Notes sur les modèles de jeux, leurs logiques temporelles, et leurs extensions temporisées

François Laroussinie
LIAFA, Univ. Paris 7 – CNRS UMR 7089, France
francoisl@liafa.jussieu.fr

1. Introduction

Model checking. La vérification automatique des systèmes *réactifs*, *critiques* et/ou *embarqués* est aujourd'hui un problème très important. D'une part, parce que l'usage de ces systèmes s'est considérablement étendu ces dernières années et d'autre part, parce que leur complexité s'est largement accrue. Leur développement, depuis leur spécification jusqu'à leur implémentation, requiert donc des approches particulières. Dans ce contexte, les méthodes formelles de vérification constituent un axe de recherche important et en plein essor. Parmi celles ci, les techniques de vérification *automatique* – le *model checking*[12, 26] – se sont beaucoup développées depuis les premiers travaux de Clarke et Sifakis, elles ont été appliquées avec succès sur de nombreuses études de cas et elles sont de plus en plus utilisées dans le milieu industriel [13, 7].

Le model checking consiste à modéliser le système à vérifier sous la forme d'un modèle mathématique qui représente son comportement, à énoncer formellement la propriété à vérifier (par exemple que « tout message envoyé par le serveur sera reçu par le destinataire »), puis à utiliser un outil (un *model checker*) pour décider automatiquement si la propriété est vérifiée ou non par le modèle.

Le succès de cette approche repose sur l'existence de modèles adaptés pour représenter le comportement d'un système, de langages de spécification de propriétés assez riches pour énoncer des propriétés complexes (comme les logiques temporelles), et enfin sur une algorithmique efficace pour traiter des systèmes de grande taille. De fait, les succès du model checking ont été obtenus par une recherche très active dans chacun de ces trois domaines.

Systèmes temps-réel. La vérification des systèmes temps-réel, c'est-à-dire les systèmes pour lesquels il est nécessaire de prendre en considération des informations quantitatives sur l'écoulement du temps (on veut, par exemple, pouvoir spécifier que le délai entre deux actions est de 100 secondes, ou que l'alarme doit se déclencher avant un certain *time-out*, *etc.*) pose des problèmes supplémentaires.

L'introduction des automates temporisés par Alur et Dill au début des années 90 [2, 1] a constitué un important progrès pour la vérification des systèmes temps-réel. Ces automates sont munis d'horloges qui progressent de manière synchrone avec le temps et chaque transition peut contenir des contraintes de franchissement sur la valeur courante des horloges. L'intégration des aspects temps-réel se fait ainsi de manière élégante et naturelle. Ces modèles bénéficient aussi de bonnes propriétés de décidabilité, même si les algorithmes sont bien plus complexes que ceux mis en oeuvre pour les classiques structures de Kripke. Des model checkers pour les automates temporisés existent [22, 30] et ont même été appliqués avec succès sur des problèmes réels (voir le site web de l'outil UPPAAL : http://www.uppaal.com/).

Systèmes ouverts. Enfin une caractéristique commune des systèmes embarqués réside dans le fait qu'ils sont contraints d'interagir avec d'autres composants, et cette dimension doit évidemment être prise en compte dans leur développement. Il faut donc garantir des propriétés d'interopérabilité. On souhaite analyser le système à vérifier comme un système *ouvert* et assurer son bon fonctionnement *dans son interaction* avec son environnement. Leur analyse pose des problèmes particuliers : il est déjà plus difficile d'énoncer leurs propriétés de correction (cela requiert des formalismes spécifiques : il faut vérifier des propriétés *quel que soit* le comportement de l'environnement) et les algorithmes de vérification sont souvent plus complexes. C'est sur cette spécificité que nous allons nous concentrer dans ce document.

Des jeux . . . pour quoi faire ? L'utilisation des jeux est classique en informatique. Elle permet par exemple d'illustrer certaines caractéristiques de formalismes logiques. Le model checking du μ-calcul (*i.e.* vérifier la satisfaction d'une formule de μ-calcul par une structure de Kripke) peut

s'exprimer sous la forme d'un jeu à deux joueurs [28]. Un autre exemple classique (et plus direct) concerne la bisimulation que l'on peut aussi définir comme un simple jeu à deux joueurs : le premier essaie de montrer que les deux systèmes à comparer sont différents (et choisit pour cela certaines transitions censées les distinguer), tandis que le second joueur essaie lui de prouver que les deux systèmes sont bien équivalents (et réplique au premier en choisissant d'autres transitions censées être équivalentes). Cette définition offre un autre point de vue, souvent plus intuitif (il suffit de l'avoir présenté en cours pour s'en rendre compte !). Cet usage des jeux est très classique mais ce n'est pas le point de vue que nous utiliserons ici.

Dans ces notes, nous allons présenter plusieurs modèles de jeux dont l'objectif est de modéliser l' *interaction* entre différents processus. Chaque joueur (représentant un certain processus) contribuera à l'évolution globale du système en choisissant ses propres actions (coups). Pour cela nous utiliserons les CGS, *Concurrent Game Structures* [4], comme modèle de référence pour nos différents jeux. On pourra alors observer ou vérifier que tel joueur ou telle coalition de joueurs peut garantir telle ou telle propriété Φ : on dira alors que ce joueur (ou cette coalition) a une stratégie gagnante pour Φ.

Après avoir modélisé un système global $\mathcal{S}$ sous la forme d'un jeu, nous aurons besoin d'énoncer certaines propriétés. Par exemple, on peut modéliser un protocole de communication sous la forme d'un jeu où interagissent :

- un émetteur (E) qui émet des messages et reçoit des accusés de réception (AR) ;
- un récepteur (R) qui reçoit des messages et envoie des AR ;
- un réseau qui transmet les messages avec plus ou moins de pertes.

Dans un tel système, on peut chercher si il est possible (et si oui, comment ?) pour l'émetteur de transmettre une suite de messages au récepteur. Il faudra probablement faire des hypothèses d'équité pour éviter que tous les messages ne soient perdus. On pourrait aussi s'intéresser au temps de transmission en fonction du degré de fiabilité du réseau, *etc.*

Si la réponse est positive, on voit bien qu'une *stratégie gagnante* pour la « coalition » composée de E et R (ils coopèrent) pour assurer la propriété ci-dessus correspond à un algorithme de protocole de communication.

La question des systèmes ouverts est ainsi liée aux problèmes de contrôle [27] où l'on s'intéresse, étant donnés un système S, son environnement E et un objectif Φ, à la synthèse d'un contrôleur C tel que $(E|(S|C)) \models \Phi$: on cherche à contrôler S avec C (via la synchronisation) de manière à ce que, plongé dans son environnement E, le système complet vérifie Φ. Clairement la synthèse de stratégies utilisée dans la théorie des jeux correspond bien à la même notion.

On peut aussi raffiner le problème de contrôle dans deux directions : d'une part, on peut s'intéresser juste à l'existence de stratégies (sans les construire), on parle alors de *contrôlabilité* ; et d'autre part, on peut fixer des conditions sur la forme que doit avoir le contrôleur, par exemple on peut souhaiter que C soit véritablement implémentable (par ex. qu'il ne suppose pas une mémoire infinie ou un temps de réaction infiniment petit...).

