Université de Mons
Faculté des Sciences
Département d'Informatique
Service d'Informatique Théorique

Résolution de jeux de sûreté joués sur graphes

Directeur : M^me Véronique BRUYÈRE

Mémoire réalisé par
Florent HUYLENBROECK

Rapporteurs : M^r Prénom NOM
M^r Prénom NOM

en vue de l'obtention du grade de
Master en Sciences Informatiques

Année académique 2021-2022

Remerciements

Nous remercions ...

Table des matières

Chapitre 1

Introduction

Contexte

Définition du problème

Le probleme dumodel-checking consiste a vérifier qu'un système informatique satisfait une spécification quand ceux-ci sont donnes sous la forme de modèles mathématiques. Des spècifications typiques sont : est- ce que le systeme peut atteindre unétat de deadlock ? Est-ce qu'une requète reçoit toujours une reponse ? Les modeles utilisés peuvent varier : les comportements du système informatique peuventétre modelisè par un automate acceptant des mots infinis, tandis que la specification peutètre modelisèe par une formule de logique temporelle LTL. Plutot que de verifier qu'un système informatique satisfait une spécification, on peut aller plus loin en envisageant la synthese de controleur. Dans le but de definir les interactions d'un système informatique avec son environnement, on considere ici un graphe orienté dont les sommets sont partagès entre le système et l'environnement. Une interaction est alors un chemin infini dans le graphe tel qu'en tout sommet systeme (resp. de l'environnement), c'est lui qui décide quel arc suivrea partir de ce sommet. L'objectif du systeme est, par exemple, d'éviter unètat de deadlock quoique fasse l'environnement, ou encore de repondrea une requéte quoique fasse l'environnement. Pour y arriver, il a besoin d'une strategie gagnante qui n'est rien d'autre qu'un programme de controle. La synthese de controleur revient donca construire (quand c'est possible) une strategie gagnante (contre l'environnement) pour un objectif donnè du système. Dans ce projet, on propose d'etudier ce problème de synthése pour desjeux de surete´ joues sur graphes. Pour ces jeux, le systeme a pour objectif d'éviter de passer par certains sommets du graphe.

Présentation et limitations des solutions existantes

Quand le graphe est fini, il existe des algorithmes simples qui indiquent si le systeme peut y parvenir et qui dans ce cas indiquent comment jouer (voir par exemple le livre [1]). Quand le graphe est infini, l'article [2] propose algorithme partiel qui utilise des SAT solveurs.

Objectif du travail et idées principales

Dans le cadre du projet, l'etudiant sera amenè´ a comprendre ces algorithmes et a reproduire les expérimentations de l'article [2], et d'envisager une implèmentation de calcul de strategie gracea la structure de données "binary decision diagrams" [3,4]

Brève description du contenu, chapitre par chapitre

Chapitre 2

Jeux joués sur graphes

Avant de proposer des méthodes de résolution pour les jeux de sûreté, nous allons d'abord introduire les notions liées aux jeux à duration infinie, joués par deux joueurs sur un graphe, les stratégies et nous intéresser à la condition de victoire pour deux types de jeu en particulier : les jeux de sûreté et les jeux d'atteignabilité.

2.1 Arènes

Une *arène* est un graphe $A = (V_0, V_1, E)$ composé de deux ensembles disjoints, non-vides de sommets V_0 et V_1, avec $V_0 \cup V_1 = V$, et d'un ensemble d'arcs $E \subseteq V \times V$. De plus, chaque sommet d'une arène doit posséder au moins un successeur.

La figure 2.1 représente une arène. Les sommets carrés appartiennent à V_0 et les sommets ronds à V_1. L'ensemble E correspond aux flèches reliant ces sommets.

2.2 Coups, parties et objectifs

Au début de la partie, un *pion* est placé sur un sommet du graphe. Ce pion est un marquage d'un sommet de l'arène qui va être modifié tour à tour par les joueurs. Dans la suite de ce rapport, *déplacer le pion* référera à l'action de retirer le marquage du sommet courant, et de marquer un autre sommet du graphe.
Le jeu est joué par deux *joueurs* numérotés j_0 et j_1 qui déplacent le pion le long en suivant les arcs de l'arène. Le joueur qui déplace le pion est le joueur à qui appartient le sommet courant où se trouve le pion. Un *coup* est l'action d'un joueur de déplacer le pion le long d'un arc du graphe, depuis un sommet lui appartenant,

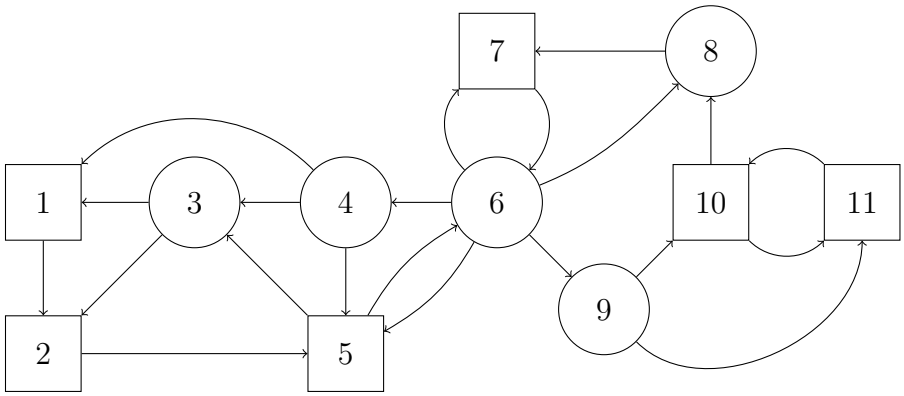


FIGURE 2.1 – Exemple d'une arène

vers un de ses successeurs.
Une séquence infinie de coups forme une *partie*.

