

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

## **Résolution de jeux de sûreté joués sur graphes**

Directeur : M<sup>me</sup> Véronique BRUYÈRE

Mémoire réalisé par  
Florent HUYLENBROECK

Rapporteurs : M<sup>r</sup> Prénom NOM  
M<sup>r</sup> Prénom NOM

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



Année académique 2021-2022

# Remerciements

Nous remercions ...

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Jeux joués sur graphes</b>	<b>3</b>
2.1	Arènes . . . . .	3
2.2	Coups, parties et objectifs . . . . .	3
2.2.1	Jeux de sûreté . . . . .	4
2.2.2	Jeux d'atteignabilité . . . . .	4
2.3	Stratégies et ensembles gagnants . . . . .	5
<b>3</b>	<b>Cas fini</b>	<b>6</b>
3.1	Résolution via les attracteurs . . . . .	6
3.2	Algorithme . . . . .	10
3.2.1	Complexité . . . . .	12
3.2.2	Exemple . . . . .	14
<b>4</b>	<b>Cas infini</b>	<b>17</b>
4.1	Alphabets, automates et transducteurs . . . . .	17
4.2	Exemple de jeu de sûreté infini . . . . .	18
4.3	Jeux de sûreté rationnels . . . . .	19
4.4	Apprentissage . . . . .	20
4.5	Résolution par la programmation logique . . . . .	21
4.6	Algorithmes . . . . .	21
4.6.1	Complexité . . . . .	21
	<b>Conclusion</b>	<b>22</b>
	<b>Annexes</b>	<b>24</b>
<b>A</b>	<b>Première annexe</b>	<b>24</b>
<b>B</b>	<b>Deuxième annexe</b>	<b>25</b>

# Chapitre 1

## Introduction

### Contexte

#### Définition du problème

Le problème du model-checking consiste à vérifier qu'un système informatique satisfait une spécification quand ceux-ci sont donnés sous la forme de modèles mathématiques. Des spécifications typiques sont : est-ce que le système peut atteindre un état de deadlock ? Est-ce qu'une requête reçoit toujours une réponse ? Les modèles utilisés peuvent varier : les comportements du système informatique peuvent être modélisés par un automate acceptant des mots infinis, tandis que la spécification peut être modélisée par une formule de logique temporelle LTL. Plutôt que de vérifier qu'un système informatique satisfait une spécification, on peut aller plus loin en envisageant la synthèse de contrôleur. Dans le but de définir les interactions d'un système informatique avec son environnement, on considère ici un graphe orienté dont les sommets sont partagés entre le système et l'environnement. Une interaction est alors un chemin infini dans le graphe tel qu'en tout sommet système (resp. de l'environnement), c'est lui qui décide quel arc suivra à partir de ce sommet. L'objectif du système est, par exemple, d'éviter un état de deadlock quoique fasse l'environnement, ou encore de répondre à une requête quoique fasse l'environnement. Pour y arriver, il a besoin d'une stratégie gagnante qui n'est rien d'autre qu'un programme de contrôle. La synthèse de contrôleur revient donc à construire (quand c'est possible) une stratégie gagnante (contre l'environnement) pour un objectif donné du système. Dans ce projet, on propose d'étudier ce problème de synthèse pour des jeux de sécurité joués sur graphes. Pour ces jeux, le système 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 système 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 un algorithme partiel qui utilise des SAT solveurs.

**Objectif du travail et idées principales**

Dans le cadre du projet, l'étudiant sera amené à comprendre ces algorithmes et à reproduire les expérimentations de l'article [2], et d'envisager une implémentation de calcul de stratégie grâce à 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* réfèrera 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 des 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, vers un de