Plan. Dans un premier temps, nous allons considérer le cas non-temporisé. Nous verrons un modèle de jeu et une logique temporelle permettant d'exprimer des propriétés adéquates sur ces modèles. Cela nous permettra d'illustrer certaines difficultés dans l'analyse de ces modèles. Dans un second temps, nous présenterons plusieurs extensions temps-réel de ces différents formalismes. Nous nous limiterons à une présentation informelle et nous renvoyons les lecteurs intéressés à des articles de la littérature.

2. Modèles de jeux

Nous allons présenter un modèle de jeux concurrents, les CGS (*Concurrent Game Structures*). Nous verrons ensuite plusieurs variantes de ces jeux et quelques exemples.

Dans la suite, on suppose fixé un ensemble de propositions atomiques AP que l'on utilisera d'une part, pour étiqueter les états de contrôle des différents modèles que nous considérerons et d'autre part, dans les formules de logiques temporelles.

2.1. Les CGS

La définition formelle des CGS est la suivante :

Définition: [CGS [4]] Une *structure de jeu concurrent* (CGS) est un 7-uplet $\mathcal{S} = \langle Q, q_0, \ell, \mathsf{Agt}, \mathbb{M}, \mathsf{Mv}, \mathsf{Edg} \rangle$ où :

- Q est ensemble fini d'états de contrôle et $q_0 \in Q$ désigne l'état initial du jeu ;
- $\ell : Q \to \mathcal{P}(\mathsf{AP})$ est une fonction d'étiquetage qui associe à tout état de contrôle le sous-ensemble de propositions atomiques qu'il vérifie ;
- $\mathsf{Agt} = \{A_1, \ldots, A_k\}$ est un ensemble fini de k joueurs (appelés aussi les *agents*) ;
- $\mathbb{M}$ est un alphabet (fini) de coups pour les joueurs ;
- $\mathsf{Mv} : Q \times \mathsf{Agt} \to \mathcal{P}(\mathbb{M})$ associe à chaque état q et chaque joueur A, les coups possibles pour A depuis q ;
- $\mathsf{Edg} : Q \times \mathbb{M}^k \to Q$ est la *table de transitions* de $\mathcal{S}$ qui décrit l'évolution du système : $\mathsf{Edg}(q, m_1, \ldots, m_k)$ est le nouvel état du système lorsque A_i joue m_i pour $i = 1, \ldots, k$ depuis l'état q.

Étant donné l'état courant du jeu q, chaque joueur A_i choisit, de manière indépendante, un coup m_i et la table de

transitions fournit alors le nouvel état du système , et le tour suivant commence alors... On note $\mathsf{Next}(q, A_i, m_i)$ l'ensemble des états de Q qu'il est possible d'atteindre lorsque le joueur A_i joue le coup m_i ; et $\mathsf{Next}(q)$ désigne l'ensemble des états r atteignables depuis q (*i.e.* pour lesquels il existe des coups $m_1, \ldots, m_k$ tels que $r = \mathsf{Edg}(q, m_1, \ldots, m_k)$).

Pour illustrer la sémantique des CGS, nous pouvons considérer le fameux jeu « chifoumi » : chaque joueur choisit la pierre (pi), les ciseaux (c), le puits (pu) ou la feuille (f) ; ensuite la pierre tombe dans le puits, mais casse les ciseaux qui coupent la feuille, *etc.* La CGS à deux joueurs, avec $\mathbb{M} = \{f, pi, c, pu\}$, de la figure 1 représente ce problème.

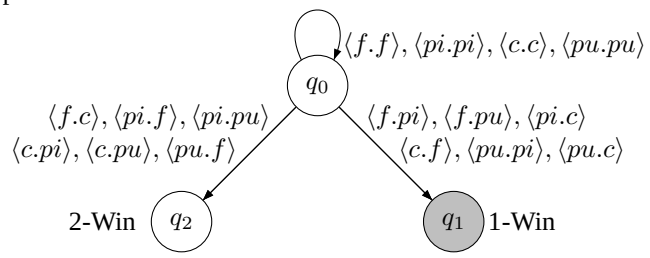


FIG. **1. Le jeu "chifoumi"**

Une paire $\langle c_1, c_2 \rangle$ représente le choix de c_1 par A_1 et c_2 par A_2. On peut aussi représenter la table de transition de la manière suivante :

		A_2 :			
	q_0	feuille	pierre	ciseaux	puits
A_1 :	feuille	q_0	q_1	q_2	q_1
	pierre	q_2	q_0	q_1	q_2
	ciseaux	q_1	q_2	q_0	q_2
	puits	q_2	q_1	q_1	q_0

Cet exemple illustre le caractère concurrent du modèle des CGS. Notons aussi que la table de transitions peut vite devenir assez grande : en effet, celle-ci est potentiellement exponentielle dans le nombre d'agents. Ceci a motivé l'introduction de CGS *symboliques* permettant une représentation plus concise de la table à l'aide de fonctions booléennes [18].

Ce modèle de jeux est assez général. Il en existe d'autres, par exemple les ATS (pour *Alternating Transition System*) [3] : dans ce modèle, à chaque tour, chaque joueur choisit un sous-ensemble d'états (ceux qu'il souhaite atteindre) parmi une liste fixée de sous-ensembles. Dès que chaque joueur a sélectionné un tel coup, la nouvelle configuration du jeu est obtenue en faisant l'intersection de tous ces sous-ensembles (par construction on suppose que celle-ci est toujours un singleton). Ce type de modèle peut se coder assez simplement avec les CGS (l'inverse est un peu plus difficile [21]) et toutes les notions que nous verrons ci-dessous s'adaptent aisément à ce cadre.

Notons aussi que l'équivalence entre modèles de jeux requiert l'utilisation de critères particuliers. En effet, si l'on compare deux systèmes de transitions, il est assez naturel d'utiliser la bisimulation comme critère d'équivalence, mais lorsqu'on compare des jeux, on doit tenir compte de la notion de coups plutôt que des simples transitions (« y-a-t-il un coup pour le joueur A dans le modèle 1 qui soit équivalent à un coup pour A dans le modèle 2 ? »), et pour cela on peut utiliser la *bisimulation alternante* [5].

Il est aussi possible de considérer des *jeux à tours* (*turn-based games*). Cette famille de jeux est très utilisée, elle bénéficie de bonnes propriétés algorithmiques. Dans ce cadre, pour chaque état q, il y a au plus un joueur qui dispose de plusieurs coups, les autres n'ont pas le choix. Cela revient à affecter chaque état à un joueur particulier (celui qui « a la main » dans cet état). Formellement, dans les CGS à tours, on impose donc que pour tout q, il existe $i \in \{1, \ldots, k\}$ tel que $|\mathsf{Mv}(q, A_i)| \geq 1$ et $|\mathsf{Mv}(q, A_j)| = 1$ pour tout $j \neq i$.