Définition 2.2.1. *Une* partie *d'un jeu est une séquence infinie* $\pi = v_0 v_1 ...$ *où* $\forall i \in \mathbb{N}, v_i \in V$, *et* $(v_i, v_{i+1}) \in E$.

L'*objectif* Ω d'un jeu joué sur graphes est ce qui va définir la condition de victoire des joueurs. Dans ce rapport nous allons nous intéresser à deux types de jeux, les jeux de sûreté et les jeux d'atteignabilité. L'objectif pour ces deux types de jeux est défini par un sous ensemble $F \subseteq V$.

2.2.1 Jeux de sûreté

Un jeu de sûreté est défini par un tuple $\mathfrak{G} = (A, F)$ avec A une arène et $F \subseteq V$ un sous ensemble de sommets dits *sûrs*. L'objectif d'un jeu de sûreté est que tous les sommets visités lors de la partie soient des sommets sûrs.

Définition 2.2.2. *Soit* $\mathfrak{G} = (A, F)$ *un jeu de sûreté, une partie* $\pi = v_0 v_1 ...$ *est gagnante pour le joueur* j_0 *si* $\forall i \in \mathbb{N}, v_i \in F$

2.2.2 Jeux d'atteignabilité

Un jeu d'atteignabilité est défini par un tuple $\mathfrak{G} = (A, F)$ avec A une arène et $F \subseteq V$ un sous ensemble de sommets *à atteindre*. L'objectif d'un jeu de sûreté est qu'au moins un sommet de F soit visité au cours de la partie.

Définition 2.2.3. *Soit* $\mathfrak{G} = (A, F)$ *un jeu d'atteignabilité, une partie* $\pi = v_0 v_1 ...$ *est gagnante pour le joueur* j_0 *si* $\exists i \in \mathbb{N}, v_i \in F$

Les jeux d'atteignabilité sont en dualité avec les jeux de sûreté.

2.3 Stratégies et ensembles gagnants

Les coups des joueurs sont décidés par la *stratégie* adoptée par ces derniers. Une stratégie est une fonction $s_{j_n} : V^*V_n \to V$ qui indique vers quel sommet le joueur j_n va déplacer le pion depuis le sommet courant $v \in V_n$ selon la séquence de déplacements précédents.
Une stratégie peut être *sans mémoire* $s_{j_n} : V_n \to V$ si elle ne prend en compte que le sommet actuel où se trouve le pion.

Une stratégie s_{j_n} est *gagnante* pour le joueur j_n si toutes les parties *jouées selon cette stratégie* mènent à une victoire du joueur j_n.

Définition 2.3.1. *Une partie* $\pi = v_0v_1...$ *est dite* jouée selon une stratégie s_{j_n} *si* $\forall i \in \mathbb{N}, v_{i+1} = s_{j_n}(v_0v_1...v_i)$ *et* $v_i \in V_n$

La notion de stratégie gagnante nous permet de définir un *ensemble gagnant*.

Définition 2.3.2. *Soit* $\mathfrak{G} = (A, F)$ *un jeu, avec* $A = (V_0, V_1, E)$*, l'*ensemble gagnant $W \subseteq V$ *est l'ensemble* $W = \{v \in V \mid j_0$ *possède une stratégie gagnante à partir de* $v\}$

Chapitre 3

Cas fini

Afin de résoudre les jeux de sûreté joués sur des graphes, nous allons distinguer les arènes possédant un nombre fini de sommets de celles en possèdant un nombre infini.

Définition 3.0.1. *Une* arène finie *est une arène $A = (V_0, V_1, E)$ pour laquelle V_0 et V_1 sont des ensembles finis.*

Dans cette section, nous allons définir la notion d'attracteur, et l'appliquer afin de calculer les ensembles gagnantes des arènes finies, et les stratégies gagnantes des deux joueurs. Nous allons aussi proposer un algorithme qui calcule les attracteurs d'une arène finie.

3.1 Résolution via les attracteurs

Une méthode pour calculer les ensembles gagnants des jeux de sûreté et d'atteignabilité se base sur le principe d'attracteur.

Définition 3.1.1. *Soit un jeu $\mathfrak{G} = (A, F)$ avec $A = (V_0, V_1, E)$ une arène finie et F un sous-ensemble de sommets, tel que $F \subseteq V$, soit $i \in \mathbb{N}$, le i^e attracteur pour le joueur j_n est l'ensemble :*
$Attr^i_{j_n} = \{v \in V \mid$ le joueur j_n peut forcer une visite d'un sommet de F depuis v en $\leq i$ déplacements$\}$

Afin de construire cet objet en incrémentant la valeur de i, [1] nous donne la formule de construction par induction suivante :

$$\begin{aligned} Attr^0_{j_n}(F) &= F \\ Attr^{i+1}_{j_n}(F) &= Attr^i_{j_n}(F) \\ &\quad \cup \{v' \in V_n \mid \exists (v, v') \in E : v \in Attr^i_{j_n}(F)\} \\ &\quad \cup \{v' \in V \setminus V_n \mid \forall (v, v') \in E : v \in Attr^i_{j_n}(F)\} \end{aligned} \tag{3.1}$$