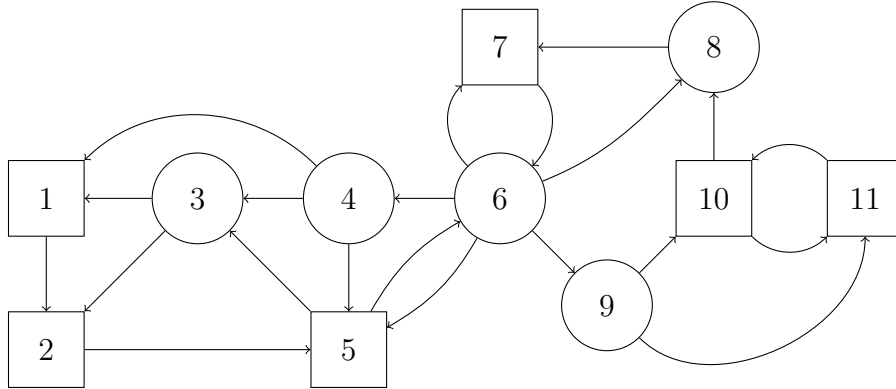


FIGURE 2.1 – Exemple d'une arène

ses successeurs.

Une séquence infinie de coups forme une *partie*.

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

L'objectif  $\Omega$  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  $F \subseteq V$ . Dans ces jeux, les deux joueurs jouent selon des objectifs opposés.

### 2.2.1 Jeux de sûreté

Un jeu de sûreté est défini par un tuple  $\mathfrak{G} = (A, I, F)$  avec  $A$  une arène,  $I \subseteq F$  un ensemble de sommets de départ et  $F \subseteq V$  un sous ensemble de sommets dits *sûrs*. L'objectif pour le joueur  $j_0$  d'un jeu de sûreté est que tous les sommets visités lors de la partie soient des sommets sûrs. L'objectif pour le joueur  $j_1$  est donc qu'au moins un sommet hors de  $F$  soit visité.

**Définition 2.2.2.** Soit  $\mathfrak{G} = (A, I, F)$  un jeu de sûreté, une partie  $\pi = v_0 v_1 \dots$  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  $\mathfrak{G} = (A, I, F)$  avec  $A$  une arène,  $I \subseteq V \setminus F$  un ensemble de sommets de départ et  $F \subseteq V$  un sous ensemble de sommets à *atteindre*. L'objectif pour le joueur  $j_0$  d'un jeu de sûreté est qu'au

moins un sommet de  $F$  soit visité au cours de la partie. L'objectif du joueur  $j_1$  est que seuls des sommets hors de  $F$  soient visités.

**Définition 2.2.3.** Soit  $\mathfrak{G} = (A, I, F)$  un jeu d'atteignabilité, une partie  $\pi = v_0v_1\dots$  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. En effet, on remarque que l'objectif du joueur  $j_0$  (resp.  $j_1$ ) d'un jeu de sûreté est similaire à celui du joueur  $j_1$  (resp.  $j_0$ ) pour un jeu d'atteignabilité.

## 2.3 Stratégies et ensembles gagnants

Les coups des joueurs sont décidés par la *stratégie* adoptée par ces derniers. Une stratégie est une fonction  $s_{j_n} : V^*V_n \rightarrow V$  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* (ou *positionnelle*)  $s_{j_n} : V_n \rightarrow V$  si elle ne prend en compte que le sommet actuel où se trouve le pion.

Une stratégie est gagnante à partir d'un sommet  $v$  si une partie jouée selon cette stratégie

**Définition 2.3.1.** Une partie  $\pi = v_0v_1\dots$  est dite jouée selon une stratégie  $s_{j_n}$  (à partir d'un sommet  $v_0 \in V$ ) si  $\forall i \in \mathbb{N}, v_i \in V_n, v_{i+1} = s_{j_n}(v_0v_1\dots v_i)$

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

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



## 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 possédant 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 gagnants 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  $\mathfrak{G} = (A, I, F)$  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  $i^{\text{e}}$  attracteur pour le joueur  $j_n$  est l'ensemble :

$\text{Attr}_{j_n}^i(F) = \{v \in V \mid \text{le joueur } j_n \text{ peut forcer une visite d'un sommet de } F \text{ depuis } v \text{ en } \leq i \text{ déplacements}\}$

Afin de construire cet objet en incrémentant la valeur de  $i$ , [2] nous donne la formule de construction par induction suivante :

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

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,  $Attr_{j_n}^0(F)$  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  $Attr_{j_n}^i(F)$  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  $Attr_{j_n}^i(F)$  pour que le sommet soit ajouté à l'attracteur suivant.