2.2. Coalitions, exécutions, stratégies,...

Une coalition A est un sous-ensemble d'agents. On étend naturellement la notion de coups à une coalition (*i.e.* un coup pour chaque joueur de la coalition), de même on utilisera $\mathsf{Next}(q, A, m)$ où $A \subseteq \mathsf{Agt}$ et m est un coup pour A.
Lorsque m représente un coup pour A et m' un coup pour $\mathsf{Agt} \backslash A$, on note $m \cdot m'$ le coup complet et alors $\mathsf{Next}(q, \mathsf{Agt}, m \cdot m') = \{\mathsf{Edg}(q, m \cdot m')\}$.

Une exécution est une séquence infinie $\rho = q_0 \to q_1 \to q_2 \ldots$ telle que $q_{i+1} \in \mathsf{Next}(q_i)$ pour tout i.

Une *stratégie* pour un joueur A_i est une fonction f_{A_i} qui associe un coup pour A_i à tout préfixe fini d'une exécution. Un tel préfixe fini représente l'état courant (le dernier état du préfixe) du jeu ainsi que son histoire (composée de tous les états visités depuis le début du jeu). On a bien sûr $f_{A_i}(q_0 \to \cdots \to q_n) \in \mathsf{Mv}(q_n, A_i)$. On note $\mathsf{Strat}(A_i)$ l'ensemble des stratégies pour A_i. Là encore, on étend aisément cette notion aux coalitions.
On dit qu'une stratégie f_A est une stratégie *positionnelle* (ou *sans mémoire*), lorsqu'elle ne dépend que de l'état courant du jeu. Dans ce cas, f_A est une fonction de Q dans $\mathbb{M}$. On peut généraliser cette restriction sur les ressources dont disposent les stratégies, en considérant des stratégies à mémoire bornée par un entier [29, 24] : une mémoire de taille k se code, par exemple, à l'aide d'un automate fini à k états que l'on utilise comme « mémoire auxiliaire » pour décider du mouvement à faire (celui-ci dépend alors de l'état courant du jeu et de l'état courant de la mémoire).

Limiter les ressources des stratégies est évidemment une question importante dès lors que l'on se pose la question

de l'implémentation d'une stratégie sous la forme d'un contrôleur réel.

Enfin, étant donnée une stratégie F_A pour une coalition $A \subseteq \mathsf{Agt}$, on note $\mathsf{Out}(q, F_A)$ l'ensemble des exécutions depuis q engendrées (ou permises) par F_A (NB : « Out » pour *outcomes*). Bien sûr, *in fine*, le choix de l'exécution dépendra des coups choisis par $\mathsf{Agt} \setminus A$ mais F_A restreint le champ des exécutions possibles.

Dans l'introduction, nous avons mentionné le cas d'un protocole de communication que l'on peut formaliser sous la forme d'un jeu. On peut aussi suggérer le problème classique du passage à niveau : des trains et des voitures peuvent arriver au passage, et il s'agit de trouver un algorithme pour contrôler la barrière de manière à garantir qu'elle soit baissée lorsque les trains passent mais suffisamment ouverte pour que les voitures n'attendent pas indéfiniment pour passer. Un tel problème peut se voir comme un jeu opposant une cheminote Alice, une conductrice Alix et un garde barrière Bob… Y a-t-il une stratégie pour Bob de manière à éviter un accident sans bloquer Alix ? Notons sur cet exemple à trois joueurs, que malgré les apparences, Bob et Alix ne doivent pas coopérer sinon une solution triviale serait qu'Alix renonce à tout trajet en voiture et que Bob ferme tranquillement la barrière… Bob doit réussir quels que soient les comportements de Alice et Alix que l'on pourrait, en fait, modéliser par un unique agent.

3. Logiques temporelles pour la spécification de propriétés sur les jeux

La logique temporelle est un bon formalisme pour énoncer des propriétés sur le comportement des systèmes réactifs [25] : elle permet de spécifier l'ordonnancement temporelle des évènements (*i.e.* exprimer que tel évènement est avant tel autre mais après un troisième, *etc.*). Par exemple, on pourra écrire « il est toujours vrai que tout problème est suivi (plus tard) par le déclenchement d'une alarme » avec la formule $\mathbf{G}\,(\texttt{pb} \Rightarrow \mathbf{F}\,\texttt{alarme})$: l'opérateur temporelle **F** permet de dire « plus tard » et **G** – son dual – permet de dire « toujours ».

Il existe de nombreuses logiques temporelles qui se différencient notamment par les opérateurs temporels autorisés (**U** : « jusqu'à » , **F** : « un jour », **G** : « toujours », **X** : « au prochain état », **W** : « jusqu'à ou toujours », **S** : « depuis », …) et par le modèle de temps considéré [15]. On distingue les logiques temporelles de temps linéaires (la plus connue étant LTL) et celles de temps arborescent (comme CTL ou CTL*). Pour les premières, le système S à spécifier est vu comme un ensemble d'exécutions (et chaque état le long d'une telle exécution a donc un unique successeur) : une spécification pour S est alors un ensemble de formules de LTL (contenant des opérateurs temporels **X** et **U**) qui doivent être vérifiées pour toutes les exécutions de S. Dans le cadre du temps arborescent, les propriétés s'interprètent sur les états du système de transitions modélisant S : chaque état peut avoir plusieurs successeurs. Les logiques de temps arborescent contiennent donc en plus des opérateurs temporels **U** et **X**, des quantificateurs sur les chemins pour préciser si la propriété temporelle doit être vérifiée sur *un* chemin ou sur *tous* les chemins… On écrira donc la propriété précédente $\mathbf{AG}\,(\ \texttt{pb} \Rightarrow \mathbf{AF}\,\texttt{alarme})$: pour toutes les exécutions (**A**), il est toujours vrai (**G**) que tout problème est suivi le long de tous les chemins (**A**) par le déclenchement de l'alarme un jour (**F**). Mais on pourra aussi écrire une propriété comme $\mathbf{AG}\,(\texttt{pb} \Rightarrow \mathbf{EF}\,\texttt{alarme})$ qui signifie que de tout état contenant un problème, il existe *un* chemin le long duquel l'alarme se déclenchera, ce qui est très différent ! L'introduction des quantificateurs de chemins permet donc l'expression de propriétés assez complexes.

Lorsqu'on s'intéresse aux jeux, il est naturel de raisonner sur les stratégies dont disposent les agents pour garantir telle ou telle propriété. Par exemple, on pourrait être intéressé par savoir si il existe une stratégie pour la coalition A pour atteindre un état gagnant win. Ce type de quantification sur *les chemins engendrés par une stratégie pour* A ne peut pas s'exprimer avec les seules quantifications existentielles et universelles sur les chemins : la logique ATL [4] a été proposée pour permettre ce type de propriétés à l'aide de quantificateurs de la forme $\langle\!\langle A \rangle\!\rangle$ qui signifie « il existe une stratégie pour la coalition A telle que… ». Formellement la syntaxe de ATL est la suivante :

Définition: [Syntaxe de ATL]

$$\begin{array}{rcl} \mathsf{ATL} \ni \phi_s, \psi_s & ::= & P \mid \neg\phi_s \mid \phi_s \vee \psi_s \mid \langle\!\langle A \rangle\!\rangle\, \phi_p \\ \phi_p & ::= & \mathbf{X}\,\phi_s \mid \phi_s\,\mathbf{U}\,\psi_s \mid \phi_s\,\mathbf{R}\,\psi_s \end{array}$$

avec $P \in \mathsf{AP}$ et $A \subseteq \mathsf{Agt}$.

