geq}$. En effet, la vérification des modalités $\mathsf{U}_{\sim k}$ avec $\sim \in \{\leq, \geq\}$ se fait de manière plus simple que lorsque les contraintes "$= c$" sont autorisées. Par exemple si l'on cherche à vérifier $\mathsf{G}(a \Rightarrow \mathsf{F}_{<20}\, b)$, si on a déjà rencontré un a et que on attend encore le b, rencontrer un nouvel état a ne nécessite pas de mesurer le temps le séparant de b : celui-ci doit

intervenir à moins de 20 u.t. du premier a et donc a fortiori, il se situera à moins de 20 u.t. du second.

3.7 Conclusion sur la vérification des SKD

Le tableau 3.7 présente le récapitulatif des résultats exposés dans ce chapitre pour les modèles à temps discret. Il en ressort qu'il est possible d'avoir des algorithmes de model checking temporisés efficaces (polynomiaux) à condition de choisir la bonne logique **ou** la bonne sémantique : $TCTL_{\leq,\geq}$ ou les structures de Kripke à durées 0/1.

<table>
<tr><th colspan="2"></th><th>SKD$^{0/1}$
[EMSS92, LST00]</th><th>sém. saut</th><th>sém. cont.</th></tr>
<tr><td rowspan="2">$TCTL$</td><td>$\leq, \geq$</td><td colspan="3">P-complet</td></tr>
<tr><td>$\leq, \geq, =$</td><td>PTIME-c</td><td>Δ_2^p-c</td><td>PSPACE-c</td></tr>
<tr><td rowspan="2">$TLTL$</td><td>$\leq, \geq$</td><td colspan="3">PSPACE-complet</td></tr>
<tr><td>$\leq, \geq, =$</td><td colspan="3">EXPSPACE-complet</td></tr>
<tr><td rowspan="2">$TCTL^*$</td><td>$\leq, \geq$</td><td colspan="3">PSPACE-complet</td></tr>
<tr><td>$\leq, \geq, =$</td><td colspan="3">EXPSPACE-complet</td></tr>
</table>

FIG. 3.7 – Résultats pour les SKD

De tous ces résultats, celui concernant les SKD avec sémantique continue et la logique $TCTL_{\leq,\geq}$ est sûrement le plus intéressant en pratique dans la mesure où cette sémantique permet la synchronisation de plusieurs automates et où la logique est déjà très expressive.

3.8 Des SKD probabilistes

Avant de présenter les résultats pour les automates temporisés, nous décrivons succinctement un travail portant sur les SKD avec probabilités et non-déterminisme [LS05].

Une structure de Kripke probabiliste avec durées (SKPD) est un quadruplet $\mathcal{S} = (Q, q_0, D, l)$ où Q est l'ensemble (fini) d'états, q_0 est l'état initial, l est la fonction d'étiquetage des états par des propositions atomiques et $D \subseteq Q \times \mathcal{I} \times \mathsf{Dist}(Q)$ est une *relation de transition probabiliste* : pour

$(q, \rho, \mu) \in D$, ρ est un intervalle fini désignant les durées possibles de la transition et μ est une distribution sur les états de Q.

Le comportement d'une SKPD est le suivant : à l'arrivée dans un état q, on choisit de manière non-déterministe une transition $(q, \rho, \mu) \in D$, puis on choisit toujours manière non-déterministe une durée $d \in \rho$, et on peut alors évoluer dans l'état $q' \in Q$ selon la probabilité $\mu(q')$.

Comme pour les SKD, on peut définir plusieurs sémantiques pour les SKPD : une sémantique de saut et plusieurs sémantiques continues. Contrairement aux SKD, la sémantique des SKPD se définit non pas en terme de STT mais en terme de *processus de Markov temporisé* (PMT) : un PMT est un quadruplet $M = (S, s_0, \rightarrow, l)$ où S est l'ensemble (fini) des états, $s_0 \in S$ est l'état initial, $\rightarrow \subseteq S \times \mathbb{N} \times \mathsf{Dist}(S)$ est la relation de transition probabiliste (*NB :* une durée fixée est associée à la transition) et l étiquette les états avec des propositions atomiques. Ici, nous nous intéressons à des SKPD et des PMT *fortement non-zénon* où tout cycle de probabilité non-nulle a une durée strictement positive. Nous renvoyons à [LS05] pour une description détaillée de ces formalismes.

Pour énoncer des propriétés temporisées et probabilistes, nous utilisons la logique *PTCTL* (*Probabilistic Timed CTL*) qui contient des modalités $\mathbb{P}_{\bowtie\lambda}(\phi_1 \,\mathsf{U}_{\sim c}\, \phi_2)$ avec $\bowtie \in \{<, \leq, \geq, >\}$, $\lambda \in \mathbb{R}_+$, $\sim \in \{<, \leq, =, \geq, >\}$ et $c \in \mathbb{N}$. Un état s d'un PMT vérifie $\mathbb{P}_{\bowtie\lambda}(\phi_1 \,\mathsf{U}_{\sim c}\, \phi_2)$ ssi la probabilité p qu'une exécution issue de s vérifie $\phi_1 \,\mathsf{U}_{\sim c}\, \phi_2$, est telle que $p \bowtie \lambda$, *quels que soient les choix non-déterministes effectués le long de l'exécution.* Par exemple, un état s vérifie $\mathbb{P}_{\geq\lambda}(\Phi_1 \mathsf{U}_{\sim c} \Phi_2)$ ssi la probabilité **minimale**, pour les choix non-déterministes, d'avoir une exécution satisfaisant $\Phi_1 \mathsf{U}_{\sim c} \Phi_2$ est supérieure à λ.

Il est alors possible d'énoncer qu'un problème est suivi par le déclenchement d'une alarme en moins de 100 millisecondes avec une probabilité supérieure à 0.99 :

$$\mathsf{problème} \;\Rightarrow\; \mathbb{P}_{\geq 0.99}(\mathsf{F}_{\leq 100}\ \mathsf{alarme})$$

La logique *PTCTL* contient aussi l'opérateur $\mathbb{D}_{\bowtie\zeta}(\Phi)$ pour énoncer une contrainte sur l'espérance de la distance minimale pour atteindre un état vérifiant Φ.

Le premier résultat porte sur la vérification d'un processus de Markov temporisé :

Proposition 2 ([LS05]) *Étant donnés un processus de Markov temporisé fortement non-zénon $M = (S, s_0, \rightarrow, l)$, et Φ une formule de PTCTL où la plus grande constante est c_Φ, décider $M \models \Phi$ peut se faire en temps $O\big(|\Phi| \cdot ((|S| \cdot | \rightarrow | \cdot c_\Phi) + poly(|M|))\big)$*

L'idée de l'algorithme est la suivante. Considérons le cas d'un opérateur $\mathbb{P}_{\leq\lambda}(\Phi_1 \mathsf{U}_{\sim c}\, \Phi_2)$. On suppose que l'on connaît la valeur de vérité des sous -formules Φ_1 et Φ_2 pour tous les états du PMT M. On va alors calculer, pour $i = 0, \ldots, c$, la probabilité maximale, notée $f(s, i)$, que les exécutions issues de l'état s vérifient $\Phi_1 \mathsf{U}_{\sim i}\, \Phi_2$. Notons que le terme $f(s, i+1)$ se définit aisément à partir des $f(s', j)$ pour $s' \in S$ et $j \leq i$.

Par exemple, pour $\mathbb{P}_{\leq\lambda}(\Phi_1 \mathsf{U}_{\leq c}\, \Phi_2)$, nous avons l'algorithme suivant [2] :

$\underline{\mathbb{P}_{\leq\lambda}(\Phi_1 \mathsf{U}_{\leq c}\, \Phi_2)}$:
  for $i := 0$ to c
    for $j := 0$ to n
      if $s_j \models \Phi_2$ then let $f(s_j, i) := 1$
      else
        if $s_j \not\models \Phi_1 \vee \Phi_2$ then let $f(s_j, i) := 0$
        else let $f(s_j, i) := \max\limits_{(s_j, d, \nu) \in \rightarrow} \sum\limits_{s' \in S} \nu(s') \cdot f(s', i - d)$

Dans l'énoncé de la proposition 2, le terme $poly(M)$ qui apparaît dans l'évaluation de la complexité, provient du calcul de la probabilité maximale (ou minimale) pour un état de vérifier $\Phi_1 \mathsf{U} \Phi_2$ dans un processus de Markov : ce type de Until non-borné peut se traiter en pratique à l'aide de méthodes de programmation linéaire. La complexité polynomiale vient de la possibilité d'utiliser la méthode de l'ellipsoïde [BA95].

Comme pour les SKD, on peut donc définir pour les SKPD une sémantique de saut ou des sémantiques continues (avec états intermédiaires ou états d'attente) en terme de PMT. Pour vérifier qu'une SKPD vérifie une formule $\Phi \in PTCTL$, on peut utiliser l'algorithme ci-dessus sur le processus de Markov temporisé représentant sa sémantique. Néanmoins la taille de ce PMT est exponentielle dans la taille de la SKPD initiale (du fait des constantes entières représentant les durées), il est donc préférable de développer des algorithmes ad-hoc pour les SKPD. Nous avons notamment obtenu les résultats suivants où $PTCTL_{\leq,\geq}$ désigne le fragment de $PTCTL$ sans les contraintes exactes "$= c$" dans les opérateurs U :

[2] où s_0, s_1,...,s_n est une énumération de S telle que si $i > j$, alors il n'existe pas de chemin entre i et j de durée nulle et de probabilité strictement positive.

Théorème 8 ([LS05]) *Étant données $\mathcal{S} = (Q, q_0, D, l)$ une SKPD et Φ une formule de $PTCTL_{\leq,\geq}$ dans laquelle la constante maximale est c_Φ, nous avons :*

- *Pour vérifier si Φ est satisfait par $\mathcal{S}$ pour la sémantique de saut, il existe un algorithme en temps $O\Big(|\Phi| \cdot \Big((|Q| \cdot |D| \cdot c_{max}) + poly(|\mathcal{S}|)\Big)\Big)$*
- *Pour vérifier si Φ est satisfait par $\mathcal{S}$ pour la sémantique continue, il existe un algorithme en temps $O\Big((|\Phi|^3 \cdot |D|^3 \cdot c_{max}) + poly(|\Phi| \cdot |D| \cdot |\mathcal{S}|)\Big)$*

Ces algorithmes sont exponentiels en $|\Phi|$ dans la mesure où la constante c_{max} est codée en binaire. Néanmoins ils montrent que la complexité en programme est polynomiale.

Nous avons aussi étudié la complexité de ces problèmes et de plusieurs variantes. Tout d'abord, nous avons montré que vérifier la (simple) formule $\mathbb{P}_{\geq\lambda}(\mathsf{F}_{\leq c}P)$ pour les SKPD sans non-déterminisme (dans tout état q, il existe un unique $(q, \rho, \mu) \in D$ et ρ est réduit à une seule durée) et munie de la sémantique de saut était déjà un problème NP-dur. En effet, on peut facilement coder le problème (NP-dur) "K-th largest subset" [GJ79] (étant donné un ensemble fini $X = \{x_1, \ldots, x_n\}$ d'entiers et deux entiers K et B, la question est de savoir s'il existe au moins K sous-ensembles de X différents tels que la somme de leurs éléments soit inférieure à B) sous la forme d'un problème de model checking. Il suffit de considérer la SKPD sans non-déterminisme de la figure 3.8 et la formule $\mathbb{P}_{\geq \frac{K}{2^n}}(\mathsf{F}_{\leq B}P)$.

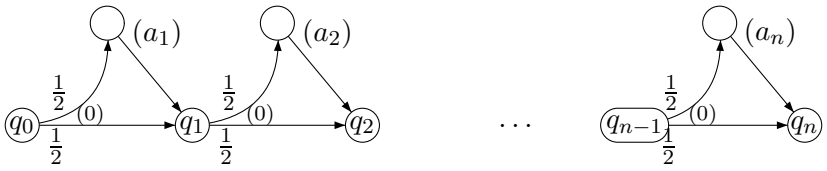

FIG. 3.8 – SKPD pour le K-th largest subset.

Cela entraîne que le model checking pour les SKPD est NP-dur et coNP-dur même pour une structure sans non-déterminisme et munie de la sémantique de saut (la plus simple). Ce résultat souligne la difficulté du problème et explique la présence de la constante c_{max} dans les algorithmes décrits ci-dessus.

Nous avons aussi considéré le cas du fragment *qualitatif* $PTCTL^{0/1}$ où l'on ne peut comparer les probabilités (minimales ou maximales) qu'avec les constantes 0 ou 1. Nous avons montré que le model checking de $PTCTL^{0/1}$

sur les SKPD avec la sémantique de saut était un problème PSPACE-dur. Le tableau suivant résume les résultats obtenus pour les structures de Kripke probabilistes avec durées dans [LS05]. Le symbole † signale que la complexité en programme est polynomiale.