Par cette construction, on obtient une séquence d'attracteurs  $Attr_{j_n}^0(F) \subseteq Attr_{j_n}^1(F) \subseteq \dots$  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  $Attr_{j_n}(F) = \bigcup_{i=0}^{|V|} Attr_{j_n}^i(F)$

**Théorème 3.1.1.** *Pour un jeu d'atteignabilité, cette construction de l'attracteur pour  $j_0$  vers  $F$  donnera l'ensemble gagnant  $W_0$  de  $j_0$ , ainsi que la stratégie gagnante positionnelle des deux joueurs.*

**Preuve.** On a que  $Attr_{j_0}(F) \subseteq W_0$  car

- $\forall v \in Attr_{j_0}^{i+1} \cap V_0, v$  possède un successeur dans  $Attr_{j_0}^i$ .
- $\forall v \in Attr_{j_0}^{i+1} \cap V_1$ , tous les successeurs de  $v$  sont dans  $Attr_{j_0}^i$ .
- $Attr_{j_0}^0(F) \subseteq F$

Donc  $j_0$  peut gagner la partie à partir de tous les sommets de  $Attr_{j_0}(F)$ . Sa stratégie gagnante est, pour chaque coup depuis un sommet de  $Attr_{j_0}^{i+1}(F)$ , de déplacer le pion vers un sommet dans  $Attr_{j_0}^i(F)$  afin de se rapprocher progressivement de  $Attr_{j_0}^0(F) \subseteq F$ , 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  $j_1$  dans ce cas est 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. La stratégie gagnante de  $j_1$  dans ce cas est de suivre n'importe lequel de ces arcs. 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  $W_0 = Attr_{j_0}(F)$ .  $\square$

Ces deux cas exhaustifs nous permettent aussi de déduire la stratégie gagnante du joueur  $j_1$ . A chaque coup, il doit déplacer le pion vers un sommet hors de l'attracteur.

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.

Ainsi, un jeu de sûreté joué selon ces stratégies sera toujours gagnant pour le joueur  $j_0$  si  $I \subseteq W_0$ .

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.  $\square$

La figure 3.1 représente, en gris, l'ensemble gagnant du joueur  $j_0$  d'un jeu d'atteignabilité joué sur l'arène de la figure 2.1, avec  $F = \{1, 2, 11\}$ .

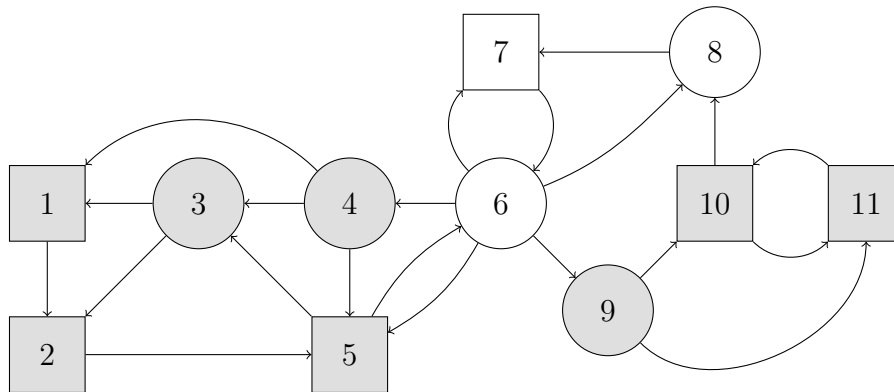


FIGURE 3.1 – Illustration d'un ensemble gagnant

## 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:    $out\_degrees \leftarrow$  tableau de taille  $|G|$  ▷ Pré-traitement
3:   for  $j$  allant de 0 à  $|G| - 1$  do
4:     if  $G.players[j] \neq p$  then
5:       for  $pred$  in  $G.predecessors[j]$  do
6:          $out\_degrees[pred] \leftarrow out\_degrees[pred] + 1$ 
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  $\leftarrow$  tableau de taille  $|G|$  ▷ Initialisation
11:   for index in F do
12:     attractor[index]  $\leftarrow$  1
13:   end for
14:   attractor_new  $\leftarrow$  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  $\neq$  0 do ▷ Calcul de l'attracteur
16:     to_check  $\leftarrow$  attractor_new
17:     attractor_new  $\leftarrow$  []
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]  $\leftarrow$  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]  $\leftarrow$  1
34:     end for
35:     i  $\leftarrow$  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  $Attr_{j_n}^{i+1}(F)$  dans 3.1 :