On interprète les formules de ATL sur des états d'une CGS $\mathcal{S}$:

$$\begin{array}{l} q \models_{\mathcal{S}} \langle\!\langle A \rangle\!\rangle\, \phi_p \text{ ssi } \exists F_A \in \mathsf{Strat}(A).\ \forall \rho \in \mathsf{Out}(q, F_A).\ \rho \models_{\mathcal{S}} \phi_p, \\ \rho \models_{\mathcal{S}} \mathbf{X}\,\phi_s \text{ ssi } \rho[1] \models_{\mathcal{S}} \phi_s, \\ \rho \models_{\mathcal{S}} \phi_s\,\mathbf{U}\,\psi_s \text{ ssi } \exists i.\ \rho[i] \models_{\mathcal{S}} \psi_s \text{ et } \forall 0 \leq j < i.\ \rho[j] \models_{\mathcal{S}} \phi_s \\ \rho \models_{\mathcal{S}} \phi_s\,\mathbf{R}\,\psi_s \text{ ssi } \forall i : \Big(\exists j < i.\ \rho(j) \models_{\mathcal{S}} \phi_s\Big) \ \vee\ \rho(i) \models_{\mathcal{S}} \psi_s \end{array}$$

Notons que la quantification existentielle sur les chemins **E** de CTL ou CTL* s'exprime avec $\langle\!\langle \mathsf{Agt} \rangle\!\rangle$: si tous les agents coopèrent, alors… Et la quantification universelle s'exprime avec $\langle\!\langle \emptyset \rangle\!\rangle$: si personne ne coopère et que l'on a une certaine propriété, c'est que celle-ci est vraie pour

toutes les exécutions… CTL est donc incluse dans ATL (et une structure de Kripke peut être vue comme une CGS à un seul joueur).

Voici quelques exemples de formules :

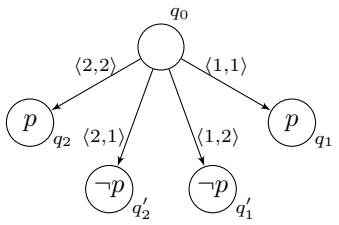
- $\langle\!\langle$Contrôleur$\rangle\!\rangle\, \mathbf{G}\,(\neg \mathsf{Pb})$: existe-t-il une stratégie pour le contrôleur de manière à ce que la proposition Pb ne soit jamais vraie ?
- $\langle\!\langle A\rangle\!\rangle\, \mathbf{F}\left(\neg\langle\!\langle B\rangle\!\rangle\, \mathbf{F}\,\mathsf{P} \;\wedge\; \neg\langle\!\langle C\rangle\!\rangle\, \mathbf{F}\,\mathsf{P}\right)$: existe-t-il une stratégie pour A de manière à atteindre un état où ni B, ni C n'ont de stratégie pour atteindre P ?
- $\langle\!\langle A\rangle\!\rangle\, \mathbf{F}\left(\neg\langle\!\langle B\rangle\!\rangle\, \mathbf{F}\,\mathsf{P} \wedge \neg\langle\!\langle C\rangle\!\rangle\, \mathbf{F}\,\mathsf{P} \wedge \langle\!\langle B,C\rangle\!\rangle\, \mathbf{F}\,\mathsf{P}\right)$: existe-t-il une stratégie pour A de manière à atteindre un état où ni B, ni C n'ont de stratégie pour atteindre P, mais où B et C peuvent coopérer ensemble pour atteindre P ?
- $\langle\!\langle$Emetteur,Recepteur$\rangle\!\rangle\, \mathbf{F}\,(\mathsf{msg\text{-}ok})$: existe-t-il une stratégie pour l'émetteur et le récepteur pour atteindre un état où la proposition msg-ok est vérifiée ?

Nous renvoyons à [4] pour une présentation détaillée de la logique ATL, ici nous voulons juste insister sur quelques aspects particuliers de cette logique temporelle.

Les quantificateurs $\langle\!\langle A\rangle\!\rangle$ ne sont pas toujours faciles à manier. Par exemple, si Φ_1 et Φ_2 sont des propriétés de chemins (disons de la forme [1] $\phi\,\mathbf{U}\,\psi$), on sait bien que $\mathbf{E}(\Phi_1 \vee \Phi_2)$ est équivalente à $\mathbf{E}\Phi_1 \vee \mathbf{E}\Phi_2$ (et donc par dualité $\mathbf{A}(\Phi_1 \wedge \Phi_2)$ est équivalente à $\mathbf{A}\Phi_1 \wedge \mathbf{A}\Phi_2$). Mais ce n'est pas le cas pour $\langle\!\langle A\rangle\!\rangle$, on a :

$$\langle\!\langle A\rangle\!\rangle\,(\Phi_1 \vee \Phi_1) \quad \not\Leftarrow \quad \langle\!\langle A\rangle\!\rangle\,\Phi_1 \vee \langle\!\langle A\rangle\!\rangle\,\Phi_1$$

Les CGS ne sont pas des jeux déterminés [4, 16] : cela s'illustre pour les formules de ATL de la manière suivante : si une coalition A n'a pas de stratégie pour garantir Φ, cela ne signifie pas que la coalition $\mathsf{Agt}\backslash A$ a une stratégie pour garantir $\neg\Phi$. Ainsi la formule $\neg\langle\!\langle A\rangle\!\rangle\,\varphi$ n'est pas équivalente à $\langle\!\langle \mathsf{Agt}\backslash A\rangle\!\rangle\,\neg\varphi$. Il suffit de considérer la CGS ci-dessous dans laquelle ni le joueur 1, ni le joueur 2 n'ont de stratégie pour assurer $\mathbf{X}\,P$ ou $\mathbf{X}\,\neg P$… et donc $q_0 \not\models \langle\!\langle A_1\rangle\!\rangle\,\mathbf{X}\,P$ et $q_0 \not\models \langle\!\langle A_2\rangle\!\rangle\,\mathbf{X}\,\neg P$.



Par contre, si l'on considère des CGS à tours, les choses se simplifient et on a bien $\neg\langle\!\langle A\rangle\!\rangle\,\varphi \equiv \langle\!\langle \mathsf{Agt}\backslash A\rangle\!\rangle\,\neg\varphi$

[1] même si ce type de formule ne fait pas partie de CTL ou ATL mais plutôt de CTL* ou ATL*.