L'intuition derrière cette formule est la suivante :
Initialement, le joueur j_n ne peut forcer une visite d'un sommet de F que depuis un sommet de F. On a donc que l'attracteur de départ, $Attr^0_{j_n}(F)$ ne contient que les sommets de F.
Ensuite, en incrémentant le nombre de coups, autrement dit la valeur de i, on va considérer l'ajout des sommets qui sont des prédécesseurs des sommets de l'attracteur courant, car ceux-ci mettront, dans le pire des cas, un coup de plus à atteindre un sommet de F. Donc, si un sommet est prédécesseur d'un sommet de l'attracteur sans en faire partie lui-même, il y a deux possibilités (correspondant aux deux ensembles unis à $Attr^i_{j_n}(F)$ dans la formule 3.1). Soit le sommet appartient à j_n et ce sera à lui de jouer un coup à partir de ce sommet. Il pourra ainsi décider de s'approcher d'un sommet de F. Il ne faut donc qu'un seul successeur dans l'attracteur courant pour être ajouté à l'attracteur suivant. Par contre si le sommet appartient au joueur opposé à j_n, alors il faut s'assurer que peu importe le coup qu'il joue, il se rapproche d'un sommet de F. Il est donc nécessaire que tous les successeurs du sommet soient dans $Attr^i_{j_n}(F)$ pour que le sommet soit ajouté à l'attracteur suivant.

Par cette construction, on obtient une séquence d'attracteurs $Attr^0_{j_n}(F) \subseteq Attr^1_{j_n}(F) \subseteq ...$ laquelle sera fixe à partir d'une certaine itération $k \leq |V|$ vu que V est un ensemble fini et qu'à chaque itération, au moins un sommet de V est ajouté à l'attracteur. On notera $Attr_{j_n}(F) = \bigcup_{i=0}^{|V|} Attr^i_{j_n}(F)$

Théorème 3.1.1. *Pour un jeu d'atteignabilité, cette construction de l'attracteur pour j_0 vers F donnera l'ensemble gagnant de j_0.*

Preuve. On a que $Attr_{j_0}(F) \subseteq W_0$ car
- $\forall v \in Attr^{i+1}_{j_0} \cap V_0, v$ possède un successeur dans $Attr^i_{j_0}$.
- $\forall v \in Attr^{i+1}_{j_0} \cap V_1$, tous les successeurs de v sont dans $Attr^i_{j_0}$.
- $Attr^0_{j_0}(F) \subseteq F$

Donc j_0 peut gagner la partie à partir de tous les sommets de $Attr_{j_0}(F)$. Il lui suffit, à chaque coup depuis un sommet de $Attr^{i+1}_{j_0}(F)$, de déplacer le pion vers un sommet dans $Attr^i_{j_0}(F)$ afin de se rapprocher progressivement de $Attr^0_{j_0}(F) \subseteq F$, ce qui est possible par la manière dont est construit l'attracteur. Cette construction explique aussi que le joueur opposé sera forcé d'en faire autant. Cette stratégie est donc gagnante pour j_0 depuis chaque sommet de l'attracteur.

Pour montrer que $W_0 \subseteq Attr_{j_0}(F)$, il faut montrer que j_0 ne peut pas gagner la partie à partir d'un sommet hors de $Attr_{j_0}(F)$, autrement dit que j_1 peut forcer le pion à rester en dehors de $Attr_{j_0}(F)$ depuis tout sommet hors de $Attr_{j_0}(F)$.
Soit un sommet $v \in V_1 \setminus Attr_{j_0}(F)$, alors v possède au moins un arc (v, v') avec $v' \notin Attr_{j_0}(F)$, sinon on aurait $v \in Attr_{j_0}(F)$. La stratégie gagnante pour j_1 est

donc de déplacer le pion le long de cet arc afin de rester hors de l'attracteur.
Soit un sommet $v \in V_0 \setminus Attr_{j_0}(F)$, alors tous les arcs v mènent vers un sommet hors de $Attr_{j_0}(F)$, sinon on aurait $v \in Attr_{j_0}(F)$. j_0 ne peut donc pas entrer dans l'attracteur depuis ce sommet et ne peut donc pas forcer de visite d'un sommet de F depuis ce sommet. Ces deux cas étant exhaustifs, et ayant montré que j_0 est contraint de rester hors de l'attracteur dans ces deux cas, on a bien que $W_0 \subseteq Attr_{j_0}(F)$
L'inclusion étant vérifiée dans les deux sens, on en déduit $W_0 = Attr_{j_0}(F)$. □

Ainsi nous avons montré que l'on peut construire l'ensemble gagnant W_0 du joueur j_0 pour un jeu d'atteignabilité en utilisant le principe d'attracteur. On obtient aussi immédiatement l'ensemble gagnant $W_1 = V \setminus W_0$ du joueur j_1. Chaque joueur peut gagner la partie à partir de chaque sommet de leur ensemble gagnant respectif en adoptant les stratégies énoncées ci-dessus.
De plus, la dualité entre un jeu d'atteignabilité et un jeu de sûreté nous permet d'énoncer le théorème suivant :

