M. Turuani

Logique Temporelle Temporisée pour la Vérification de Programmes : Expressivité et Complexité

Research Report LSV–99–8, Dec. 1999

Laboratoire Spécification et Vérification

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Research Report LSV–99–8, Lab. Spécification et Vérification, CNRS & ENS de Cachan, France, Dec. 1999
First published as *Mémoire de DEA, DEA Programmation, Paris, Sep. 1999.*

Stage réalisé au Laboratoire de Spécification et de Vérification de l'Ecole Normale Supérieure de Cachan, sous la direction de Mr P.Schnoebelen et Mr F.Laroussinie que je remercie de m'avoir guidé tout ce temps.

Table des matières

Résumé

On étudie $TCTL_s$, une extension de la logique temporelle CTL. $TCTL_s$ permet d'énoncer des propriétés quantitatives sur les comportements et peut donc être vue comme une logique temporisée (ou "temps-réel").

Par rapport à d'autres extensions similaires, $TCTL_s$ a deux avantages : on utilise des structures de Kripke classiques, et le model-checking se fait en temps polynômial.

Notre étude se concentre sur le pouvoir d'expression de divers fragments de $TCTL_s$, et sur la complexité du model-checking. Nos résultats sur l'expressivité de $TCTL_s$ ont une portée plus générale et s'appliquent aux extensions de CTL pour les automates temporisés.

1 Introduction

Logiques temporelles. Les logiques temporelles (CTL, LTL) [Pnu77, Eme90] sont très utilisées pour la spécification de systèmes réactifs [Eme90]. Elles utilisent un modèle simple et disposent d'algorithmes de model-checking[1] efficaces [QS82, CES86, SBB$^+$99]. Avec les logiques temporelles, on peut exprimer des propriétés réactives complexes, par exemple : pour tout chemin, à tous moments sur ces chemins, si une alarme se déclenche, alors elle conduit inévitablement à une réponse, ce qui s'écrit $\mathsf{AG}(alarme \Rightarrow \mathsf{AF}\ reponse)$.

L'un des avantages de ces logiques est qu'elles reposent sur un modèle simple et général (une structure de Kripke, c'est-à-dire un automate pour lequel chaque état vérifie un certain nombre de propositions atomiques). Un autre avantage de ces logiques est l'existence d'algorithmes de model-checking efficaces.

Logiques temporisées (ou temps-réel). Les logiques temporelles simples, comme CTL ou LTL, ne permettent pas (ou pas facilement) d'exprimer des aspects *quantitatifs* sur le temps. Elles se limitent en fait à des aspects "avant/après". Cette limitation est parfois gênante : par exemple, on souhaite souvent pouvoir écrire aisément le fait qu'il existe un chemin menant à une certaine situation *avant un délai maximal*.

Les logiques temps-réel [AH92] permettent d'exprimer ce genre de propriétés pour un modèle de temps continu, sur des automates temporisés. Si on reprend l'exemple précédent, on exprime que la réponse se produit moins de 5 unités de temps après l'alarme, en écrivant $\mathsf{AG}(alarme \Rightarrow \mathsf{AF}_{\leq 5}\ reponse)$. De telles logiques sont actuellement mises en oeuvre dans des outils tels que HyTech [HHW97], Uppaal [LPY97], ou Kronos [Yov97] (voir [SBB$^+$99]) mais leur model-checking est au moins *PSPACE-dur* [ACD93, AL99].

Une autre extension : $RTCTL$. Il existe une autre extension de CTL étudiée par Emerson, Mok, Sistla, et Srinivasan dans [EMSS91] : $RTCTL$. À la différence des logiques temps-réel sur des automates temporisés, $RTCTL$ utilise

[1] Le model-checking est la vérification automatique qu'un modèle donné satisfait une formule donnée

une notion de temps discret, et conserve les mêmes structures de Kripke que CTL. De plus, le model-checking pour $RTCTL$ est linéaire en temps, et on pourra exprimer des aspects quantitatifs sur l'ordonnancement des actions à la manière des logiques temporisées. Le temps est ici discret : il "avancera" d'une unité à chaque transition. Du point de vue de la modélisation, cette approche permet de ne pas abandonner les modèles habituellement utilisés.

Notre travail de stage. On cherche ici à étendre la logique proposée dans [EMSS91] tout en conservant un model-checking polynômial en temps : nous proposons la logique $TCTL_s$, pour laquelle on considère des intervalles comme contraintes temporelles, et où toutes les transitions ne font pas nécessairement "avancer" le temps.

La première partie de notre étude est consacrée à l'expressivité de $TCTL_s$ et de ses fragments (Chapitres 4 et 5). La question ici est de savoir ce qu'apporte l'ajout de contraintes $\leq k$, $\geq k$, et $= k$. Nous montrerons qu'elles ne sont pas redondantes et que $= k$ est, en un certain sens, plus coûteuse que $\leq$ ou $\geq$ (Chapitre 6).

Ensuite, nous étudierons une notion de bisimulation caractérisant l'égalité selon $TCTL_s$ (Chapitre 7).

Nous verrons enfin quels résultats on peut attendre d'une autre extension, $TCTL_c$, décrite dans [LST99] (Chapitre 10). Elle est plus générale que $TCTL_s$, mais son model-checking est coûteux dans le cas général.

Travaux connexes Emerson et al. ont déjà étudié certaines extensions de $RTCTL$, mais sans jamais chercher à mesurer l'expressivité. [ET97] propose des contraintes "régulières" mais le model-checking devient coûteux. [ES89] introduit une variante de notre *tick*, et considère des contraintes avec paramètres : là aussi, le model-checking devient coûteux.

[YMW97] introduit une logique semblable à $TCTL_c$, mais n'étudie ni la complexité structurelle, ni l'expressivité.

Nos algorithmes de model-checking pour $TCTL_s$ étendent ceux de [EMSS91, ET99] (N.B : [ET99] a été publié pendant la rédaction de ce mémoire). Nos résultats de complexité pour $TCTL_c$ s'inspirent de [AL99].

2 Syntaxe de $TCTL_s$

Soit $\mathbb{N}$ l'ensemble des entiers naturels positifs. Ce seront les seuls entiers considérés.

Notons $PA = \{prop1, prop2, ..., reponse, ...\}$ un ensemble de propositions atomiques, contenant une proposition particulière $tick$.

Définition 2.1 ($TCTL_s$) *Les formules de $TCTL_s$ sont contruites sur la grammaire :*

$f := \neg f' \mid f_1 \wedge f_2 \mid \mathsf{E}\, f_1 \,\mathsf{U}_I\, f_2 \mid \mathsf{A}\, f_1 \,\mathsf{U}_I\, f_2 \mid \mathsf{EX}\, f' \mid \mathsf{AX}\, f' \mid prop_1 \mid ..$
$I := [a_1..b_1] \cup [a_2..b_2] \cup ... \cup [a_n..b_n]$

avec $prop_1, prop_2, \ ... \in PA$.
I est appelé contrainte de temporisation. *Les a_i et b_i sont des entiers tels que* $0 \leq a_1 \leq b_1 < a_2 \leq b_2 < ... \leq b_n$, *et* $b_n \in \mathbb{N} \cup \{\infty\}$

I est un terme, mais il s'interprète évidemment comme un ensemble d'entiers : $[\![I]\!] \subseteq \mathbb{N}$. On confondra souvent ces deux notions.

Intuitivement, $\mathsf{E}\, f_1 \,\mathsf{U}_I\, f_2$ correspond à "Il existe un chemin vérifiant f_1 jusqu'à vérifier un f_2 se produisant après une durée comprise dans I. De la même manière, $\mathsf{A}\, f_1 \,\mathsf{U}_I\, f_2$ correspond à "*Tous les chemins* vérifient $f_1 \ \mathsf{U}_I \ f_2$".

On définit également des combinateurs dérivés :

Définition 2.2 (Combinateurs dérivés) *Posons :*

- $false$: $p \wedge \neg p$, *pour* $p \in PA$,
- $true$: $\neg false$,
- $\mathsf{EF}_I f$: $\mathsf{E}\, true \,\mathsf{U}_I\, f$ *"Il existe un état atteignable vérifiant f pour un temps dans I"*,
- $\mathsf{AF}_I f$: $\mathsf{A}\, true \,\mathsf{U}_I\, f$ *"Inévitablement le long de chaque chemin, il existe un état vérifiant f pour un temps dans I"*,
- $\mathsf{EG}_I f$: $\neg(\mathsf{AF}_I \neg f)$ *"Il existe un chemin pour lequel chaque état ayant un temps dans I vérifie f"*,
- $\mathsf{AG}_I f$: $\neg(\mathsf{EF}_I \neg f)$ *"Pour tout chemin, tout état ayant un temps dans I vérifie f"*.

On peut également définir des notations spécifiques pour des contraintes temporelles de forme particulière : quand I vaut $[0..\infty]$, on omet parfois de l'écrire pour retrouver une formule de CTL, et on note $\leq k$ pour $[0..k]$, $\geq k$ pour $[k..\infty]$, etc...

3 Sémantique

3.1 Structure de Kripke

Les formules de $TCTL_s$ sont évaluées sur des structures de Kripke :

Définition 3.1 (Structure de Kripke) *Une structure de Kripke est un triplet $A = \langle Q, R, l \rangle$, où Q est un ensemble d'états, $R \subseteq Q \times Q$ est une relation de transitions entre les états, et où $l : Q \to 2^{PA}$ associe à chaque $q \in Q$ l'ensemble $l(q) \subseteq PA$ des propositions atomiques vérifiées dans q.*

Ici, on notera pour tous $q, q' \in Q$, $q \to_R q'$ ssi $(q, q') \in R$.

La proposition atomique $tick$ a un sens particulier : informellement, elle repère les états pour lesquels on observe que le temps vient d'avancer. On considère que le temps “avance” juste avant cet état, i.e. $tick \in l(q)$ ssi toute transition $q' \rightarrow_R q$ fait “avancer” le temps. A l'inverse, $tick \notin l(q)$ et $q' \rightarrow q$ implique que passer de q' à q ne prend aucun temps.

Définition 3.2 (Successeurs et Prédécesseurs) *Pour une structure* $A =< Q, R, l >$ *donnée, et* $\forall q \in Q$*, on définit :*

1. $Succ(q) \stackrel{def}{=} \{r \in Q \;\; tq \; q \rightarrow_R r\}$,
2. $Pred(q) \stackrel{def}{=} \{r \in Q \;\; tq \; r \rightarrow_R q\}$

On suppose de plus que tout état a au moins un successeur (il n'y a pas d'état “bloqué”).

Définition 3.3 (Chemins) *On appelle chemin dans A une séquence infinie d'états en successeurs directs, généralement notés* w*. Les états d'un tel chemin seront notés* $w_0 \rightarrow_R \; .. \; \rightarrow_R w_n \rightarrow_R ...$*, avec* $\forall i, (w_i, w_{i+1}) \in R$*.*

Notons,

1. $Chemins(A)$: l'ensemble des chemins de A.
2. $Chemins(q)$, $q \in Q$: l'ensemble des chemins de A partant de q.

Ainsi, $Chemins(A) = \bigcup_{q \in Q} Chemins(q)$.

