La vérification à l’aide des jeux

Clément Tamines

Séminaire jeunes

5 mars 2020

Pour ceux qui ne me connaissent pas. . .

En résumé :

- Clément Tamines
- Bachelier et Master en Informatique
- Doctorat avec Véronique Bruyère et Jean-François Raskin (ULB)
- Théorie des jeux

Figure: Moi.

Sujet de ce séminaire : **introduire** des notions de **théorie des jeux**

Plan de la présentation

Présenter les concepts de **vérification** et de **synthèse**

- Quels problèmes essayent-ils de résoudre et comment ?

Introduire des notions de **théorie des jeux**

- Concepts de jeux sur graphe à deux joueurs
- Jeux d'accessibilité et de parité
- Discuter d'algorithmes permettant de les résoudre

Introduire brièvement mes **thèmes de recherche**

- Solveurs partiels
- Notion de pareto-optimalité dans les jeux multijoueurs

Systèmes réactifs

Systèmes informatiques qui **interagissent** avec leur **environnement**

- Ils doivent pouvoir réagir aux évènements dans cet environnement
- Il faut donc considérer tous les évènements possibles de celui-ci

Ces systèmes sont **grands et complexes** et donc **sujets aux erreurs**

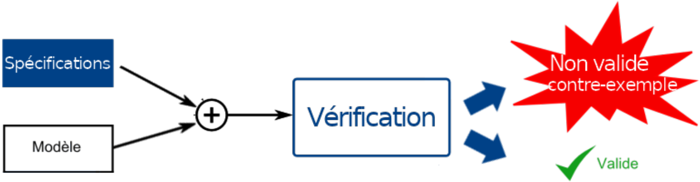
- Nécessité de nombreux tests pour assurer leur bon fonctionnement
- Il est difficile de tester toutes les possibilités de combinaisons d'évènements
- Certains systèmes sont critiques, ne tolèrent pas les bugs

La vérification formelle

On aimerait une **preuve formelle** du bon fonctionnement du système

- Montrer que le système a toujours un comportement correct
- Quels que soient les évènements de l'environnement

La **vérification formelle** nous permet d'effectuer cette tâche



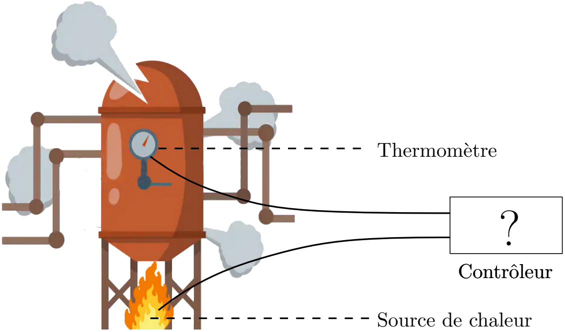
Un algorithme va **confirmer** si le système **respecte la spécification** et ce **quels que soient les évènements** produits par l'environnement

La synthèse de contrôleur

On veut effectuer le **travail inverse**

- On a un environnement qui peut produire plusieurs évènements
- On a une spécification que notre système doit respecter
- On veut générer un contrôleur qui respecte la spécification

Contrôleur de **régulation de température** où la spécification demande $[T_{min}, T_{max}]$

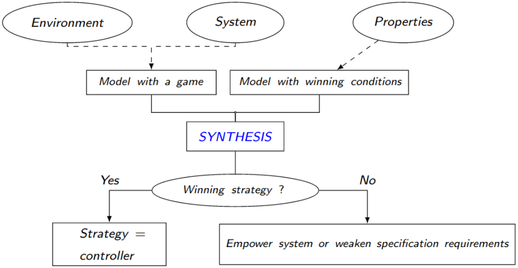


Synthèse via les jeux

On peut voir notre problème de synthèse **comme un jeu**

- Le système est un joueur, l'environnement aussi
- But du système : fonctionner correctement i.e., respecter la spécification
- On suppose que l'environnement veut l'en empêcher

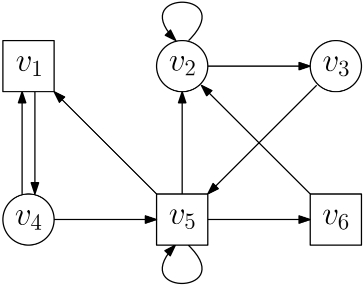
⇒ le système et l'environnement sont donc **antagonistes**



Une **façon de jouer** pour le système qui gagne à tous les coups = **un contrôleur**

Jeu sur graphe

Jeu à **deux joueurs** : joueur 1 (système) et joueur 2 (environnement)