	SKPD sans déterm.		DPS	
	sem. saut	sem. cont.	sem. saut	sem. cont.
$PTCTL^{0/1}_{\leq,\geq}$	P-complet	P-complet	P-dur dans PSPACE	P-dur dans EXPTIME$^{(\dagger)}$
$PTCTL^{0/1}$	Δ_2^p-complet	PSPACE-complet	PSPACE-dur dans EXPTIME	PSPACE-dur dans EXPTIME
$PTCTL_{\leq,\geq}$	NP-dur et coNP-dur dans EXPTIME$^{(\dagger)}$			
$PTCTL$	Δ_2^p-dur dans EXPTIME	PSPACE-dur dans EXPTIME	PSPACE-dur dans EXPTIME	PSPACE-dur dans EXPTIME

Chapitre 4

Automates temporisés

4.1 Introduction

Les automates temporisés ont été introduits par R. Alur et D. Dill dans les années 90 [AD90]. Il s'agit d'automates classiques munis d'horloges qui évoluent de manière continue avec le temps. Des conditions sur la valeur de ces horloges permettent d'intégrer des contraintes temps-réel dans la description du comportement de ces modèles. Depuis leur introduction, de nombreuses variantes d'automates temporisés ont été proposées, ici nous nous contentons d'une définition simple de ces modèles et nous mentionnons juste quelques variantes classiques.

À chaque transition d'un automate temporisé, est associée une *garde* (une contrainte sur la valeur des horloges) décrivant *quand* la transition *peut* être exécutée, et un ensemble d'horloges qui doivent être remises à zéro lors du franchissement de la transition. Chaque état de contrôle contient un *invariant* (une contrainte sur les horloges) qui peut *restreindre* le temps d'attente dans l'état et donc, d'une certaine manière, forcer l'exécution d'une transition d'action. Cette notion d'invariant a donc pour principale fonction d'introduire de la *vivacité* dans le modèle, comme peuvent le faire des conditions d'équité (du type Büchi ou Müller).

Le domaine de temps $\mathbb{T}$ que nous considérons est $\mathbb{R}_+$ mais la plupart des résultats présentés ici ne sont pas modifiés lorsque l'on considère $\mathbb{N}$.

4.2 Définitions

Quelques notations. Soit X un ensemble d'horloges à valeur dans $\mathbb{R}_+$, on note ${\mathbb{R}_+}^X$ l'ensemble des valuations pour X. Comme expliqué dans le

chapitre 2 pour les horloges de formules, on utilise $v + d$ pour désigner la valuation incrémentée de $d \in \mathbb{R}_+$ et $v[r \leftarrow 0]$ pour celle où les horloges de $r \subseteq X$ ont été remises à zéro.

On appelle *contrainte atomique simple* (ou *rectangulaire*) une formule de la forme $x \bowtie k$ avec $x \in X$, $k \in \mathbb{N}$ et $\bowtie \in \{=, <, \leq, >, \geq\}$. On appelle *contrainte atomique diagonale* (ou *triangulaire*) une formule de la forme $x - x' \bowtie k$. On note $\mathcal{C}(X)$ l'ensemble des combinaisons booléennes de contraintes atomiques sur X.

Formellement, un automate temporisé est défini comme suit :

Définition 7 (Automate temporisé) *Un* automate temporisé A *est un 7-uplet* $(Q, q_0, \Sigma, X, \rightarrow_A, \mathtt{Inv}, l)$ *avec :*

- Q *est un ensemble fini* d'états de contrôle *ou* localités*,*
- $q_0 \in Q$ *est l'*état initial*,*
- Σ *est un alphabet d'actions,*
- X *est un ensemble fini d'*horloges *(à valeurs réelles positives),*
- $\rightarrow_A \subseteq Q \times \mathcal{C}(X) \times \Sigma \times 2^X \times Q$ *est un ensemble fini de* transitions*;* $e = \langle q, g, a, r, q' \rangle \in \rightarrow_A$ *représente une transition de* q *vers* q'*,* g *est la garde associée à* e*,* $a \in \Sigma$ *est l'étiquette de la transition, et* r *est l'ensemble d'horloges devant être remises à zéro. On note* $q \xrightarrow{g,a,r} q'$ *une telle transition,*
- $\mathtt{Inv} : Q \rightarrow \mathcal{C}(V)$ *associe un* invariant *à chaque état de contrôle,*
- $l : Q \rightarrow 2^{Prop}$ *étiquette les états de contrôle par les propositions atomiques qu'ils vérifient.*

Un exemple d'automate temporisé est donné sur la figure 4.1. Cet automate décrit une minuterie : on part de la localité initiale OFF, à tout moment on peut appuyer sur le bouton (action b) et passer dans l'état ON avec l'horloge x remise à zéro. Le temps peut s'écouler tant que la valeur de l'horloge est inférieure à 10 (à tout moment une pression sur le bouton permet de remettre x à zéro) et lorsque $x = 10$, la transition i ramène l'automate dans l'état OFF.

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

La sémantique d'un automate temporisé est définie sous la forme d'un système de transitions temporisé où les transitions sont soit des transitions de temps, soit des transitions d'action.

Définition 8 (Sémantique des automates temporisés) *La sémantique*

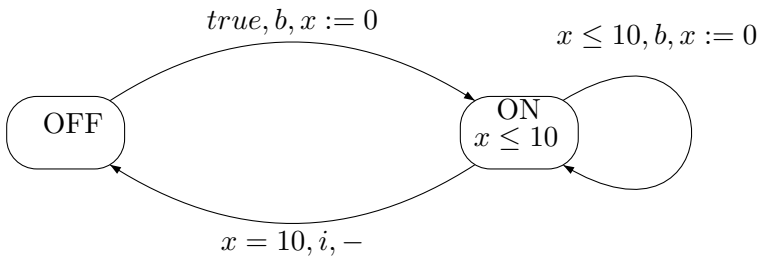

FIG. 4.1 – Exemple d'automate temporisé : une minuterie

d'un automate temporisé $A = (Q, q_0, \Sigma, X, \rightarrow_A, \mathit{Inv}, l)$ *est définie par le STT* $\mathcal{T}_A = (S, s_{init}, \rightarrow, l)$ *:*

- $S = Q \times {\mathbb{R}_+}^X$,
- $s_{init} = (q_0, v_0)$ *avec* $v_0(x) = 0,\ \forall x \in X$,
- $\rightarrow$ *correspond à deux types de transitions :*
  - *Les transitions d'actions :* $(q, v) \xrightarrow{a} (q', v')$ *ssi* $\exists q \xrightarrow{g,a,r} q' \in \rightarrow_A$ *et* $v \models g$, $v' = v[r \leftarrow 0]$ *et* $v' \models \mathit{Inv}(q')$.
  - *Les transitions de temps : Soit* $t \in \mathbb{R}_+$. $(q, v) \xrightarrow{t} (q, v+t)$ *ssi* $v+t' \models \mathit{Inv}(q)$ *pour tout* $0 \leq t' \leq t$.
- $l(q, v) = l(q)$ *pour pour tout* q.

Informellement : le système part de la configuration initiale (état de contrôle q_0 et toutes les horloges à zéro), puis effectue deux types de transitions : Les transitions d'action si la valeur courante des horloges le permet (ce type de transition s'effectue de manière *instantanée*, leur durée est nulle, et certaines horloges peuvent être remises à zéro) et les transitions de temps qui augmentent toutes les horloges d'une même durée (les horloges sont *synchrones*) en respectant l'invariant associé à l'état courant.

Comme expliqué dans le chapitre 2, nous avons noté les étiquettes des STT de la forme $(0, a)$ avec $a \in \Sigma$ par a, et celles de la forme (t, δ) par t ; le STT associé à un automate temporisé a donc ses transitions étiquetées par un élément de Σ ou un élément de $\mathbb{R}_+$. Évidemment lorsque nous considérons l'accessibilité ou le model checking de propriétés énoncées en logique temporelle, l'étiquetage des transitions n'intervient pas, seules les propositions atomique sur les états nous intéressent.

On peut noter que les systèmes de transitions engendrés par un auto-

mate temporisé vérifient deux propriétés de fermeture suivantes (additivité temporelle) : (1) si $s \xrightarrow{t} s'$ et $s' \xrightarrow{t'} s''$ avec $t, t' \in \mathbb{R}_+$, alors $s \xrightarrow{t+t'} s''$; et (2) si $s \xrightarrow{t} s'$ avec $t \in \mathbb{R}_+$, alors $\forall 0 \leq t' \leq t$, $\exists s''$ t.q. $s \xrightarrow{t'} s''$ et $s'' \xrightarrow{t-t'} s'$. Enfin notons aussi que toute exécution non-Zénon de ces STT vérifie la propriété de "variabilité finie" : pour tout intervalle de temps fini, le nombre d'étiquetages différents de propositions atomiques est fini.

Exécutions équitables, non-Zénon etc. La densité de $\mathbb{R}_+$ requiert une discussion sur le type d'exécutions que nous souhaitons considérer pour représenter le comportement d'un système. En effet, la définition du STT $\mathcal{T}_A$ rend possible l'existence d'exécutions convergentes composées de transitions de temps où le temps progresse mais sans diverger. Or lorsque l'on s'intéressera à vérifier des propriétés du type " il est toujours vrai qu'après l'envoi d'un message, sa réception a lieu dans un délai inférieur à 10 unité de temps (ut)", on souhaite vérifier qu'après l'émission, *toute* exécution contient un état "réception" en moins de 10 ut, et cette quantification sur "toutes les exécutions" ne doit pas porter sur ces exécutions convergentes. On souhaite donc disposer de critères d'acceptation sur les exécutions afin de se restreindre à celles qui sont pertinentes pour le système étudié. Par exemple :

- $\mathsf{Exec}_{\text{div}}$: Restriction aux exécutions divergentes : pour tout $t \in \mathbb{R}_+$, il existe un préfixe σ de π (noté $\pi = \sigma \cdot \pi'$) tel $\mathsf{Time}(\rho) > t$.
- $\mathsf{Exec}_{\text{inf}}$: Restriction aux exécutions comprenant un nombre infini de transitions d'action.

Enfin, on peut noter Exec^F la combinaison des deux critères ci-dessus, c.-à-d. les exécutions divergentes contenant un nombre infini de transitions d'action.

Une exécution d'un STT associée à un automate temporisé sera décrite par une séquence de la forme $(q_0, v_0) \xrightarrow{t_0} \xrightarrow{a_1} (q_1, v_1) \xrightarrow{t_1} \xrightarrow{a_2} \ldots \xrightarrow{a_n} (q_n, v_n)$: une alternance de transitions de temps et d'action.

Mots et langages temporisés. On associe naturellement un mot temporisé $(w, t) \in \Sigma^\omega \times \mathbb{T}^\omega$ à une exécution π : w_i est alors le symbole de la i-ème transition d'action de π et t_i est la date à laquelle cette transition a eu lieu, *i.e.* $\mathsf{Time}(\pi|i)$. On peut alors étudier les propriétés des langages temporisés selon les différentes conditions d'acceptation (Büchi, Müller, etc.) — voir notamment [AD94].

Remarque 1 *Dans notre cadre, une exécution peut donc se voir soit comme une évolution continue qui associe à chaque date une* [1] *valuation de propositions atomiques, soit comme le mot associé, c'est-à-dire une séquence d'évènements (actions) datés. On retrouve ces deux points de vue dans la littérature à travers les notions de sémantique "pointwise" (évènement daté) ou "interval-based" (évolution continue : à chaque intervalle de temps est associé une valuation des propositions atomiques) [Ras99]. Dans ce document, nous adoptons le point de vue "interval-based" pour les logiques temporelles, et le point de vue "pointwise" pour les logiques modales.*

Sous-classes. On peut considérer plusieurs familles d'automates temporisés qui correspondent à des sous-classes du modèle décrit ci-dessus.

- Automates sans contrainte diagonale (AT^{df}) : les gardes sont restreintes aux combinaisons booléennes de contraintes atomiques simples. Cette sous-classe est très souvent étudiée. Il est bien connu que l'ajout des contraintes diagonales n'augmente pas l'expressivité du modèle dans la mesure où, pour tout automate A, il est possible de construire un automate temporisé A' sans contrainte diagonale bisimilaire à A (voir [BDGP98]). Un résultat récent a néanmoins montré que les contraintes diagonales apportaient de la *concision* [BC05].
- Automates à une horloge (1C-AT) : la temporisation est donc simplifiée , nous verrons que cette classe admet des propriétés particulières pour le model checking. De manière analogue, on notera 2C-AT les automates temporisés à deux horloges.
- Automate "temps réel" (*real-time automata*) : il s'agit d'AT avec une seule horloge qui est remise à zéro à chaque transition d'action [HJ96]. Ces modèles ont notamment été étudiés du point de vue théorie du langage [Dim00].

Extensions des automates temporisés. Les AT peuvent être vus comme des *systèmes hybrides linéaires* [Hen96] particuliers avec des variables dynamiques de pente 1, des contraintes et des mises à jour particulières. Notons que les résultats de décidabilité pour les AT sont très sensibles à toute variation sur ces trois caractéristiques du modèle. Quelques classes décidables existent néanmoins, par exemple les automates hybrides rectangulaires [HKPV98], ou certaines variantes avec mises à jour étendues [BDFP04]. Nous renvoyons à [HKPV98, ACH+95] pour des présentations générales sur les systèmes hybrides linéaires et leurs sous-classes décidables.

[1] ou plusieurs, si des actions sont effectuées en temps 0

Composition parallèle. La composition parallèle sur les automates temporisés se définit de manière standard avec une table de synchronisation sur les actions de Σ. Pour l'écoulement du temps, on suppose que les STT associés aux automates sont fortement synchronisés.