- $Attr_{j_n}^i(F)$  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*.
- $\{v' \in V_n \mid \exists(v, v') \in E : v \in Attr_{j_n}^i(F)\}$ . 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.
- $\{v' \in V \setminus V_n \mid \forall(v, v') \in E : v \in Attr_{j_n}^i(F)\}$ . 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  $Attr_p^0(F) \subseteq Attr_p^1(F) \subseteq \dots$  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 décrémente à chaque nouvel attracteur calculé. Cependant, la condition d'arrêt ne vérifie que 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  $O(n + m + n) = O(n + m)$ .

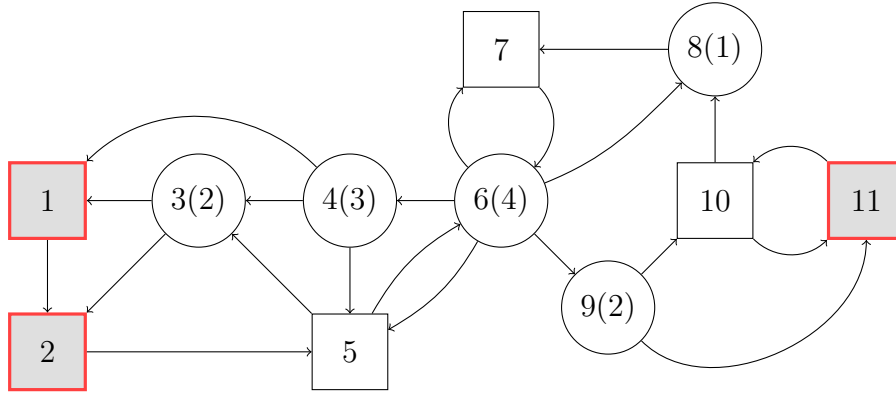
Nous obtenons donc une complexité totale de  $O(n + m + (n + m)) = O(2(n + m)) = O(n + m)$ .  $\square$



### 3.2.2 Exemple

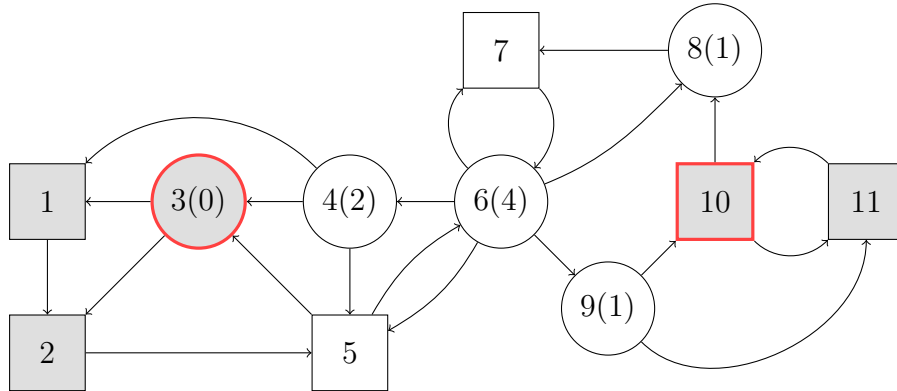
Afin d'illustrer le fonctionnement de l'algorithme, considérons l'arène de la figure 2.1 et calculons l'attracteur pour le joueur  $j_0$  avec  $F = \{1, 2, 11\}$ , dans le but de calculer les ensembles gagnants des deux joueurs pour un jeu d'atteignabilité. Supposons  $i$  négatif afin de calculer l'attracteur 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 :