Par exemple, on peut définir l'automate suivant :
- $Q = \{p, q\}$
- $R = \{(q, p), (p, p)\}$
- $l = \begin{cases} q \rightarrow \{prop\} \\ p \rightarrow \{prop, tick\} \end{cases}$

En prenant pour convention dans la représentation d'une telle structure de griser les états vérifiant $tick$, on peut voir cet automate comme dans la figure 1.

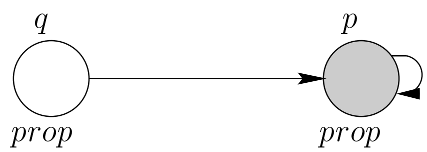


Fig. 1 – Un exemple de structure de Kripke

On a alors : $Chemins(q) = \{q \rightarrow p \rightarrow p \rightarrow ...\}$
Notons w_q ce chemin.

Définition 3.4 (Notion de temps sur un chemin) *Posons, pour un chemin* w *et un entier* i *:*

$$time^i_w \stackrel{def}{=} |\{j \mid 0 < j \leq i \wedge w_j \models tick\}| .$$

C'est le “temps” (nombre de ticks) écoulé pour aller de w_0 à w_i par w.
Par exemple, on a dans la structure de Kripke de la figure 1 : $time^1_{w_q} = 1$.

3.2 Sémantique

On veut ici définir la notion “la formule f est vraie dans l'état q de A”, notée $q \models f$.

Définition 3.5 (Sémantique de $TCTL_s$) *Si f est une formule de $TCTL_s$, et $q \in Q$, définissons $q \models f$ par induction sur la structure de f de la manière suivante :*

1. $q \models prop$, $prop \in PA$, *ssi* $prop \in l(q)$.
2. $q \models \neg f$ *ssi on n'a pas* $q \models f$.
3. $q \models f_1 \wedge f_2$ *ssi* $q \models f_1$ *et* $q \models f_2$.
4. $q \models \mathsf{E}\, f_1\, \mathsf{U}_I\, f_2$ *ssi* $\exists w \in Chemins(q)$, $w \models f_1\ \mathsf{U}_I\ f_2$.
5. $q \models \mathsf{A}\, f_1\, \mathsf{U}_I\, f_2$ *ssi* $\forall w \in Chemins(q)$, $w \models f_1\ \mathsf{U}_I\ f_2$.
6. $q \models \mathsf{EX}\, f$ *ssi* $\exists q' \in Succ(q)$, $q' \models f$.
7. $q \models \mathsf{AX}\, f$ *ssi* $\forall q' \in Succ(q)$, $q' \models f$.
8. $w \models f_1\ \mathsf{U}_I\ f_2$ *ssi* $\exists i$ *tq* $w_i \models f_2$, $\forall 0 \leq j < i$, $w_j \models f_1$, *et* $time^i_w \in I$.

Ainsi, EX et AX ne prennent pas les ticks en compte, i.e. EX ne se réduit pas à $\mathsf{EF}_{[1..1]}$. La présence des opérateurs EX et AX n'est donc pas redondante avec $\mathsf{E}\,\mathsf{U}_I$ et $\mathsf{A}\,\mathsf{U}_I$.

Remarque 3.6 $\mathsf{E}\, prop'\, \mathsf{U}_{\geq a+1}\, prop$ *n'est pas équivalente*[2] *à* $\mathsf{EG}_{\leq a+1}\ \mathsf{E}\, prop'\, \mathsf{U} prop$. *De même,* $\mathsf{A}\, prop'\, \mathsf{U}_{\geq a+1}\, prop$ *n'est pas équivalente à* $\mathsf{AG}_{\leq a+1}\ \mathsf{A}\, prop'\, \mathsf{U} prop$.

En effet, si on reprend la structure de Kripke de la figure 1 et que l'on ajoute une proposition atomique $prop'$ vérifiée nulle-part, on remarque que :
- $q \models \mathsf{EG}_{\leq 1}\ \mathsf{E}\, prop'\, \mathsf{U} prop$ car $q \models \mathsf{E}\, prop'\, \mathsf{U} prop$ ($q \models prop$), et $p \models \mathsf{E}\, prop'\, \mathsf{U} prop$.
- Mais on n'a *pas* $q \models \mathsf{E}\, prop'\, \mathsf{U}_{\geq 1}\, prop$ car on n'a *pas* $q \models prop'$.

On peut également définir la notion de validité (une formule vraie sur n'importe quel état de n'importe quelle structure) :

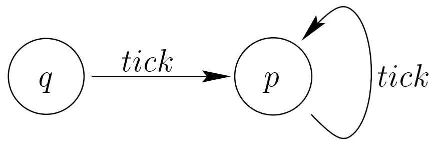
Définition 3.7 *Pour toute formule f de $TCTL_s$, f est valide (noté $\models f$) ssi* $\forall A = \langle Q, R, l \rangle$, $\forall q \in Q$, $q \models f$

[2]La notion d'équivalence est définie au chapitre 3.6. C'est simplement le fait que deux formules soient toujours simultamément vraies ou fausses.

3.3 En pratique : *tick* sur les transitions

Dans la pratique, il peut sembler plus naturel de placer la proposition *tick* sur les transitions plutôt que sur les états. En effet, les états désignent en général des configurations possibles du système à des instants donnés : Il est alors plus naturel d'attribuer un temps d'exécution à la transition entre deux états qu'à un état lui-même.

Par exemple, on aurait pu choisir cette structure à la place de celle de la figure 1 :



Il se trouve en fait que la différence entre mettre les *tick* sur les transitions ou sur les états n'est qu'apparente : on peut traduire les structures avec les *ticks* sur les transitions dans le cadre $TCTL_s$. On obtient une nouvelle structure de taille linéairement plus grande que la précédente, et on conserve la valeur de vérité des formules. (D'où des complexités semblables pour le model-checking.) Voir pour cela l'Annexe.

Ainsi, d'un point de vue théorique, mettre les *tick* sur les états ou les transitions est équivalent. Mais pour établir nos résultats, il est ici plus aisé de considérer les *ticks* sur les états, car cela nous permet de conserver les mêmes structures de Kripke que pour CTL. Nous suivrons donc cette approche.

3.4 Taille d'une formule

Pour mesurer la complexité des algorithmes sur les formules (model-checking, traduction, ...), il est nécessaire de définir la taille (notée $|f|$) des formules de $TCTL_s$. Posons pour cela :

Définition 3.8 (Taille)

1. $|prop| \stackrel{def}{=} 1$ *pour* $prop \in PA$
2. $|\neg f| \stackrel{def}{=} |\mathsf{EX}\, f| \stackrel{def}{=} |\mathsf{AX}\, f| \stackrel{def}{=} 1 + |f|$
3. $|f \wedge g| \stackrel{def}{=} 1 + |f| + |g|$
4. $|\mathsf{E}\, f\, \mathsf{U}_I\, g| \stackrel{def}{=} 1 + |f| + |g| + |I|$
5. $|\mathsf{A}\, f\, \mathsf{U}_I\, g| \stackrel{def}{=} 1 + |f| + |g| + |I|$
6. *Si* $I = \bigcup_{i=1}^{n} [a_i..b_i]$*, alors* $|I| \stackrel{def}{=} \sum_i (\lceil log_2(a_i) \rceil + \lceil log_2(b_i) \rceil)$ *en prenant* 1 *pour* $log_2(\infty)$*,* $log_2(0)$ *et* $log_2(1)$*.*

Intuitivement, $|f|$ est la taille d'une écriture linéaire (et en base 2 pour les entiers) de f.

Certaines applications manipulent les formules de logique sous la forme de dags (graphes acycliques). Cette structure permet de partager des sous-formules identiques et la différence de taille entre le dag et une représentation linéaire des formules peut être importante. On définit donc $|f|_{dag}$:

Définition 3.9 (Taille dag) *Posons :*

$$|f|_{dag} \stackrel{def}{=} \begin{cases} \textit{nombre de sous-formules distinctes de } f \\ + \sum\limits_{I \textit{ dans } f} |I| \end{cases}$$

Alors, le model-checking de CTL est en $O(|Structure| \times |formule|_{dag})$.

Remarque 3.10 *Quand toutes les sous-formules de f sont distinctes, $|f| = |f|_{dag}$.*

Enfin, on peut définir la hauteur temporelle d'une formule :

Définition 3.11 (Hauteur temporelle) *Posons :*

1. $ht(prop) \stackrel{def}{=} 1$, *pour* $prop \in PA$
2. $ht(\neg f) \stackrel{def}{=} ht(f)$
3. $ht(f \wedge g) \stackrel{def}{=} max(ht(f), ht(g))$
4. $ht(\mathsf{EX}\, f\,) \stackrel{def}{=} ht(\mathsf{AX}\, f\,) \stackrel{def}{=} 1 + ht(f)$
5. $ht(\mathsf{E}\, f\, \mathsf{U}_I\, g\,) \stackrel{def}{=} ht(\mathsf{A}\, f\, \mathsf{U}_I\, g\,) \stackrel{def}{=} 1 + max(ht(f), ht(g))$

On a : $\forall f \in TCTL_s$, $ht(f) \leq |f|_{dag} \leq |f| \leq 2^{|f|_{dag}}$

3.5 Sous-classes de $TCTL_s$:

On aura besoin dans la suite de considérer des fragments de $TCTL_s$, construits avec des contraintes de temporisation plus restreintes. Pour cela, on dira qu'une contrainte est de type $\leq$ si elle est de la forme $[0..k]$ (c. à d. "$\leq k$"), de type $\geq$ si elle est de la forme $[k..\infty]$, et de type $=$ si elle est de la forme $[k..k]$. Pour $Ctr \subseteq \{\leq, \geq, =\}$, on note $TCTL_s^{Ctr}$ le fragment de $TCTL_s$ n' utilisant que les contraintes temporelles de Ctr.