4.3 Résultats classiques sur le model checking des automates temporisés

4.3.1 Graphe des régions

La présence des valuations d'horloges produit un nombre infini de configurations pour un automate temporisé. Il est donc nécessaire d'utiliser des techniques particulières pour vérifier des propriétés classiques comme l'accessibilité ou plus généralement des propriétés décrites avec des logiques temporelles.

Nous rappelons ici une technique bien connue, le *graphe des régions*, proposée par Alur et Dill pour analyser le comportement des automates temporisés. Tout d'abord, nous considérons le problème de l'accessibilité d'un état de contrôle. Les problème est donc le suivant : Étant donnés un AT A et un état de contrôle q_F, est-ce que q_F est atteignable depuis la configuration initiale s_{init} ? Pour cela, nous allons remplacer ce problème d'accessibilité sur le système de transitions infini $\mathcal{T}_A$ par un problème d'accessibilité sur un graphe fini — l'automate des régions – $\mathcal{R}_A$. Les états de $\mathcal{R}_A$ correspondent à des ensembles de configurations de A vérifiant les mêmes propriétés d'accessibilité. Plus précisément ses états sont des éléments de $Q \times ({\mathbb{R}_+}^X)_{/\sim}$ où $\sim$ est une équivalence sur les valuations d'horloges telle que (1) $({\mathbb{R}_+}^X)_{/\sim}$ est fini et (2) pour tout u et v avec $u \sim v$, on a : (q, u) et (q, v) permettent d'atteindre les mêmes états de contrôle.

Notons que pour avoir les mêmes états accessibles, il suffit de pouvoir exécuter les mêmes transitions d'action (donc de satisfaire les mêmes gardes), d'arriver dans des configurations équivalentes (après les remises à zéro) et de pouvoir simuler les mêmes transitions de temps. Effectivement la relation $\sim$ peut se définir comme une bisimulation de temps abstrait (au sens où l'on n'exige pas d'attendre les mêmes durées depuis deux états équivalents).

L'équivalence $\sim$ dépend de la constante maximale — notée M — apparaissant dans les gardes et les invariants de l'automate considéré. On note $\mathcal{C}_M(X)$ le sous-ensemble de $\mathcal{C}(X)$ où les constantes appartiennent à $\{0, 1, \ldots, M\}$. L'ensemble $\mathcal{C}_M(X)$ est fini modulo équivalences booléennes.

En fait, pour $\sim$, il suffit de prendre l'équivalence $\equiv_{\mathcal{C}_M(X)}$ induite par

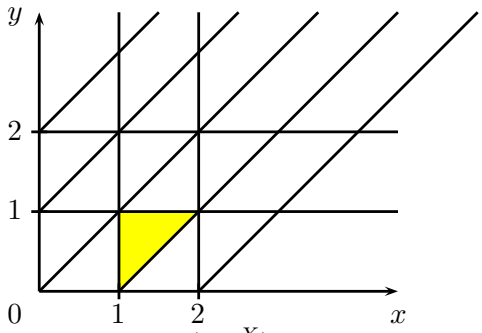

FIG. 4.2 – Exemple de $(\mathbb{R}_+{}^X)_{/\equiv_M}$ avec $M = 2$ et $X = \{x, y\}$

l'ensemble de contraintes de $\mathcal{C}_M(X)$. On peut en effet montrer que $\equiv_{\mathcal{C}_M(X)}$ est une bisimulation de temps abstrait et elle est clairement d'indice fini.

Si on considère le cas de $X = \{x, y\}$ avec $M = 2$, l'équivalence $\equiv_{\mathcal{C}_M(X)}$ engendre 72 classes d'équivalences représentées sur la Figure 4.2 sous la forme de points, segments ou surfaces. À l'avenir nous noterons $\equiv_M$ l'équivalence $\equiv_{\mathcal{C}_M(X)}$. La zone grisée correspond aux valuations vérifiant $(1 < x < 2) \wedge (0 < y < 1) \wedge (x > y > x - 1)$.

Dans le cas des automates temporisés sans contrainte diagonale, on utilise habituellement une autre équivalence définie de manière ad-hoc (voir par exemple [AD94]). Dans tous les cas, le nombre de classes d'équivalence est exponentiel dans le nombre d'horloges et exponentiel dans le codage de la constante M.

Le *graphe des régions* désigne le graphe fini dont les états sont des éléments de $(\mathbb{R}_+{}^X)_{/\equiv_M}$ et muni de transitions représentant l'écoulement du temps. Une *région* de A désigne une classe d'équivalence γ de $\equiv_M$. On appelle l'*automate des régions* le graphe fini $\mathcal{R}_A$ dont les états sont des éléments de $Q \times (\mathbb{R}_+{}^X)_{/\equiv_M}$. L'automate des régions peut se voir comme le résultat d'un produit particulier entre entre A et le graphe des régions.

Étant donnée une valuation u, on note $[u]$ la classe d'équivalence de $\equiv_M$ contenant u. L'équivalence $\equiv_M$ étant une bisimulation de temps abstrait pour les gardes de $\mathcal{C}_M(X)$, toutes les valuations d'une classe γ de $(\mathbb{R}_+{}^X)_{/\equiv_M}$ vérifient les même gardes de $\mathcal{C}_M(X)$, on note $\gamma \models \phi$ lorsque les valuations de γ vérifient ϕ. De plus, étant données deux valuations u et v de γ, les valuations $u[r \leftarrow 0]$ et $v[r \leftarrow 0]$ appartiennent à la même classe (notée $\gamma[r \leftarrow 0]$) pour tout $r \subseteq X$. Enfin on définit une opération `Succ` sur les classes : $\texttt{Succ}(\gamma)$ est la région, si elle existe, $[u + t] \neq \gamma$ telle que $u \in \gamma$, $t \in \mathbb{R}_+$, et $([u + t'] = \gamma \vee [u + t'] = [u + t])$ pour tout $0 \leq t' < t$. Sinon on

définit $\texttt{Succ}(\gamma) = \gamma$ dans le cas où toutes les horloges sont supérieures à M. $\texttt{Succ}(\gamma)$ correspond à la première région atteignable depuis γ par écoulement du temps (ou γ elle-même si le temps ne fait plus changer de région).

Toutes ces opérations sont aisément calculables à partir des définitions de $\equiv_M$. Cela permet de définir l'automate des régions de A [ACD93] : $\mathcal{R}_A = (S', s'_0, \rightarrow, \Sigma)$ avec :

- $S' = Q \times ({\mathbb{R}_+}^X)_{/\equiv_M}$,
- $s'_0 = (q_0, \gamma_0)$ avec $\gamma_0 = [v_0]$,
- $\rightarrow$ se décompose en deux types de transitions :
  - Les actions : $(q, \gamma) \xrightarrow{a} (q', \gamma')$ ssi $\exists q \xrightarrow{g,a,r} q' \in \rightarrow$, $\gamma \models g$, $\gamma' = \gamma[r \leftarrow 0]$ et $\gamma' \models \texttt{Inv}(q')$.
  - Les délais : $(q, \gamma) \xrightarrow{\delta} (q, \texttt{Succ}(\gamma))$ ssi $\texttt{Succ}(\gamma) \models \texttt{Inv}(q)$.

Il y a aussi *correspondance entre les exécutions* de A et celles de $\mathcal{R}_A$: Pour toute exécution $(q_0, v_0) \xrightarrow{t_0}\xrightarrow{a_1} (q_1, v_1) \xrightarrow{t_1}\xrightarrow{a_2} \ldots \xrightarrow{a_n} (q_n, v_n)$ de A, il existe une exécution $(q_0, \gamma_0) \xrightarrow{\delta^*}\xrightarrow{a_1} (q_1, \gamma_1) \xrightarrow{\delta^*}\xrightarrow{a_2} \ldots \xrightarrow{\delta^*}\xrightarrow{a_n} (q_n, \gamma_n)$ dans $\mathcal{R}_A$ (et réciproquement).

En conclusion, on a le résultat classique suivant :

Théorème 9 ([AD94]) *Un état de contrôle q_F est atteignable dans un automate temporisé A ssi un état de la forme (q_F, γ) est accessible dans $\mathcal{R}_A$.*

Cela donne clairement un algorithme pour décider de l'accessibilité d'un état de contrôle.

Model checking *TCTL.* La technique que nous venons de présenter s'adapte à la vérification de formules de *TCTL* [ACD93]. Par exemple, pour la version de *TCTL* avec opérateurs $\mathsf{E\_U}_{\sim k}\_$ et $\mathsf{A\_U}_{\sim k}\_$, il suffit de construire un graphe des régions avec une horloge supplémentaire x_Φ afin de mesurer les délais séparant deux configurations le long d'une exécution. Pour étiqueter les états (q, γ) de l'automate des régions qui satisfont $\mathsf{E}\varphi_1\mathsf{U}_{\sim c}\varphi_2$, il suffit de vérifier si la formule (*CTL*) $\mathsf{E}\varphi_1\mathsf{U}(P_{[x_\Phi \sim c]} \wedge \varphi_2)$ est vraie en $(q, \gamma[x_\Phi \leftarrow 0])$ où $P_{[x_\Phi \sim c]}$ est une proposition atomique étiquetant les régions où $x_\Phi \sim c$: l'horloge x_Φ n'étant jamais remise à zéro le long du chemin, la proposition $P_{[x_\Phi \sim c]}$ indique bien que le temps écoulé pour atteindre φ_2 satisfait $\sim c$.

Si on considère les versions de *TCTL* avec des horloges de formule, il suffit d'ajouter ces horloges au graphe de régions et d'adapter l'algorithme d'étiquetage classique de *CTL* dans le même esprit que ci-dessus.

4.3.2 Complexité de l'accessibilité

L'accessibilité d'un état de contrôle est un problème difficile :

Théorème 10 ([AD94]) *L'accessibilité d'un état de contrôle dans un automate temporisé est un problème PSPACE-complet.*

Ce résultat est montré pour le test du vide pour le langage reconnu par un automate temporisé mais il est clairement équivalent à celui de l'accessibilité. L'idée est de coder le comportement d'une machine de Turing $\mathcal{M}$ bornée en espace linéaire sur un mot w sous la forme d'un automate temporisé.

Une construction possible [AL02], que nous évoquerons par la suite, consiste à définir un automate temporisé $\mathcal{A}_{\mathcal{M},w}$ dont les états de contrôle sont des paires (q, i) où q est un état de $\mathcal{M}$ et i un entier indiquant la position de la tête de lecture sur le ruban ($1 \leq i \leq |w|$). Le contenu des cases du ruban est codé avec des horloges. On suppose que l'alphabet de $\mathcal{M}$ est composé de deux lettres $\{a, b\}$, on va utiliser deux horloges pour chaque case $i : x_i$ et y_i. On adopte la convention que la case i contient un a (resp. un b) ssi $x_i = y_i$ (resp. $x_i > y_i$). Ce codage a la particularité d'être stable vis-à-vis de l'écoulement du temps. De plus, écrire un a consistera à remettre x_i et y_i à zéro, et écrire un b consiste à laisser passer un temps strictement positif puis à remettre y_i à zéro. Enfin, un simple test $x_i = y_i$ ou $x_i > y_i$ suffit pour tester la valeur d'une case du ruban.

Il reste à définir les transitions de $\mathcal{A}_{\mathcal{M},w}$. Supposons qu'il existe une transition de $\mathcal{A}_{\mathcal{M},w}$ de la forme suivante [2] $q \xrightarrow{a/b/r} q'$, alors on définira les transitions $(q, i) \xrightarrow{g,\text{step},r} (q', i+1)$ si $i + 1 \leq |w|$ et où g est $x_i = y_i$ et r est $\{y_i\}$. Afin de garantir qu'un délai strictement positif sépare deux transitions discrètes, on ajoutera une horloge t et on complétera chaque garde avec la contrainte $t > 0$.

Enfin il faut rajouter une transition dont l'objectif est d'écrire sur le ruban le mot initial w. La taille de $\mathcal{A}_{\mathcal{M},w}$ est bien polynomiale en $\mathcal{M}$. Une transition d'action de l'automate temporisé correspond à une transition de la machine de Turing $\mathcal{M}$. Ici nous utilisons des contraintes diagonales, mais nous pourrions faire un autre codage : la complexité de l'accessibilité ne dépend pas de la présence ou non des contraintes diagonales. Enfin, notons aussi que nous n'utilisons pas l'étiquetage des transitions de l'automate : toutes les transitions sont étiquetées par le même label.

Deux résultats plus précis ont été montré par Courcoubetis et Yannakakis :

[2] $a/b/r$ signifie "lire a, écrire b et déplacer la tête de lecture vers la droite".

Théorème 11 ([CY92]) – *L'accessibilité d'un état de contrôle dans un automate temporisé à trois horloges (ou plus) est PSPACE-complet.*
– *L'accessibilité d'un état de contrôle dans un automate temporisé où les constantes appartiennent à* $\{0, 1\}$ *est PSPACE-complet.*

Bien sûr si l'on borne le nombre d'horloges et la constante maximale, on obtient un problème NLOGSPACE-complet dans la mesure où la taille du graphe des régions ne dépend que de ces deux paramètres.

4.3.3 Complexité du model checking de *TCTL*

Le model checking de *TCTL* n'est pas plus difficile que l'accessibilité :

Théorème 12 ([ACD93]) *Le model checking de TCTL sur les automates temporisés est un problème PSPACE-complet.*

Le coté PSPACE-dur vient directement de la possibilité d'énoncer les propriétés d'accessibilité dans les AT. Pour montrer qu'il est dans PSPACE, il suffit de considérer un algorithme de model checking à la volée sur le graphe des régions : pour vérifier une propriété Φ, il est suffisant de ne stocker que $|\Phi|$ configurations et de procéder de manière non-déterministe.