Théorème 3.1.2. *Cette méthode de résolution pour les jeux d'atteignabilité permet aussi de résoudre les jeux de sûreté.*

Preuve. Soit un jeu de sûreté $\mathfrak{G} = (A, F)$ avec A une arène finie. on construit $Attr_{j_1}(V \setminus F)$, autrement dit la liste de sommets depuis lesquels j_1 peut forcer une visite d'un sommet hors de F. En adoptant la même stratégie que j_0 dans un jeu d'atteignabilité, cet attracteur donne l'ensemble gagnant de j_1 pour un jeu de sûreté. De manière analogue, j_0, en adoptant la stratégie du joueur j_1 du jeu d'atteignabilité, ne pourra gagner le jeu de sûreté que depuis les sommets hors de cet attracteur. □

La figure suivante représente, en gris, l'ensemble gagnant du joueur j_0 d'un jeu d'atteignabilité joue sur l'arène de la figure 2.1, avec $F = \{1, 2, 11\}$.

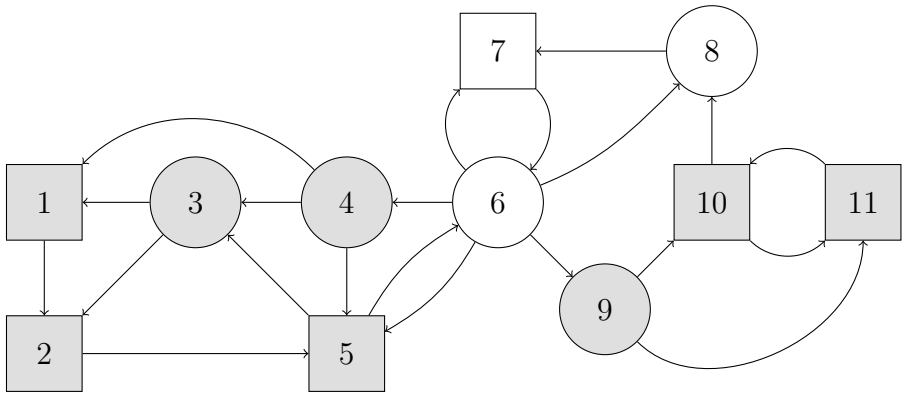


FIGURE 3.1 – Illustration d'un ensemble gagnant

3.2 Algorithme

Nous allons maintenant proposer un algorithme de calcul du i^e attracteur. Pour faciliter la lecture, l'algorithme sera découpé en 3 phases suivies d'explications.

Algorithm 1 Attracteur

Entrées **G** : Graphe, structure de données composée d'un tableau à deux dimensions $predecessors$ de prédécesseurs (la liste de prédécesseurs d'un noeud i est stockée à la i^e entrée du tableau) et une liste $players$ (le i^e noeud appartient au joueur dont le numéro figure en i^e entrée de $players$).
F : Liste de numéros de sommets.
p : Numéro de joueur.
i : Nombre d'itération pour la construction de l'attracteur, une valeur négative calculera l'attracteur complet.

Sortie $Attr_p^i(F)$

```
1: procedure ATTRACTOR(G, F, p, i)
2:     out_degrees ← tableau de taille |G|                ▷ Pré-traitement
3:     for j allant de 0 à |G| − 1 do
4:         if G.players[j] ≠ p then
5:             for pred in G.predecessors[j] do
6:                 out_degrees[pred] ← out_degrees[pred] + 1
7:             end for
8:         end if
9:     end for
```

L'algorithme commence par une phase de pré-traitement au cours de laquelle on va calculer le *demi-degré extérieur* (le nombre d'arcs sortants) de chaque noeud n'appartenant pas au joueur p. Pour cela, on initialise une liste $out\_degrees$ de la taille du nombre de noeuds du graphe. Ensuite, on parcourt le tableau des prédécesseurs du graphe $G.predecessors$. On incrémente l'indice de $out\_degrees$ correspondant à chaque noeud rencontré dans ce tableau car s'il est prédécesseur d'un autre noeud, alors un arc en sort pour aller vers celui-ci.

```
10:     attractor ← tableau de taille |G|                    ▷ Initialisation
11:     for index in F do
12:         attractor[index] ← 1
13:     end for
14:     attractor_new ← F
```

On initialise ensuite le tableau $attractor$ qui va indiquer quels sommets sont marqués comme appartenant à l'attracteur courant. On y marque les sommets de F. Cette étape correspond au calcul de $Attr_p^0(F)$.
On initialise ensuite la liste $attractor\_new$. Cette liste va contenir, après chaque itération de la boucle principale, les sommets qui ont été ajoutés à l'attracteur lors de cette itération. On ajoute initialement les sommets de F à cette liste car $Attr_p^0(F)$ est déjà calculé.

```
15:     while attractor_new non vide and i ≠ 0 do        ▷ Calcul de l'attracteur
16:         to_check ← attractor_new
17:         attractor_new ← [ ]
18:         for index in to_check do
19:             for pred in G.predecessors[index] do
20:                 if attractor[pred] = 0 then
21:                     if G.players[pred] = p then
22:                         attractor_new.append(pred)
23:                     else
24:                         out_degrees[pred] ← out_degrees[pred] − 1
25:                         if out_degrees[pred] = 0 then
26:                             attractor_new.append(pred)
27:                         end if
28:                     end if
29:                 end if
30:             end for
31:         end for
32:         for index in attractor_new do
33:             attractor[index] ← 1
34:         end for
35:         i ← i − 1
36:     end while
37:     return attractor
38: end procedure
```

