

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>
<b>3</b>	<b>Cas fini</b>	<b>5</b>
<b>4</b>	<b>Cas infini</b>	<b>13</b>
	<b>Conclusion</b>	<b>14</b>
	<b>Annexes</b>	<b>15</b>
<b>A</b>	<b>Première annexe</b>	<b>15</b>
<b>B</b>	<b>Deuxième annexe</b>	<b>16</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

### Jeux de sûreté

Un *jeu de sûreté* se joue sur une *arène*  $A = (V_0, V_1, E)$  composée 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'arêtes  $E \subseteq V \times V$ . De plus, chaque sommet d'une arène doit posséder au moins un successeur dans l'arène, autre que lui-même.

Un tel jeu est défini par un triplet  $\mathfrak{G} = (A, I, F)$ , avec  $A$  une arène,  $I \subseteq V$  un ensemble de sommets initiaux et  $F \subseteq V$  un ensemble de sommets *sûrs*.

Au début de la partie, un pion est placé sur un sommet de l'ensemble  $I$ .

Le jeu est joué par deux *joueurs* numérotés  $j_0$  et  $j_1$  qui déplacent tour à tour le pion le long des arêtes de l'arène. Une *partie* d'un jeu est une séquence infinie  $\pi = v_0 v_1 \dots$  où  $v_0 \in I, \forall i \in \mathbb{N}, v_i \in V$ , et  $(v_i, v_{i+1}) \in E$ .

Les coups des joueurs sont décidés par la *stratégie* adoptée par ces derniers. Une stratégie est une fonction  $s_{j_n} : V^* V_n \rightarrow V$  ( $n \in \{0, 1\}$ ) qui indique vers quel sommet déplacer le pion selon la séquence de déplacement précédents. Une stratégie  $s_{j_n}$  est *gagnante* si une partie jouée selon cette stratégie satisfait une condition de victoire pour le joueur  $j_n$ . Une partie est dite jouée selon une stratégie  $s_{j_n}$  si  $\forall i \in \mathbb{N}, v_{i+1} = s_{j_n}(v_0 v_1 \dots v_i)$  et  $v_i \in V_n$ .

Une stratégie peut être *sans mémoire*  $s_{j_n} : V_n \rightarrow V$  si elle ne prend en compte que le sommet actuel où se trouve le pion.

La *condition de victoire* d'un jeu de sûreté est que tous les sommets visités au cours de la partie soient de l'ensemble  $F$ . Autrement dit, soit une partie  $\pi = v_0 v_1 \dots, \forall i \in \mathbb{N}, v_i \in F$ .

Calculer une stratégie gagnante pour un joueur revient généralement à calculer un ensemble gagnant  $W \subseteq V$  pour ce joueur.

$$W_n = \{v \in V \mid j_n \text{ gagne à partir de } v\}$$

Cette définition d'un ensemble gagnant donne immédiatement une stratégie pour le joueur  $j_0$ . A chaque tour,  $j_0$  n'a qu'à bouger le pion vers un sommet dans  $W$ .

## Jeux d'atteignabilité

Un *jeu d'atteignabilité* est similaire à un jeu de sûreté  $\mathfrak{G} = (A, I, F)$ , sauf que  $F \subseteq V$  est un ensemble de sommet à atteindre, et donc  $I \subseteq V \setminus F$ . Une partie  $\pi = v_0 v_1 \dots$  est donc gagnante pour le joueur  $j_0$  si  $\exists i \in \mathbb{N}, v_i \in F$ .

## Chapitre 3

### Cas fini

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

#### Résolution via les attracteurs

Une méthode pour calculer les ensembles gagnants des jeux de sûreté se base sur le principe d'attracteur. Intéressons nous d'abord au cas des jeux d'atteignabilité :

**Définition 3.0.1.** Soit un jeu d'atteignabilité  $\mathfrak{G} = (A, I, F)$  avec  $A = (V_0, V_1, E)$  une arène finie, le  $i^e$  attracteur pour le joueur  $j_n$  est l'ensemble :  
 $Attr_{j_n}^i = \{v \in V \mid \text{le joueur } j_n \text{ peut forcer une visite d'un sommet de } F \text{ depuis } v \text{ en } \leq i \text{ déplacements}\}$