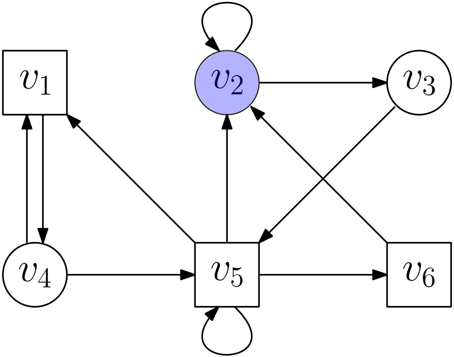
Joué sur une arène de jeu

- états V
- arcs E
- pas de deadlock
- V partitionné entre les joueurs : V_1 ($\bigcirc$) et V_2 ($\square$)

Partie

Partie : **chemin infini** dans V^ω qui commence en un état initial

Règle : **déplace un jeton**, le propriétaire de l'état actuel décide où aller



Partie $\pi = v_1\, v_4\, v_5\, v_2 v_2{}^\omega$

Objectif

Le but d'un joueur est de satisfaire son **objectif**

- Ensemble $\Omega \subseteq V^\omega$ de parties qui respectent une certaine propriété
- Joueurs antagonistes : si l'objectif de 1 est Ω, celui de 2 est $V^\omega \setminus \Omega$
- Exemple : $\Omega_1 =$ parties qui visitent v_1 alors $\Omega_2 =$ parties qui évitent v_1

Joueur 1 **gagne** une partie π si celle-ci **satisfait** cette propriété : $\pi \in \Omega_1$

Résoudre un jeu revient à calculer

- Les états depuis lesquels chaque joueur peut s'assurer de gagner
- La façon dont les joueurs jouent pour ce faire

Stratégies

Une stratégie pour un joueur $j \in \{1, 2\}$ est une **fonction** $V^* \times V_j \to V$

- Dicte les choix d'un joueur
- Pour lui permettre de satisfaire son objectif

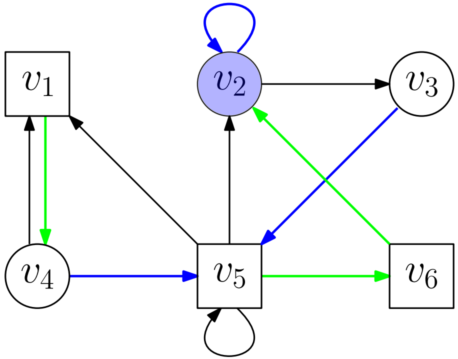
On parle de stratégie **gagnante** pour un joueur

- Depuis un état v
- Assure à ce joueur de satisfaire son objectif
- Quelles que soient les actions de son adversaire

Stratégie sans mémoire

Le choix de l'arc en un état donné ne dépend **que de cet état**

- Stratégie σ_1 pour le joueur 1 (en bleu)
- Stratégie σ_2 pour le joueur 2 (en vert)

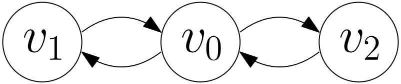


Partie $\pi = v_1\ v_4\ v_5\ v_6\ v_2 {v_2}^{\omega}$

Stratégie à mémoire finie

Le choix de l'arc en un nœud dépend de ce nœud **et du passé** (histoire)

Exemple : passer infiniment souvent par v_1 et v_2



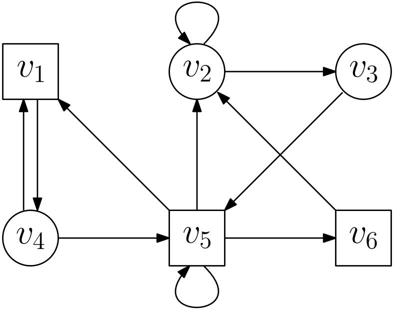
Nécessite de se rappeler les nœuds visités !

$\Rightarrow$ Une telle stratégie est encodée par un automate

Jeu d'accessibilité

Ajout d'un **ensemble cible** $U \subseteq V$

- Objectif du joueur 1 : atteindre un état de U (accessibilité)
- Objectif du joueur 2 : éviter les états de U (sûreté)



Ex : $U = \{v_5\}$, $P = v_1\ v_4\ v_5\ v_2{}^{\omega}$ est gagnante pour le joueur 1

Attracteur

Région gagnante du joueur 1 :

- Ensemble d'états à partir duquel le joueur 1 est certain d'atteindre U
- Il a donc une stratégie gagnante depuis les états de cet ensemble
- Quelles que soient les actions du joueur 2