D'autres logiques. On trouve dans la littérature, d'autres formalismes pour énoncer des propriétés sur les jeux. Il y a des extensions naturelles de ATL comme ATL* ou ATL+. On peut aussi étendre ATL avec des *contextes stratégiques* ce qui augmente beaucoup son expressivité et permet d'énoncer des propriétés complexes sur les stratégies des différents agents [8]. Il y a aussi d'autres possibilités, comme SL (pour *Strategy Logic*) [11] ou GL (pour *Game Logic*) [4]. Ici nous allons juste présenter rapidement cette dernière logique.

GL est une logique qui permet de raisonner explicitement sur les arbres d'exécutions engendrés par une stratégie d'un joueur (ou d'une coalition). On dispose d'un quantificateur existentiel de stratégies pour A ($\exists A$), de quantificateurs existentiel et universel de chemins ($\exists$ et $\forall$) dans un arbre d'exécution (*i.e.* engendré par une stratégie pour une coalition $A \subseteq \mathsf{Agt}$) et des modalités temporelles habituelles. La syntaxe est définie comme suit (GL_t contient les formules d'*arbres*, et GL_p les formules de chemins) :

Définition: [Syntaxe de GL [4]]

$$\begin{array}{rcl}
\mathsf{GL} \ni \phi_s, \psi_s & ::= & P \mid \neg\phi_s \mid \phi_s \vee \psi_s \mid \exists A.\phi_t \\
\mathsf{GL}_t \ni \phi_t, \psi_t & ::= & \phi_s \mid \neg\phi_t \mid \phi_t \vee \psi_t \mid \exists \phi_p \\
\mathsf{GL}_p \ni \phi_p, \psi_p & ::= & \phi_t \mid \neg\phi_p \mid \phi_p \vee \psi_p \mid \mathbf{X}\,\phi_p \mid \phi_p\,\mathbf{U}\,\psi_p
\end{array}$$

avec $A \subseteq \mathsf{Agt}$ et $P \in \mathsf{AP}$.

La sémantique est assez naturelle : un état q satisfait $\exists A.\phi_t$ ssi il existe une stratégie $F_A \in \mathsf{Strat}(q, A)$ telle que l'arbre T_{F_A} engendré par F_A depuis q vérifie ϕ_t.

La formule $\exists A.(\exists\mathbf{G}\,P_1 \wedge \exists\mathbf{G}\,P_2)$ énonce l'existence d'une stratégie pour A telle qu'il existe (dans le cadre de cette stratégie) un chemin où P_1 est toujours vraie et un chemin où P_2 est toujours vraie. Cette propriété est inexprimable avec ATL* [4] mais elle l'est dès que l'on dispose de contextes stratégiques [8].

4. Model checking

Ici le problème du modèle checking consiste, étant données une CGS $\mathcal{S}$ et une formule $\Phi \in \mathsf{ATL}$, à décider si $\mathcal{S}$ vérifie Φ ou non. L'algorithme procède comme suit : pour toute sous-formule ψ de Φ, on calcule les états vérifiant ψ (on note $[\![\psi]\!]$ ce sous-ensemble). On commence par les propositions atomiques (il suffit alors de considérer la fonction d'étiquetage ℓ), on traite facilement les opérateurs booléens (une fois que l'on a traité les sous-formules…) et il reste à considérer le cas des modalités $\langle\!\langle A\rangle\!\rangle\,\mathbf{X}\,\_$, $\langle\!\langle A\rangle\!\rangle\,\_\,\mathbf{U}\,\_$ et $\langle\!\langle A\rangle\!\rangle\,\_\,\mathbf{R}\,\_$.

Lorsqu'on connaît les états vérifiant φ, on peut déterminer ceux vérifiant $\langle\!\langle A\rangle\!\rangle\,\mathbf{X}\,\varphi$, ces états sont les *prédécesseurs contrôlables* pour A de l'ensemble $[\![\varphi]\!]$, on

le note $\mathsf{CPre}(A, [\![\varphi]\!])$ où CPre est défini de la manière suivante : $\mathsf{CPre}(A, S) \stackrel{\text{def}}{=} \{q \in Q \mid \exists m_A \in \mathsf{Mv}(q, A) \text{ t.q. } \mathsf{Next}(q, A, m_A) \subseteq S\}$

$\mathsf{CPre}(A, S)$ contient les états pour lesquels A dispose d'une stratégie qui permet d'accéder en une transition à un état de S.

Le traitement des modalités $\langle\!\langle A \rangle\!\rangle \_ \mathbf{U} \_$ et $\langle\!\langle A \rangle\!\rangle \_ \mathbf{R} \_$ est le plus caractéristique. Ici nous ne présenterons que le premier cas. Étant donnée une formule $\Psi \stackrel{\text{def}}{=} \langle\!\langle A \rangle\!\rangle \varphi_1 \mathbf{U} \varphi_2$, on va trouver tous les états vérifiant Ψ en calculant le plus petit point fixe de la fonction $f : 2^Q \to 2^Q$ suivante :

$$f(Z) \stackrel{\text{def}}{=} [\![P_2]\!] \cup \Big([\![P_1]\!] \cap \mathsf{CPre}(A, Z) \Big) \qquad (1)$$

On peut exprimer cette propriété en utilisant le μ-calcul alternant avec $\mu Z. \Big(P_2 \vee (P_1 \wedge \langle\!\langle A \rangle\!\rangle \mathbf{X}\, Z) \Big)$.

On retrouve ici le même schéma de point fixe que pour l'opérateur $\mathbf{E} \_ \mathbf{U} \_$ de CTL, avec une différence : l'utilisation de $\langle\!\langle A \rangle\!\rangle \mathbf{X}$ au lieu $\mathbf{EX}$. On s'intéresse bien aux prédécesseurs contrôlables (par la coalition A) et non aux seuls prédécesseurs.

Pour illustrer le calcul du point fixe, on considère l'exemple de la formule $\langle\!\langle A_1 \rangle\!\rangle \mathbf{F}\, P$ interprétée sur la CGS de la figure 2 (où les boucles sans étiquette correspondent à l'unique transition possible pour ces états). On suppose que q_3 et q_4 sont les seuls états qui vérifient P.

Le calcul du point fixe détecte tous les états vérifiant Φ (coloriés sur la figure). On commence par trouver q_3 et q_4, puis q_2 (grâce au coup 1, A_1 peut assurer d'arriver en q_3 ou q_4), puis q_1 (grâce au coup 1), puis q_0 (par le coup 2).

Notons que la complexité du calcul de $\mathsf{CPre}(A, S)$ est déterminante pour la complexité du model checking de ATL : cette complexité varie selon que l'on considère des CGS « basiques », des CGS symboliques ou des ATS. Cela donne les complexités suivantes :

Théorème: Le model checking de ATL...

- sur les CGS est un problème P-complet [4] ;
- sur les CGS symboliques est Δ_3^p-complet [21] ;
- sur les ATS est Δ_2^p-complet [21].