La première étape de la boucle principale est de copier la liste $attractor\_new$

dans une nouvelle liste to_check afin de garder une trace des nouveaux sommets à traiter. On vide ensuite la liste $attractor\_new$ afin de pouvoir éventuellement y ajouter des nouveaux sommets pendant l'itération courante. La boucle principale de cet algorithme se base ensuite sur la construction par induction de $Attr_{j_n}(F)$. On y retrouve les 3 éléments de l'union qui constitue $Attr^{i+1}_{j_n}(F)$ dans 3.1 :

- $Attr^{i}_{j_n}(F)$ se retrouve aux lignes 32-34. A chaque étape, on ne crée pas un nouvel attracteur mais on marque dans $attractor$ les nouveaux sommets présents dans $attractor\_new$.
- $\{v' \in V_n \mid \exists (v, v') \in E : v \in Attr^{i}_{j_n}(F)\}$. Dans la boucle intérieure, lignes 21-22, si un prédécesseur du noeud en cours de traitement appartient au joueur cible, alors il est ajouté à $attractor\_new$ afin d'être ajouté à l'attracteur.
- $\{v' \in V \setminus V_n \mid \forall (v, v') \in E : v \in Attr^{i}_{j_n}(F)\}$. Dans la boucle intérieure, lignes 23-28, si un prédécesseur du noeud en cours de traitement n'appartient pas à p, alors il est ajouté à l'attracteur si tous ses successeurs sont aussi dans l'attracteur. C'est à cette étape que le pré-traitement joue un rôle. A chaque fois qu'un noeud est rencontré dans la liste des prédécesseurs d'un autre noeud, on décrémente la valeur correspondante dans $out\_degrees$. Si cette valeur atteint 0, cela veut dire que tous les successeurs de ce noeud font partie de l'attracteur (car on ne visite les prédécesseurs d'un noeud que s'il a été ajouté à l'attracteur). On peut donc l'ajouter à son tour à l'attracteur.

Le calcul s'arrête quand aucun noeud n'est ajouté à l'attracteur au cours d'une itération ($attractor\_new$ est vide). Cela veut dire que le point fixe de la séquence d'attracteur $Attr^0_p(F) \subseteq Attr^1_p(F) \subseteq ...$ est atteint et que l'attracteur complet a été calculé.

L'algorithme peut aussi retourner le i^e attracteur si on lui passe en entrée une valeur de i positive (et inférieure au nombre d'itération qu'il faut pour atteindre le point fixe). En effet, i intervient dans le calcul de la condition d'arrêt. Celui-ci est décrémenté à chaque nouvel attracteur calculé. Cependant, la condition d'arrêt ne vérifie qui si $i \neq 0$. Une valeur négative de i en entrée assurera donc le calcul de l'attracteur complet, car celui-ci ne causera pas l'arrêt de la boucle principale.

3.2.1 Complexité

Considérons un graphe G possédant n noeuds et m arcs. Alors l'algorithme *Attractor* possède une complexité dans le pire des cas en $O(n + m)$.

Preuve. Afin de calculer la complexité totale de l'algorithme, intéressons-nous à la complexité des 3 étapes principales :

— *Pré-traitement* (lignes 1-9)
Le calcul du demi-degré extérieur à l'aide d'une structure de données telle que décrite dans l'en-tête de l'algorithme se fait en temps $O(m)$. En effet, il s'agit d'itérer sur la liste de prédécesseurs et, pour chaque noeud rencontré, incrémenter son demi-degré extérieur. Le graphe possédant m arcs, il y aura au plus m éléments dans la liste des prédécesseurs. Le coût pour chacun de ces prédécesseurs étant en $O(1)$, la complexité de cette opération sera en $O(m)$.

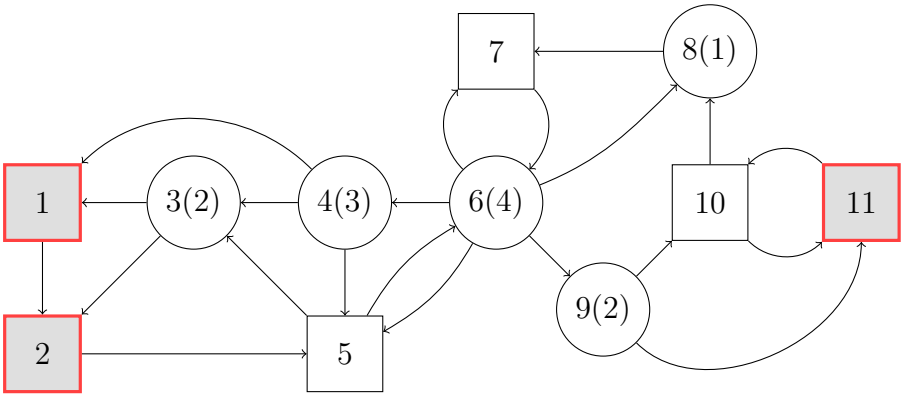
— *Initialisation* (lignes 10-14)
Cette étape se fait en temps $O(n)$ car il y a au plus n noeuds dans le graphe, donc vers lesquels on souhaite construire l'attracteur. Ainsi, au maximum n opérations en $O(1)$ seront effectuées lors de cette étape.