On peut alors retrouver les notations de [EMSS91] étendues aux ticks :

$$\begin{array}{rclrcl} RTCTL &=& TCTL_s^{\leq} & RTCTL^{\geq} &=& TCTL_s^{\leq,\geq} \\ RTCTL^{=} &=& TCTL_s^{\leq,=} & CRTCTL &=& TCTL_s^{\leq,\geq,=} \end{array}$$

3.6 Équivalences de formules

La définition d'équivalence que nous utiliserons est l'équivalence sémantique classique :

Définition 3.12 (Équivalence) *Deux formules f et g sont équivalentes, noté $f \equiv g$, ssi pour toute structure de Kripke A, et pout tout état q de A, on a $q \models f$ ssi $q \models g$, i.e. $f \equiv g$ ssi $\models f \Leftrightarrow g$*

Cette équivalence est substitutive :

Proposition 3.13

$$\forall f_1,\ f_2 \in TCTL_s,\ \forall \Omega \text{ contexte de } TCTL_s,\ f_1 \equiv f_2 \Rightarrow \Omega[f_1] \equiv \Omega[f_2]$$

Un contexte est une formule comportant une ou plusieurs occurence(s) d'une proposition atomique supplémentaire, qui est remplacée par f_1 dans $\Omega[f_1]$, f_2 dans $\Omega[f_2]$

Preuve: Ceci se montre aisément par récurrence sur la structure de Ω. ■

Définition 3.14 *On notera $f \overset{A}{\equiv} g$ quand f et g sont équivalentes pour une structure de Kripke $A = \langle Q, R, l\rangle$ donnée, i.e. $\forall q \in Q$, $q \models f$ ssi $q \models g$*

Remarque 3.15 *Cette équivalence est également substitutive.*

Proposition 3.16 (Limitation à $|Q|$) *On a, pour une structure $A = \langle Q, R, l\rangle$ donnée, si $k \geq |Q|$ ou $k = \infty$:*

1. $\mathsf{E}\, f\, \mathsf{U}_{\leq k}\, g \overset{A}{\equiv} \mathsf{E}\, f\, \mathsf{U}_{\leq |Q|-1}\, g$
2. $\mathsf{A}\, f\, \mathsf{U}_{\leq k}\, g \overset{A}{\equiv} \mathsf{A}\, f\, \mathsf{U}_{\leq |Q|-1}\, g$

Preuve: En effet, si $A = \langle Q, R, l\rangle$, $m = |Q|$ et $m \leq k$, on a :
- Si $q \models \mathsf{E}\, f\, \mathsf{U}_{\leq k}\, g$, alors $\exists w \in Chemins(q)$ tq $w \models f\ \mathsf{U}_{\leq k}\ g$. Soit $i \in Nat$ minimal tel que $w_i \models g$. Si $i < m$, alors $w \models f\ \mathsf{U}_{\leq m-1}\ g$. Sinon, $i \geq m$, et $w_0 ... w_i$ contient un état répété et, en retirant de w les boucles avant i, on obtient un chemin w' tel que $w' \models f\ \mathsf{U}_{\leq m-1}\ g$. Finalement, $q \models \mathsf{E}\, f\, \mathsf{U}_{\leq m-1}\, g$.
- Si $q \models \mathsf{A}\, f\, \mathsf{U}_{\leq k}\, g$, soit $w \in Chemins(q)$, soit i le plus petit indice tel que $w_i \models g$. Alors on a $\forall j < i$, $w_j \models f \wedge \neg g$. Si $i < m$, $w \models f\ \mathsf{U}_{[0..m-1]}\ g$. Sinon, il existe une boucle dans w, et il existe donc un chemin infini w' n'utilisant que les états de w avant i (en restant dans la boucle). On a alors $q \models \mathsf{EG}\, \neg g$, d'où contradiction : $i < m$ est le seul cas possible.

De plus, l'autre sens est trivial ($w \models f\ \mathsf{U}_{[0..|Q|-1]}\ g \Rightarrow w \models f\ \mathsf{U}_{[0..k]}\ g$ pour $k > |Q|$). ■

Remarque 3.17 *On n'a* pas $w \models f \mathsf{U}_{\leq k} g$ *ssi* $w \models f \mathsf{U}_{\leq |Q|-1} g$

4 Traduction de $TCTL_s$ dans CTL

Proposition 4.1 (Traduction en CTL) *Toute formule f de $TCTL_s$ est équivalente à une formule de CTL. De plus, il est possible de construire effectivement une formule équivalente à f de taille $O(2^{|f|})$.*

Remarque 4.2 *Nous verrons à la proposition 5.1, que toute traduction est au moins exponentielle.*

Preuve: On va tout d'abord définir un nouvel opérateur $\mathsf{A}\, f\, \mathsf{U}_{\leq/=b}\, (g/h)$. Sa définition ici n'est pas absolument nécessaire, mais elle permet de décomposer les différentes étapes de la traduction. Intuitivement, on peut dire que :

$$\mathsf{A}\, f\, \mathsf{U}_{\leq/=b}\, (g/h) \;\equiv \mathsf{A}(f\ \mathsf{U}_{\leq b}\ g\ \vee\ f\ \mathsf{U}_{=b}\ h)$$

Définition 4.3 ($\mathsf{A}\, f\, \mathsf{U}_{\leq/=b}\, (g/h)$) *Posons, pour $(f,g,h) \in TCTL_s$, $q \in Q$, et b entier positif : $q{\models}\mathsf{A}\, f\, \mathsf{U}_{\leq/=b}\, (g/h)$ ssi $\forall w \in Chemins(A)$ tq $w_0 = q$, $\exists i \in \mathbb{N}$ tq*

- *$w_i{\models}g$, $\forall j < i$, $w_j{\models}f$, et $time_w^i \leq b$*
- *ou $w_i{\models}h$, $\forall j < i$, $w_j{\models}f$, et $time_w^i = b$*

Posons de plus :

- *$|\mathsf{A}\, f\, \mathsf{U}_{\leq/=b}\, (g/h)\,| \stackrel{def}{=} |\,f\,| + |\,g| + |\,h\,| + \lceil log_2(b)\rceil$, avec $b < \infty$*
- *$TCTL_s^{\leq,=,\leq/=}$ l'extension de $TCTL_s^{\leq,=}$ avec l'opérateur $\mathsf{A}\, f\, \mathsf{U}_{\leq/=b}\, (g/h)$.*

Définition 4.4 *Soit I une contrainte temporelle et $c \in \mathbb{N}$. On définit $I - c$ comme étant l'unique contrainte temporelle telle que $[\![I - c]\!] = [\![I]\!] - c = \{n \geq 0 \mid n + c \in [\![I]\!]\}$.*

Remarque 4.5 *Cette contrainte est unique d'après sa sémantique.*

Etant donné une contrainte I, il est facile de calculer $I - c$ en temps linéaire avec l'algorithme suivant :

- $(\bigcup_i [a_i..b_i]) - c \stackrel{\text{def}}{=} \bigcup_i ([a_i..b_i] - c)$
- $[a..b] - c \stackrel{\text{def}}{=} \begin{cases} [max(0, a - c)..b - c], \text{ avec } \infty - c \stackrel{\text{def}}{=} \infty, \text{ si } 0 \leq b - c. \\ \emptyset \text{ sinon} \end{cases}$

Voyons comment on peut éliminer les contraintes temporelles générales sous la forme d'intervalles, en utilisant uniquement les contraintes $\leq$, $=$, et $\leq / =$:

Lemme 4.6 *On a les équivalences suivantes, pour a et $b \in \mathbb{N}$, f et $g \in TCTL_s^{\leq,=,\leq/=}$:*

1. $\mathsf{E}\, f\, \mathsf{U}_{\geq a}\, g \;\equiv \mathsf{E}\, f\, \mathsf{U}_{=a}\, (\mathsf{E}\, f\, \mathsf{U}\;\, g)$
2. $\mathsf{A}\, f\, \mathsf{U}_{\geq a}\, g \;\equiv \mathsf{A}\, f\, \mathsf{U}_{=a}\, (\mathsf{A}\, f\, \mathsf{U}\;\, g)$
3. $\mathsf{E}\, f\, \mathsf{U}_{[0..b]\cup I}\, g \;\equiv (\mathsf{E}\, f\, \mathsf{U}_{\leq b}\, g\ ) \vee (\mathsf{E}\, f\, \mathsf{U}_I\, g\ )$
4. $\mathsf{A}\, f\, \mathsf{U}_{[0..b]\cup I}\, g \;\equiv \mathsf{A}\, f\, \mathsf{U}_{\leq/=b}\, (g/(\mathsf{A}\, f\, \mathsf{U}_{I-b}\, g\ ))$
5. $\mathsf{E}\, f\, \mathsf{U}_{[a+1..b+1]\cup I}\, g \;\equiv \mathsf{E}\, f\, \mathsf{U}_{=a+1}\, (\mathsf{E}\, f\, \mathsf{U}_{[0..b-a]\cup I-a-1}\, g\ )$
6. $\mathsf{A}\, f\, \mathsf{U}_{[a+1..b+1]\cup I}\, g \;\equiv \mathsf{A}\, f\, \mathsf{U}_{=a+1}\, (\mathsf{A}\, f\, \mathsf{U}_{[0..b-a]\cup I-a-1}\, g\ )$

Preuve: Les preuves de correction de ces équivalences sont aisées d'après la définition de $\models$. ■

On peut alors montrer que :

Lemme 4.7 *$TCTL_s$ n'est pas plus expressif que $TCTL_s^{\leq,=,\leq/=}$. De plus, il existe une traduction donnant des formules dont la taille dag est linéaire en celle des formules de départ.*

Preuve: On utilise les égalités du lemme 4.6 comme des règles de réécriture en les appliquant de gauche à droite. Pour réécrire une formule de $TCTL_s$ vers $TCTL_s^{\leq,=,\leq/=}$, on appliquera ces règles par récurrence sur la structure de la formule. Ceci termine car la somme des b_i finis des termes I non triviaux (non réduits à $= a$, $\leq a$, etc...) ne peut que diminuer par application des règles 3 à 6, et on épuise les intervals $\geq a$ par application des règles 1 et 2. On peut ainsi réécrire toute formule de $TCTL_s$ dans $TCTL_s^{\leq,=,\leq/=}$ par induction sur sa structure (l'équivalence est substitutive), et récurrence sur I pour chaque $\mathsf{E}\,\_\,\mathsf{U}_I$ et $\mathsf{A}\,\_\,\mathsf{U}_I$. ■

De plus, si on note $trad(f)$ pour la traduction de f, on a :

$$|trad(\mathsf{E}\,f\,\mathsf{U}_{[a..b]\cup I}\,g\,)| \;\leq 3\times\; |\mathsf{E}\,f\,\mathsf{U}_I\,g\;| \;+\lceil log_2(a)\rceil+\lceil log_2(b)\rceil+4, \text{ donc}$$
$$|trad(\mathsf{E}\,f\,\mathsf{U}_I\,g\,)| \;= O(3^{\text{nb d'intervalles dans } I}\times(\,|trad(f)|\;+\;|trad(g)|\;+\;|I|\,)),$$
$$\text{et de même pour }\;|trad(\mathsf{E}\,f\,\mathsf{U}_{[a..b]\cup I}\,g\,)|\,.$$

On a donc :

$$|\text{formule traduite}|\; = O(3^{\text{nb d'intervalles dans la formule}}\times\;|\text{formule de départ}|\,)$$

Mais un algorithme tel qu'un model-checker utilisant cette traduction partagerait le calcul pour chaque occurence de f, ce qui correspond à partager les sous-formules identiques dans la représentation des formules. Dans ce cas, l'augmentation de la taille de la formule est compensée par la diminution de I :

$$|\text{formule traduite}|_{\,dag} = O(\,|\mathsf{E}\,f\,\mathsf{U}_I\,g\;|_{\,dag})$$

L'expansion de $TCTL_s$ vers $TCTL_s^{\leq,=,\leq/=}$ est alors linéaire.