On peut aussi mentionner que pour les propriétés énoncées avec ATL, il suffit de considérer des stratégies positionnelles (*i.e.* sans mémoire). Mais ce n'est pas le cas de ATL^* : par exemple, la formule $\langle\!\langle A \rangle\!\rangle (\mathbf{F}\, P_1 \wedge \mathbf{F}\, P_2)$ énonce l'existence d'une stratégie pour atteindre un état vérifiant P_1 et un vérifiant P_2 et une telle stratégie peut nécessiter de faire deux choix différents pour A dans un même état (le premier pour atteindre P_1, le second pour P_2).

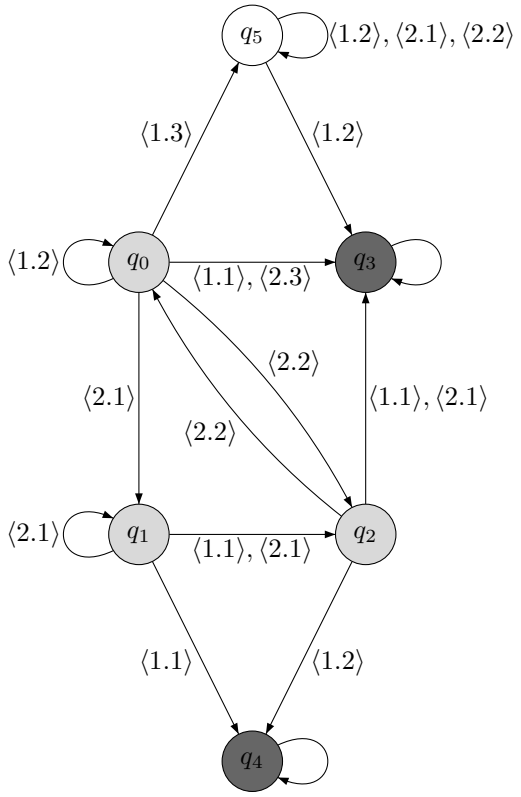


FIG. **2. Calcul de $[\![\langle\!\langle A_1 \rangle\!\rangle \mathbf{F}\, P]\!]$.**

5. Extensions temporisées

On considère à présent des modèles temps-réel (ou temporisés), c'est-à-dire munis d'une information quantitative sur le temps séparant deux états le long d'une exécution. Étant donné un préfixe fini π d'une exécution, on note $\mathsf{Durée}(\pi)$ la durée de π (mesurée dans l'unité de temps du modèle).

On commence par présenter une version temporisée d'ATL. Nous verrons ensuite plusieurs modèles de jeux temporisés sur lesquels on peut interpréter les formules de cette logique.

5.1. La logique TATL

La logique TATL [17] étend ATL avec des contraintes temps-réel de la même manière que TCTL [1] étend CTL :

Définition: [Syntaxe de TATL]

$$\begin{aligned} \mathsf{TATL} \ni \phi_s, \psi_s \quad &::= \quad P \mid \neg\phi_s \mid \phi_s \vee \psi_s \mid \langle\!\langle A \rangle\!\rangle \phi_p \\ \phi_p \quad &::= \quad \phi_s \, \mathbf{U}_{\sim c} \, \psi_s \mid \phi_s \, \mathbf{R}_{\sim c} \, \psi_s \end{aligned}$$

avec $P \in \mathsf{AP}$ et $A \subseteq \mathsf{Agt}$.

On définit sa sémantique de manière standard :

$$
\begin{aligned}
q \models_{\mathcal{S}} \langle\!\langle A \rangle\!\rangle \phi_p \text{ ssi } & \exists F_A \in \mathsf{Strat}(A).\ \forall \rho \in \mathsf{Out}(q, F_A). \\
& \rho \models_{\mathcal{S}} \phi_p, \\
\rho \models_{\mathcal{S}} \phi_s \mathbf{U}_{\sim d} \psi_s \text{ ssi } & \exists i.\ \rho[i] \models_{\mathcal{S}} \psi_s \text{ et } \mathsf{Durée}(\rho_{|i}) \sim d \\
& \text{et } \forall 0 \leq j < i.\ \rho[j] \models_{\mathcal{S}} \phi_s \\
\rho \models_{\mathcal{S}} \phi_s \mathbf{R}_{\sim d} \psi_s \text{ ssi } & \rho \models_{\mathcal{S}} \neg(\neg\phi_s \mathbf{U}_{\sim d} \neg\psi_s).
\end{aligned}
$$

Notons que l'opérateur $\mathbf{X}$ a disparu : il n'a pas de sens dans le cas des modèles à temps dense car il n'y a pas de notion de *prochain* état.

5.2. Le cas du temps discret

On présente d'abord une extension temporisée assez simple des CGS. Il s'agit ici d'associer une durée entière à chaque transition d'une CGS. On appelle ces modèles des TDCGS (*Tight Durational CGS*) [20].

Formellement une TDCGS est une CGS où la table de transitions est de la forme $\mathsf{Edg} : Q \times \mathbb{M}^k \to \mathbb{N} \times Q$. Si $\mathsf{Edg}(q, m) = (d, q')$, alors on note $\mathsf{Edg}_\tau(q, m)$ la durée d et $\mathsf{Edg}_s(q, m)$ l'état q'.

Comment décider si un état d'une TDCGS $\mathcal{S}$ satisfait ou non une formule $\phi \stackrel{\text{def}}{=} \langle\!\langle A \rangle\!\rangle P_1 \mathbf{U}_{\leq d} P_2$ avec $P_1, P_2 \in \mathsf{AP}$? Pour cela, on va calculer un ensemble de fonctions $v_i : Q \to \mathbb{N}$ pour $i \in \mathbb{N}$ de la manière suivante :

$$
\begin{cases}
\text{si } q \models P_2 : & v_0(q) = 0 \\
\text{si } q \models \neg P_2 : & v_0(q) = +\infty
\end{cases}
$$

$$
\begin{cases}
\text{si } q \models P_2 : \quad v_{i+1}(q) = 0 \\
\text{si } q \models \neg P_1 \wedge \neg P_2 : \ v_{i+1}(q) = +\infty \\
\text{Sinon :} \\
v_{i+1}(q) = \min\limits_{m \in \mathsf{Mv}(q,A)} \max\limits_{m' \in \mathsf{Mv}(q,\bar{A})} \Big(\mathsf{Edg}_\tau(q, m \cdot m') \\
\qquad\qquad + v_i(\mathsf{Edg}_s(q, m \cdot m')) \Big)
\end{cases}
$$

La valeur $v_i(q)$ correspond à la durée minimale que peut garantir A pour accéder à un état vérifiant P_2 (en ayant vérifié P_1 auparavant) en au plus i tours.

Considérons l'exemple de TDCGS de la figure 3 (la valeur de vérité des propositions atomiques est donnée dans le tableau ci-dessous) et la formule $\langle\!\langle A_1 \rangle\!\rangle P_1 \mathbf{U}_{\leq c} P_2$.