4.3.4 Complexité du model checking de *TLTL*

Pour le temps linéaire, nous avons les résultats suivants :

Théorème 13 ([AFH96]) – *Le model checking de TLTL sur les automates temporisés est un problème indécidable.*
– *Le model checking de* $TLTL_{\leq,\geq}$ *sur les automates temporisés est un problème PSPACE-complet.*

Ces résultats soulignent clairement l'importance des contraintes exactes "=c" dans les formalismes de spécification. Le premier est basé sur la preuve d'indécidabilité de la logique *MTL*. On peut noter que ce résultat dépend fortement du choix de la sémantique "interval based" (*cf.* la remarque 1) : dans [OW05] il est montré qu'avec la sémantique "pointwise" la logique *MTL* est décidable.
Le second provient du fait que $TLTL_{\leq,\geq}$ correspond à la logique $MITL_{0,\infty}$, c'est à dire lorsque les intervalles associés aux modalités Until contiennent soit la borne gauche 0, soit la borne droite ∞.

4.4 Complexité du model checking des logiques modales temporisées

Le premier résultat porte sur $L_{\mu,\nu}$ et ses variantes :

Théorème 14 ([AL02]) *Le model checking de $L_{\mu,\nu}$, $L^{-}_{\mu,\nu}$ et L_{ν} sur les automates temporisés est EXPTIME-complet.*

- La preuve de dureté se fait pour le plus petit fragment, c'est-à-dire L_{ν}. On peut coder la non-acceptation d'un mot w par une machine de Turing alternante bornée en espace linéaire sous la forme d'un problème de model checking pour une formule de L_{ν}.
  L'idée de cette réduction consiste à reprendre la construction de $\mathcal{A}_{\mathcal{M},w}$ pour la machine de Turing bornée en espace linéaire décrite précédemment avec des transitions Acc dans les configurations gagnantes, et d'utiliser la formule de L_{ν} suivante pour coder le caractère alternant de $\mathcal{M}$ [3] : $\Phi = \max(X, [\mathrm{Acc}]\perp \wedge [\delta]\,[\mathrm{step}]\,\langle\delta\rangle\,\langle\mathrm{step}\rangle\,X)$. On énonce ainsi que l'état courant (OU) n'est pas acceptant et que quelle que soit la transition de la machine $\mathcal{M}$ choisie ($[\delta][a]$), il existe alors une transition depuis cet état (ET) qui amène à un état qui n'est pas acceptant etc.
- L'appartenance à EXPTIME s'explique par le fait que l'on peut utiliser un algorithme standard de μ-calcul sur le graphe des régions. Ce dernier a une taille exponentielle en $(|\Phi| + |A|)^2$ et l'algorithme pour le μ-calcul s'exécute en $O((|\Phi| \cdot |\mathcal{R}_A|)^{alt})$ où alt désigne le nombre d'alternations dans Φ [EL86], que l'on peut majorer par Φ. On obtient donc un algorithme exponentiel en $\Phi \cdot (|\Phi| + |A|)^2$.

On a même un résultat plus fin :

Théorème 15 ([AL02]) – *La complexité en* programme *du model checking de $L_{\mu,\nu}$, $L^{-}_{\mu,\nu}$ et L_{ν} est EXPTIME-complete.*
– *La complexité en* spécification *du model checking de $L_{\mu,\nu}$, $L^{-}_{\mu,\nu}$ et L_{ν} est EXPTIME-complete.*

Le premier point découle directement de la construction précédente : la formule utilisée pour énoncer la non-acceptation du mot w par la machine de Turing alternante $\mathcal{M}$ ne dépend ni de w, ni de $\mathcal{M}$.

La résultat pour la complexité en spécification s'obtient à l'aide d'un codage légèrement différent. L'idée est d'utiliser une formule $\Phi_{\mathcal{M},w}$ de L_{ν} pour énoncer la non-acceptation de w par $\mathcal{M}$ [4] sur l'automate *nil* — un

[3] On suppose qu'il y a une alternation stricte des états OU/ET dans l'automate $\mathcal{M}$.
[4] $\mathcal{M}$ désigne une machine de Turing alternante bornée linéairement en espace.

automate sans horloge ni transition, qui ne peut que laisser passer du temps. La formule $\Phi_{\mathcal{M},w}$ va utiliser des horloges pour coder le contenu du ruban (comme pour l'automate, nous aurons deux horloges x_i et y_i pour chaque cellule i), $|w|$ horloges pour coder la position de la tête de lecture (p_i sera inférieure à 1 ssi la tête de lecture se trouve à la position i) et $|Q_{\mathcal{M}}|$ horloges pour indiquer l'état de contrôle courant de $\mathcal{M}$ (p_j sera inférieure à 1 ssi l'état de contrôle courant est j). Il est alors possible de coder toutes les transitions de $\mathcal{M}$ dans la formule de L_ν ainsi que son comportement alternant.

Les fragments de $L_{\mu,\nu}$. Nous renvoyons à [AL02] pour une présentation détaillée de tous les résultats concernant les nombreuses logiques modales temporisées que l'on peut définir comme fragment de $L_{\mu,\nu}$. Ici, nous mentionnons juste deux points :

- Le model checking des logiques L_s, *SBLL* et $L_{\forall S}$ est PSPACE-complet puisque chaque problème $A \models \varphi$ peut se ramener à un problème d'accessibilité sur un produit $A \times T_\varphi$ où T_φ est un *automate de test* de taille polynomial en φ. Chacune de ces logiques impose des restrictions sur les disjonctions et les modalités $\langle a \rangle$ et $\langle \delta \rangle$: on peut montrer que lever une seule de ces restrictions rend le model checking EXPTIME-complet.
- Même si l'on supprime les opérateurs de point fixe dans $L_{\mu,\nu}$, $L^-_{\mu,\nu}$ et L_ν, le model checking reste PSPACE-dur. La seule présence d'horloges et des modalités de base suffit à énoncer des problèmes difficiles et ce, même sur le processus *nil* (*i.e.* la complexité en spécification est PSPACE-dure). En fait, tous ces problèmes sont PSPACE-complets.

4.5 Complexité du model checking des automates à 1 ou 2 horloges

4.5.1 Accessibilité

Le premier résultat pour les 1C-AT concerne l'accessibilité :

Théorème 16 ([LMS04]) *L'accessibilité d'un état de contrôle dans un automate temporisé à une horloge est un problème NLOGSPACE-complet.*

Le coté NLOGSPACE-dur vient du cas non temporisé. Pour montrer le coté NLOGSPACE-facile, on utilise un algorithme non déterministe basé sur l'idée suivante. Soit A un 1C-AT et soit x son horloge. On note $\{0, c_1, \ldots, c_k\}$ les constantes apparaissant dans les contraintes de A. Une configuration de A est une paire (q, v) où v désigne la valeur courante de x. Pour décider si

une transition $q \xrightarrow{g,a,r} q'$ est tirable, il suffit de connaître dans quel intervalle $]c_i; c_{i+1}[$, $[c_i; c_i]$ ou $]c_k; \infty[$ la valeur v se situe. Il suffit donc de coder une configuration par une paire (q, n) où n désigne le *numéro* de l'intervalle ($n \leq 2k+2$ et $k \leq |A|$). Étant donnée une configuration codée de cette manière, il suffit, pour décider de la valeur de vérité d'une garde $x \sim c$, de compter le nombre n_1 de contraintes distinctes inférieures à c et le nombre n_2 de contraintes supérieures à c : l'intervalle numéro n vérifie $x \sim c$ ssi ssi $\frac{n}{2} \geq n_1$ et $\frac{n}{2} \leq k - n_2$ si n est pair, et $\frac{n-1}{2} \geq n_1$ et $\frac{n+1}{2} \leq k - n_2$ si n est impair. Le fait d'utiliser le numéro de l'intervalle permet d'obtenir un algorithme en espace logarithmique.

Ce résultat signifie que la complexité de l'accessibilité dans les automates temporisés à une horloge est identique à celle de l'accessibilité dans les automates classiques.

Par contre, dès que l'on considère deux horloges, nous observons un saut de complexité :

Théorème 17 ([LMS04]) *L'accessibilité d'un état de contrôle dans un automate temporisé à deux horloges est un problème NP-dur.*

En effet, considérons une instance du problème SUBSET-SUM : $\{a_1, \ldots, a_p\}$ des entiers et une valeur entière b. On peut réduire ce problème au problème de l'accessibilité de G dans l'automate temporisé de la figure 4.3.

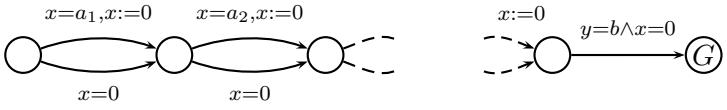

FIG. 4.3 – Codage de SUBSET-SUM avec un 2C-AT

Notons que cet automate à deux horloges est très simple puisque l'horloge x est remise à zéro à chaque transition. Considérer deux horloges au lieu d'une horloge unique induit donc un saut de complexité même dans ce cas particulier.

4.5.2 Model checking *TCTL*

Automates à une horloge. Lorsque l'on considère le model checking des automates temporisés à une horloge pour une formule de *TCTL*, nous retrouvons la complexité des automates avec un nombre quelconque d'horloges :

Théorème 18 ([LMS04]) *Le model checking de TCTL sur les automates temporisés à une horloge est un problème PSPACE-complet.*

L'appartenance à PSPACE provient de la complexité du model checking de *TCTL* dans le cas général. Le coté dur peut se montrer avec la même technique que celle utilisée pour prouver que le model checking *TCTL* sur les SKD avec sémantique continue est PSPACE-dur (Théorème 6).

Par contre, nous avons le résultat suivant qui montre qu'il est possible d'avoir des algorithmes efficaces sur les automates à une horloge lorsque l'on utilise la logique $TCTL_{\leq,\geq}$:

Théorème 19 ([LMS04]) *Étant donnée un automate temporisé à une horloge* $A = (Q, q_0, \Sigma, \{x\}, \rightarrow_A, \mathtt{Inv}, l)$ *et une formule* Φ *de* $TCTL_{\leq,\geq}$*, décider* $\mathcal{T}_A \models \Phi$ *peut se faire en temps* $O\big(|\Phi|^5 \cdot |Q|^3 \cdot |\rightarrow_A|^3\big)$.

L'algorithme pour $TCTL_{\leq,\geq}$ sur les 1C-AT est assez proche de celui proposé pour le model checking de $TCTL_{\leq,\geq}$ sur les SKD à sémantique continue, néanmoins ici nous sommes face à quelques difficultés supplémentaires car les horloges ne sont pas toujours remises à zéro après chaque transition.

L'objectif est donc le calcul d'unions d'intervalles $\mathsf{Sat}[q, \varphi]$ pour toutes les sous-formules φ de Φ. Outre le fait que nous manipulons des positions à valeurs dans $\mathbb{R}_+$ (mais les bornes de ces intervalles sont toujours des entiers), une autre différence par rapport aux SKD est que les intervalles peuvent être ouverts ou fermés, à gauche comme à droite. On note Cst_A l'ensemble des constantes apparaissant dans les contraintes de A (on suppose que $0 \in \mathsf{Cst}_A$).

Considérons le cas de $\xi \stackrel{\text{def}}{=} \mathsf{E}\varphi\mathsf{U}_{\leq c}\psi$. Soit $\mathbb{B}$ l'ensemble de constantes composé de Cst_A et de toutes les bornes des intervalles de $\mathsf{Sat}[q, \varphi]$ et $\mathsf{Sat}[q, \psi]$ pour tout q. On note $\mathbb{B} \stackrel{\text{def}}{=} \{b_0, b_1, \ldots, b_k\}$ en supposant $b_0 = 0$ et $b_i < b_{i+1}$. On appelle $\mathcal{I}_{\mathbb{B}}$ l'ensemble d'intervalles $[b_0; b_0]$, $]b_0; b_1[$, $[b_1; b_1]$, $\ldots$, $]b_k; \infty[$. Tout intervalle $\lambda \in \mathcal{I}_{\mathbb{B}}$ vérifie les mêmes gardes de l'automate A, et uniformément φ ou $\neg\varphi$, et ψ ou $\neg\psi$.

Le point clé de l'algorithme consiste à construire un graphe G dont les états vont correspondre à des paires (q, λ), que l'on va utiliser pour construire les $\mathsf{Sat}[q, \xi]$. Les transitions de G sont soit des transitions d'action (de A), soit des transition de temps (abstrait : il n'y a pas de durée) qui permettent de passer à l'intervalle adjacent.

Les positions contenues dans un intervalle $\lambda \in \mathcal{I}_{\mathbb{B}}$ ont un comportement suffisamment proche pour que l'on puisse leur associer une fonction de durée $\delta^{\psi}_{q,\lambda} : \lambda \rightarrow \mathbb{R}_+$ qui associe à chaque position de l'intervalle la durée minimale pour atteindre un état vérifiant ψ (le long d'un chemin vérifiant φ) et *pour que ces fonctions de durée aient une structure particulière* : elles sont constantes pour une partie gauche de λ puis sont décroissante (pente -1)

pour la partie droite. La partie gauche de l'intervalle comprend les positions dont un plus court chemin remet x à zéro avant de laisser passer du temps. La partie droite (pente -1) contient les positions dont un plus court chemin laisse passer du temps (pour atteindre au moins la borne droite) avant de remettre x à zéro [5]. On peut bien sûr avoir une partie gauche vide ou une partie droite vide. La figure 4.4 donne un exemple de fonctions de durée pour une séquence d'intervalles.