Construction par induction :

$$\begin{aligned} Attr_{j_0}^0(F) &= F \\ Attr_{j_0}^{i+1}(F) &= Attr_{j_0}^i(F) \\ &\quad \cup \{v' \in V_0 \mid \exists (v, v') \in E : v \in Attr_{j_0}^i(F)\} \\ &\quad \cup \{v' \in V_1 \mid \forall (v, v') \in E : v \in Attr_{j_0}^i(F)\} \end{aligned} \tag{3.1}$$

Par cette construction, on obtient une séquence d'attracteurs  $Attr_{j_0}^0(F) \subseteq Attr_{j_0}^1(F) \subseteq \dots$  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 donc  $Attr_{j_0}(F) = \bigcup_{i=0}^{|V|} Attr_{j_0}^i$

Cette construction de l'attracteur correspond à l'ensemble gagnant de  $j_0$ .



**Preuve.** En effet on a que  $W_0 \subseteq Attr_{j_0}$  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  $W_0$ .

Pour montrer que  $Attr_{j_0}(F) \subseteq W_0$ , 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 une arête  $(v, v')$  avec  $v' \notin Attr_{j_0}(F)$ , sinon on aurait  $v \in Attr_{j_0}(F)$ .

Soit un sommet  $v \in V_0 \setminus Attr_{j_0}(F)$ , alors toutes les arêtes  $v$  mènent vers un sommet hors de  $Attr_{j_0}(F)$ , sinon on aurait  $v \in Attr_{j_0}(F)$ .

Ces deux cas étant exhaustifs, on a bien que  $Attr_{j_0}(F) \subseteq W_0$

L'inclusion étant vérifiée dans les deux sens, on a bien que  $W_0 = Attr_{j_0}(F)$ . □

$Attr_{j_0}(F)$  étant l'ensemble gagnant pour  $j_0$ , il lui permet d'établir sa stratégie. A chaque tour, le joueur va déplacer le pion d'un sommet de  $Attr_{j_0}^{i+1}(F)$  vers un sommet de  $Attr_{j_0}^i(F)$  jusqu'à atteindre  $Attr_{j_0}^0(F) = F$

De cette stratégie découle immédiatement la stratégie pour le joueur  $j_1$  : déplacer le pion vers des sommets hors de l'attracteur car, à partir d'un tel sommet,  $j_0$  ne peut pas forcer une visite d'un sommet de  $F$ .

Cette méthode de résolution pour les jeux d'atteignabilité permet aussi de résoudre les jeux de sûreté. Soit un jeu de sûreté  $\mathfrak{G} = (A, I, F)$  avec  $A$  une arène finie. Si l'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  $F$ , on peut en déduire une stratégie pour  $j_0$  de manière analogue à la stratégie pour le joueur  $j_1$  dans un jeu d'atteignabilité. Il suffit à  $j_0$  de ne pas visiter de sommets de cet attracteur. Ainsi,  $j_1$  ne pourra pas forcer de visite de  $F$ .

**Algorithm 1** Attracteur

---

**Entrées** **G** : Graphe, structure de donnée composée d'un tableau à deux dimensions *predecessors* de prédecesseurs (la liste de prédecesseurs d'un noeud *i* est stockée à la *i*<sup>e</sup> entrée du tableau) et une liste *players* (le *i*<sup>e</sup> noeud appartient au joueur dont le numéro figure en *i*<sup>e</sup> entrée de *players*).

**F** : Liste de numéro 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:       out_degrees[j]  $\leftarrow$  demi-degré extérieur du noeud j
6:     end if
7:   end for
8:   attractor  $\leftarrow$  tableau de taille  $|G|$  ▷ Initialisation
9:   for index in F do
10:    attractor[index]  $\leftarrow$  1
11:  end for
12:  attractor_new  $\leftarrow F$ 
13:  while attractor_new non-vidé and i  $\neq 0$  do ▷ Calcul de l'attracteur
14:    to_check  $\leftarrow$  attractor_new
15:    attractor_new  $\leftarrow$  [ ]
16:    for index in to_check do
17:      for pred in G.predecessors[index] do
18:        if attractor[pred] = 0 then
19:          if G.players[pred] = p then
20:            attractor_new.append(pred)
21:          else
22:            out_degrees[pred]  $\leftarrow$  out_degrees[pred] - 1
23:            if out_degrees[pred] = 0 then
24:              attractor_new.append(pred)
25:            end if
26:          end if
27:        end if
28:      end for
29:    end for

```

---

---