Le calcul des v_i est décrit par le tableau ci-dessous :

$v_i(q)$	i=0	i=1	i=2	i=3
q_0 (P_1, $\neg P_2$)	$+\infty$	$+\infty$	21	21
q_1 (P_1, P_2)	0	0	0	0
q_2 (P_1, $\neg P_2$)	$+\infty$	20	20	20
q_3 ($\neg P_1$, P_2)	0	0	0	0
q_4 (P_1, $\neg P_2$)	$+\infty$	$+\infty$	$+\infty$	$+\infty$

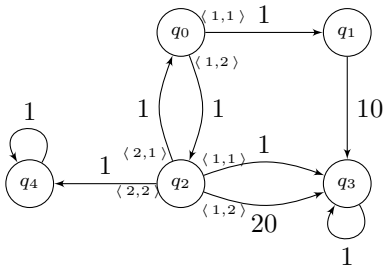


FIG. **3. Exemple de TDCGS**

Le calcul des v_i converge (ici en 3 itérations). Finalement on déduit de la dernière colonne du tableau que $\langle\!\langle A_1 \rangle\!\rangle P_1 \mathbf{U}_{\leq 25} P_2$ est vraie pour q_0, mais $\langle\!\langle A_1 \rangle\!\rangle P_1 \mathbf{U}_{\leq 20} P_2$ ne l'est pas.

On peut montrer que le calcul de $[\![\langle\!\langle A \rangle\!\rangle P_1 \mathbf{U}_{\leq d} P_2]\!]$ peut se faire en temps $O(|Q| \cdot |\mathsf{Edg}|)$.

Notons aussi qu'il existe une autre extension où l'on associe à chaque transition un intervalle d'entiers correspondant aux durées possibles que peut prendre la transition. On peut alors utiliser des agents particuliers qui ont pour seule action de choisir le temps des transitions. Le model-checking de ces modèles, les DCGS, procède de la même manière que celui des TDCGS avec la même complexité :

Théorème: [Complexité du model checking pour les (T)DCGS [20]]

- Le model checking de TATL sur les TDCGS ou les DCGS est un problème EXPTIME-complet.
- Le model checking de TATL$_{\leq,\geq}$ sur les TDCGS ou les DCGS est un problème P-complet.

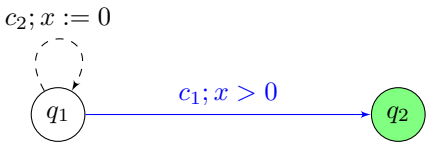
Des problèmes difficiles. La borne inférieure de complexité EXPTIME-dur ci-dessus repose sur le simple problème de décider si la formule $\langle\!\langle A \rangle\!\rangle \mathbf{F}_{=c} P$ est vraie ou non sur une TDCGS. Ce type de problème peut s'exprimer facilement sous la forme d'un jeu sur les graphes pondérés. On appelle le jeu du compte-à-rebours (*count-down game*), le jeu à deux joueurs suivant : on dispose d'un graphe $(Q, \mathcal{T})$ avec $\mathcal{T} \subseteq Q \times \mathbb{N}_{>0} \times Q$, une configuration du jeu est une paire $(q, c) \in Q \times \mathbb{N}$. A chaque tour, le joueur A_1 choisit un entier d tel que (a) $0 < d \leq c$ et (b) $\exists (q, d, q') \in \mathcal{T}$. Ensuite le joueur A_2 choisit une transition (q, d, q') dans $\mathcal{T}$. La nouvelle configuration est alors $(q', c - d)$. Le joueur A_1 gagne le jeu si une configuration de la forme $(-, 0)$ est atteinte. En fait, décider de l'existence d'une stratégie gagnante pour A_1 dans un tel jeu depuis une configuration (q_0, c) est un problème EXPTIME-complet [19]. On comprend bien dès lors que beaucoup de problèmes mélangeant des aspects quantitatifs (comme le temps) et des aspects d'interaction (avec plusieurs agents) appartiennent à des classes de complexité élevées.

5.3. Le cas du temps dense

Dans cette dernière section, nous allons présenter informellement deux types de jeux temporisés basés sur un modèle de temps dense (il en existe d'autres, comme par exemple [23, 6]).

Les TGA. Un des modèles les plus connus est celui des *Timed Game Automata*, les automates de jeux temporisés[14]. C'est une extension assez naturelle des automates temporisés d'Alur et Dill sous la forme d'un jeu à deux joueurs. L'automate dispose d'horloges (à valeur dans $\mathbb{R}_{\geq 0}$) qui avancent de manière synchrone avec le temps, les transitions d'action sont gardées par des contraintes sur la valeur courante des horloges (spécifiant ainsi **quand** la transition peut être tirée) et des opérations de remise à zéro d'horloges peuvent aussi être effectuées lors du franchissement de ces transitions. Les transitions sont partitionnées en deux sous-ensembles : le premier contient les transitions du joueur 1 et le second celles du joueur 2. A chaque tour, à partir d'une configuration (q, v) où v désigne la valuation courante des horloges du jeu, chaque joueur A_i choisit un coup composé d'un délai d'attente d_i après lequel il souhaite jouer et d'une transition t_i (que A_i contrôle) à tirer. Alors la nouvelle configuration (q', v') est soit l'état correspondant au choix de A_1 – c'est à dire l'attente de d_1 unités de temps suivie de la transition t_1 – si $d_1 < d_2$, soit celle correspondant au choix de A_2 si $d_2 < d_1$, soit elle provient d'un choix non-déterministe entre les deux configurations précédentes lorsque $d_1 = d_2$.

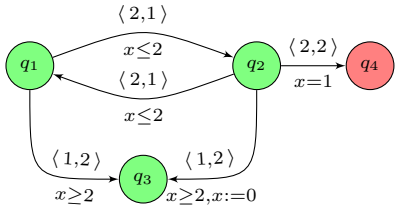
Considérons l'exemple ci-dessous :



Si depuis $(q_1, 0)$, A_1 joue $(1.5, c_1)$ et A_2 joue $(0.8, c_2)$, alors on arrive en $(q_1, 0)$. Mais si A_1 joue $(0.5, c_1)$ et A_2 joue $(0.8, c_2)$, alors on arrive en $(q_2, 0.5)$.

Toujours sur l'exemple ci-dessus, on peut voir que le joueur A_2 semble disposer d'une stratégie gagnante pour éviter à jamais l'arrivée dans q_2 : il lui suffit de jouer à chaque tour $(0, c_2)$…Le joueur A_1 devant toujours donner un délai strictement négatif, son coup ne sera jamais pris ! Il y a bien sûr un problème sémantique dans cette « stratégie gagnante » : le joueur A_2 bloque l'écoulement du temps…Pour éviter ce problème, il est important de ne retenir que les stratégies non-Zéno (qui ne bloque pas le temps). Plusieurs solutions pont été proposées, ici nous mentionnons celle de [14] : un joueur ne gagne la partie que s'il assure son objectif et que le temps diverge, ou alors si le temps converge c'est par la seule faute de son adversaire. Ainsi, avec cette définition, le joueur A_1 a bien une stratégie pour atteindre q_2 (exercice : laquelle ?).

Les TCGS. Les TCGS (pour *Timed CGS*) sont des CGS temporisées [9]. L'idée est ici de garder le caractère concurrent des CGS et d'y ajouter du temps quantitatif. La figure ci-dessous donne un exemple de TCGS : on voit que les transition sont toujours étiquetées avec le vecteur de coups correspondant complété par une garde et une éventuelle remise à zéro.