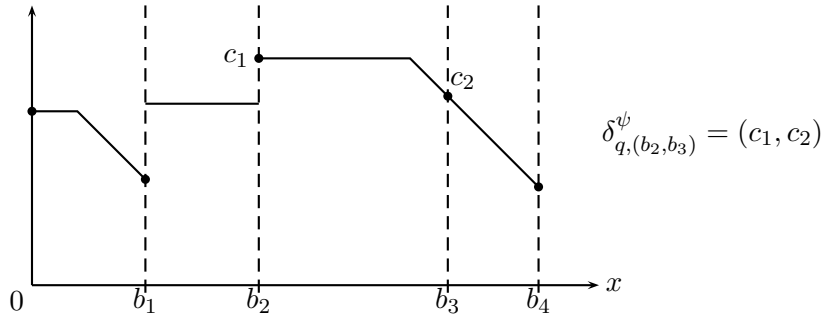

FIG. 4.4 – Exemple de fonctions de durée

Ce genre de fonctions peut se définir avec une paire (c_1, c_2) pour les intervalles ouverts : représentant les valeurs de la limite à gauche et de la limite à droite de $\delta^{\psi}_{q,\lambda}$ sur l'intervalle λ si il est ouvert. On les définit avec une unique valeur pour les intervalles fermés (restreints à un point).

On peut appliquer sur le graphe G une variante de l'algorithme de Bellman-Ford où l'on calcule les fonctions de durée associées à chaque noeud par approximations successives (pour cela on étend les opérations min et + sur les fonctions de durée).

Finalement il reste à construire les ensembles $\mathsf{Sat}[q, \xi]$ en fonction du seuil c. Cette sélection peut amener à casser un intervalle λ en deux, on peut montrer que d'une part $|\mathsf{Sat}[q, \xi]|$ est toujours borné par $2 \cdot |\xi| \cdot |{\rightarrow_A}|$ et d'autre part que le nombre de nouvelles constantes pouvant apparaître dans la sélection de sous-intervalles est borné par $|{\rightarrow_A}| + \sum_q |\mathsf{Sat}[q, \psi]|$.

L'algorithme de Bellman-Ford a une complexité en $O(|V_G| \cdot |{\rightarrow_G}|)$ et cela donne une complexité [6] en $O(|Q|^3 \cdot |{\rightarrow_A}|^3 \cdot |\xi|^4)$.

[5] Pour un état dans $(q,]b_i; b_{i+1}[)$, cela ne signifie pas, contrairement au cas des SKD, que l'on va passer par l'état $(q, [b_{i+1}; b_{i+1}])$ car il se peut que des transitions d'actions aient lieu avant l'attente et que la borne b_{i+1} soit atteinte pour un autre état q'.

[6] La complexité mentionnée dans [LMS04] est inférieure mais elle est erronée en raison d'une mauvaise évaluation du nombre de nouvelles constantes pouvant être créées par les coupures d'intervalles.

Automates à deux horloges. Le premier résultat porte sur le model checking de *CTL*, c'est à dire sans contraintes temps-réel, nous avons :

Théorème 20 ([LMS04]) *Le model checking de CTL sur les automates temporisés à deux horloges est un problème PSPACE-complet.*

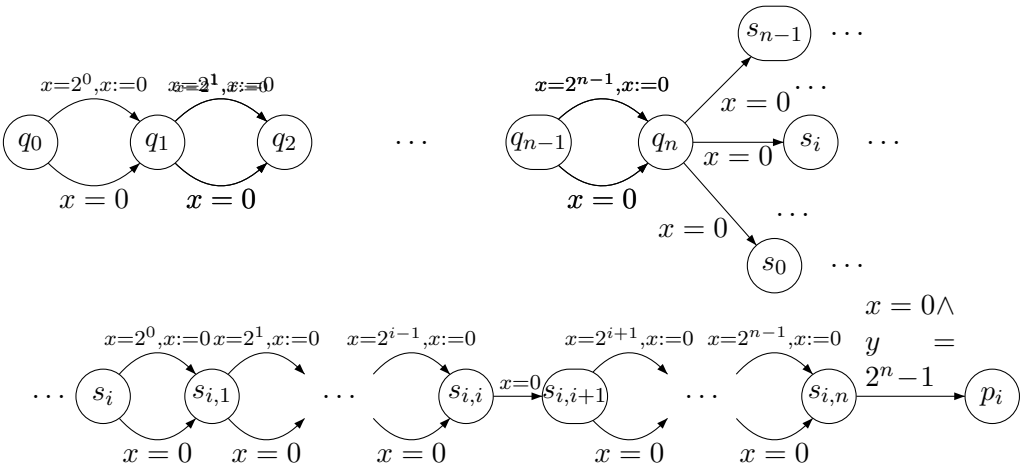

FIG. 4.5 – L'automate A_Φ à deux horloges pour coder QBF avec *CTL*

La preuve se base sur une réduction de QBF ($\Phi \stackrel{\text{def}}{=} \forall p_0 \exists p_1 \ldots \exists p_{n-1} \cdot \varphi$) à un problème de model checking pour un 2C-AT et une formule de *CTL*. L'automate A_Φ est décrit par la figure 4.5 et la formule est :

$$\overline{\Phi} \stackrel{\text{def}}{=} \Big(\mathsf{A}q_0\mathsf{U}\Big(q_1 \wedge (\mathsf{E}q_1\mathsf{U}(q_2 \wedge \ldots \mathsf{U}(q_n \wedge \overline{\varphi})))\Big)\Big)$$

avec $\overline{\varphi} \stackrel{\text{def}}{=} \varphi[p_i \leftarrow \mathsf{EF}p_i]$. Un chemin depuis q_0 est interprété comme une valuation pour les p_i : passer par la transition de durée 2^i affecte $\top$ à p_i, alors que passer par celle de durée 0 lui affecte $\bot$.

On peut remarquer que l'automate A_Φ a une horloge qui est remise à zéro après chaque transition, il s'agit donc, là encore, d'une sous-classe simple des 2C-AT.

Notons que ce résultat est à comparer avec celui sur les 1C-AT pour $TCTL_{\leq,\geq}$: dans les deux cas, le système complet (automate + formule) comprend deux horloges et la logique à vérifier est de temps arborescent mais

la complexité est très différente. Le fait de pouvoir utiliser des contraintes exactes dans l'automate semble déterminant de ce point de vue.

On en déduit directement le corollaire suivant :

Corollaire 1 *Le model checking de $TCTL_{\leq,\geq}$ ou TCTL sur les automates temporisés à deux horloges est un problème PSPACE-complet.*

Automates temps-réel. La vérification des automates à une horloge remise à zéro à chaque transition a la même complexité que les 1C-AT :

Théorème 21 ([LMS04]) – *L'accessibilité dans les automates temps-réel est un problème NLOGSPACE-complet.*
– *Le model checking TCTL dans les automates temps-réel est un problème PSPACE-complet.*
– *Le model checking $TCTL_{\leq,\geq}$ dans les automates temps-réel est un problème P-complet.*

Notons que le coté PSPACE-dur pour *TCTL* provient du fait que la preuve pour les 1C-AT utilise en fait un automate temps-réel.

4.5.3 Model checking L_ν, *TLTL*, etc.

La complexité du model checking des autres logiques pour les automates à une ou deux horloge(s) se déduit des autres résultats de complexité. Pour L_ν et $L_{\mu,\nu}$, le model checking est EXPTIME-complet pour le processus *nil* (sans horloge), cette complexité vaut donc aussi pour les 1C-AT et les 2C-AT.

Pour les logiques de temps linéaire, le résultat d'indécidabilité vaut toujours pour le model checking de *TLTL* sur les 1C-AT et les 2C-AT, il repose sur la possibilité de coder une exécution d'une machine à deux compteurs avec une formule et l'automate (sans horloge) n'a qu'à générer toutes les exécutions possible pour un certain alphabet. Pour $TLTL_{\leq,\geq}$, le model checking est toujours PSPACE-complet : il est déjà PSPACE-dur pour *LTL* et dans PSPACE pour les automates à n horloges.

4.6 Bisimulation temporisée

Tester si deux automates temporisés sont bisimilaires est un problème difficile :

Théorème 22 ([LS00b]) – *Vérifier toute relation comprise entre la simulation forte et la bisimulation forte sur les automates temporisés est un problème EXPTIME-dur.*
– *Tester si deux automates temporisés sont fortement (bi)similaires est un problème EXPTIME-complet.*

Pour le premier point, le coté "dur" vient de la construction d'une structure $\mathcal{S}$ contenant deux états s et s' à partir d'un mot w et d'une machine de Turing alternante en espace polynomial $\mathcal{M}$ telle que (1) si $\mathcal{M}$ n'accepte pas w, alors s et s' sont bisimilaires, et (2) si $\mathcal{M}$ accepte w, alors s' ne simule pas s. Ainsi pour toute relation R entre la simulation et la bisimulation, on a $\mathcal{M}$ n'accepte pas w ssi sRs'. La "structure" $\mathcal{S}$ peut être soit une composition parallèle d'automates finis (un système non plat), soit un automate temporisé.

Pour le second point, l'appartenance à EXPTIME vient du fait que la simulation ou la bisimulation se code facilement avec le μ-calcul temporisé. En effet, considérons deux automates sans invariant A_1 et A_2 et leur composition parallèle $(A_1|A_2)$ où leurs actions respectives sont non-synchronisées (sauf le temps) mais seulement étiquetées par le numéro de l'automate exécutant la transition. Alors on a : A_1 est bisimilaire à A_2 ssi $(A_1|A_2) \models Z$ avec :

$$Z =_\nu \bigwedge_{a\in\Sigma} [a_1]\,\langle a_2\rangle\, Z \wedge \bigwedge_{a\in\Sigma} [a_2]\,\langle a_1\rangle\, Z \wedge [\delta]\, Z$$

Si les automates ont des invariants, il faut les enlever (un invariant dans un automate bloque l'écoulement du temps pour les deux composants) et compléter la formule ci-dessus mais cela ne pose pas de problème particulier [7] . Nous utilisons ici une composition parallèle $(A_1|A_2)$ mais nous pourrions construire un unique automate de taille polynomiale.

4.7 Vérification des compositions parallèles d'AT

Les résultats que nous avons présentés pour les problèmes de vérification sur les AT de la forme $A \models \varphi$ — à l'exception de ceux portant sur les logiques de temps linéaire —, montrent que leur complexité correspond exactement à celle du problème correspondant pour les compositions parallèles de systèmes non temporisés : du point de vue de la complexité, une horloge se comporte comme un composant.

[7] La construction est donnée dans le guide d'utilisation de CMC (voir la section suivante).

Le fait de considérer des compositions parallèles d'automates temporisés $(A_1|\ldots|A_n)$ ne change pas la complexité :

Théorème 23 ([AL02]) – *L'accessibilité dans les compositions parallèles d'automates temporisés est un problème PSPACE-complet.*
– *Le model checking des compositions parallèles d'automates temporisés pour TCTL est un problème PSPACE-complet.*
– *Le model checking des compositions parallèles d'automates temporisés pour $L_{\mu,\nu}$ ou L_ν est un problème EXPTIME-complet.*
– *Tester la (bi)simulation de deux compositions parallèles d'automates temporisés est un problème EXPTIME-complet.*

Les deux premiers résultats sont obtenus en utilisant un algorithme à la volée sur la composition parallèle. Les deux derniers reposent sur le fait que l'algorithme de model checking appliqué au graphe des régions du produit des automates reste un algorithme dans EXPTIME.

Ces résultats sur la complexité des compositions parallèles d'AT masquent le fait qu'en pratique, ces problèmes sont plus difficiles que ceux de la forme $A \models \varphi$. En effet, pour résoudre un problème $(A_1|\ldots|A_n) \models \varphi$, il faut traiter deux types différents d'explosion combinatoire : l'une provient des contraintes temps-réel (les horloges), la seconde provient de la mise en parallèle. Et si ces deux aspects ne s'additionnent pas du point de vue de la complexité théorique, ils ne se traitent pas facilement *ensemble* en pratique. Par exemple, il est bien connu que l'explosion combinatoire de la partie contrôle (la mise en parallèle) est souvent bien prise en charge par des structures de données comme les BDD [BCM+90]. De même, la partie "contraintes temporisées" peut être efficacement manipulée par des DBM [BM83, Dil90] (même si des problèmes existent [Bou03]). Il reste qu'il n'existe pas de structure de données qui permette une manipulation efficace de ces deux aspects ensemble. Cela reste un point sensible dans l'application du model checking temporisé aujourd'hui. C'est ce qui avait motivé notre intérêt pour la vérification compositionnelle.