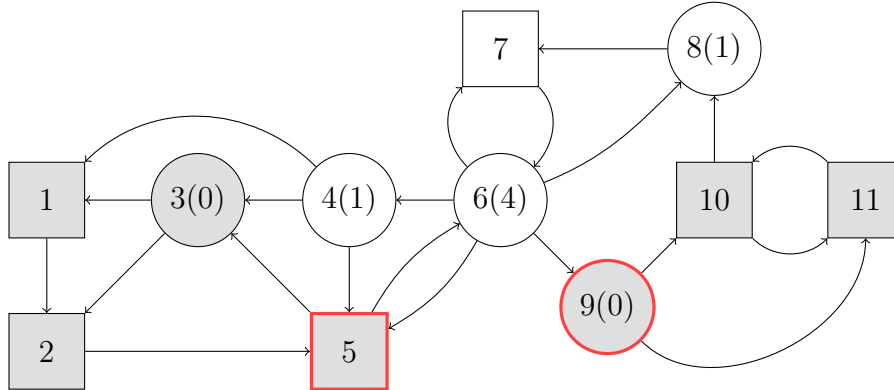
Nous avons donc bien  $Attr_{j_0}^0(F) = \{1, 2, 11\}$ .

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 :



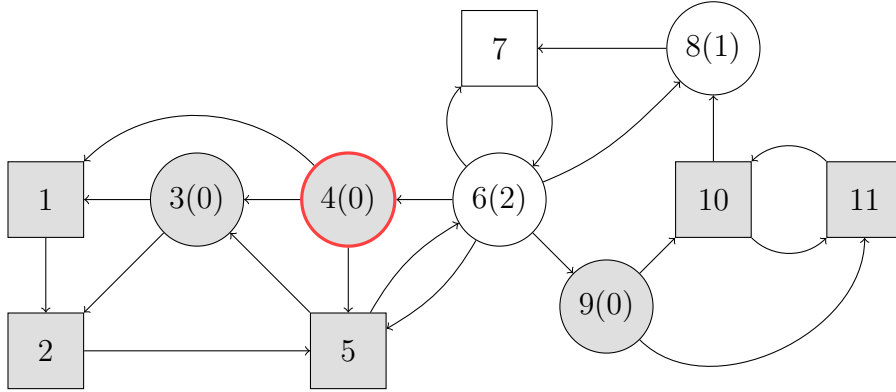
On a  $Attr_{j_0}^1(F) = \{1, 2, 3, 10, 11\}$ .

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.



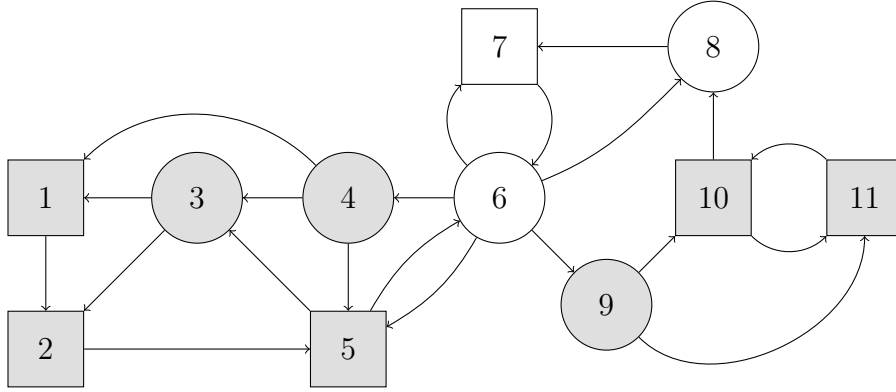
Nous obtenons  $Attr_{j_0}^2(F) = \{1, 2, 3, 5, 9, 10, 11\}$

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  $Attr_{j_0}^3(F) = \{1, 2, 3, 4, 5, 9, 10, 11\}$

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  $Attr_{j_0}^4(F) = Attr_{j_0}^3(F) = Attr_{j_0}(F) = \{1, 2, 3, 4, 5, 9, 10, 11\}$

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*  $\Sigma$  est un ensemble fini, non vide, de symboles. Un *mot* de longueur  $n \in \mathbb{N}$  sur l'alphabet  $\Sigma$  est une séquence de symboles  $u = u_1u_2\dots u_n$  avec  $\forall i \in \mathbb{N}, x_i \in \Sigma$ . Le mot vide est noté  $\lambda$ .