De plus, on peut éliminer la contrainte temporelle $\leq/=$ en n'utilisant que les contraintes $\leq$ et $=$:

Lemme 4.8 *$TCTL_s^{\leq,=,\leq/=}$ n'est pas plus expressif que $TCTL_s^{\leq,=}$*

Preuve: En effet, on a les équivalences suivantes :

$$\begin{cases}\mathsf{A}\,f\,\mathsf{U}_{\leq/=b}\,(g/h) \;\equiv\; & \neg\mathsf{E}\,(f\wedge\neg g)\,\mathsf{U}_{\leq b-1}\,(\neg f\wedge\neg g) \quad (\text{pour } b>0)\\ & \wedge\neg\mathsf{E}\,(f\wedge\neg g)\,\mathsf{U}_{=b}\,(tick\wedge\neg\mathsf{A}\,f\,\mathsf{U}_{\leq/=0}\,(g/h)\,)\\ \text{et } \mathsf{A}\,f\,\mathsf{U}_{\leq/=0}\,(g/h) & \equiv\mathsf{A}\,f\,\mathsf{U}_{\leq 0}\,(g\vee h)\end{cases}$$

Pour un chemin w donné et $b>0$, il n'y a que deux cas où $f\;\mathsf{U}_{\leq/=b}(g/h)$ ne se produit pas :

1. soit on a $\neg f$ avant d'avoir g pour un temps strictement inférieur à b (sur w), auquel cas on a $w\models(f\wedge\neg g)\;\mathsf{U}_{\leq b-1}(\neg f\wedge\neg g)$

2. sinon, on a $w\models\mathsf{G}_{\leq b-1}(f \wedge \neg g)$. Il ne faut donc pas avoir $\mathsf{A}\, f\, \mathsf{U}_{\leq/=0}\, (g/h)$ au premier état distant exactement de b *ticks* : c'est $\neg\mathsf{E}\, f \wedge \neg g\, \mathsf{U}_{=b}\, (tick \wedge \neg\mathsf{A}\, f\, \mathsf{U}_{\leq/=0}\, (g/h)\ )$

L'autre sens est évident. ■

On remarque que comme pour le lemme 4.7, la duplication des sous formules conduit à une explosion exponentielle de la taille de la formule, mais seulement une augmentation linéaire pour la taille dag.

Voyons comment on peut éliminer les contraintes temporelles $=$ et $\leq$ en se ramenant uniquement à des formules de CTL. On a tout d'abord :

Lemme 4.9 *On a les équivalences suivantes :*

1. $\mathsf{E}\, f\, \mathsf{U}_{=0}\, g \ \equiv \mathsf{E}\, f\, \mathsf{U}_{\leq 0}\, g \ \equiv g \vee (f \wedge \mathsf{EX}\, \mathsf{E}\, (f \wedge \neg tick)\, \mathsf{U}\, (g \wedge \neg tick) \ \ )$
2. $\mathsf{E}\, f\, \mathsf{U}_{=a+1}\, g \ \equiv f \wedge \mathsf{EX}\, \mathsf{E}\, (f \wedge \neg tick)\, \mathsf{U}\, (tick \wedge \mathsf{E}\, f\, \mathsf{U}_{=a}\, g\ )$
3. $\mathsf{E}\, f\, \mathsf{U}_{\leq a+1}\, g \ \equiv g \vee f \wedge \mathsf{EX}\, \mathsf{E}\, (f \wedge \neg tick)\, \mathsf{U}\, (g \vee (tick \wedge \mathsf{E}\, f\, \mathsf{U}_{\leq a}\, g\ ))$
4. $\mathsf{A}\, f\, \mathsf{U}_{=0}\, g \ \equiv \mathsf{A}\, f\, \mathsf{U}_{\leq 0}\, g \ \equiv g \vee (f \wedge \mathsf{AX}\, \mathsf{A}\, (f \wedge \neg tick)\, \mathsf{U}\, (g \wedge \neg tick) \ \ )$
5. $\mathsf{A}\, f\, \mathsf{U}_{=a+1}\, g \ \equiv f \wedge \mathsf{AX}\, \mathsf{A}\, (f \wedge \neg tick)\, \mathsf{U}\, (tick \wedge \mathsf{A}\, f\, \mathsf{U}_{=a}\, g\ )$
6. $\mathsf{A}\, f\, \mathsf{U}_{\leq a+1}\, g \ \equiv g \vee f \wedge \mathsf{AX}\, \mathsf{A}\, (f \wedge \neg tick)\, \mathsf{U}\, (g \vee (tick \wedge \mathsf{A}\, f\, \mathsf{U}_{\leq a}\, g\ ))$

Preuve: De même que pour les équivalences du lemme 4.6, on peut voir aisément que celles-ci sont correctes en procédant par cas selon la forme des différents chemins possibles dans une structure de Kripke. Par exemple, un état q_0 vérifie $\mathsf{E}\, f\, \mathsf{U}_{=a+1}\, g$ si et seulement si il existe $q_0 \to q_1 \to \ldots \to q_n$ tels que $\forall i < n$, $q_i\models f$, $\forall 0 < i < n$, $q_i\models\neg tick$, et $q_n\models tick \wedge \mathsf{E}\, f\, \mathsf{U}_{=a}\, g$. D'où le 2. ■

Grâce à ces équivalences, il est possible, par récurrence sur les contraintes temporelles, de traduire $TCTL_s^{\leq,=}$ dans CTL :

Lemme 4.10 $TCTL_s^{\leq,=}$ *n'est pas plus expressif que* CTL

Preuve: En effet, on peut utiliser les équivalences précédentes comme des règles en les orientant de gauche à droite.

L'application de ces règles termine, car les bornes des contraintes temporelles ne peuvent que diminuer (et il n'y a pas création de contraintes temporelles supplémentaires). cela permet de réécrire toute formule de $TCTL_s^{\leq,=}$ dans CTL par induction sur la structure de la formule et récurrence sur chaque borne de sa contrainte temporelle. ■

Ici, outre le dédoublement des sous-formules identiques, on a également une expansion linéaire en la valeur des bornes des contraintes lors de la récurrence, alors que la taille de toute borne est $\lceil log_2(\text{borne})\rceil$: on obtient, si P est l'ensemble des bornes apparaissant dans les contraintes temporelles de la formule initiale, dans le pire des cas :

$$|\text{formule traduite}|_{\,dag} = O(2^{|\text{formule initiale}|})$$

En combinant les lemmes 4.6 à 4.10, on a bien que $TCTL_s$ n'est pas plus expressif que CTL, mais au prix d'une traduction exponentielle pour la taille dag. ■

De plus $\mathsf{EF}_{\leq n} a$ se réécrit en une formule de hauteur temporelle au moins exponentielle en $|\mathsf{EF}_{\leq n} a|_{dag}$ d'après le théorème 5.1 .

En revanche, nous verrons dans le corollaire 5.1 que $TCTL_s - \mathsf{X}$ ne se réécrit pas dans $CTL - \mathsf{X}$ ($TCTL_s$ et CTL privés des opérateurs EX et AX).

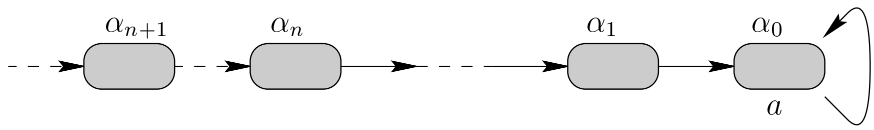
5 Expressivité

Bien que $TCTL_s$ puisse se réécrire dans CTL, cette traduction augmente la hauteur temporelle de la formule de manière au moins exponentielle. Voyons comment se réécrivent au mieux $\mathsf{U}_{=}$ et $\mathsf{U}_{\leq}$.

Proposition 5.1 *$TCTL_s$ contient des formules pour lesquelles tout équivalent dans CTL est de hauteur temporelle au moins exponentielle.*

Preuve: Pour tout $n \in \mathbb{N}$, pour toute proposition atomique *prop*, et pour toute formule $f \in CTL$ équivalente à $\mathsf{EF}_{\leq n} prop$, on a $|f| > n \sim 2^{|\mathsf{E}\, True\, \mathsf{U}_{\leq n}\, prop| - 2}$

En effet, considérons la structure de Kripke suivante :



avec $\alpha_0 \models prop \wedge tick$, et $\forall n > 0$, $\alpha_n \models \neg prop \wedge tick$

Considérons de plus l'hypothèse H suivante :

$$\forall f \in CTL,\ \forall n \geq ht(f),\ (\alpha_n \models f \Leftrightarrow \alpha_{n+1} \models f)$$

Si H est vérifiée, comme $\mathsf{EF}_{\leq n} prop$ est vraie sur α_n mais *fausse* sur α_{n+1}, toute formule de CTL équivalente à $\mathsf{EF}_{\leq n} prop$ doit être de taille strictement plus grande que n. De plus, on montre que H est effectivement vérifiée par induction sur la structure de f. Notons H_f cette propriété à f fixée :

1. $\forall prop \in PA$, H_f est vraie car tous les α_n, pour $n \geq 1$, vérifient les mêmes propositions atomiques (α_0 n'est pas considéré car $|prop| > 0$).
2. Soit $f \notin PA$, et soit $n \geq ht(f)$ (donc > 1). Considérons la structure de f :
   (a) Si $f = \mathsf{EX}\, g$ (ou $f = \mathsf{AX}\, g$, car il n'y a dans cet automate qu'un seul chemin possible partant d'un état donné), alors : $\alpha_n \models f$ ssi $\alpha_{n-1} \models g$ ssi $\alpha_n \models g$ (en appliquant H_g pour $n-1$) ssi $\alpha_{n+1} \models f$.
   (b) Si $f = \mathsf{E}\, g\, \mathsf{U} h$ (ou $f = \mathsf{A}\, g\, \mathsf{U} h$), on a :
   - Si $\alpha_n \models f$, alors $\alpha_{n+1} \models f$ car :
     - Si $\alpha_n \models h$, en appliquant H_h pour n, on a $\alpha_{n+1} \models h$, et donc $\alpha_{n+1} \models f$.

– Sinon, $\alpha_n \models g$ nécessairement, et donc en appliquant H_g pour n, on a $\alpha_{n+1} \models g$.

Ainsi $\alpha_{n+1} \models f$ (il n'y a qu'un seul chemin partant de α_{n+1}, et son second état est α_n).

– La réciproque est également vraie, pour les mêmes raisons.

(c) Sinon, $f = \neg g$ ou $f = g \wedge h$, et il suffit d'appliquer H_g et H_h pour n.

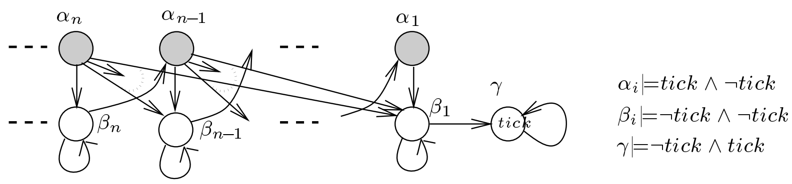
On a donc bien H_f, et H est bien vérifiée. ■

Corollaire 5.2 *$TCTL_s - \mathsf{X}$ ($TCTL_s$ privé des opérateurs EX et AX) est strictement plus expressif que $CTL - \mathsf{X}$.*

Preuve: En effet, si on regarde l'exemple de la preuve précédente, aucune formule de $CTL - \mathsf{X}$ ne peut distinguer α_n de α_{n+1} : Ils sont stuttering bisimilaires. ■

Proposition 5.3 *$TCTL^{\leq,\geq}$ dans $TCTL^{\leq}$ contient des formules pour lesquelles tout équivalent dans $TCTL^{\leq}$ est de hauteur temporelle au moins exponentielle.*

Preuve: En effet, considérons la structure suivante :



On a alors :

– $\forall n > 1$, $\alpha_n \models \mathsf{EF}_{\geq n-1} prop$, mais $\alpha_{n-1} \models \neg \mathsf{EF}_{\geq n-1} prop$

– $\forall f \in TCTL^{\leq}$, $\forall n > ht(f)$, $\alpha_n \models f$ ssi $\alpha_{n+1} \models f$, et $\beta_n \models f$ ssi $\beta_{n+1} \models f$.