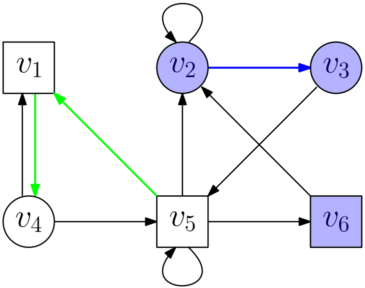
Cet ensemble est appelé **attracteur**

⟶ Résoudre les jeux d'accessibilité revient à les calculer

⟶ Les stratégies gagnantes sont construites lors de leur calcul

Exemple d'attracteur

Attracteur pour le **joueur 1** de **l'ensemble cible** $\{v_3\}$

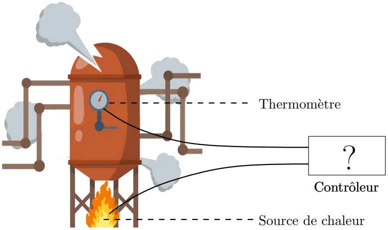


Construction par étapes

- Initialement : $\{v_3\}$
- Ajout des prédécesseurs de $\{v_3\}$ qui appartiennent au joueur 1
- Ajout des prédécesseurs de $\{v_3\}$ qui appartiennent au joueur 2 si tous les arcs sortants arrivent en l'attracteur
- Répété jusqu'à ce qu'aucun nœud ne soit ajouté

$\Rightarrow$ Algorithme polynomial : $\mathbf{O}(n+m)$

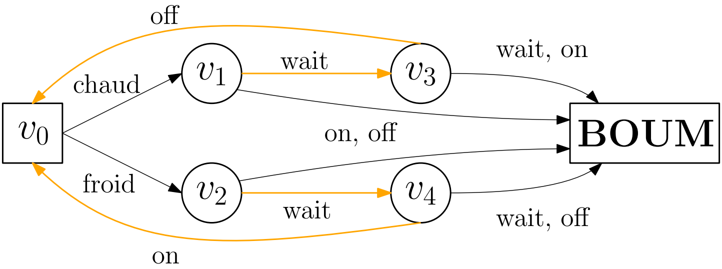
Retour à la synthèse de contrôleur



Modéliser le problème **par un jeu**

Résoudre ce jeu

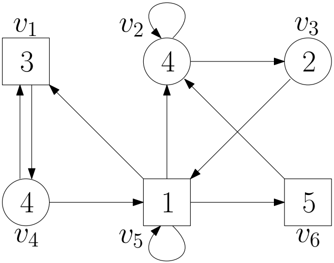
On obtient ainsi un **contrôleur**



Jeux de parité

Ajout d'une **fonction de priorité** $\alpha : V \to \mathbb{N}$
$\to$ on associe à une partie la suite de priorités correspondante

Objectif : **priorité maximale** apparaissant **infiniment souvent** dans π
$\to$ si celle-ci est paire (impaire) le joueur 1 (2) gagne



$$\pi = v_1\, v_4\, v_5\, v_6\, (v_2\, v_3\, v_5)^\omega$$
$$\alpha(\pi) = 3\ 4\ 1\ 5\ (4\ 2\ 1)^\omega$$

$\to$ le joueur 1 gagne la partie

Résoudre les jeux de parité

Jeux de parité $\in$ NP $\cap$ co-NP, de nombreux algorithmes existent

Algorithme	Complexité	Référence
Recursive	$\mathcal{O}(e \cdot n^d)$	[4]
Small Progress Measures	$\mathcal{O}(d \cdot e \cdot (\frac{n}{d})^{d/2})$	[1]
Big-Step	$\mathcal{O}(e \cdot n^{\frac{1}{3}d})$	[3]
Dominion Decomposition	$\mathcal{O}(n^{\sqrt{n}})$	[2]

Mon mémoire : résoudre des jeux de parité par réduction aux jeux de sûreté

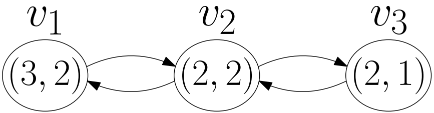
1. Transformer un jeu de parité G en jeu de sûreté particulier G'
2. Résoudre le jeu de sûreté G' comme précédemment
3. Obtenir la solution du jeu de parité G depuis celle du jeu de sûreté G'

Jeux de parité généralisée

Ajout de k **fonctions de priorité**, $1 \leq i \leq k$, $\alpha_i : V \to \mathbb{N}$

On considère les suite de priorités correspondantes

- Joueur 1 : conjonction d'objectifs de parité
- Joueur 2 : disjonction complémentaire d'objectifs de parité



$$\pi = v_1 \, (v_2 \, v_3)^\omega$$
$$\alpha_1(\pi) = 3 \, (2 \; 2)^\omega$$
$$\alpha_2(\pi) = 2 \, (2 \; 1)^\omega$$

Différence importante : le joueur 1 a une stratégie à mémoire finie

Solveur partiel

Algorithmes qui **résolvent partiellement** les jeux de parité en **temps polynômial**
$\Rightarrow$ ils ne donnent donc pas toujours la solution complète !