4.7.1 Model checking compositionnel

Le model checking compositionnel consiste, étant données une composition $(A_1|\ldots|A_n)$ et une formule φ, à construire une formule φ/A_1 telle que $(A_1|\ldots|A_n) \models \varphi$ ssi $(A_2|\ldots|A_n) \models \varphi/A_1$. En répétant ce processus, $n-1$ fois on se ramène à vérifier que l'automate *nil* vérifie ou non la propriété $\varphi/A_1/\ldots/A_n$. On évite ainsi la construction du modèle associé au produit des automates. Néanmoins la complexité est reportée dans la formule, dont

$(\varphi_1 \wedge \varphi_2)/q = (\varphi_1/q) \ \wedge \ (\varphi_2/q)$	$Z/q \ = \ Z^q$
$(\varphi_1 \vee \varphi_2)/q = (\varphi_1/q) \ \vee \ (\varphi_2/q)$	
$\Big(\langle a \rangle \ \varphi\Big)/q = \bigvee_{q \xrightarrow{g,b,r} q' \text{t.q.} f(b,c)=a} g \wedge (r \ \underline{\texttt{in}} \ \texttt{Inv}(q')) \wedge \langle c \rangle \Big(r \ \underline{\texttt{in}} \ \varphi/q'\Big)$	
$\Big([a] \ \varphi\Big)/q = \bigwedge_{q \xrightarrow{g,b,r} q' \text{t.q.} f(b,c)=a} (g \wedge r \ \underline{\texttt{in}} \ \texttt{Inv}(q')) \ \Rightarrow \ [c] \Big(r \ \underline{\texttt{in}} \ \varphi/q'\Big)$	
$\langle \delta \rangle \ \varphi/q \ = \ \langle \delta \rangle \Big(\texttt{Inv}(q) \wedge \varphi/q\Big)$	$[\delta] \ \varphi/q \ = \ [\delta] \Big(\texttt{Inv}(q) \Rightarrow \varphi/q\Big)$
$(x + c \bowtie y + d)/q = (x + c \bowtie y + d)$	$(x \ \underline{\texttt{in}} \ \varphi)/q = x \ \underline{\texttt{in}} \ \Big(\varphi/q\Big)$

TAB. 4.1 – Définition des règles pour construire φ/q.

la taille va croître au fur et à mesure que l'on va *quotienter* les automates. Dans cette opération, le comportement des automates est codé dans la formule. Cette méthode peut néanmoins s'avérer efficace car on applique après chaque quotient des règles de simplification pour essayer de garder la formule aussi petite que possible. De plus, dans la construction du quotient on n'intègre dans la formule que la partie du comportement de l'automate qui est pertinente vis-à-vis de la propriété à vérifier. Cette méthode a été proposée pour le model checking des systèmes non temporisés par Andersen [And95] et nous l'avons étendu pour les systèmes temporisés pour la logique L_ν [LL95, LL98].

Ici nous rappelons la définition du quotient d'une formule par un état de contrôle d'un automate et pour une fonction de synchronisation f. Le tableau 4.1 décrit cette construction pour les principales modalités de L_ν. On peut remarquer que les horloges de l'automate deviennent des horloges de formule et que le nombre de variables de point fixe peut augmenter (au pire chaque variable donne lieu à $|Q|$ variables)

Nous avons alors le théorème suivant :

Théorème 24 ([LL98]) *Étant donnés deux automates temporisés* $A_1 = (Q^1, q_0^1, \Sigma, X^1, \rightarrow_{A_1}, \mathtt{Inv}_1, l_1)$ *et* $A_2 = (Q^2, q_0^2, \Sigma, X^2, \rightarrow_{A_2}, \mathtt{Inv}_2, l_2)$*, une fonction de synchronisation* f *et une formule de* L_ν*, nous avons pour toute configuration étendue* $((q_1, v_1), (q_2, v_2))$ *de* $(A_1|A_2)_f$ *et* u *une valuation pour les horloges de formule :*

$$\Big(((q_1, v_1), (q_2, v_2), u) \models \varphi\Big) \;\; ssi \;\; ((q_2, v_2), uv_1) \models \varphi/q_1$$

Depuis janvier 96, nous développons un outil, CMC [8] (pour Compositional Model Checking) qui implémente cette méthode compositionnelle. En plus du quotient, il existe des règles de simplification pour éviter une trop forte croissance de la taille de la formule. Enfin, un algorithme de calcul de point de fixe est aussi nécessaire pour résoudre la dernière étape de la méthode ($nil \models \varphi$) qui, rappelons-le (cf. le théorème 15), est déjà un problème EXPTIME-complet.

Model checking compositionnel hybride. Nous avons aussi étendu cette méthode compositionnelle aux systèmes hybrides linéaires [CL00]. Pour cela, nous avons défini une extension de L_ν avec des variables dynamiques. Bien sûr, la vérification de ces systèmes est un problème indécidable, l'intérêt de notre méthode est essentiellement de permettre de passer d'un problème de model checking de la forme $(A_1|\ldots|A_n) \models \varphi$ où φ est exprimée dans un formalisme de haut niveau, à un système d'équations (de contraintes) sur les variables dynamiques que l'on peut ensuite essayer de résoudre en utilisant des calculs sur les polyèdres. L'outil CMC a ainsi été étendu pour vérifier des systèmes à chronomètres (on utilise toujours des DBM et on procède par sur-approximations dans les calculs de points fixes).

4.8 Conclusion

Les résultats présentés dans ce chapitre sont récapitulés dans le tableau 4.2, ceux sans référence bibliographique s'obtiennent facilement à partir des autres résultats.

Il ressort de ce tableau qu'en général, du point de vue de la complexité du model checking, ajouter des horloges revient à ajouter des composants, c'est-à-dire considérer la vérification d'une composition parallèle plutôt que celle d'un unique automate (ce constat ne s'applique pas aux logiques de

[8] `http://www.lsv.ens-cachan.fr/~fl/cmcweb.html`

temps linéaire : la présence de contraintes $= c$ ou d'horloges de formule rend le problème indécidable). Il y a donc un coût à la vérification temporisée.

Il faut aussi souligner que les automates temporisés à une horloge bénéficient de propriétés remarquables. Tout d'abord, ils permettent d'utiliser des algorithmes de vérification efficaces, notamment celui en temps polynomial pour $TCTL_{\leq,\geq}$. Ceci est évidemment important en pratique. Ce résultat prolonge celui concernant les SKD à sémantique continue, mais il est encore plus intéressant dans la mesure où le temps continue est plus naturel pour la modélisation. Ensuite il faut noter combien l'ajout d'une seconde horloge entraîne un saut de complexité élevée : passer de une à deux horloges se paie cher du point de vue complexité. La classe des 1C-AT est donc une classe à part avec de bonnes propriétés. D'autres études ont déjà souligné cela : l'inclusion de langages temporisés de mots finis est décidable pour les 1C-AT [OW04] (mais indécidable pour les mots infinis [ADOW05]) et le test du vide pour les automates alternants temporisés à une horloge est aussi décidable [LW05].

	$A \models \varphi$				$(A_1 \mid \ldots \mid A_n) \models \varphi$
	non-temporisé	1C-AT	2C-AT	nC-AT	
Accessibilité	NLOGSPACE-C [VW94]	NLOGSPACE-C [LMS04]	NP-dur [LMS04]	PSPACE-C [CY92]	PSPACE-C
CTL	P-C [CES86, QS83]	P-C	PSPACE-C [LMS04]	PSPACE-C	PSPACE-C
TCTL	-	PSPACE-C [LMS04]	PSPACE-C	PSPACE-C [ACD93]	PSPACE-C
$TCTL_{\leq,\geq}$	-	P-C [LMS04]	PSPACE-C	PSPACE-C	PSPACE-C
L_ν	P-C [CS93]	EXPTIME-C [AL02]	EXPTIME-C	EXPTIME-C [AL02]	EXPTIME-C
$L_{\mu,\nu}$	UP $\cap$ coUP [Jur98]	EXPTIME-C [AL02]	EXPTIME-C	EXPTIME-C [AL02]	EXPTIME-C
LTL	PSPACE-C [SC85]	PSPACE-C	PSPACE-C	PSPACE-C	PSPACE-C
TLTL	-	indéc.	indéc.	indéc. [AFH96]	indéc.
$TLTL_{\leq,\geq}$		PSPACE-C	PSPACE-C	PSPACE-C [AFH96]	PSPACE-C

TAB. 4.2 – Complexité du model checking pour les automates temporisés ($\mathbb{T} = \mathbb{R}_+$)

Chapitre 5

Perspectives

Mes perspectives de recherche s'articulent autour de deux thématiques : les modèles temporisés pour la vérification et le contrôle temporisé. Pour le premier thème, on peut dissocier deux approches : d'une part la recherche d'algorithmes efficaces pour le model checking temporisé, on cherche alors à simplifier la temporisation pour gagner en complexité et éventuellement combiner ces aspects temps-réel à d'autres dimensions (probabilistes, coûts, etc.). D'autre part, on peut au contraire, chercher à étendre les modèles existants pour les rendre plus proches de domaines d'application particuliers.

Model checking temporisé efficace. Les différents travaux présentés dans ce document montrent qu'il existe des algorithmes efficaces pour le model checking temporisé à condition de choisir le bon modèle et la bonne logique. On peut observer que ces algorithmes ont été obtenus de manière ad-hoc, ils ne résultent pas de l'application des méthodes standards sur des cas particuliers : il faut les développer spécialement pour les modèles considérés pour tenir compte de toutes leurs spécificités.

Ce bilan est en soi intéressant puisqu'il permet d'envisager une application de la vérification temporisée à des modèles de taille plus importante. Et même si la restriction sur la temporisation est forte, ces modèles peuvent être suffisants pour certains systèmes.

Mais surtout, une autre motivation de ces travaux est que les contraintes temps-réel ne sont qu'un des aspects à prendre en considération lors de la vérification d'un système complexe. Par exemple, modéliser un système embarqué nécessite souvent de considérer des contraintes liées aux ressources (par ex. l'énergie), de manipuler des données, d'intégrer des aspects probabilistes... Et bien sûr un tel système sera décrit sous la forme de plusieurs

composants reliés par des mécanismes de communication. Chacun de ces aspects est potentiellement une source d'explosion combinatoire et peut entraîner des sauts de complexité importants.

Par exemple, la vérification des protocoles de communication CSMA-CD ou CSMA-CA nécessite la prise en considération du temps et d'aspects probabilistes : ces protocoles peuvent être modélisés par des *automates temporisés probabilistes* [KNS02, DFH+05]. Actuellement, la vérification de ces modèles procède en deux étapes : d'abord on construit un modèle où le temps est abstrait, puis on y introduit les probabilités et ce modèle est alors analysé avec un outil comme PRISM [KNP04]. L'abstraction du temps se fait soit sur la base de l'automate des régions (ou l'automate des zones), soit en considérant un domaine de temps discret (voir [KNS03] pour une comparaison de ces approches). Il est clair que l'approche basée sur la construction des régions ou des zones est très coûteuse et ne peut pas être utilisée sur des systèmes un peu complexes.

Dès lors, il peut être intéressant d'avoir des modèles présentant des mécanismes de temporisation simples qui peuvent se combiner à d'autres aspects, par exemple probabilistes, sans entraîner un coût trop élevé. C'est ce qui a motivé notre travail sur les SKD probabilistes présenté à la fin du chapitre 3.

La vérification du protocole ABR [BFKM03] illustre aussi ce phénomène. En effet, sa modélisation ne nécessitait qu'une seule horloge, mais comportait plusieurs composants et surtout plusieurs paramètres dont il fallait trouver les valeurs correctes. La difficulté reposait donc sur cette combinaison temps+paramètres+composants.

Le problème des ressources est aussi une question importante. Ces dernières années, plusieurs travaux ont porté sur des algorithmes de détection de chemins (ou comportements) *optimaux* dans des automates temporisés munis de coûts. Là encore, la définition de modèles particuliers avec un nombre restreint d'horloges et différentes notions de coût reste un thème de recherche prometteur.

La définition de modèles et de langages de spécification combinant plusieurs aspects quantitatifs comme le temps, les probabilités et les coûts, est donc un sujet important et encore assez peu exploré. Les recherche sur les modèles à temporisation simple sont un point de départ intéressant pour cet objectif.

D'autres modèles temporisés. Certains domaines d'application nécessitent des modèles particuliers, parfois plus riches que les automates temporisés classiques, il peut donc être intéressant de développer des méthodes de

vérification spéciales pour tirer parti de leurs propriétés.

C'est le cas, par exemple, pour les problèmes d'ordonnancement qui utilisent naturellement des chronomètres (horloges pouvant s'arrêter et repartir) pour modéliser la préemption. Mais si le model checking des systèmes à chronomètres est un problème indécidable en général, l'usage de ces chronomètres est ici suffisamment particulier pour espérer pouvoir obtenir des algorithmes sur des modèles dédiés à l'ordonnancement. Cela permettrait d'appliquer les techniques du model checking à des problèmes non solubles par des méthodes standards (par exemple : l'ordonnancement préemptif avec incertitude sur les durées et les périodes des tâches).

Les systèmes sur puces, *Systems on Chip*, sont aussi un domaine d'application intéressant. La taille et la complexité de ces systèmes a considérablement augmenté ces dernières années, et l'usage de méthodes formelles pour leur validation est un domaine en plein essor. La vérification de ces systèmes est un terrain possible pour l'utilisation des modèles à temps discret lorsqu'on les observe à un haut niveau d'abstraction. Les modèles à temps dense restent nécessaires dès lors que l'on souhaite analyser finement leur comportement ou que l'on s'intéresse à leur vitesse en vue de les connecter à d'autres composants.