```

30:      for  $index$  in  $attractor\_new$  do
31:           $attractor[index] \leftarrow 1$ 
32:      end for
33:       $i \leftarrow i - 1$ 
34:  end while
35:  return  $attractor$ 
36: end procedure

```

---

## Explication

Cet algorithme se base sur la construction par induction de  $Attr_{j_n}^i(F)$ .

Le cas de départ de la construction se retrouve dans l'étape d'initialisation de l'algorithme. On y ajoute les noeuds de  $F$  dans l'attracteur.

L'étape du calcul de l'attracteur représente la deuxième étape de la construction. Identifions les 3 éléments de l'union qui constitue  $Attr_{j_0}^{i+1}(F)$  dans l'algorithme :

- $Attr_{j_0}^i(F)$  se retrouve à la ligne 30. A chaque étape on ne crée pas un nouvel attracteur mais on ajoute de nouveaux sommets à l'ancien.
- $\{v' \in V_0 \mid \exists(v, v') \in E : v \in Attr_{j_0}^i(F)\}$ . Dans la boucle intérieure, ligne 19, si un prédecesseur du noeud en cours de traitement appartient au joueur cible, alors il est ajouté à l'attracteur.
- $\{v' \in V_1 \mid \forall(v, v') \in E : v \in Attr_{j_0}^i(F)\}$ . Dans la boucle intérieure, ligne 21, si un prédecesseur du noeud en cours de traitement appartient au joueur ennemi, 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. Afin de se souvenir de l'appartenance des successeurs du noeud à l'attracteur, on calcule initialement son demi-degré extérieur. Ainsi, à chaque fois qu'un noeud est rencontré dans la liste des prédecesseurs d'un autre noeud, on décremente son demi-degré extérieur. 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 prédecesseurs d'un noeud que s'il a été ajouté à l'attracteur). On peut donc l'ajouter à son tour à l'attracteur.

## Complexité

Considérons un graphe  $G$  possédant  $n$  noeuds et  $m$  arêtes. 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 boucles principales :

