Synthèse de Stratégies Infinies sans Déterminisation

Aidan Harding* ath@cs.bham.ac.uk
Mark Ryan* mdr@cs.bham.ac.uk
Pierre-Yves Schobbens† pys@info.fundp.ac.be

*School of Computer Science
University of Birmingham
Edgbaston, Birmingham B15 2TT, UK

†Institut d'Informatique
Facultés Universitaires de Namur
Rue Grandgagnage 21
5000 Namur, Belgium

Résumé :
Nous présentons un algorithme pour la synthèse de stratégies dans les jeux oméga-réguliers. Cet algorithme ne nécessite pas de déterminisation, il n'utilise que deux points fixes imbriqués, il se prête à une implémentation symbolique, ce qui indique qu'il sera efficient en pratique. Il est destiné à la vérification de systèmes orientés-agents décrivant les protocoles d'interaction entre agents, par rapport à une logique de coopération/compétition bâtie sur une logique temporelle quelconque.

1 Introduction

La synthèse de stratégies est une question importante dans de nombreux domaines, mais dans cet article nous la considérons dans le cadre des systèmes multi-agents. Elle permet notamment de résoudre les problèmes suivants :

- La vérification : Étant donné un système, nous pouvons vérifier que certains agents peuvent bien exercer leurs droits, et la synthèse nous dit de quelle façon.
- Planification : la synthèse nous donne un plan adaptatif et multi-agents : pour chaque situation possible, elle indique à chacun les actions à effectuer, et assure l'obtention du résultat désiré quoi que fassent les autre agents pour l'empêcher. De plus, elle traite des objectifs temporels quelconques, par exemple de maintenir une propriété tout en cherchant à atteindre une série d'objectifs.
- La construction de techniques d'attaque ou de défense pour les protocoles de sécurité : [8] montre comment des propriétés de sécurité qui n'avaient pu être formalisées jusqu'à présent, comme la non-répudiation, etc. sont modélisables dans notre approche. La synthèse fournit alors soit le plan d'attaque, soit le plan de défense suivant que le protocole est correct ou non.
- La construction de tests, de manuels pour l'utilisateur : Les systèmes sont bâtis pour permettre à certaines fonctions d'être exercées suivant les désirs des utilisateurs. La synthèse fournit la séquence exacte d'opérations à effectuer pour obtenir une de ces fonctions.

1.1 Plan

L'article rappelle d'abord des définitions classiques, et d'autres qui le sont moins. La Section 3 présente l'algorithme de synthèse de stratégies. La Section 4 présente des applications et extensions possibles.

2 Jeux et Automates

Notre algorithme produit des stratégies pour un jeu infini. Il nous faut donc définir les parties où le premier joueur (appelé A) gagne, ce qu'on appelle la *condition pour gagner*. Comme les parties sont infinies, il nous faut trouver une façon finie de décrire un langage infini de mots (parties) infinis. La solution classique est un automate de Büchi. Les algorithmes connus de synthèse de stratégies requièrent un automate déterministe. Cependant, les automates de Büchi déterministes sont nettement moins expressifs. Nous devons donc utiliser une famille d'automates plus puissants. L'exemple classique est l'algorithme de Safra [9], qui déterminise un automate de Büchi et produit un automate de Rabin déterministe. Cependant, cet algorithme est très coûteux, et produit un automate très grand dans une famille complexe qui est elle-même très coûteuse à traiter. Donc, nous proposons de rester dans les automates de Büchi non-déterministes, mais nous requérons une condition assez originale : l'automate doit être partiellement déterministe, et pour le reste déterministe en arrière. Ce mélange étrange est indispensable, puisque nous savons que les automates de Büchi déterministes tant en avant qu'en arrière manquent de l'expressivité nécessaire.

2.1 Jeux infinis

Un jeu se compose de :
– Σ, les *positions* qui forment l'*arène*.
– Σ_A, Σ_O une partition de Σ en deux : les positions A joue et où O joue.
– $\delta_G \subset \Sigma \times \Sigma$, une relation de *coups* totale (car le jeu est infini).
– $W_A \subset \Sigma^\omega$, la condition pour que A gagne, qui est un ensemble de mots infinis, autrement dit un langage.

Un *coup* est un couple de positions de la relation du jeu. Une *partie (mot)* est une suite infinie de positions liées par un coup.