Le premier point est immédiat. Pour le second, on procède par induction sur la structure de f : Les cas $\wedge$, $\neg$, et $prop$ sont immédiats. De plus :

– Si $f = \mathsf{EX}\, g$ ou $f = \mathsf{AX}\, g$, il suffit d'utiliser la récurrence et $Succ(\alpha_{n+1}) = \{\beta_{n+1}\} \cup Succ(\alpha_n)$

– Si $f = \mathsf{A}\, g\, \mathsf{U}_{\leq k}\, h$, pour $k \in \mathbb{N} \cup \{\infty\}$:
  – si $\beta_{n+1} \models f$ pour $n > ht(f)$, alors comme on a un chemin infini ne passant que par β_{n+1}, on a nécessairement $\beta_{n+1} \models h$, et donc par récurrence $\beta_n \models f$, et donc $\beta_n \models f$.
  – Si $\beta_n \models f$ pour $n > ht(f)$, alors $\beta_{n+1} \models f$ de la même manière.
  – si $\alpha_{n+1} \models f$ pour $n > ht(f)$, alors $\alpha_n \models f$ car $Succ(\alpha_n) \subseteq Succ(\alpha_{n+1})$.
  – ou si $\alpha_n \models f$ pour $n > ht(f)$, alors de tous les chemins partant de α_{n+1}, seul celui passant pas β_{n+1} ne correspond pas à un chemin partant de α_n. Mais comme $\alpha_n \models h$ ou $\beta_n \models f$, on a $\alpha_{n+1} \models h$ ou $\beta_{n+1} \models f$, d'où $\alpha_{n+1} \models f$.

– Si $f = \mathsf{E}\, g\, \mathsf{U}_{\leq k}\, h$, pour $k \in \mathbb{N} \cup \{\infty\}$:
  – si $\beta_{n+1} \models f$ pour $n > ht(f)$, alors soit $\beta_{n+1} \models h$, auquel cas $\beta_n \models f$ par récurrence, soit $\beta_n \models g$ et $\alpha_n \models \mathsf{E}\, g\, \mathsf{U}_{\leq k-1}\, h$, auquel cas :
    – si $\alpha_n \models h$, alors on applique simplement la récurrence.

  - si $\beta_n \models \mathsf{E}\, g\, \mathsf{U}_{\leq k-1}\, h$, alors $\beta_n \models f$
  - sinon, $\exists i < n$ tq $\beta_i \models \mathsf{E}\, g\, \mathsf{U}_{\leq k-1}\, h$, et $\alpha_{n-1} \models f$ (rec), d'où $\beta_n \models f$.
- si $\beta_n \models f$ pour $n > ht(f)$, alors : si $\beta_n \models h$, alors par récurrence, $\beta_{n+1} \models f$. Sinon, $\beta_{n+1} \models f$, et $\alpha_{n-1} \models \mathsf{E}\, g\, \mathsf{U}_{\leq k-1}\, h$, et :
  - si $\alpha_{n-1} \models h$, alors $\alpha_n \models h$, d'où $\beta_{n+1} \models f$.
  - sinon, $\alpha_n \models g$ (rec), et comme $Succ(\alpha_{n-1}) \subseteq Succ(\alpha_n)$, on a $\alpha_n \models \mathsf{E}\, g\, \mathsf{U}_{\leq k-1}\, h$, d'où $\beta_{n+1} \models f$.
- si $\alpha_{n+1} \models f$ pour $n > ht(f)$, alors :
  - si $\alpha_{n+1} \models h$, alors $\alpha_n \models f$ par récurrence.
  - si $\beta_{n+1} \models f$, alors $\beta_n \models f$ (voir plus haut), d'où $\alpha_n \models f$
  - si $\exists i < n$ tq $\beta_i \models f$, alors α_n y mène également, donc $\alpha_n \models f$ ($\alpha_n \models g$ par récurrence).
- ou si $\alpha_n \models f$ pour $n > ht(f)$, alors $\alpha_{n+1} \models f$ par récurrence si $\alpha_n \models h$, ou par $Succ(\alpha_n) \subseteq Succ(\alpha_{n+1})$ et $\alpha_{n+1} \models g$ (rec) sinon.

$\blacksquare$

6 Model-checking

Dans cette section on décrit de façon détaillée comment fonctionne notre algorithme de model-checking pour $TCTL_s$.

6.1 Structures de données

Nommons $A = \langle Q, R, l \rangle$ la structure de Kripke considérée. Le model-checking sur $TCTL_s$ est une extension de celui sur CTL [Eme90, SBB+99]. On utilise de plus les traductions linéaires (selon la taille dag) vues lors du passage de $TCTL_s$ à CTL pour se ramener à une logique plus simple. On donnera uniquement les algorithmes sur un fragment, le cas général s'en déduisant simplement avec les réductions.

On utilise les mêmes structures de données que pour CTL :

Pour stocker et utiliser la structure de Kripke, on utilise un ensemble d'états chainés par successeurs et prédécesseurs. Cela permet un parcours aisé du graphe, mais ne permet pas le model-checking symbolique [BCM+92].

Pour stocker et utiliser $l : Q \to 2^{PA}$, on considère une fonction totale modifiable $Q \times PA \to Bool$ (par exemple, un tableau) dont les éléments seront notés $q.a$, pour $q \in Q$, et $a \in PA$. On suppose par défaut que $\forall a \in PA$, $\forall q \in Q$, $q.a$ ssi $a \in l(q)$. On écrit simplement $q.a$ pour $a \in l(q)$, et $q.\neg a$ ou $\neg q.a$ pour $a \notin l(q)$. De plus, on a besoin d'utiliser d'autres formules que les propositions atomiques. On considère que l'on peut également étiqueter les états par des formules.

On suppose également l'existence d'une propriété suplémentaire b, qui sert de manière interne aux algorithmes.

6.2 Restriction de cette logique

Comme la complexité du model-checking n'est pas la même selon la nature des formules $TCTL_s$ à vérifier, on considère également $TCTL_s^{\leq,\geq}$. On a :

1. Sur $A = \langle Q, R, l \rangle$, on a $TCTL_s^{\leq,\geq} =_A TCTL_s^{\leq |Q|}$, car :
   - $\mathsf{E}\, f\, \mathsf{U} g \ \stackrel{A}{\equiv}\ \mathsf{E}\, f\, \mathsf{U}_{\leq |Q|+k}\, g \ \stackrel{A}{\equiv}\ \mathsf{E}\, f\, \mathsf{U}_{\leq |Q|}\, g$
   - $\mathsf{A}\, f\, \mathsf{U} g \ \stackrel{A}{\equiv}\ \mathsf{A}\, f\, \mathsf{U}_{\leq |Q|+k}\, g \ \stackrel{A}{\equiv}\ \mathsf{A}\, f\, \mathsf{U}_{\leq |Q|}\, g$
   - $\mathsf{E}\, f\, \mathsf{U}_{\geq a+1}\, g \ \equiv\ \mathsf{EG}_{\leq a}\ \mathsf{E}\, f\, \mathsf{U}_{\geq 1}\, g$
   - $\mathsf{A}\, f\, \mathsf{U}_{\geq a+1}\, g \ \equiv\ \mathsf{AG}_{\leq a}\ \mathsf{A}\, f\, \mathsf{U}_{\geq 1}\, g$
   - $\mathsf{E}\, f\, \mathsf{U}_{\geq 1}\, g$ et $\mathsf{A}\, f\, \mathsf{U}_{\geq 1}\, g$ se réécrivent aisément avec $\mathsf{E}\, f\, \mathsf{U}_{=1}\, g$ et $\mathsf{A}\, f\, \mathsf{U}_{=1}\, g$
2. $TCTL_s = TCTL_s^{\leq,=,\leq/=}$ par le lemme 4.7.

Grâce à cela, on a :

Proposition 6.1 (Complexité du model-checking) *Le model-checking de $f \in TCTL_s^{\leq,\geq}$ sur $A = \langle Q, R, l \rangle$ se fait en :*

$$O(\,|A|\ \times\ |f|\,),\ avec\ \ |A|\ \stackrel{def}{=}\ |Q|\ +\ |R|$$

Preuve: En effet, le model-checking de $f \in TCTL_s^{\leq,\geq}$ utilisera les fonctions $Verif_{\mathsf{E}\ \mathsf{U}_{\leq}}$, $Verif_{\mathsf{A}\ \mathsf{U}_{\leq}}$, $Verif_{\mathsf{E}\ \mathsf{U}_{\geq 1}}$ et $Verif_{\mathsf{A}\ \mathsf{U}_{\geq 1}}$ qui travaillent en temps $O(\ |\ A\ |\ )$ (si les sous-formules ont déja été vérifiées), ce qui donne bien $O(\,|A|\ \times\ |f|\,)$ pour la formule en entier. ■

Proposition 6.2 *Le model-checking de $f \in TCTL_s$ sur $A = \langle Q, R, l \rangle$ se fait en :*

$$O((\,|Q|^{\,3} +\ |R|\,) \times\ |f|\,)$$

Preuve: Ici, le model-checking sur $TCTL_s$ utilise les fonctions précédentes plus $Verif_{\mathsf{E}\ f\ \mathsf{U}_{=a}\ g}$, qui elle travaille en temps $O(\ |\ Q\ |\ ^3 +\ \ |\ R\ |\ )$, et utilise la traduction de $TCTL_s^{\leq,=,\leq/=}$ dans $TCTL_s^{\leq,=}$ pour traiter $\mathsf{A}\, f\, \mathsf{U}_{=a}\, g\ =\ \mathsf{A}\, f\, \mathsf{U}_{\leq/=a}\, (false/g)$. ■

6.3 Méthodes

Le but du model-checking sur f est ici d'affecter la valeur de chaque $q.f$, $q \in Q$. Pour cela, on procède par récurrence sur la structure de f, en appelant au besoin les procédures suivantes :

- $Verif_{f=\mathsf{E}\ g\ \mathsf{U}_{\leq b}\ h}$: réalise $\forall q \in Q,\ q.f := q {\models} \mathsf{E}\, g\, \mathsf{U}_{\leq b}\, h$, pour $b \leq\ |Q|$, en temps $O(\,|A|\,)$
- $Verif_{f=\mathsf{A}\ g\ \mathsf{U}_{\leq b}\ h}$: réalise $\forall q \in Q,\ q.f := q {\models} \mathsf{A}\, g\, \mathsf{U}_{\leq b}\, h$, pour $b \leq\ |Q|$, en temps $O(\,|A|\,)$
- $Verif_{f=\mathsf{E}\ g\ \mathsf{U}_{=1}\ h}$: réalise $\forall q \in Q,\ q.f := q {\models} \mathsf{E}\, g\, \mathsf{U}_{=1}\, h$, sous l'hypothèse $(q.h \wedge q.tick) \Rightarrow \forall q' \in Pred(q),\ q'.h$, en temps $O(\,|Q|\,)$

- $Verif_{f=\mathsf{A}\,g\,\mathsf{U}_{=1}\,h}$: réalise $\forall q \in Q,\ q.f := q \models \mathsf{A}\,g\,\mathsf{U}_{=1}\,h$, sous l'hypothèse $(q.h \wedge q.tick) \Rightarrow \forall q' \in Pred(q),\ q'.h$, en temps $O(\,|Q|\,)$
- $Verif_{\mathsf{E}\,g\,\mathsf{U}_{=a}\,h}$: réalise $\forall q \in Q,\ q.f := q \models \mathsf{E}\,g\,\mathsf{U}_{=a}\,h$, pour $0 < a < \infty$, en temps $O((\,|Q|\,^3 + \,|R|\,) \times \lceil log_2(a) \rceil)$

Comme on procède par récurrence, on suppose toujours que les sous-formules ont déjà toutes été marquées. Si on a une formule de $TCTL_s^{\leq,\geq}$, on se ramène à une formule de $TCTL_s^{\leq\,|Q|}$ comme vu précédement (expansion linéaire hormis les sous-formules). Si on a une formule de $TCTL_s$, on se ramène à une formule de $TCTL_s^{\leq,=}$.