— *Pré-traitement*

Le calcul du demi-degré extérieur à l'aide d'une structure de données telle que décrite dans l'entête 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. Les graphes possédant  $m$  arêtes, il y aura au plus  $m$  éléments dans la liste des prédécesseurs.

— *Initialisation*

Cette étape se fait en temps  $O(n)$  car il y a au plus  $n$  noeuds dans le graphes, donc pour lesquels on souhaite construire l'attracteur.

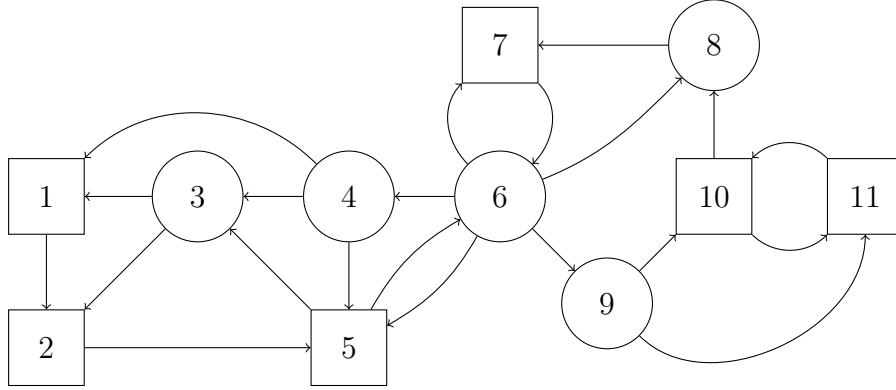
— *Calcul de l'attracteur*

Considérons une valeur de  $i$  négative pour le pire des cas. Il y aura au maximum  $n$  passages dans la boucle extérieure (ligne 13) car, à chaque étape, *attractor\_new* doit contenir au moins un noeud de  $G$  hors de l'attracteur courant. L'évaluation de la condition d'arrêt se fait en  $O(1)$ . La boucle à la ligne 16 itère sur les noeuds d'*attractor\_new* et, pour chacun d'entre eux, la boucle à la ligne 17 va itérer sur ses prédécesseurs. Toutes les opérations à l'intérieur de cette boucle sont en temps constant  $O(1)$ . On a donc que, dans le pire des cas, l'algorithme va effectuer des opérations en  $O(1)$  pour chaque prédécesseur de chaque noeud de  $G$ . Finalement, la boucle à la ligne 30 effectuera dans le pire des cas  $n$  fois une opération en  $O(1)$ , car au plus  $n$  noeuds seront ajoutés à l'attracteur. Cela nous donne une complexité en  $O(n$  (évaluation de la condition d'arrêt)  $+m$  (boucle intérieure)  $+n$  (ajout à l'attracteur)  $) = O(2n + m) = O(n + m)$ .

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

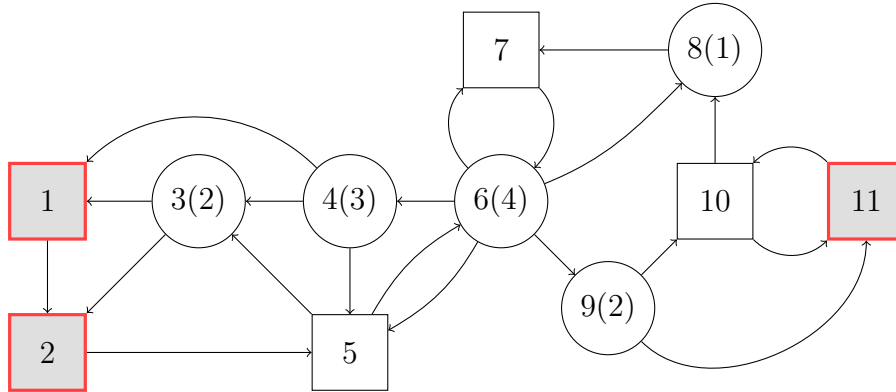
## Exemple

Considérons l'arène suivante :