En général, lorsqu'on utilise des modèles formels pour un système particulier, on a souvent besoin d'adapter leur sémantique ou de tenir compte de certains problèmes liés au codage du système dans le formalisme cible. C'est ce type de problème qui a motivé notre travail dans [BBBL05] : par exemple, la modélisation d'un programme mettant à jour des variables peut contenir des états intermédiaires qui ne sont pas toujours pertinents au sens où ils ne correspondent pas à des états existants dans le système initial. Une propriété d'accessibilité n'a pas à considérer ces états là mais seulement ceux où le système reste un temps non nul. Ces questions sont aussi abordées dans les problèmes de l'*implémentabilité* : lorsqu'on passe d'un modèle formel à une implémentation, on peut naturellement se trouver face à de petites variations sur le comportement (par ex. des horloges). On souhaite donc que les propriétés vérifiées sur le modèle supportent ce passage et les perturbations engendrées.

A travers ces questions, on essaie d'adapter les modèles formels (extension, sémantique etc.) aux contraintes posées par des domaines d'application particuliers.

Contrôle temporisé. Le problème de la synthèse de contrôleur est, d'une certaine manière, une généralisation du model checking : on part d'un sys-

tème $\mathcal{S}$ à contrôler (un processus plongé dans un environnement) et d'une propriété de correction Φ, et on cherche à construire un *contrôleur* $\mathcal{C}$ tel que $\mathcal{C} \otimes \mathcal{S} \models \Phi$ où $\mathcal{C} \otimes \mathcal{S}$ représente le système contrôlé. La synthèse de contrôleur pose des problèmes particuliers dans le cas temporisé. Les notions d'événements observables et contrôlables sont plus délicates. Le type du contrôleur offre aussi plusieurs possibilités : contrôle échantillonné, non-Zénon, etc. Dans [BCL05], nous avons proposé une méthode qui permet de ramener la question de l'existence d'un contrôleur (sous forme de STT) à un problème de model checking pour une extension de L_ν. Je souhaite poursuivre ce travail dans plusieurs directions.

D'abord la question de la forme des stratégies utilisées pour les objectifs de contrôle énoncés avec des formules de L_ν demanderait à être étendue. La solution proposée dans [BCL05] ne permet pas de rendre compte de l'expressivité de la logique choisie. Ce problème rejoint celui, plus général, de la définition des stratégies pour le contrôle temporisé.

Actuellement, le passage de l'existence d'un contrôleur à sa construction effective n'est pas résolu. Cette question est liée aux problèmes de satisfaisabilité des logiques. En effet, les propriétés de compositionnalité de L_ν font que la synthèse de contrôleur se réduit directement à celui de la satisfaisabilité. Ce problème est souvent indécidable pour les logiques temporisées, mais il reste un problème ouvert pour L_ν. Notons aussi que les choix sémantiques influent fortement sur ces questions et qu'il est fort possible que la définition de variantes de L_ν munies d'une sémantique appropriée permette d'obtenir des résultats de décidabilité. Enfin, au delà de la question de la décidabilité, la mise au point de techniques symboliques pour résoudre ces questions sera nécessaire pour les appliquer concrètement.

L'énoncé de la contrôlabilité d'un système demande des formalismes particuliers. La différence de statut entre les évènements contrôlables et incontrôlables a motivé la définition d'un formalisme comme *ATL* [AHK02] dans le cadre non temporisé. Des tels formalismes où il est possible de quantifier sur les stratégies sont des extensions très intéressantes des logiques temporelles classiques. De telles logiques pour le cadre temporisé restent aujourd'hui à définir, ce problème est aussi relié à la question de la forme des stratégies.

Enfin la question du contrôle rejoint aussi le problème évoqué précédemment de l'analyse quantitative sur des coûts associés aux systèmes : chercher des exécutions où la fonction de coût vérifie telle ou telle propriété peut se voir aussi sous la forme d'un problème de contrôle particulier.

Annexe A

Autres travaux — expressivité des logiques temporelles

A.1 Logique temporelle avec passé

La plupart des logiques temporelles classiques (*CTL* ou *LTL*) ne contiennent pas d'opérateurs du passé (S pour "Since" et X^{-1} pour "Previous") permettant de faire référence aux instants antérieurs du système. Pourtant, les opérateurs du passé facilitent l'expression de nombreuses propriétés et rendent les spécifications plus naturelles.

Nous avons travaillé sur la hiérarchie classique des logiques du temps arborescent et étudié l'effet, en terme d'expressivité, de l'ajout des opérateurs du passé. Nous avons aussi introduit un nouvel opérateur (noté N pour "Now") utile pour la spécification avec des opérateurs du passé. Nous avons notamment montré dans quel cas il est possible d'obtenir des traductions de logique avec passé vers des logiques "pur futur". Les techniques mises en oeuvre sont proches de celles utilisées dans [Gab89] pour le temps linéaire. Ces résultats ont été publiés dans [LS97, LS00a].

Pour les logiques de temps linéaire, nous avons montré que les opérateurs du passé apportaient de la *concision*, c'est-à-dire qu'il existe des formules utilisant des opérateurs du passé pour lesquelles toutes les formules de *LTL* équivalentes ont une taille au moins exponentielle par rapport à la formule de départ. Ce résultat donne un fondement théorique au constat (bien connu [LPZ85]) selon lequel les opérateurs du passé sont *pratiques* pour énoncer des propriétés. Nous avons aussi montré que la logique PLTL$+\mathsf{N}$ est

exponentiellement plus succincte que PLTL. Nous avons enfin proposé un algorithme de model checking et de satisfaisabilité pour PLTL+N fondé sur les automates alternants et montré que ces deux problèmes étaient EXPSPACE-complet. Ces résultats ont été publiés dans [LMS02b].

A.2 Logique temporelle quantitative

Nous avons étudié des extensions quantitatives de *CTL* interprétées sur des structures de Kripke avec "tick" : le temps est alors discret et avance lorsqu'on arrive dans un état étiqueté par la proposition "tick". Nous avons étudié des questions d'expressivité de plusieurs variantes de ces logiques ($TCTL$, $TCTL_{\leq,\geq}$, $TCTL_h$, ...) et obtenu des résultats de concision. En effet, dans le cadre des structures de Kripke avec "tick", chacune de ces logiques peut se traduire en *CTL*, mais cette traduction a un coût exponentiel. Nous avons aussi considéré ces problèmes dans le cas des logiques sans opérateur Next. Voir [LST03].

A.3 Logique modale temporisée L_ν

Dans [LLW95], nous avons avons travaillé sur l'expressivité de la logique modale temporisée L_ν (cf. chapitre 2). Nous avons notamment montré comment construire des formules caractéristiques (vis-à-vis de la bisimulation forte temporisée $\approx$) pour les automates temporisés : étant donné un automate temporisé A, on construit une formule Φ_A telle que, pour tout automate B, on a $B \models \Phi_A$ ssi $A \approx B$. Ce type de construction peut aussi s'obtenir avec la méthode du quotient utilisée pour le model checking compositionnel.

En ce qui concerne les problèmes de satisfaisabilité de L_ν, nous avons notamment décrit un algorithme, pour, étant donnés une formule Φ et deux entiers N et C, construire, s'il existe, un automate A avec N horloges et C comme constante maximale tel que $A \models \Phi$. Le problème de la satisfaisabilité reste un problème ouvert.

A.4 Sémantique "presque partout" pour $TCTL$

Nous avons proposé une variante de l'opérateur Until, notée U^{a} ("Until almost everywhere") dont la sémantique est la suivante : un chemin vérifie $\varphi \mathsf{U}^{\mathrm{a}}_{\sim c} \psi$ ssi il existe un intervalle de mesure non nulle vérifiant uniformément ψ et contenant une position p à distance $\sim c$ de l'état initial, et tel que l'ensemble des positions précédant p et vérifiant $\neg\varphi$ est de mesure nulle. Cette

sémantique est motivée par la volonté de ne pas considérer les états intermédiaires où l'on ne fait que passer en temps nul, le long d'une exécution. Il est courant que de tels états apparaissent dans les modèles, mais ils sont généralement dus à des contraintes liées aux formalismes utilisés et ne sont pas pertinents pour la vérification.

Nous avons donc introduit cette version de Until dans *TCTL* et montré qu'elle n'était pas exprimable avec la version standard de *TCTL*. Cette preuve enrichit des techniques souvent utilisées pour les logiques non-temporisées. Nous avons aussi montré que l'inverse (exprimer le Until standard avec le Until "presque partout") n'était pas non plus possible. Enfin, nous avons montré que le model checking était décidable (toujours via le graphe des régions) sans surcoût de complexité. Ces résultats sont présentés dans [BBBL05].

A.5 Logique modale temporisée pour le contrôle

Le problème du contrôle consiste à synthétiser un contrôleur qui permet de piloter un système de manière à ce qu'il vérifie une certaine propriété de correction. C'est un problème classique dans le cas non-temporisé. Nous avons défini une extension de L_ν avec un nouvel opérateur ($[\delta\rangle$) pour exprimer la *contrôlabilité* d'un système. L'existence d'un contrôleur se ramène alors à un problème de model checking sur le système (un automate temporisé) à contrôler.

Nous avons aussi montré que cet opérateur augmentait l'expressivité de L_ν mais que le model checking restait EXPTIME-complet. Cette extension a été implémentée dans CMC. Ces résultats sont décrits dans [BCL05]

Bibliographie

[ABBL98] L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen. The power of reachability testing for timed automata. In V. Arvind and R. Ramanujam, editors, *Proceedings of the 18th Conference on Fundations of Software Technology and Theoretical Computer Science (FSTTCS'98)*, volume 1530 of *Lecture Notes in Computer Science*, pages 245–256, Chennai, India, December 1998. Springer.

[ABBL03] L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen. The power of reachability testing for timed automata. *Theoretical Computer Science*, 300(1-3) :411–475, May 2003.

[ABL98] L. Aceto, A. Burgueño, and K. G. Larsen. Model checking via reachability testing for timed automata. In *Proc. 4th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS '98), Lisbon, Portugal, Mar. 1998*, volume 1384 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 1998.

[ACD93] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1) :2–34, 1993.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1) :3–34, 1995.

[AD90] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proc. 17th Int. Coll. Automata, Languages, and Programming (ICALP '90), Warwick University, England, July 1990*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[ADOW05] P. A. Abdulla, J. Deneux, J. Ouaknine, and J. Worrell. Decidability and complexity results for timed automata via channel machines. In *Proc. 32nd Int. Coll. Automata, Languages, and Programming (ICALP 2005), Lisboa, Portugal, July 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 1089–1101. Springer, 2005.

[AFH96] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1) :116–146, 1996.

[AH92] R. Alur and T. A. Henzinger. Logics and models of real time : A survey. In *Real-Time : Theory in Practice, Proc. REX Workshop, Mook, NL, June 1991*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer, 1992.

[AH93] R. Alur and T. A. Henzinger. Real-time logics : Complexity and expressiveness. *Information and Computation*, 104(1) :35–77, 1993.

[AH94] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1) :181–203, 1994.

[AHK02] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5) :672–713, 2002.

[AL02] L. Aceto and F. Laroussinie. Is your model checker on time ? On the complexity of model checking for timed modal logics. *Journal of Logic and Algebraic Programming*, 52-53 :7–51, August 2002.

[Alu91] R. Alur. *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Stanford Univ., August 1991. Available as Tech. Report STAN-CS-91-1378.

[And95] H. R. Andersen. Partial model checking (extended abstract). In *Proc. 10th IEEE Symp. Logic in Computer Science (LICS '95), San Diego, CA, USA, June 1995*, pages 398–407. IEEE Comp. Soc. Press, 1995.

[BA95] A. Bianco and L. d. Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. 15th Conf. Found. of Software Technology and Theor. Comp. Sci. (FST&TCS '95), Bangalore, India, Dec. 1995*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 1995.

[BBBL05] H. Bel Mokadem, B. Bérard, P. Bouyer, and F. Laroussinie. A new modality for almost everywhere properties in timed automata. In M. Abadi and L. de Alfaro, editors, *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, Lecture Notes in Computer Science, San Francisco, CA, USA, August 2005. Springer. To appear.

[BC05] P. Bouyer and F. Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 2005. To appear.

[BCL05] P. Bouyer, F. Cassez, and F. Laroussinie. Modal logics for timed control. In M. Abadi and L. de Alfaro, editors, *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, Lecture Notes in Computer Science, San Francisco, CA, USA, August 2005. Springer. To appear.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking : 10^{20} states and beyond. In *Proc. 5th IEEE Symp. Logic in Computer Science (LICS '90), Philadelphia, PA, USA, June 1990*, pages 428–439. IEEE Comp. Soc. Press, 1990.

[BCM05] P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of tptl and mtl. Technical Report 05, Laboratoire Specification et Vérification, May 2005.

[BDFP04] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3) :291–345, August 2004.

[BDGP98] B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2) :145–182, November 1998.

[BFKM03] B. Bérard, L. Fribourg, F. Klay, and J.-F. Monin. A compared study of two correctness proofs for the standardized algorithm of ABR conformance. *Formal Methods in System Design*, 22(1) :59–86, January 2003.

[BM83] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In *Proc. IFIP 9th World Computer Congress*, volume 83 of *Information Processing*, pages 41–46. North-Holland/ IFIP, 1983.