La *concaténation* de deux mots  $u = u_1u_2\dots u_n$  et  $v = v_1v_2\dots v_m$  ( $n, m \in \mathbb{N}$ ) est le mot  $w = u \cdot v = u_1u_2\dots u_nv_1v_2\dots v_m$  de longueur  $n + m$ . L'ensemble  $\Sigma^*$  contient tous les mots possibles sur l'alphabet  $\Sigma$ . 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.
- $\Sigma$  est l'alphabet lu par l'automate.
- $\delta$  est une fonction  $Q \times \Sigma \rightarrow Q$  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  $u = u_1u_2\dots u_n$ ,  $n \in \mathbb{N}$  est une séquence d'états  $q_0q_1\dots q_n$  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  $\Sigma$  et  $\Gamma$  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  $q_0q_1\dots q_n$ ,  $n \in \mathbb{N}$  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  $\lambda$  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 existe un transducteur  $\mathcal{T}$  tel que  $\mathcal{R}(\mathcal{T}) = R$ .

## 4.2 Exemple de jeu de sûreté infini

Afin d'illustrer le cas infini, nous allons considérer l'arène infinie simple représentée à la figure 4.1. Il s'agit d'un tableau infini, à une dimension, fermé à gauche. Chaque sommet est décrit par le numéro du joueur à qui il appartient, et un nombre  $k \in \mathbb{N}$ . Il existe un sommet appartenant à chaque joueur pour chaque valeur possible de  $k$  et ceux-ci sont arrangés dans l'ordre croissant de cette valeur, en alternant les sommets de  $j_0$  et  $j_1$ . L'ensemble  $F$  est représenté par les sommets grisés. Il est décrit par une certaine valeur non nulle de  $k$ , à partir de laquelle tous les sommets font partie de  $F$  (à la figure 4.1,  $k = 2$ ).

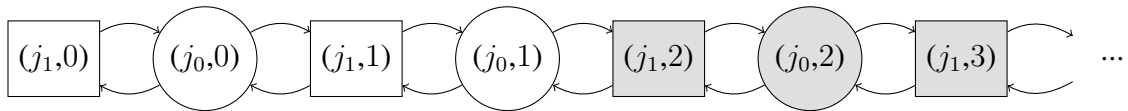


FIGURE 4.1 – Exemple d'arène infinie

Nous appellerons le jeu joué sur cette arène  $\mathfrak{G}_k$ . Les règles du jeu sont différentes pour cet exemple. Les joueurs bougent le pion chacun à leur tour. Le joueur  $j_0$  ne peut déplacer le pion que vers la droite ou ne pas le déplacer du tout, et le joueur  $j_1$  ne peut le déplacer que vers la gauche (s'il n'est pas arrivé au bout de l'arène) ou ne pas le déplacer. Au début de la partie, le pion est placé sur un sommet appartenant à  $F$ , et le joueur  $j_0$  commence. Les objectifs des joueurs restent les mêmes,  $j_0$  veut rester dans  $F$  tandis que  $j_1$  essaie d'en sortir. Cet exemple est simple car le joueur  $j_0$  peut gagner, peu importe  $F$ , en choisissant de bouger le pion à droite à chaque coup. Il reste néanmoins intéressant à résoudre.

### 4.3 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.

**Définition 4.3.1.** Une arène rationnelle sur l'alphabet  $\Sigma$  est une arène  $A_\Sigma = (V_0, V_1, E)$  où  $V_0, V_1$  sont des langages réguliers et  $E$  est une relation régulière.

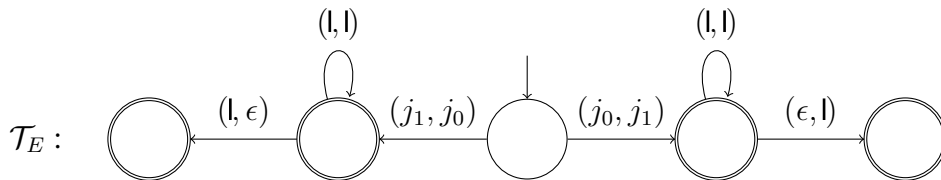
**Définition 4.3.2.** Un jeu de sûreté rationnel sur l'alphabet  $\Sigma$  est un jeu  $\mathfrak{G}_\Sigma = (A_\Sigma, I, F)$  où  $A_\Sigma$  est une arène rationnelle sur  $\Sigma$  et  $I, F \subseteq \Sigma^*$  sont des langages réguliers.