Les noeuds ronds appartiennent au joueur  $j_0$ , et les carrés à  $j_1$ . Appliquons l'algorithme *Attractor* sur celui-ci, avec comme ensemble de départ  $F = \{1, 2, 11\}$  (il est intéressant de noter que l'ensemble  $F$  ne doit pas forcément être connexe) et pour le joueur  $j_0$ . De plus nous allons supposer  $i$  négatif afin de calculer l'attracteur complet.

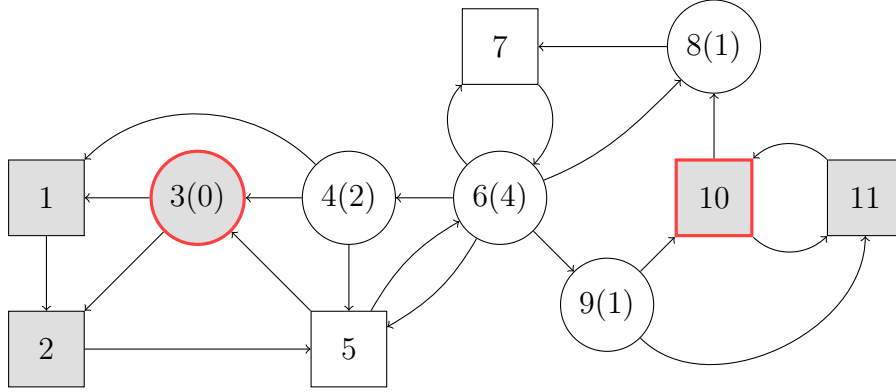
L'étape de pré-traitement sera rendue visuelle en ajoutant aux noeuds de  $j_1$  la valeur correspondant dans le tableau *out\_degrees*. Les noeuds faisant partie de l'attracteur courant *attractor* seront colorés en gris et ceux étant ajoutés à l'attracteur à l'itération précédente (les noeuds de la 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  $Attr_{j_0}^0(F) = \{1, 2, 11\}$ .

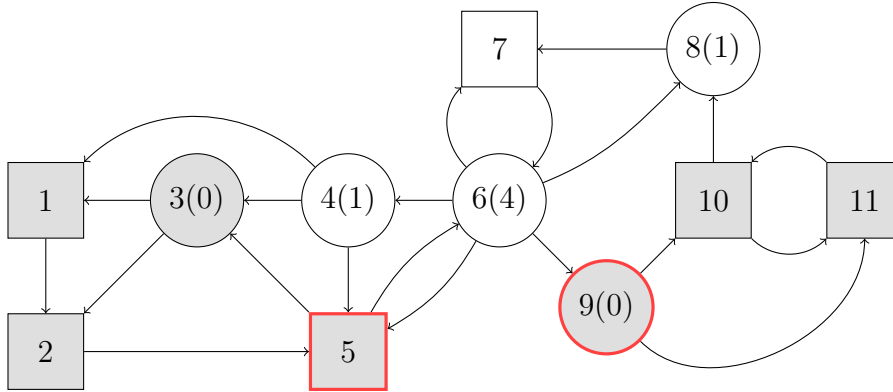
La première itération du calcul de l'attracteur va ajouter les noeuds 3 et 10 à l'attracteur. En effet, la valeur de *out\_degrees* de 3 va être décrémentée deux fois, une fois en suivant les prédecesseurs de 1 et une autre fois en suivant ceux de 2. Cette valeur atteignant 0, il sera ajouté à l'attracteur. 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 :



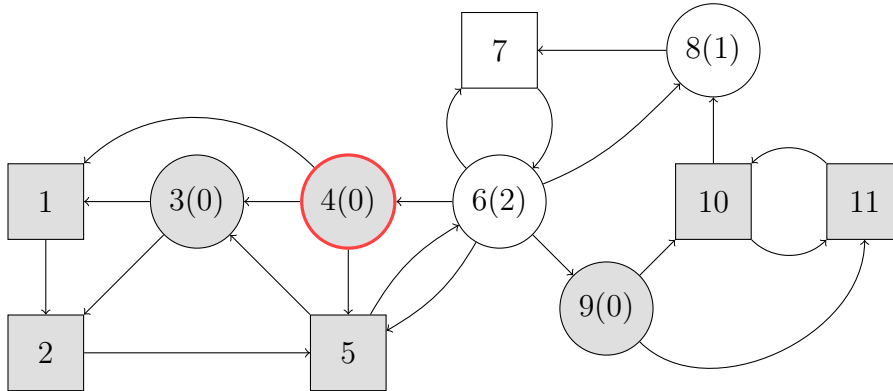
Qui correspond à  $Attr_{j_0}^1(F) = \{1, 2, 3, 10, 11\}$ .

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.



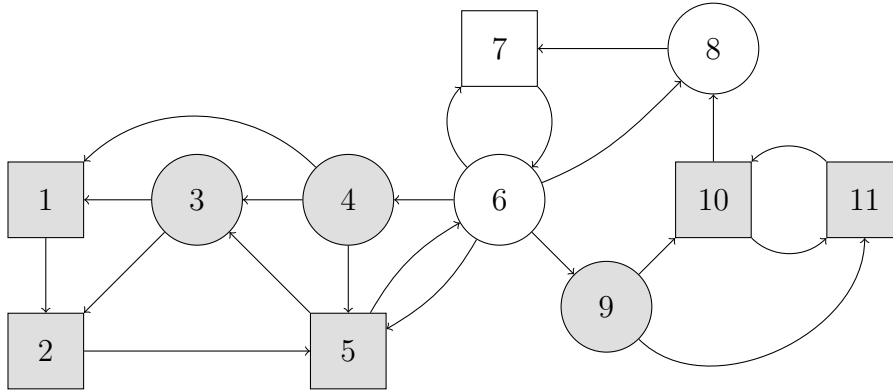
Nous obtenons  $Attr_{j_0}^2(F) = \{1, 2, 3, 5, 9, 10, 11\}$

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 décrétementée une fois car il est prédecesseur de 4. Aucun noeud n'est ajouté à *attractor\_new*, l'algorithme s'arrête.



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

## **Chapitre 4**

### **Cas infini**



# 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.

## **Annexe A**

### **Première annexe**

## **Annexe B**

### **Deuxième annexe**