[Bou03] P. Bouyer. Untameable timed automata ! In H. Alt and M. Habib, editors, *Proceedings of the 20th Annual Symposium on*

Theoretical Aspects of Computer Science (STACS'03), volume 2607 of *Lecture Notes in Computer Science*, pages 620–631, Berlin, Germany, February 2003. Springer.

[CC95] S. Campos and E. M. Clarke. Real-time symbolic model checking for discrete time models. In T. Rus and C. Rattray, editors, *Theories and Experiences for Real-Time System Development*, volume 2 of *AMAST Series in Computing*, pages 129–145. World Scientific, 1995.

[CCG00] S. Campos, E. M. Clarke, and O. Grumberg. Selective quantitative analysis and interval model checking : Verifying different facets of a system. *Formal Methods in System Design*, 17(2) :163–192, 2000.

[CCM+94] S. Campos, E. M. Clarke, W. R. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *Proc. 15th IEEE Real-Time Systems Symposium (RTSS '94), San Juan, Puerto Rico, Dec. 1994*, pages 266–270. IEEE Comp. Soc. Press, 1994.

[CES83] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications : A practical approach. In *Proc. 10th ACM Symp. Principles of Programming Languages (POPL '83), Austin, TX, USA, Jan. 1983*, pages 117–126, 1983.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2) :244–263, 1986.

[CL00] F. Cassez and F. Laroussinie. Model-checking for hybrid systems by quotienting and constraints solving. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 373–388, Chicago, Illinois, USA, July 2000. Springer.

[CS93] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2(2) :121–147, 1993.

[CTM+99] S. Campos, M. Teixeira, M. Minea, A. Kuehlmann, and E. M. Clarke. Model checking semi-continuous time models using BDDs. In *Proc. 1st Int. Workshop on Symbolic Model Checking*

(SMC'99), Trento, Italy, July 1999, volume 23(2) of *Electronic Notes in Theor. Comp. Sci.* Elsevier Science, 1999.

[CY92] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4) :385–415, 1992.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[DFH+05] M. Duflot, L. Fribourg, Th. Hérault, R. Lassaigne, F. Magniette, S. Messika, S. Peyronnet, and C. Picaronny. Probabilistic model checking of the CSMA/CD protocol using PRISM and APCM. In m. R. A. Huth, editor, *Proceedings of the 4th International Workshop on Automated Verification of Critical Systems (AVoCS'04)*, volume 128 of *Electronic Notes in Theoretical Computer Science*, pages 195–214, London, UK, May 2005. Elsevier Science Publishers.

[Dil90] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Int. Workshop Automatic Verification Methods for Finite State Systems (CAV '89), Grenoble, June 1989*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1990.

[Dim00] C. Dima. Real-time automata and the Kleene algebra of sets of real numbers. In *Proc. of STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000*, volume 1770 of *Lecture Notes in Computer Science*, pages 279–289, 2000.

[DLS02] S. Demri, F. Laroussinie, and Ph. Schnoebelen. A parametric analysis of the state explosion problem in model checking (extended abstract). In H. Alt and A. Ferreira, editors, *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 620–631, Antibes Juan-les-Pins, France, March 2002. Springer.

[DS02] S. Demri and Ph. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, 174(1) :84–103, April 2002.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited : On branching versus linear time temporal logic. *Journal of the ACM*, 33(1) :151–178, 1986.

[EJS01] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for the μ-calculus and its fragments. *Theoretical Computer Science*, 258(1–2) :491–522, 2001.

[EL86] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proc. 1st IEEE Symp. Logic in Computer Science (LICS '86), Cambridge, MA, USA, June 1986*, pages 267–278. IEEE Comp. Soc. Press, 1986.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. v. Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier Science, 1990.

[EMSS92] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Systems*, 4(4) :331–352, 1992.

[Gab89] D. M. Gabbay. The declarative past and imperative future : Executable temporal logic for interactive systems. In *Proc. Workshop Temporal Logic in Specification, Altrincham, UK, Apr. 1987*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer, 1989.

[GHKK05] H. Gruber, M. Holzer, A. Kiehn, and B. König. On timed automata with discrete time – structural and language theoretical characterization. In *Proc. 9th Int. Conf. Developments in Language Theory (DLT 2005)*, volume 3572 of *Lecture Notes in Computer Science*, pages 272 – 283. Springer, June 2005.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[Gla90] R. J. v. Glabbeek. The linear time – branching time spectrum. In *Proc. Theories of Concurrency (CONCUR '90), Amsterdam, NL, Aug. 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.

[GO01] P. Gastin and D. Oddoux. Fast LTL to büchi automata translation. In *Proc. 13th Int. Conf. Computer Aided Verification (CAV 2001), Paris, France, July 2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proc. 11th IEEE Symp. Logic in Computer Science (LICS '96), New Brunswick, NJ, USA, July 1996*, pages 278–292. IEEE Comp. Soc. Press, 1996.

[HJ96] D. V. Hung and W. Ji. On the design of hybrid control systems using automata models. In *Proc. 16th Conf. Found. of Software Technology and Theor. Comp. Sci. (FST&TCS'96), Hyderabad, India, Dec. 1996*, volume 1180 of *Lecture Notes in Computer Science*, pages 156–167. Springer, 1996.

[HKPV98] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1) :94–124, 1998.

[HKV02] D. Harel, O. Kupferman, and M. Y. Vardi. On the complexity of verifying concurrent transition systems. *Information and Computation*, 173(2) :143–161, 2002.

[HKWT95] T. A. Henzinger, P. W. Kopke, and H. Wong-Toi. The expressive power of clocks. In *"Proc. 22nd Int. Coll. Automata, Languages, and Programming (ICALP '95), Szeged, Hungary, July 1995"*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer, 1995.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1) :137–161, 1985.

[HMP94] T. A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112(2) :273–337, 1994.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2) :193–244, 1994.

[JM96] L. Jategaonkar and A. R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154(1) :107–143, 1996.

[Jur98] M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68(3) :119–124, November 1998.

[KNP04] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 2.0 : A tool for probabilistic model checking. In *Proc. 1st Int. Conf. on Quantitative Evaluation of Systems (QEST 2004), Enschede, The Netherlands, Sep. 2004*, pages 322–323. IEEE Comp. Soc. Press, 2004.

[KNS02] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of the ieee 802.11 wireless local area network

protocol. In *Proc. 2nd Joint Int. Workshop on Process Algebra and Performance Modelling and Probabilistic Methods in Verification (PAPM-PROBMIV 2002)*, volume 2399 of *lncs*, pages 169–187. Springer, 2002.

[KNS03] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the ieee1394 firewire root contention protocol. *Formal Aspects of Computing*, 14(3) :295–318, 2003.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4) :255–299, 1990.

[KVdR83] R. Koymans, J. Vytopil, and W.-P. de Roever. Real-time programming and asynchronous message passing. In *Proc. 2nd ACM Symp. Principles of Distributed Computing (PODC '83), Montreal, Canada, Aug. 1983*, pages 187–197, 1983.

[KVW00] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2) :312–360, 2000.

[Lar90] K. G. Larsen. Proof systems for Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72 :265–288, 1990.

[LL95] F. Laroussinie and K. G. Larsen. Compositional model-checking of real time systems. In I. Lee and S. A. Smolka, editors, *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *Lecture Notes in Computer Science*, pages 529–539, Philadelphia, Pennsylvania, USA, August 1995. Springer.

[LL98] F. Laroussinie and K. G. Larsen. CMC : A tool for compositional model-checking of real-time systems. In S. Budkowski, A. R. Cavalli, and E. Najm, editors, *Proceedings of IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE'XI) and Protocol Specification, Testing and Verification (PSTV'XVIII)*, volume 135 of *IFIP Conference Proceedings*, pages 439–456, Paris, France, November 1998. Kluwer Academic Publishers.

[LLW95] F. Laroussinie, K. G. Larsen, and C. Weise. From timed automata to logic – and back. In J. Wiedermann and P. Hájek, editors, *Proceedings of the 20th International Symposium on Mathematical Fundations of Computer Science (MFCS'95)*,

volume 969 of *Lecture Notes in Computer Science*, pages 27–41, Prague, Czech Republic, August 1995. Springer.

[LMS01] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking CTL^+ and FCTL is hard. In F. Honsell and M. Miculan, editors, *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'01)*, volume 2030 of *Lecture Notes in Computer Science*, pages 318–331, Genova, Italy, April 2001. Springer.

[LMS02a] F. Laroussinie, N. Markey, and Ph. Schnoebelen. On model checking durational Kripke structures (extended abstract). In M. Nielsen and U. Engberg, editors, *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'02)*, volume 2303 of *Lecture Notes in Computer Science*, pages 264–279, Grenoble, France, April 2002. Springer.

[LMS02b] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 383–392, Copenhagen, Denmark, July 2002. IEEE Computer Society Press.

[LMS04] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking timed automata with one or two clocks. In Ph. Gardner and N. Yoshida, editors, *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401, London, UK, August 2004. Springer.

[LMS05] F. Laroussinie, N. Markey, and P. Schnoebelen. Efficient timed model checking for discrete-time systems. Submitted to Theoretical Computer Science, January 2005.

[LPY95] K. G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Proc. 10th Int. Conf. Fundamentals of Computation Theory (FCT '95), Dresden, Germany, Aug. 1995*, volume 965 of *Lecture Notes in Computer Science*, pages 62–88. Springer, 1995.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Proc. Logics of Programs Workshop, Brooklyn College, NY, USA, June 1985*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer, 1985.

[LS95] F. Laroussinie and Ph. Schnoebelen. A hierarchy of temporal logics with past. *Theoretical Computer Science*, 148(2) :303–324, September 1995.

[LS97] F. Laroussinie and Ph. Schnoebelen. Specification in CTL+Past, verification in CTL. In C. Palamidessi and J. Parrow, editors, *Proceedings of the 4th International Workshop on Expressiveness in Concurrency (EXPRESS'97)*, volume 7 of *Electronic Notes in Theoretical Computer Science*, Santa Margherita Ligure, Italy, September 1997. Elsevier Science Publishers.

[LS00a] F. Laroussinie and Ph. Schnoebelen. Specification in CTL+past for verification in CTL. *Information and Computation*, 156(1-2) :236–263, January 2000.

[LS00b] F. Laroussinie and Ph. Schnoebelen. The state-explosion problem from trace to bisimulation equivalence. In J. Tiuryn, editor, *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2000)*, volume 1784 of *Lecture Notes in Computer Science*, pages 192–207, Berlin, Germany, March 2000. Springer.

[LS05] F. Laroussinie and J. Sproston. Model checking durational probabilistic systems. In V. Sassone, editor, *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 140–154, Edinburgh, U.K., April 2005. Springer.

[LST00] F. Laroussinie, Ph. Schnoebelen, and M. Turuani. On the expressivity and complexity of quantitative branching-time temporal logics. In G. H. Gonnet, D. Panario, and A. Viola, editors, *Proceedings of the 4th Latin American Symposium on Theoretical Informatics (LATIN 2000)*, volume 1776 of *Lecture Notes in Computer Science*, pages 437–446, Punta des Este, Uruguay, April 2000. Springer.

[LST03] F. Laroussinie, Ph. Schnoebelen, and M. Turuani. On the expressivity and complexity of quantitative branching-time temporal logics. *Theoretical Computer Science*, 297(1-3) :297–315, March 2003.

[LW05] S. Lasota and I. Walukiewicz. Alternating timed automata. In *Proc. 8th Int. Conf. Foundations of Software Science and*

Computation Structures (FOSSACS 2005), Edinburgh, Scotland, UK, Apr. 2005, volume 3441 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2005.

[Mar04] N. Markey. Past is for free : On the complexity of verifying linear temporal properties with past. *Acta Informatica*, 40(6-7) :431–458, May 2004.

[McM93] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic, 1993.

[MS04] N. Markey and Ph. Schnoebelen. TSMV : A symbolic model checker for quantitative analysis of systems. In *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems (QEST'04)*, pages 330–331, Enschede, The Netherlands, September 2004. IEEE Computer Society Press.

[OW04] J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata : Closing a decidability gap. In *Proc. 19th IEEE Symp. Logic in Computer Science (LICS 2004), Turku, Finland, July 2004*, pages 54–63. IEEE Comp. Soc. Press, 2004.

[OW05] J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *Proc. 20th IEEE Symp. Logic in Computer Science (LICS 2005), Chicago, IL, USA, June 2005*, 2005.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[QS83] J.-P. Queille and J. Sifakis. Fairness and related properties in transition systems. A temporal logic to deal with fairness. *Acta Informatica*, 19(3) :195–220, 1983.

[Ras99] J. F. Raskin. *Logics, Automata and Classical Theories for Deciding Real Time*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, 1999.

[Saw03] Z. Sawa. Equivalence checking of non-flat systems is EXPTIME-hard. In *Proc. 14th Int. Conf. Concurrency Theory (CONCUR 2003), Marseille, France, Sep. 2003*, volume 2761 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2003.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3) :733–749, 1985.

[Sch01] Ph. Schnoebelen. *Spécification et vérification des systèmes concurrents*. Mémoire d'habilitation, Université Paris 7, Paris, France, October 2001.

[Sti01] C. Stirling. *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer, 2001.

[Var95] M. Y. Vardi. Alternating automata and program verification. In *Computer Science Today. Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1995.

[VW94] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1) :1–37, 1994.