— *Calcul de l'attracteur*
Considérons une valeur de i négative pour le pire des cas.
L'étape du calcul de l'attracteur agit comme un parcours du graphe. En effet, une fois un noeud visité et marqué comme appartenant à l'attracteur, ce noeud ne sera plus visité. Chaque noeud n'est donc visité qu'une seule fois et, dans le pire des cas, tous les noeuds du graphe seront visités.
La condition d'arrêt (ligne 15) est vérifiée au maximum n fois (une fois pour chaque sommet, car il faut au minimum un sommet dans $attractor\_new$ pour satisfaire la condition et, dans le pire des cas, tous les sommets seront ajoutés un par un à l'attracteur). Le coût de cette évaluation est $O(1)$.
Pour chaque sommet traité, on effectue une action en $O(1)$ (ligne 19) afin de récupérer la liste des ses prédécesseurs dans la structure de données G, une autre opération en $O(1)$ (lignes 32-34) afin de marquer ce sommet dans l'attracteur. En comptant aussi la condition d'arrêt et l'opération arithmétique (ligne 35), le coût total par sommet est donc $O(1)$.
Ensuite, pour chaque prédécesseur (autrement dit pour chaque arc), l'algorithme n'effectue que des opérations en $O(1)$: ajouts à une liste (lignes 22 et 26), accès à un tableau via l'indice, comparaisons et opérations arithmétiques (lignes 20, 21, 24 et 25). Le coût total par arc est donc $O(1)$. Nous obtenons un coût total pour le parcours de $n \cdot O(1) + m \cdot O(1) = O(n + m)$.
Il faut ajouter à cela le coût de copier la liste $attractor\_new$ dans to_check. Etant donné que cette liste ne peut contenir qu'une fois chaque sommet à travers toutes les itérations, le coût total des copies vaudra $O(n)$. La complexité de l'étape de calcul est donc $O(n + m + n) = O(n + m)$.

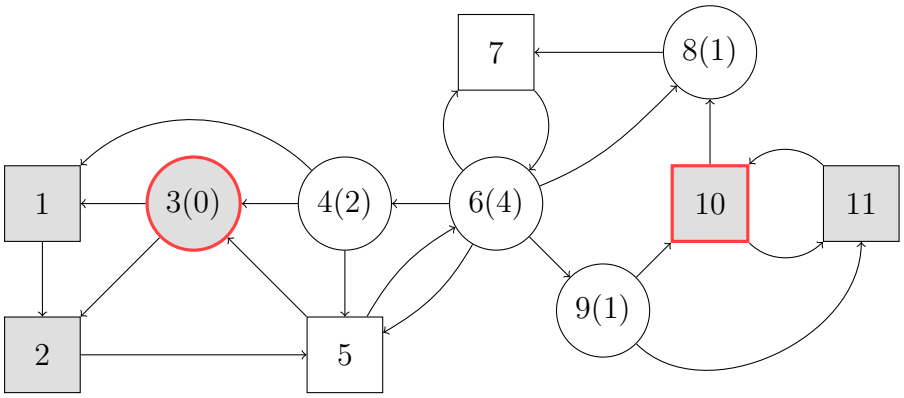
Nous obtenons donc une complexité totale de $O(n+m+(n+m)) = O(2(n+m))$ $= O(n + m)$. □

3.2.2 Exemple

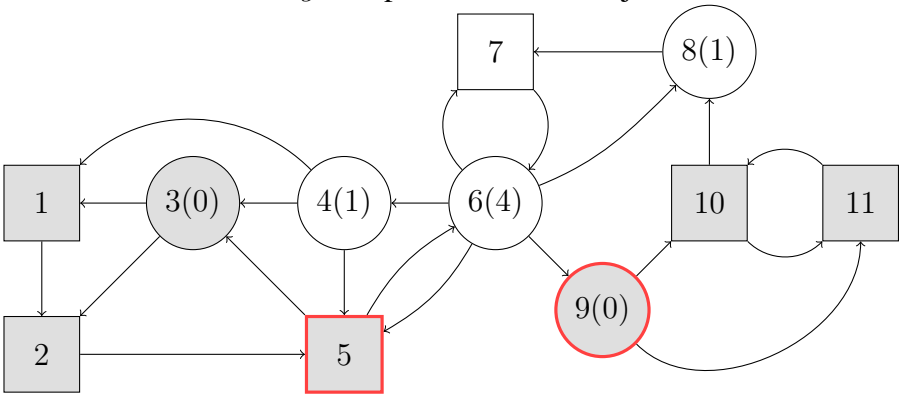
Afin d'illustrer le fonctionnement de l'algorithme, considérons l'arène de la figure 2.1 et calculons l'attracteur pour le joueur j_0 avec $F = \{1, 2, 11\}$, dans le but de calculer les ensembles gagnants des deux joueurs pour un jeu d'atteignabilité. Supposons i négatif afin de calculer l'attracteur complet.
L'étape de pré-traitement sera rendue visuelle en ajoutant aux noeuds de j_1 la valeur correspondant dans le tableau $out\_degrees$. Les noeuds faisant partie de l'attracteur courant $attractor$ seront colorés en gris et ceux étant ajoutés à l'attracteur à l'itération précédente (les noeuds de la liste $attractor\_new$) seront entourés en rouge. Nous obtenons donc, avant l'entrée dans la boucle principale de l'algorithme, la représentation suivante :