Le jeu se déroule comme suit : chaque joueur A choisit un délai d'attente d comme pour les TGA, complété par une fonction f de $\mathbb{R}^+$ dans $\mathbb{M}$ qui donne pour chaque délai positif le coup souhaité pour A. Ce coup (d, f) signifie que A souhaite jouer $f(d)$ après un délai d, mais que s'il *devait* jouer à une autre date (*i.e.* contraint par un de ses adversaires) alors il fournit aussi une fonction générale indiquant ses choix.

Dès lors que chaque joueur A_i a choisi un coup (d_i, f_i), le système changera d'état après $d \stackrel{\text{def}}{=} \min(d_1, \ldots, d_k)$ unités de temps et la transition (d'automate temporisée) choisie est donnée par la table de transitions $\mathsf{Edg}(q, f_1(d), \ldots, f_k(d))$.

Considérons l'exemple ci-dessus. Notons qu'il manque des transitions : par exemple, depuis q_1, le coup $\langle 1.2 \rangle$ lorsque x est inférieur strictement à 2 n'est pas représenté. On suppose que ces absences correspondent à des boucles sur le même état de contrôle. Depuis l'état $(q1, x = 1.2)$, supposons que A_1 choisisse (d_1, f_1) avec $d_1 = 0.9$ et $f_1(d) = 2$ si $d \leq 0.5$ et $f_1(d) = 1$ si $d > 0.5$. De plus, supposons que A_2 choisisse (d_2, f_2) avec $d_2 = 1.4$ et $f_2(d) = 2$ pour $d \leq 1$ et $f_2(d) = 1$ pour $d > 1$. Alors le système exécutera la transition $q_2 \rightarrow q_3$ étiquetée par $\langle 1.2 \rangle$ et gardée par $x \geq 2$ après un délai de 0.9 (et à cette date x vaudra donc 2.1).

Comme pour les TGA, le model checking est décidable. Ici on peut construire une « CGS des régions » qui est bisimilaire (au sens des jeux et du temps qualitatif) à la CGS infinie représentant la sémantique de la TCGS.

Finalement, on obtient les résultats suivants concernant ces modèles :

Théorème:

– Le problème du model checking de TATL sur les TGA est EXPTIME-complet [14].

– Le problème du model checking de TATL sur les TCGS est EXPTIME-dur et dans EXPSPACE [9].

Notons aussi que l'outil UppAal dispose d'une version dédiée – UppAal Tiga [2] – à l'analyse d'une variante des TGA (on peut vérifier des propriétés d'atteignabilité au moyen d'algorithmes à la volée [10]) : des outils existent donc pour ces modèles et là encore des études de cas ont été résolues avec eux.

6. Conclusion

Ce document est loin d'être exhaustif, il a juste pour objectif de présenter quelques modèles de jeux et logiques temporelles adaptées. Les aspects temps-réel rendent la définition et l'analyse de ces modèles encore plus complexes. C'est la raison pour laquelle nous avons rappelé les modèles pour le cas non-temporisé car ils permettent d'aborder une série de questions sémantiques importantes que l'on retrouve aussi dans le cadre temporisé. Ce domaine de recherche est riche et très dynamique, de nombreuses questions restent aujourd'hui ouvertes.

Références

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

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

[3] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In W.-P. de Roever, H. Langmaack, and A. Pnueli, editors, *Revised Lectures of the 1st International Symposium on Compositionality : The Significant Difference (COMPOS'97)*, volume 1536 of *LNCS*, pages 23–60. Springer, 1998.

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

[5] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In D. Sangiorgi and R. de Simone, editors, *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *LNCS*, pages 163–178. Springer, 1998.

[6] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proc. Symp. System Structure & Control*, pages 469–474. Elsevier, 1998.

[7] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[8] Th. Brihaye, A. Da Costa, F. Laroussinie, and N. Markey. ATL with strategy contexts and bounded memory. In *Proceedings of the Symposium on Logical Foundations of Computer Science (LFCS'09)*, volume 5407 of *LNCS*, pages 92–106, Deerfield Beach, FL, USA, Jan. 2009. Springer.

[2] `http://www.cs.aau.dk/~adavid/tiga/`

[9] Th. Brihaye, F. Laroussinie, N. Markey, and G. Oreiby. Timed concurrent game structures. In L. Caires and V. T. Vasconcelos, editors, *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR'07)*, volume 4703 of *Lecture Notes in Computer Science*, pages 445–459, Lisbon, Portugal, Sept. 2007. Springer.

[10] F. Cassez, A. David, E. Fleury, K. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR'05*, volume 3653 of *LNCS*, pages 66–80. Springer, 2005.

[11] K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR'07)*, LNCS, pages 59–73. Springer-Verlag, Sept. 2007.

[12] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Prog. Lang. Syst.*, 8(2) :244–263, 1986.

[13] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[14] L. de Alfaro, M. Faella, T. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *CONCUR'03*, volume 2761 of *LNCS*, pages 144–158. Springer, 2003.

[15] 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, 1990.

[16] V. Goranko and G. van Drimmelen. Complete axiomatization and decidability of alternating-time temporal logic. *Theoretical Computer Science*, 353(1-3) :93–117, Mar. 2006.

[17] T. Henzinger and V. Prabhu. Timed alternating-time temporal logic. In *FORMATS'06*, volume 4202 of *LNCS*, pages 1–17. Springer, 2006.

[18] W. Jamroga and J. Dix. Do agents make model checking explode (computationally) ? In M. Pechoucek, P. Petta, and L. Zsolt Varga, editors, *Proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05)*, volume 3690 of *LNCS*. Springer, 2005.

[19] M. Jurdziński, F. Laroussinie, and J. Sproston. Model checking probabilistic timed automata with one or two clocks. In *Proceedings of the 13th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lecture Notes in Computer Science*, pages 170–184, Braga, Portugal, Mar. 2007.

[20] F. Laroussinie, N. Markey, and G. Oreiby. Model checking timed ATL for durational concurrent game structures. In E. Asarin and P. Bouyer, editors, *Proceedings of the 4th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 245–259, Paris, France, Sept. 2006. Springer.

[21] F. Laroussinie, N. Markey, and G. Oreiby. On the expressiveness and complexity of ATL. *Logical Methods in Computer Science*, 4(2 :7), May 2008.

[22] 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.

[23] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS'95*, volume 900 of *LNCS*, pages 229–242. Springer, 1995.

[24] R. Mazala. Infinite games. In *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*, pages 23–42. Springer-Verlag, 2002.

[25] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Comp. Soc. Press, 1977.

[26] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. 5th Int. Symp. on Programming, Turin, Italy, Apr. 1982*, volume 137 of *LNCS*, pages 337–351. Springer, 1982.

[27] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1) :81–98, 1989.

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

[29] W. Thomas. On the synthesis of strategies in infinite games. In *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *LNCS*, pages 1–13. Springer-Verlag, Mar. 1995.

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