Pour un jeu ω-régulier, la condition $W_A = \mathcal{L}(B)$ est présentée par un automate de Büchi satisfaisant les conditions de la Section 2.2 ci-dessous. Ces jeux sont *déterminés* [1], c'est-à-dire qu'en chaque position un des joueurs a une stratégie gagnante.

2.2 Automates pour gagner

Rappelons qu'un *automate de Büchi* B se compose de :
– Σ, son *alphabet* (ici, les positions)
– Q_B, ses *états* finis
– $\phi_0 \subset Q_B$, ses *états initiaux*
– $\rightarrow_B \subset Q_B \times \Sigma \times Q_B$, sa *relation de transition*
– N_f, ses *états accepteurs*

Une *exécution* de l'automate est une suite infinie d'états $q_0 \rightarrow_B^{a_0} q_1 \rightarrow_B^{a_1} \dots$ qui suit la relation de transition. Une exécution est *acceptée* ssi elle visite N_f infiniment souvent. Le *mot d'*une exécution est $a_0 a_1 \dots$. Une exécution est *initiale* ssi son premier état est initial : $q_0 \in \phi_0$. Les mots d'exécutions initiales acceptées forment le *langage* de B, $\mathcal{L}(B)$. On écrit $B(\phi)$ pour l'automate B avec ϕ comme états initiaux. Le langage d'un état s est alors $\mathcal{L}(B(\{s\}))$.

Un automate est *globalement déterministe en arrière* ssi deux états distincts ont des langages disjoints. Il est *déterministe en arrière* [3] ssi deux états s, t convergeant vers un même état $s' : s \rightarrow_B^a s', t \rightarrow_B^a s'$ doivent être identiques : $s = t$.

Mais ces propriétés sont trop strictes pour représenter tous les langages ω-réguliers. Donc, nous les affaiblissons par une équivalence $\sim$ entre états telle que deux états équivalents ont le même langage :

$$s \sim t \rightarrow \mathcal{L}(B(\{s\})) = \mathcal{L}(B(\{t\})) \qquad (1)$$

Ceci nous permet de définir la classe plus puissante des automates *globalement déterministes en arrière hors de* $\sim$: deux états s, t ont des langages disjoints ou sont équivalents (et donc ont le même langage). De même, ils sont *déterministes en arrière hors de* $\sim$ ssi $\forall s' \sim t'$, si $s \rightarrow_B^a s'$ et $t \rightarrow_B^a t'$ alors $s \sim t$.

Comme toujours, $[X]_\sim = \{s \mid \exists t \in X. s \sim t\}$ note la fermeture X par $\sim$. Un ensemble d'états est *représentatif* s'il contient un état dans chaque classe de $\sim$; il est *sous-représentatif* s'il en contient au plus un. Nous demandons un N_0 représentatif.

Un état mort a un langage vide. Ces états peuvent être retirés, et nous présentons un algorithme pour ce faire en Section 2.3. Dans ce cas, la propriété globale est plus forte que la variante locale : si $s' \sim t', s \rightarrow^a s', t \rightarrow^a t'$ alors $a.\mathcal{L}(B(\{t\}))$ est non vide et dans leur intersection, donc $s \sim t$.

Un automate est *déterministe dans* $\sim$ ssi $s \sim t, s \rightarrow^b s', t \rightarrow^b t'$ implique $s' = t'$.

Un automate est *globalement total en arrière* ssi tout mot est dans le langage d'un état. Il est *(localement) total en arrière* si chaque état a une transition en arrière pour chaque position (symbole).

Exemple. L'automate $Q_B = \{d, e, f, g\}$ où $\Sigma = \{p, q\}$, $d \rightarrow_B^p d, d \rightarrow_B^q d, d \rightarrow_B^p e, f \rightarrow_B^q e, e \rightarrow_B^p f, g \rightarrow_B^q f, f \rightarrow_B^p g, d \rightarrow_B^q g$, $N_f = \{f\}$, $\phi_0 = \{f\}$ est déterministe et total en arrière. Cependant, $(qp)^\omega \in [\mathcal{L}(B(g)) \cap \mathcal{L}(B(f))]$, tandis que p^ω n'est dans le langage d'aucun état. Donc cet automate n'a pas les variantes globales de ces propriétés.

Si nous supprimons d, nous préservons son langage, nous le rendons déterministe mais il n'est plus total en arrière.