Un **solveur partiel** prend un jeu en entrée et donne

- Des ensembles d'états gagnants pour chaque joueur
- Le fragment du jeu non résolu

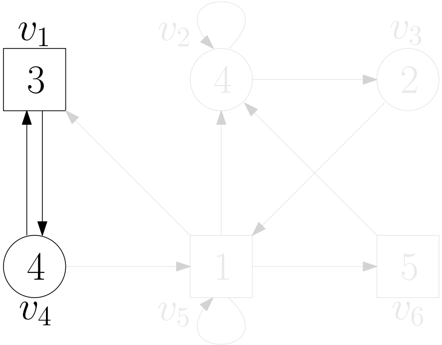
Pourquoi est-ce intéressant ?

- Algorithmes en temps polynômial
- Fonctionne bien en pratique sur les familles de graphes généralement testées

Exemple simple de solveur partiel

Identifier états du joueur 1 de **priorité paire** avec une **boucle**

- Depuis un tel état, le joueur 1 peut boucler
- Voir donc infiniment souvent un état de couleur paire
- Le joueur 1 gagne depuis l'attracteur de cet état



Questions de recherche

Nos **contributions**

- Étendre les solveurs partiels aux jeux de parité généralisée
- Combiner des solveurs partiels avec un solveur complet
- Évaluer les performances sur des exemples pratiques

Informatique **théorique**

- Classifier les problèmes en fonction de la complexité de leur résolution
- Étude des complexités théoriques des algorithmes : $P \subseteq NP \subseteq PSPACE$

En **pratique**

- Implémentation de nos algorithmes
- Benchmarks
- Compétitions de solveurs

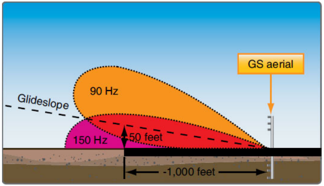
Synthèse et pareto-optimalité

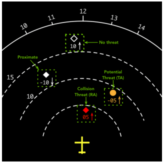
On a fait **plusieurs hypothèses**

- Deux joueurs : système et l'environnement
- Antagonistes : suppose que l'environnement veut faire échouer le système

En **pratique**

- Système interagit avec plusieurs autre systèmes (agents)
- Ces agents ont leur propre but, autre que faire échouer le système
- Environnement non antagoniste composé d'agents avec leur propre objectif

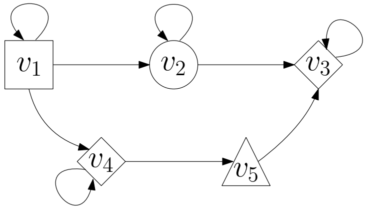




Synthèse et pareto-optimalité

Jeux à *k* **joueurs** où chacun a son propre objectif

- On a 4 joueurs 0, 1, 2, 3 et 1, 2, 3 collaborent
- Les joueurs 1, 2, 3 jouent de façon rationnelle : pareto-optimale
- On regarde qui satisfait son objectif dans 1, 2, 3 : (0, 1, 0)
- Pareto : personne ne peut améliorer son gain sans baisser celui d'un autre
  Exemple: si ils peuvent avoir (1,0,1) ils ne vont pas jouer pour avoir (0,0,1)
- Est-ce que 0 gagne contre toute stratégie pareto-optimale de 1, 2, 3 ?



Merci pour votre attention !

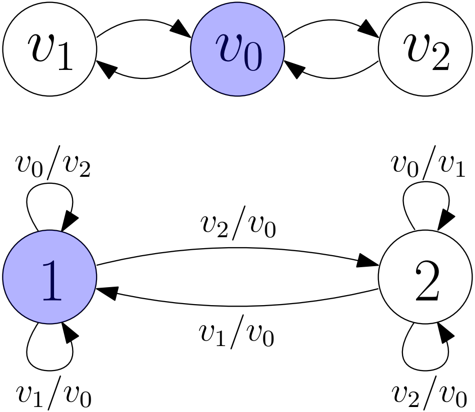
[1] M. Jurdziński.
Small progress measures for solving parity games.
In H. Reichel and S. Tison, editors, *Proc. 17th Ann. Symp. on Theoretical Aspects of Computer Science, STACS'00*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.

[2] M. Jurdziński, M. Paterson, and U. Zwick.
A deterministic subexponential algorithm for solving parity games.
In *Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithm, SODA'06*, pages 117–123. ACM, 2006.

[3] S. Schewe.
Solving parity games in big steps.
In *Proc. of the 27th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'07*, volume 4855, pages 449–460. Springer, 2007.

[4] W. Zielonka.
Infinite games on finitely coloured graphs with applications to automata on infinite trees.
Theor. Comput. Sci., 200(1-2):135–183, 1998.

Stratégie à mémoire finie



Partie $\pi = v_0\ v_2\ v_0\ v_1\ v_0\ \ldots$

Mémoire $\rho =\ 1\ 1\ 2\ 2\ 1\ \ldots$