Remarque 6.3 *On a également besoin de procédures* $Verif_{\mathsf{EX}\,\_}$ *et* $Verif_{\mathsf{AX}\,\_}$ *, mais elles sont évidentes et correspondent exactement à leurs homologues dans* CTL*. De même pour le traitement de* $\neg$ *et* $\wedge$ *.*

6.4 Procédures $Verif_{\_=\mathsf{E}\ \_\ \mathsf{U}_{\leq\_}\ \_}$ et $Verif_{\_=\mathsf{A}\ \_\ \mathsf{U}_{\leq\_}\ \_}$

Le principe de ces procédures est le même que pour $\mathsf{E}\,g\,\mathsf{U}h$ et $\mathsf{A}\,g\,\mathsf{U}h$ dans CTL, sauf qu'ici on doit compter le nombre d'états marqués par *tick* par lesquels on est déjà passé pour ne pas dépasser la valeur limite donnée. L'algorithme commence par étiqueter les états vérifiant $\mathsf{E}\,g\,\mathsf{U}_{\leq 0}\,h$, puis $\mathsf{E}\,g\,\mathsf{U}_{\leq 1}\,h$, etc... , en ne considérant en tout qu'une seule fois chaque état et chaque transition. On utilise pour celà, à la $i^{\text{ème}}$ itétation, l'ensemble E des états vérifiant $\mathsf{E}\,g\,\mathsf{U}_{=i}\,h$ qui n'ont pas déja étés vus. (i.e qui ne vérifient pas $\mathsf{E}\,g\,\mathsf{U}_{\leq i-1}\,h$) On remarque que tout état vérifiant $\mathsf{E}\,g\,\mathsf{U}_{\leq c}\,h$ avec $c \leq b$ vérifie aussi $\mathsf{E}\,g\,\mathsf{U}_{\leq b}\,h$. Ainsi l'algorithme évitera d'étiqueter deux fois le même état.

Remarque 6.4 *On utilise deux ensembles* E *pour différencier les états vérifiant* $\mathsf{E}\,f\,\mathsf{U}_{\leq c}\,g$ *(ensemble* E*) de ceux vérifiant* $\mathsf{E}\,f\,\mathsf{U}_{\leq c+1}\,g$ *(ensemble* E'*), pour c minimal. (De même pour* $\mathsf{A}\ \mathsf{U}$ *).*

```
proc Verif_{f=E g U_{≤b} h}
  ≡
  E := {s ∈ Q tq s.h}
  E' := vide
  for k := 0 to min(b, |Q| ) do
      while E <> vide do
            q := getone(E)
            q.f := true
            foreach q' ∈ Pred(q),
              if q'.g ∧ ¬q'.f then
                              if q.tick then add(E', q')   (point 1)
                                     else add(E, q')   (point 1)
                              fi
              fi
            end
```

```
        od
        swap(E, E')
    od
.
proc Verif_{f=A g U_{≤b} h}
    ≡
    E := {q ∈ Q tq q.h}
    E' := vide
    foreach q ∈ Q,  q.n := degr(q) end
    for k := 0 to min(b, |Q| ) do
        while E <> vide do
                q := getone(E)
                q.f := true
                foreach q' ∈ Pred(q),
                   if q'.g ∧ ¬q'.h then
                                    if q'.n > 1 then q'.n − −
                                           else if q.tick then add(E', q')   (point 1)
                                                   else add(E, q')   (point 1)
                                                fi
                                    fi
                   fi
                end
        od
        swap(E, E')
    od
.
```

6.5 Procédures $Verif_{f=\mathsf{E}\ g\ \mathsf{U}_{=1}\ h}$ et $Verif_{f=\mathsf{E}\ g\ \mathsf{U}_{=1}\ h}$

Le principe est encore de remonter dans le graphe à partir des états q tels que $q \models h$. Mais comme on sait que tout état vérifiant g et prédécesseur d'un état q tel que $q \models f \wedge \neg tick$, pour $f = \mathsf{E}\, g\, \mathsf{U}_{=k}\, h$ ou $f = \mathsf{A}\, g\, \mathsf{U}_{=k}\, h$, vérifie également f, on peut partir des états vérifiant f dont le successeur direct vérifie h et $tick$. On fait ensuite la clôture par prédécesseur pour des transitions n'aboutissant pas à un $tick$. On est donc en présence d'une adaptation des algorithmes de CTL pour $\mathsf{E}\, g\, \mathsf{U} h$ et $\mathsf{A}\, g\, \mathsf{U} h$ (l'adaptation concerne les tests de $q \models tick$). Ces deux procédures sont correctes pour les mêmes raisons, et on conserve la même complexité : $O(\,|A|\, \times\, |f|\,)$.

```
proc Verif_{f=E g U_{=1} h}
    ≡
    E1, E2 := vide
    foreach q ∈ Q,
       q.dejavu := false
       q.f := false
    end
```

```
foreach q →_R q',
   if q'.tick ∧ q'.h ∧ q.g then add(E_2, q)
                                q.f := true
   fi
   if q'.¬tick ∧ q'.h ∧ q.g ∧ ¬q.h then add(E_1, q)
                                        q.dejavu := true
   fi
end
while E_1 <> vide do
        q' := getone(E)
        foreach q ∈ Pred(q'),
           if q.g ∧ q'.tick ∧ ¬q.f then add(E_2, q)
                                       q.f := true
           fi
           if q.g ∧ q'.¬tick ∧ ¬q.dejavu then add(E_1, q)
                                              q.dejavu := true
           fi
        end
od
while E_2 <> vide do
        q' := getone(E)
        foreach q ∈ Pred(q'),
           if q.g ∧ ¬q.f ∧ q'.¬tick then q.f := true
                                         add(E_2, q)
           fi
        end
od
.
proc Verif_{f=A g U_{=1} h}
   ≡
   E_1, E_2 := vide
   foreach q ∈ Q,
      q.n := |Succ(q)| )
      if q.h then add(E_1, q) fi
   end
   while E_1 <> vide do
           q' := getone(E_1)
           if q'.tick then add(E_2, q') else
                            foreach q ∈ Pred(q'),
                               if q.g ∧ ¬q.tick ∧ ¬q.h then q.n − −
                                                            if q.n = 0 then add(E_1, q) fi
                               fi
                            end
           fi
   od
   foreach q ∈ Q,
```

$q.n := |Succ(q)|$
$q.f := false$
end
while $E_2 <> vide$ **do**
$q' := getone(E_2)$
foreach $q \in Pred(q')$,
if $q.g$ **then**
$q'.n_1 - -$
if $q'.n_1 = 0$ **then** $add(E, q')$ **fi**
fi
end
od
.

6.6 Procedure $Verif_{\_ = \mathsf{E}\ \_\ \mathsf{U}_{=\_}\ \_}$

Le principe du model-checking de $\mathsf{E}\, g\, \mathsf{U}_{=a}\, h$ est le suivant : on construit une nouvelle relation R'_b vérifiant : $(q, q') \in R'_b$ ssi $\exists q = q_1 \to q_2 \to \ \dots\ \to q_n = q'$ tq $|\{0 < i \leq n \text{ tq } q_i \models tick\}| = b$ et $\forall 0 < i \leq n-1,\ q_i \models g$ Pour avoir un algorithme de complexité $O((|Q|^3 + |R|) \times |f|)$, on doit construire R'_b en $\lceil log_2(b) \rceil$, i.e. par récurrence sur l'écriture binaire de b. Pour cela, on a :

- $R'_{2\times b} = \{(q, q') \text{ tq } \exists q'' \text{ tq } q'' \models g,\ (q, q'') \in R'_b \text{ et } (q'', q') \in R'_b\} = R'_b.R'_b$
- $R'_{b+1} = R'_b.R'_1 = R'_1.R'_b$

On a alors :

$$q \models \mathsf{E}\, g\, \mathsf{U}_{=a}\, h \ \text{ ssi } \exists q' \in Q \text{ tq } q' \models h \text{ et } (q, q') \in R_a$$

Cet algorithme utilisera donc les algorithmes classiques de fusion de deux relations (procédure $R_3 = Fusion(R_1, R_2)$, en temps $O(|Q|^3)$), et de fermeture réflexive et transitive d'une relation (procédure $Fermeture(R_1)$, en temps $O(|Q|^3)$). Notons $a_0, a_1, \dots, a_m$ l'écriture binaire de a : $a = \sum_{i \leq m} a_i \times 2^i$. Ainsi :

proc $Verif_{f = \mathsf{E}\, g\, \mathsf{U}_{=a}\, h}$
$\equiv$ Soient R', R'', et T trois nouvelles relations.
$T := \{q \to q' \in R \text{ tq } q.g \land q'.\neg tick\}$
$T := Fermeture(T)$
$R' := \{q \to q' \in R \text{ tq } q.g \land q'.tick\}$
$R' := Fusion(T, Fusion(R', T))$
for $k := m$ **to** 0 **do**
$T := Fusion(T, T)$
if $a_k = 1$ **then** $T := Fusion(T, R')$ **fi**
od
foreach $(q, q') \in T$ tq $q'.h,\ q.f := true$ (et faux ailleurs)
.

7 $TCTL_c$

On énonce ici des résultats de [LST99] relatifs à la logique $TCTL_c$. $TCTL_c$ est une extension supplémentaire naturelle de $TCTL_s$, exponentiellement plus succinte, mais au prix d'un model-checking moins efficace.

7.1 Définition

On utilise pour $TCTL_c$ les mêmes structures que pour $TCTL_s$. En revanche, la manière dont le temps est pris en compte est différente : Alors que dans $TCTL_s$, on place des contraintes temporelles au niveau de chaque opérateur (U_I, etc...), on définit dans $TCTL_c$ des horloges, suivant l'idée introduite dans [AH89], pouvant être mises à zéro à différents endroits de la formule, et dont on pourra tester le contenu à tout moment. Ainsi, si $Cl = \{x, y, ...\}$ désigne un ensemble d'horloges, on a :

Définition 7.1 ($TCTL_c$) *Les formules de $TCTL_c$ sont construites sur la grammaire :*

$$\begin{array}{ll} f := \neg f' & \mid f_1 \wedge f_2 \mid \mathsf{E}\, f_1 \,\mathsf{U}\, f_2 \mid \mathsf{A}\, f_1 \,\mathsf{U}\, f_2 \mid \mathsf{EX}\, f' \\ & \mid \mathsf{AX}\, f' \mid x \ \underline{in}\ f' \mid x \sim k \mid .. \end{array}$$

où $\sim$ est un opérateur de comparaison parmis $\{=,\ \leq,\ \geq,\ <,\ >\}$.