Si nous changeons les états accepteurs en $N_f = \{e\}$, nous réduisons son langage mais gardons bien sûr les variantes locales. Il devient alors globalement déterministe et total en arrière.

2.3 Suppression des états morts

La suppression des états morts peut se faire en temps linéaire par une double recherche en profondeur. Mais nous présentons ici l'algorithme d'Emerson-Lei [2] car il est plus efficace en pratique pour les implémentations symboliques, et

aussi parce qu'il est un cas particulier de notre algorithme présenté en Section 3. On l'écrit simplement en μ-calcul : $\nu z.\mu s.EX((N_f \cap z) \cup s)$.

z sont les états qui peuvent atteindre N_f k fois, où k est le nombre d'itérations du point fixe extérieur. Au point fixe, k est infini, et z sont les états non morts (vivants). s sont les états qui peuvent atteindre $z \cap N_f$ en l pas, où l est le nombre d'itérations du point fixe intérieur.

2.4 Déterminisation finie

Nous utilisons la procédure classique de déterminisation. Il est bien connu qu'elle donne des résultats incorrects pour les automates infinis, mais nous corrigeons ceci en gardant une copie de l'original B dans notre algorithme.

Le résultat est appelé M, la mémoire :
– Σ est le même alphabet de positions.
– Q_M sont des ensembles représentatifs.
– l'état initial est $[\phi_0]_\sim \cap N_0$.
– la transition δ_M est classiquement $\delta_M(\phi, a) = \{q_2 \in Q_B \mid \exists q_1 \in \phi_1.q_1 \rightarrow^a q_2\}$.

3 Synthèse

Notre algorithme prend en entrée un jeu comme ci-dessus, et calcule sa déterminisation finie M.

Les stratégies pour A que nous calculons sont fonction de la mémoire et de la position : $g, n \subset Q_M \times \delta_G$. Quand g ou n ne sont pas définies, cela signifie qu'elle renoncent, parce qu'aucun coup n'est gagnant. g est déterministe sur Σ_A, càd qu'elle renvoie au plus un successeur. n est la *la stratégie la plus générale*, tandis que g est la *stratégie la plus courte*. Ces deux stratégies ont des propriétés différentes. n contient toute stratégie gagnante possible, mais elle n'est pas nécessairement gagnante elle-même car elle ne garantit pas de se rapprocher du but. g au contraire se rapproche du but aussi vite que possible, mais elle n'évolue pas de façon monotone au cours de l'algorithme. Donc aucune variable monotone ne peut dépendre de g, sous peine de détruire la propriété de point fixe.

Notre algorithme est similaire à Emerson-Lei : Il calcule les états gagnants et la stratégie en arrière, en faisant des hypothèses optimistes sur ce qui se passe au delà de l'horizon k. Par contre il travaille sur trois états simultanément : $Q_C = \Sigma \times Q_B \times Q_M$. Ces triples sont notés (a, t, ϕ) :
– $a \in \Sigma$ est la position courante du jeu ;
– $t \in Q_B$ est l'état de l'automate B ; il ne peut être utilisé dans la stratégie, car il décrit l'évolution future du jeu ;
– $\phi \in Q_M$ est la mémoire.
Les chemins que nous calculons satisfont donc aussi les trois relations de transition. Cette nouvelle relation de transition sera notée Δ_C.

Nous calculons $z \subset Q_C$, les triples gagnants pour A. Lorsque ϕ est l'état initial de la mémoire et que $t \in \phi$, les a correspondants sont les états gagnants pour A. Les t correspondants décrivent les évolutions futures possibles du jeu, et elles sont toutes gagnantes si tous les t sont dans ϕ. Nous mettons cette dernière propriété comme invariant de notre algorithme, et la question est alors seulement de savoir si le jeu va se poursuivre infiniment sans que g ne renonce, càd si on peut trouver un t. Ceci est testé par $u = \emptyset$ dans l'algorithme ci-contre.

Comme toujours, une position de A est considérée comme gagnante dès qu'un coup est gagnant ; ce coup est enregistré dans la stratégie, et les u sont enregistrés dans s individuellement. Pour une position de O, tous les coups doivent être gagnants pour que la position soit considérée comme gagnante. Dans ce cas, tous les u sont enregistrés dans s ensemble.