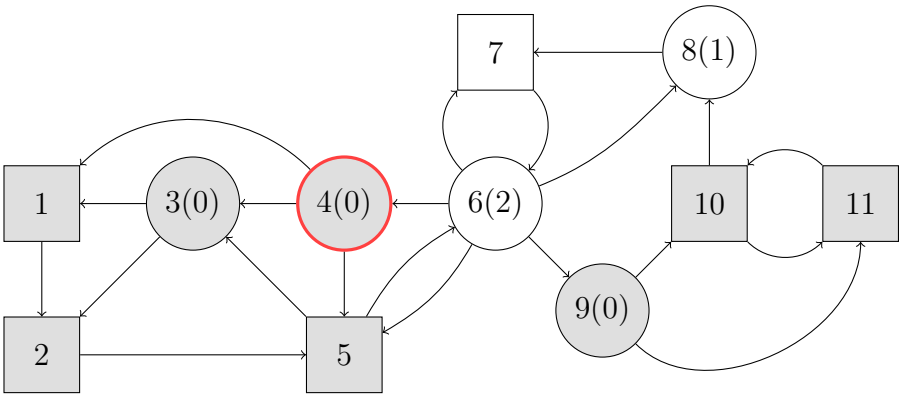
Nous avons donc bien $Attr^0_{j_0}(F) = \{1, 2, 11\}$.
La première itération du calcul de l'attracteur va ajouter les noeuds 3 et 10 à l'attracteur. En effet, la valeur de $out\_degrees$ de 3 va être décrémentée deux fois, une fois en suivant les prédécesseurs de 1 et une autre fois en suivant ceux de 2. Cette valeur atteignant 0, il sera ajouté à l'attracteur. La valeur de $out\_degrees$ de 4 sera aussi décrémentée une fois en partant de 1. Le cas de 10 est plus trivial, il appartient à j_0 et a été rencontré en suivant les prédécesseurs de 11, il est donc ajouté à l'attracteur. La valeur de $out\_degrees$ de 9 est elle aussi décrémentée car le noeud 9 est un prédécesseur de 11. Nous obtenons donc la représentation suivante :



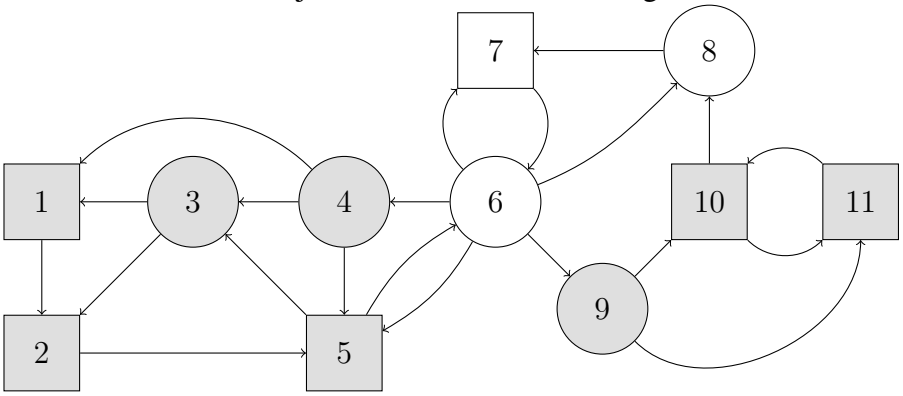
On a $Attr^1_{j_0}(F) = \{1, 2, 3, 10, 11\}$.
5 possède maintenant un successeur dans l'attracteur, il y sera donc ajouté à l'étape suivante. La valeur de $out\_degrees$ de 4 est décrémentée une fois, car il est prédécesseur de 3. Le noeud 9 est atteint une deuxième fois, cette fois depuis 10. Sa valeur de $out\_degrees$ passant à 0, il est ajouté à l'attracteur.



Nous obtenons $Attr^2_{j_0}(F) = \{1, 2, 3, 5, 9, 10, 11\}$
4 est atteint une dernière fois car il est prédécesseur de 5. Sa valeur de out_degree passant à 0, il est ajouté à l'attracteur. Cette même valeur pour 6 est décrémentée deux fois lors de cette itération car il est successeur de 5 et 9.



On obtient $Attr^3_{j_0}(F) = \{1, 2, 3, 4, 5, 9, 10, 11\}$
La valeur de $out\_degrees$ de 6 est décrémentée une fois car il est prédécesseur de 4. Aucun noeud n'est ajouté à $attractor\_new$, l'algorithme s'arrête.



L'attracteur final calculé par l'algorithme est donc $Attr^4_{j_0}(F) = Attr^3_{j_0}(F) = Attr_{j_0}(F) = \{1, 2, 3, 4, 5, 9, 10, 11\}$

Cet attracteur correspond à l'ensemble gagnant W_0 du joueur j_0. Par exemple, si le pion est initialement placé sur le sommet 5, ce sera à j_0 de le déplacer. Selon la stratégie gagnante pour le joueur j_0 décrite dans la section précédente, celui-ci déplacera le pion vers 3. Le coup suivant sera décidé par le joueur j_1. celui-ci n'aura pas le choix et devra déplacer le pion sur un sommet de F : 1 ou 2. Le joueur j_0 gagne la partie.
L'ensemble $W_1 = \{6, 7, 8\}$ est donc l'ensemble gagnant du joueur j_1. Si le pion est initialement placé sur un sommet de cet ensemble, alors j_1 gagne. Par exemple, le pion placé initialement sur 6 sera déplacé par le joueur j_1 vers 7 (en passant éventuellement par 8 selon la stratégie gagnante pour le joueur j_1 car ni 7 ni 8 ne font partie de l'attracteur). Le joueur j_0 déplacera le pion de 7 vers 6 et la partie consistera en une répétition infinie de coups entre 6, 7 et 8.