Nous allons maintenant traduire le jeu  $\mathfrak{G}_k$  en un jeu de sûreté rationnel. Pour cela, nous devons proposer un encodage qui fait correspondre à chaque arc et chaque sommet un mot unique d'un alphabet. Pour cela, nous allons considérer l'alphabet  $\Sigma = \{j_0, j_1, l\}$  afin d'associer le sommet  $(x, i) \in \{j_0, j_1\} \times \mathbb{N}$  au mot  $xl^i$  où  $l$  est l'encodage de  $i$  en unaire.

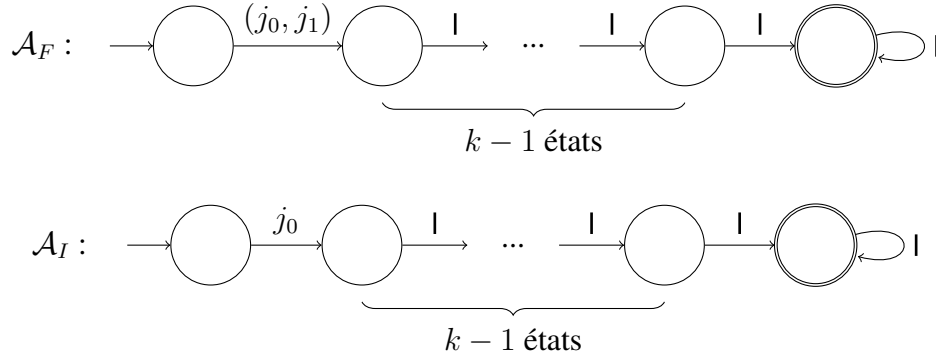
Ainsi, les automates suivants représentent les ensembles  $V_0$  et  $V_1$  :



Les arcs (l'ensemble  $E$ ) sont représentés par le transducteur suivant



Et pour finir, les automates  $\mathcal{A}_F$  et  $\mathcal{A}_I$  représentent les ensembles  $F$  et  $I$ .



## 4.4 Apprentissage

Résoudre un jeu de sûreté rationnel consiste à produire un automate fini déterministe (AFD) qui acceptera tous les mots représentant les sommets de l'ensemble gagnant  $W_0$ . Afin de produire un tel automate, nous allons utiliser le principe d'*apprentissage*. Cet apprentissage a lieu entre un *enseignant*, lequel est en connaissance du jeu de sûreté que l'on veut résoudre, et un *élève*, cherchant à en apprendre l'AFD décrivant l'ensemble gagnant. On suppose que l'alphabet utilisé est connu par les deux partis avant que l'apprentissage commence.

Avant de commencer l'apprentissage, [1] suggère d'apporter quelques précisions à la définition 2.3.2 d'ensemble gagnant.

**Définition 4.4.1.** Soit  $\mathfrak{G} = (A, I, F)$  un jeu, avec  $A = (V_0, V_1, E)$ , l'ensemble gagnant pour le joueur  $j_0$  est l'ensemble  $W \subseteq V$  qui satisfait :

- (1)  $I \subseteq W$
- (2)  $W \subseteq F$
- (3)  $\forall v \in W \cap V_0, \text{Succ}(v) \cap W \neq \emptyset$  (fermeture existentielle)
- (4)  $\forall v \in W \cap V_1, \text{Succ}(v) \subseteq W$  (fermeture universelle)

Où  $\text{Succ}(v)$  est la fonction retournant les successeurs du sommet  $v$ .