4 Conclusion

Nous avons présenté un nouvel algorithme, plus efficace, pour la synthèse de stratégies dans les jeux infinis. Cette efficacité est supposée sur base du fait notre algorithme partage les principes d'algorithmes qui sont actuellement utilisés industriellement par exemple dans Cadence SMV, mais nous devons encore la démontrer pratiquement en construisant une implémentation efficace et en l'utilisant sur des exemples de grande taille.

Pour traiter de tels exemples, nous devons généraliser les techniques utilisées dans les implémentations industrielles actuelles, comme l'abstraction (ce qui est fait dans [6]), les cônes d'influence, les ordres partiels.

On peut se demander si la construction des automates que nous requérons comme donnée de l'algorithme n'est pas trop coûteuse. En fait, toutes les logiques temporelles linéaires génèrent naturellement des automates-tableaux qui satisfont ces conditions. En particulier, les logiques LTL (ce qui donne une synthèse pour ATL* [4]), ECL [5] (ce qui donne une synthèse

$z := \{(a,t,\phi) \mid a \in \Sigma, t \in Q_B, \phi \in Q_M, t \in \phi\}$
repeat counted by k
  $g := n := \emptyset$;
  $s_0 := \{(a,t,\phi) \in z \mid t \in N_f\}$;
  repeat counted by l
    $s_l :=$ if $l > 1$ then s_{l-1} else $\emptyset$
    for $a \in \Sigma_O, \phi \in 2^Q$
      $v := \top$; – *Is the O position winning for A ?*
      $r := \emptyset$;
      for $a' \in \delta_G(a)$ do
        $u := \{(a,t,\phi) \mid \exists t'.(a',t',\delta_M(\phi,a)) \in s_{l-1}$
                $\wedge \quad t \rightarrow^a t'\}$
          $v := (v \wedge u \neq \emptyset)$
          $r := r \cup u$
      end for a'
      if v then
        $s_l := s_l \cup r$
      end if v
    end for a ;
    for $a \in \Sigma_A, \phi \in 2^Q$
      for $a' \in \delta_G(a)$ do
        $u := \{(a,t,\phi) \mid \exists t'.(a',t',\delta_M(\phi,a)) \in s_{l-1}$
                $\wedge \quad t \rightarrow^a t'\}$
        if $u \neq \emptyset$ then
          $g := g \cup \{(\phi,a,a')\}$ if $g(\phi,a) = \emptyset$;
          $n := n \cup \{(\phi,a,a')\}$;
          $s_l := s_l \cup u$;
        end if
      end for a'
    end for a
  until s stabilises
  $z := z \cap s$
until z stabilises

FIG. 1 – Synthèse de g

pour les problèmes avec temps-réel), et DC (ce qui permet de traiter des intervalles plutôt que des points).

Les stratégies synthétisées peuvent facilement être présentées sous forme de programmes à l'utilisateur. Nous avons aussi un outil [7] qui permet de jouer contre une stratégie pour avoir une intuition des raisons qui la font gagner. Il permet aussi de jouer contre des joueurs réels en réseau.

Références

[1] D.A. Martin. Borel determinacy. *Ann. Math.*, 102 :363–371, 1975.

[2] E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proceedings of the Eighteenth Hawaii International Conference on System Sciences*, volume 1, pages 277–288, North-Holland, CA, 1985. Western Periodicals Company.

[3] E. Allen Emerson and A. Prasad Sistla. Deciding full branching time logic. *Information and Control*, 61(3) :175–201, June 1984.

[4] Aidan Harding, Mark Ryan, and Pierre-Yves Schobbens. Symbolic synthesis for atl*. In Marc Pauly, editor, *Proc. Symposium on Logic in Games and Multiagent Systems*, 2002.

[5] T.A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In K.G. Larsen, S. Skyum, and G. Winskel, editors, *ICALP 98 : Automata, Languages, and Programming*, Lecture Notes in Computer Science 1443, pages 580–591. Springer-Verlag, 1998.

[6] Thomas A. Henzinger, Freddy Mand, Rupak Manjumdar, and Jean-François Raskin. Abstract interpretation of game properties. In *Symposium on Static Analsysis*, 2000.

[7] Patrick Heymans. The albert II specification animator. Technical Report CREWS-97-13, University of Namur, 1997.

[8] Kremer and Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *CONCUR : 12th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2001.

[9] Shmuel Safra. *Complexity of automata on infinite objects*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1989.