Chapitre 4

Cas infini

Résoudre les jeux de sûreté joués sur des *arènes infinies* va nécessiter une étape d'abstraction. En effet, un algorithme linéaire en terme d'arcs et de sommets aurait une complexité infinie sur un graphe de taille infinie. Nous allons donc, dans ce chapitre, introduire et utiliser des notions de la théorie des automates afin de produire un problème de taille finie, et ensuite proposer des algorithmes de programmation logique afin de résoudre celui-ci.

Définition 4.0.1. *Une* arène infinie *est une arène $A = (V_0, V_1, E)$ pour laquelle on autorise V_0 et V_1 à être des ensembles infinis.*

4.1 Alphabets, automates et transducteurs

Un *alphabet* Σ est un ensemble fini, non vide, de symboles. Un *mot* de longueur $n \in \mathbb{N}$ sur l'alphabet Σ est une séquence de symboles $u = u_1u_2...u_n$ avec $\forall i \in \mathbb{N}, x_i \in \Sigma$. Le mot vide est noté λ.
La *concaténation* de deux mots $u = u_1u_2...u_n$ et $v = v_1v_2...v_m$ $(n, m \in \mathbb{N})$ est le mot $w = u \cdot v = u_1u_2...u_nv_1v_2...v_m$ de longueur $n + m$. L'ensemble Σ^* contient tous les mots possibles sur l'alphabet Σ. Un sous ensemble $L \subseteq \Sigma^*$ est appelé un langage.

Un *automate* fini est un 5-uple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ où
- Q est un ensemble fini, non vide, d'états.
- Σ est l'alphabet lu par l'automate.
- δ est une fonction $Q \times \Sigma \to Q$ appelée *fonction de transition* de l'automate. L'expression $\delta(q_1, u) = q_2$ indique que l'automate transitionne de l'état q_1 à l'état q_2 quand le mot $u \in \Sigma^*$ est lu en entrée.
- $q_0 \subseteq Q$ est l'ensemble des états initiaux.
- $F \subseteq Q$ est l'ensemble des états finaux.

Définition 4.1.1. *Un automate* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *est* déterministe *si,* $\delta(q, a) = p_1$ *et* $\delta(q, a) = p_2$ *implique que* $p_1 = p_2$.

Un automate fini qui n'est pas déterministe est appelé *non déterministe*.

Un *parcours* d'un automate par un mot $u = u_1u_2...u_n$, $n \in \mathbb{N}$ est une séquence d'états $q_0q_1...q_n$ telle que $\forall i \in \mathbb{N}, \delta(q_i, u_i) = q_{i+1}$. Un mot u est *accepté* par un automate $\mathcal{A}$ si le parcours de l'automate par ce mot se termine sur un état final $q \in F$. L'ensemble des mots acceptés par un automate forme le langage $\mathcal{L}(\mathcal{A})$ de cet automate. Un langage L est dit *régulier* s'il existe un automate fini $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = L$.

Enfin, un *transducteur* est un automate fini non déterministe $\mathcal{T} = (Q, \hat{\Sigma}, \delta, q_0, F)$ où $\hat{\Sigma} = (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})$ avec Σ et Γ deux alphabets. Un parcours d'un transducteur ne se fait pas par un mot mais par une paire de mots (u, v) avec $u \in \Sigma^*$ et $v \in \Gamma^*$ afin de produire une séquence $q_0q_1...q_n$, $n \in \mathbb{N}$ telle que $\forall i \in \mathbb{N}, \delta(q_i, (u_i, v_i)) = q_{i+1}$. Les mots u et v peuvent être de longueur différente car λ est inclu aux deux alphabets.
Un transducteur, en acceptant des paires de mots, permet de définir une *relation* R entre deux langages. L'ensemble des paires de mots acceptées par un transducteur forme la relation $\mathcal{R}$. Pour finir, une relation est dite *régulière* s'il exite un transducteur $\mathcal{T}$ tel que $\mathcal{R}(\mathcal{T}) = R$.

4.2 Jeux de sûreté rationnels

Un jeu de sûreté rationnel est une représentation symbolique d'un jeu de sûreté, basée sur les langages rationnels et les relations rationnelles.

4.3 Apprentissage

4.4 Résolution par la programmation logique

4.5 Algorithmes

4.5.1 Complexité

Conclusion

Mettez votre conclusion ici. Dressez le bilan de votre travail effectué, en prenant du recul. Discuter de si vous avez bien réussi les objectifs du travail ou non. Présentez les perspectives futurs.

Bibliographie

[1] W. Thomas. Church's problem and a tour through automata theory. Master's thesis, RWTH Aachen, Lehrstuhl Informatik 7, 52056 Aachen, Germany, 2008.

Annexe A

Première annexe

Annexe B

Deuxième annexe