L'apprentissage se fait à l'intérieur d'une boucle de type *CEGIS* (Counterexample guided inductive synthesis, ou synthèse inductive guidée par le contre-exemple). A chaque itération de cette boucle, l'élève conjecture une AFD  $\mathcal{C}$  et l'enseignant vérifie à l'aide de la définition 4.4.1 si  $\mathcal{L}(\mathcal{C})$  est un ensemble gagnant. Si  $\mathcal{L}(\mathcal{C})$  est un ensemble gagnant, alors l'enseignant répond "oui" et l'apprentissage s'arrête. Sinon, l'enseignant retourne un *contre-exemple* pour une des règles non respectées de la définition 4.4.1 et l'apprentissage se poursuit dans l'itération suivante. Plus formellement, la définition suivante établit le protocole de vérification par l'enseignant :

**Définition 4.4.2.** *Enseignant pour jeux de sûreté rationnels*

Soit  $\mathfrak{G}_\Sigma = (A_\Sigma, I, F)$ , un jeu de sûreté rationnel, avec  $A_\Sigma = (V_0, V_1, E)$ . Soit un AFD  $\mathcal{C}$ . Un enseignant pour  $\mathfrak{G}_\Sigma$  confronté à  $\mathcal{C}$  va effectuer les vérifications suivantes :

- (1) Si  $I \not\subseteq \mathcal{L}(\mathcal{C})$ , alors l'enseignant retourne un contre-exemple positif  $u \in I \setminus \mathcal{L}(\mathcal{C})$ . Sinon,
- (2) Si  $\mathcal{L}(\mathcal{C}) \not\subseteq F$ , alors l'enseignant retourne un contre-exemple négatif  $u \in \mathcal{L}(\mathcal{C}) \setminus F$ . Sinon,
- (3) Si  $\exists u \in \mathcal{L}(\mathcal{C}) \cap V_0$ ,  $\text{Succ}(u) \cap \mathcal{L}(\mathcal{C}) = \emptyset$  alors l'enseignant choisit  $u$  et retourne un contre-exemple d'implication existentielle  $(u, \mathcal{A}) \in \Sigma^* \times \text{AFN}_\Sigma$  où  $\mathcal{L}(\mathcal{A}) = \text{Succ}(u)$ . Sinon,
- (4) Si  $\exists u \in \mathcal{L}(\mathcal{C}) \cap V_1$ ,  $\text{Succ}(u) \not\subseteq \mathcal{L}(\mathcal{C})$ , alors l'enseignant choisit  $u$  et retourne un contre-exemple d'implication universelle  $(u, \mathcal{A}) \in \Sigma^* \times \text{AFN}_\Sigma$  où  $\mathcal{L}(\mathcal{A}) = \text{Succ}(u)$ .

Où

- $\text{AFN}_\Sigma$  est l'ensemble des automates finis non-déterministes sur  $\Sigma$ .
- Un contre-exemple positif est un mot  $u$  qui devrait être accepté par  $\mathcal{C}$ , mais qui a été rejeté.
- Un contre-exemple négatif est un mot  $u$  qui devrait être rejeté par  $\mathcal{C}$ , mais qui a été accepté.
- Un contre-exemple d'implication existentielle  $(u, \mathcal{A})$  signifie que si  $\mathcal{C}$  accepte  $u$ , alors au moins un mot  $v \in \mathcal{L}(\mathcal{A})$  doit, lui aussi, être accepté, et cela n'a pas été respecté.
- Un contre-exemple d'implication universelle  $(u, \mathcal{A})$  signifie que si  $\mathcal{C}$  accepte  $u$ , alors tous les mots  $v \in \mathcal{L}(\mathcal{A})$  doivent, eux aussi, être acceptés, et cela n'a pas été respecté.

Si les 4 vérifications ne retournent pas de contre-exemples, alors l'enseignant répond "oui".

L'ordre des vérifications n'a pas d'importance.

## 4.5 Résolution par la programmation logique

## 4.6 Algorithmes

### 4.6.1 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] D. Neider. An automaton learning approach to solving safety games over infinite graphs. Technical report, Department of Electrical and Systems Engineering University of Pennsylvania, USA, 2016.
- [2] W. Thomas. Church's problem and a tour through automata theory. Technical report, RWTH Aachen, Lehrstuhl Informatik 7, 52056 Aachen, Germany, 2008.

## **Annexe A**

### **Première annexe**

## **Annexe B**

### **Deuxième annexe**