Intuitivement, on étend CTL en lui ajoutant deux constructions : $x\ \underline{\text{in}}\ f$ désigne la mise à zéro d'une horloge, x, qui sera utilisée dans f. Les horloges sont incrémentées le long des chemins désignés par les opérateurs habituels de CTL. On peut alors, à tout moment, tester la valeur d'une horloge par $x \sim k$, avec $k \in \mathbb{N}$. Pour une définition formelle de la sémantique, on se repportera à [LST99].

Ceci donne plus de souplesse à la logique. Ainsi, ou pourra exprimer aisément des formules de la forme : $x\ \underline{\text{in}}\ \mathsf{E}\, f\, \mathsf{U}(x < 10) \wedge \mathsf{E}\, g\, \mathsf{U}(x = 20) \wedge h$, i.e. il existe un chemin de durée 20 le long duquel les états vérifient successivement f, g, puis h, mais avec les transitions de f à g et de g à h respectivement après au plus 10 unités de temps et après exactement 20 unités de temps.

Intuitivement, de telles propriétés sont difficilement exprimables dans $TCTL_s$, car dans la formule $\mathsf{E}\, f\, \mathsf{U}_{\leq 10}\, g$, on ne peut pas référencer le moment où g se produit effectivement.

7.2 Résultats d'expressivité et de complexité

Voici quelques résultats d'expressivité de $TCTL_c$ dont les preuves se trouvent dans [LST99] :

Proposition 7.2 *$TCTL_c$ se traduit dans CTL.*

Proposition 7.3 *$TCTL_c$ est exponentiellement plus succinte que $TCTL_s$.*

Proposition 7.4 *Le model-checking sur $TCTL_c$ est* PSPACE-complet.

8 Conclusion

Nous avons proposé la logique $TCTL_s$, une extension de CTL qui permet de :

1. exprimer des propriétés quantitatives sur les comportements
2. conserver les mêmes structures de Kripke que pour les logiques temporelles
3. avoir un model-checking en temps polynômial

$TCTL_s$ est inspiré de $RTCTL$. Là où $RTCTL$ se limite à des contraintes "$\leq k$", nous avons ajouté dans $TCTL_s$ "$\geq k$", "$= k$", et plus généralement les intervalles. Par ailleurs, l'introduction d'une proposition "*tick*" généralise la modélisation du passage du temps. $TCTL_s$ est, à notre connaissance, la plus riche extension de $RTCTL$ qui garde un model-checking polynômial. La littérature existante, et en particulier [LST99], montre comment la plupart des possibilités d'extension restantes conduisent à un model-checking *PSPACE-dur*.

Nous avons étudié le pouvoir d'expression de différents fragments, ceci dans le but de vérifier que les apports respectifs des contraintes $\leq k$, $\geq k$, et $= k$ sont bien réels. Cette problématique n'est pas posée dans la littérature existante sur les logiques temps réel, et nos résultats s'appliquent à d'autres logiques (par exemple le $TCTL$ de [ACD93]). Une particularité de nos résultats d'expressivité tient à l'existence (en général) de traductions dans CTL. Ainsi, nos résultats énoncent essentiellement que certaines propriétés ne peuvent pas être exprimées *de façon succinte* en l'absence de tel ou tel type de contraintes.

Références

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

[AH89] R. Alur and T. A. Henzinger. A really temporal logic. In *Proc. 30th IEEE Symp. Foundations of Computer Science (FOCS'89), ? ?, ? ? 1989*, pages 164–169, 1989.

[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.

[AL99] L. Aceto and F. Laroussinie. Is your model checker on time ? In *Proc. 24th Int. Symp. Math. Found. Comp. Sci. (MFCS'99), Szklarska Poreba, Poland, Sep. 1999*, volume 1672 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1999.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking : 10^{20} states and beyond. *Information and Computation*, 98(2) :142–170, 1992.

[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.

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

[EMSS91] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Proc. 2nd Int. Workshop Computer-Aided Verification (CAV'90), New Brunswick, NJ, USA, June 1990*, volume 531 of *Lecture Notes in Computer Science*, pages 136–145. Springer, 1991.

[ES89] E. A. Emerson and J. Srinivasan. Branching time temporal logic. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Proc. REX School/Workshop, Noordwijkerhout, NL, May-June 1988*, volume 354 of *Lecture Notes in Computer Science*, pages 123–172. Springer, 1989.

[ET97] E. A. Emerson and R. J. Trefler. Generalized quantitative temporal reasoning : An automata-theoretic approach. In *Proc. 7th Int. Joint Conf. Theory and Practice of Software Development (TAPSOFT'97), Lille, France, Apr. 1997*, volume 1214 of *Lecture Notes in Computer Science*, pages 189–200. Springer, 1997.

[ET99] E. A. Emerson and R. J. Trefler. Parametric quantitative temporal reasoning. In *Proc. 14th IEEE Symp. Logic in Computer Science (LICS'99), Trento, Italy, July 1999*, pages 336–343, 1999.

[HHW97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech : A model-checker for hybrid systems. *Journal of Software Tools for Technology Transfer*, 1(1–2) :110–122, 1997.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2) :134–152, 1997.

[LST99] F. Laroussinie, Ph. Schnoebelen, and M. Turuani. On the expressivity and complexity of quantitative branching-time temporal logics. Submitted to 4th. Latin American Symposium on Theoretical Informatics (LATIN'2000), September 1999.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. Foundations of Computer Science (FOCS'77), Providence, RI, USA, Oct.-Nov. 1977*, pages 46–57, 1977.

[QS82] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. Int. Symp. on Programming, Turin, Italy, Apr. 1982*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[SBB+99] Ph. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie, and A. Petit. *Vérification de logiciels : Techniques et outils du model-checking.* Vuibert, 1999.

[YMW97] J. Yang, A. K. Mok, and F. Wang. Symbolic model checking for event-driven real-time systems. *ACM Transactions on Programming Languages and Systems*, 19(2) :386–412, 1997.

[Yov97] S. Yovine. Kronos : A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1–2) :123–133, 1997.

9 Annexe

9.1 tick sur les transitions

Soit $A = \langle Q, R \subseteq Q{\times}Q{\times}\{tick, \neg tick\}, l : Q \to 2^{PA}\rangle$ avec $tick \notin PA$. On vérifie sur cette structure les mêmes formules que pour $TCTL_s$ sans $tick$ dans les propositions atomiques, en adaptant la sémantique de $TCTL_s$ de manière évidente.

Posons $A' = \langle Q', R', l' : Q' \to 2^{PA'}\rangle$ transformé de A, avec :

- $PA' = PA \cup \{tick\}$
- $Q' = Q{\times}\{tick, \neg tick\}$
- R' tq : $\forall q_1, q_2, b, b' \in Q^2{\times}\{tick, \neg tick\}^2$, $\{(q_1, b), (q_2, b')\} \in R'$ ssi $(q_1, q_2, b') \in R$.
- $l'(q, tick) = l(q) \cup \{tick\}$, et $l'(q, \neg tick) = l(q)$

Intuitivement, on dédouble chaque état de la strucute de départ : pour l'un, toute transition aboutissant portait un *tick* dans la strucutre initiale, pour l'autre pas. Cela permet "d'oublier" l'information *tick* sur les transitions.

Proposition 9.1 *On a alors pour toute formule f de $TCTL_s$ sur PA (sans tick), pour tout état q de Q :*

$$q \models_A f \Leftrightarrow (q, tick) \models_{A'} f \Leftrightarrow (q, \neg tick) \models_{A'} f$$

Preuve: En effet, si on note respectivement (1), (2), et (3) ces propositions, et en notant b, b' et b'' comme éléments de $\{tick, \neg tick\}$, on a (1) $\Leftrightarrow$ (2) et (1) $\Leftrightarrow$ (3) par induction sur la structure de f :

- $f \in PA - tick$: par définition de Q' et l'.
- $f = \neg g$ et $f = g \wedge h$: évidents.
- $f = \mathsf{EX}\, g$:
  - Si $q \models \mathsf{EX}\, g$, alors $\exists (q, q', b) \in R$ tq $q' \models g$. Alors $\forall b'$, $(q', b') \models g$ et $\forall b''$, $((q, b''), (q', b)) \in R'$ par définition. Donc $\forall b''$, $(q, b'') \models \mathsf{EX}\, g$, et (1) $\Rightarrow$ (2) $\wedge$ (3) pour f.
  - De même, (2) $\vee$ (3) $\Rightarrow$ (1) pour f.
- $f = \mathsf{AX}\, g$: De même que pour $\mathsf{EX}\, g$ en regardant chaque successeur, on a (1) $\Rightarrow$ (2) $\wedge$ (3) et (2) $\vee$ (3) $\Rightarrow$ (1) pour f.
- $f = \mathsf{E}\, g\, \mathsf{U}_I\, h$:
  - Si $q \models f$, alors soit w le chemin correspondant, avec $\forall k$, $(w_k, w_k{+}1, c_k) \in R$. $(w_0 = q)$ Soit i le plus petit entier de I tel que $w_i \models h$. Alors $\forall b_i$, $(w_i, b_i) \models h$, et $\forall j < i$, $\forall b_j$, $(w_j, b_j) \models g$. De plus, $\forall j < i$, $\forall b_j$, $((w_j, b_j), (w_j + 1, c_j)) \in R'$ par définition. On peut donc bien construire dans A' un chemin correspondant à $(w_0, b_0) \models \mathsf{E}\, g\, \mathsf{U}_I\, h$ en suivant w, d'où (1) $\Rightarrow$ (2) $\wedge$ (3) pour f.
  - De même, (2) $\vee$ (3) $\Rightarrow$ (1) pour f.
- $f = \mathsf{A}\, g\, \mathsf{U}_I\, h$: Si $q \models f$, alors pour tout chemin $(w_i, c_i)_{i>0}$ de A partant d'un (q, b) (b quelconque), il existe un chemin correspondant dans A (comme précédement) partant de q, vérifiant $g\ \mathsf{U}_I\ h$. Donc celui de A' également. De même dans l'autre sens. On a donc (1) $\Rightarrow$ (2) $\wedge$ (3) et (2) $\vee$ (3) $\Rightarrow$ (1) pour f.

■

9.2 Bisimulation

On va définir l'équivalence $=_{TCTL_s\backslash X}$ entre états induite par la logique $TCTL_s\backslash X$. Nous verrons qu'elle correspond à une bisimulation.

Ainsi :

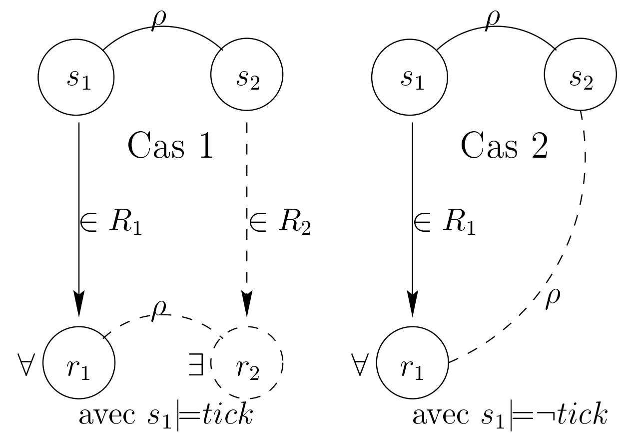
Définition 9.2 *Pour deux structures de Kripke* $A_1 = \langle S_1, R_1, l_1\rangle$ *et* $A_2 = \langle S_2, R_2, l_2\rangle$*, on note :*

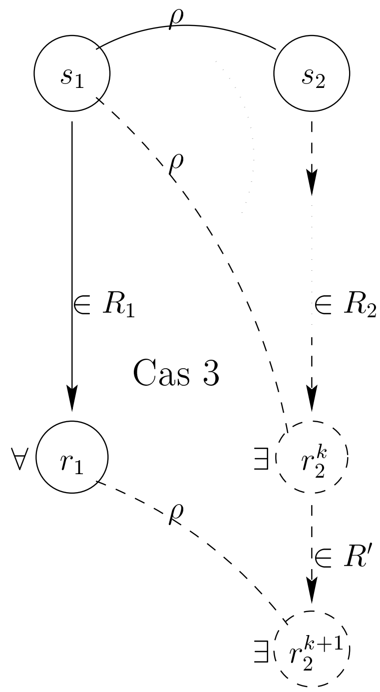
$\forall s_1, s_2 \in S_1 \times S_2,\ s_1 =_{TCTL_s\backslash X} s_2$ *ssi* $\forall f \in TCTL_s\backslash X,\ s_1 \models f \Leftrightarrow s_2 \models f$

Cette définition établit que deux états de deux structures sont dits égaux selon $TCTL_s\backslash X$ si et seulement si ils vérifient tous deux exactement le même ensemble de formules de $TCTL_s\backslash X$. On définit également la relation de bisimulation suivante, dont on verra l'équivalence avec $=_{TCTL_s\backslash X}$.

Définition 9.3 ρ *est une bisimulation entre* $A_1 = \langle S_1, R_1, l_1\rangle$ *et* $A_2 = \langle S_2, R_2, l_2\rangle$ *ssi :*

- $\rho \subseteq S_1 \times S_2$
- *Si* $s_1\ \rho\ s_2$*, alors* $l_1(s_1) = l_2(s_2)$*.*
- *Si* $s_1\ \rho\ s_2$*,* ρ *et* ρ^{-1} *vérifient la propriété de transfert suivante : (pour* ρ^{-1}*, on échangera simplement* A_1 *et* A_2*)*
  $\forall (s_1, s_2, r_1) \in S_1 \times S_2 \times S_1$ *tels que* $s_1\ \rho\ s_2$ *et* $(s_1, r_1) \in R_1$*, on a l'un des cas suivants :*
  1. $s_1 \models tick$ *et* $\exists (s_2, r_2) \in R_2$ *tq* $r_1\ \rho\ r_2$*.*
  2. $s_1 \models \neg tick$ *et* $r_1\ \rho\ s_2$*.*
  3. $s_1 \models \neg tick$ *et* $\exists r_2^1 ... r_2^k \in S_2^k$ *tq* $k \in \mathbb{N}^*$*,* $r_1\ \rho\ r_2^k$*, et* $\forall i < k$*,* $s_1\ \rho\ r_2^i$*.*





avec $s_1 \models \neg tick$

Id et $\emptyset$ sont des bisimulations, et l'union et la composée de deux bisimulations sont encore des bisimulations, de même que l'inverse d'une bisimulation. On pose $\sim \; = \; \bigcup\{\rho \;\mid\; \rho$ est une bisimulation $\}$. C'est la plus grande bisimulation, aussi appelée "la" bisimulation. C'est une relation d'équivalence par la stabilité de la notion de bisimulation par inverse, union, et composée.

Lemme 9.4 *Si $p_0 \; \rho \; q_0$, et si il existe un chemin $p_0...p_k$ tel que $p_k \models g$ et $\forall i < k$, $p_i \models f$, alors il existe un chemin $q_0...q_{k'}$ tel que $q_{k'} \models g$ et $\forall i < k'$, $q_i \models f$.*

Preuve: Il suffit de suivre le chemin $p_0...p_k$ pas à pas : Quelle que soit la propriété de transfert utilisée, elle distingue un état ou une suite d'états prolongeant $q_0..q_l$ et vérifiant les mêmes formules. ■

Proposition 9.5 *On a $q \sim q'$ ssi $q =_{TCTL_s \setminus X} q'$.*

Preuve: Séparons la preuve en deux lemmes pour plus de claireté :

Lemme 9.6 $\sim \; \subseteq \; =_{TCTL_s^{\leq,=} \setminus X} \; = \; =_{TCTL_s \setminus X}$.

Preuve:

1. $f \sim f' \Rightarrow (f =_{TCTL_s^{\leq,=} \setminus X} f' \Rightarrow f =_{TCTL_s \setminus X} f')$. On peut le montrer par induction sur la structure des formules de $TCTL_s \setminus X$:

(a) Si $f \in PA$, alors $\forall\ s, s'$, $s \sim s' \Rightarrow l(s) = l(s') \Rightarrow s =_f s'$ (avec $s =_g s' \overset{def}{\Leftrightarrow} (s \models g \Leftrightarrow s' \models g))$

(b) Si $f = g \wedge h$ ou $f = \neg g$, direct par l'induction.

(c) Si $s \models f = \mathsf{E}\, g\, \mathsf{U}_{=k}\, h$ ou $\mathsf{E}\, g\, \mathsf{U}_{\leq k}\, h$, et si $\forall\ s, s'$, $s \sim s' \Rightarrow s =_{g,h} s'$, alors $\forall\ s, s'$ tq $s \sim s'$, $s =_f s'$, et il existe $\pi_0 ... \pi_n$ succession d'états tels que $\pi_0 = s$, $\pi_n \models h$, $time^n_{\pi_0 .. \pi_n} = k$ ($\leq k$ respectivement), et $\forall\ i < n$, $(\pi_i, \pi_{i+1}) \in R$ et $\pi_i \models g$. On dira de $\pi_0 ... \pi_n$ que c'est un chemin explicitant $s \models f$. Alors d'après le Lemme 9.4, il existe un chemin explicitant $s' \models f$. (On a une série d'états vérifiant g puis h (par récurrence, car g et h sont de taille inférieure à f), avec le même nombre de ticks).

(d) De même si $f = \mathsf{A}\, g\, \mathsf{U}_{=0}\, h = \mathsf{A}\, g\, \mathsf{U}_{\leq 0}\, h$. (Comme $\sim$ est symétrique, la présence d'un chemin partant de s' ne convenant pas conduit nécessairement à l'existance d'un chemin partant de s et ne convenant pas non plus).

On a donc $\forall\ s, s' \in S_1 \times S_2$, $s \sim s' \Rightarrow s =_{TCTL_s \setminus X} s'$.

2. On peut remarquer enfin que $\sim$ étant symétrique, on peut appliquer le point précédent pour $f \sim f'$ et $f' \sim f$.

■

Lemme 9.7 $=_{TCTL_s}$ *est une bisimulation (donc* $\subseteq \sim$*).*

Preuve: $\forall\ s, s' \in S_1 \times S_2$, si $s =_{TCTL_s \setminus X} s'$, alors $l(s) = l'(s')$ car $\forall\ a \in PA$, $s \models a$ ssi $s' \models a$, et :

1. Si $s \models tick$, alors $s' \models tick$ et $\forall\ r \in S$ tq $(s, r) \in R$, on a : Si $r \models \neg tick$, alors énumérons l'ensemble des formules vérifiées par l'état r, que l'on notera $\{f_i\}_{i \in \mathbb{N}}$, avec $f_0 = \neg tick$. Posons $\forall\ i$, $\psi_i = \bigwedge_{j \leq i} f_j$. On a $\forall\ i$, $s \models \mathsf{E}\, tick\, \mathsf{U}_{=0}\, \psi_i$, d'où $s' \models \mathsf{E}\, tick\, \mathsf{U}_{=0}\, \psi_i$. Comme $f_0 = \neg tick$ et $s' \models tick$, un chemin explicitant $s' \models \mathsf{E}\, tick\, \mathsf{U}_{=0}\, \psi_i$ comporte nécessairement exactement deux états : $\exists r'_i$ tq $(s', r'_i) \in R'$ et $r'_i \models \psi_i$. Or s' n'a qu'un nombre fini de successeurs, donc l'un d'entre-eux revient infiniment souvent dans la suite $r'_0, r'_1, ..$: notons le r'. r' vérifie une infinité de ψ_i, donc tous les f_j. De plus, chaque formule de $TCTL^{\leq,=}_s$ apparaît soit comme un f_j, soit comme sa négation : on a donc $r =_{TCTL^{\leq,=}_s} r'$. Donc ici le diagramme (3) est vérifié.
2. Si $r \models tick$, alors on peut faire de même avec $\mathsf{E}\, tick\, \mathsf{U}_{=1}\, \psi_i$ et $f_0 = tick$.
3. Si $s \models \neg tick$, alors $s' \models \neg tick$ et $\forall\ r \in S$ tq $(s, r) \in R$, on a :
   – Si $r \models \neg tick$, alors : Enumérons les ensembles de formules vérifiées par s et par r, que l'on notera $\{f_i\}_{i \in \mathbb{N}}$ et $\{g_i\}_{i \in \mathbb{N}}$ respectivement. Posons $\forall\ i$, $\psi_i = \bigwedge_{j \leq i} f_j$ et $\phi_i = \bigwedge_{j \leq i} g_j$. On a alors $\forall\ i$, $s \models \mathsf{E}\, \psi_i\, \mathsf{U}_{=0}\, \phi_i$, donc $s' \models \mathsf{E}\, \psi_i\, \mathsf{U}_{=0}\, \phi_i$. Notons $\pi^i_0 ... \pi^i_{p_i}$ un chemin explicitant $s' \models \mathsf{E}\, \psi_i\, \mathsf{U}_{=0}\, \phi_i$ ne passant pas deux fois par le même état (on peut enlever les boucles car

de toute façon, on ne doit compter aucun $tick$). On a donc $p_i < |S|$. Or comme chaque état n'a qu'un nombre fini de successeurs, on n'a qu'un nombre fini de chemins de longueur plus petite que $|S|$. Il en existe donc un, noté $\sigma_0..\sigma_n$, qui explicite un nombre infini de $s' \models \mathsf{E}\, \psi_i\, \mathsf{U}_{=0}\, \phi_i$. Or $\psi_{i+1} \Rightarrow \psi_i$ et $\phi_{i+1} \Rightarrow \phi_i$, donc il les explicite tous. Donc $r =_{TCTL_s\backslash X} \sigma_n$ et $\forall\ i < n$, $s =_{TCTL_s\backslash X} \sigma_i$. On vérifie donc le diagramme (1) si $n = 0$, et le diagramme (2) sinon.
- Si $r \models tick$, alors on peut faire de même avec $\mathsf{E}\, \psi_i\, \mathsf{U}_{=1}\, tick \wedge \phi_i$.

De plus, $=_{TCTL_s\backslash X}$ est symétrique, donc on a également les propriétés de transfert dans l'autre sens.

■

On a donc l'inclusion dans les deux sens, d'où la proposition.

■

Remarque 9.8 *De manière similaire, on peut considérer une bisimulation pour $TCTL_s\backslash X$ privé de toutes les formules utilisant tick en tant que proposition atomique en prenant simplememt $l(s) - \{tick\} = l'(s') - \{tick\}$ au lieu de $l(s) = l'(s')$.*

Ceci permet d'écrire $s \sim s'$ pour les structures suivantes :