
Chapitre 11 : Théorie des jeux

Dans ce chapitre nous nous intéressons à des jeux déterministes, à **deux joueurs** et à **information totale**. Les deux joueurs, que l'on nommera **Alice** et **Bob**, sont adversaires l'un de l'autre, et jouent à tour de rôle. Le fait que les jeux soient à informations totales signifie que les deux joueurs connaissent à chaque instant l'état complet du jeu. Les échecs, le go ou le puissance 4 sont par exemple des jeux à information totale, contrairement à la plupart des jeux de cartes où les cartes d'un joueur ne sont pas connues des autres joueurs.

On se fixe pour objectif la formalisation de tels jeux au moyen des objets mathématiques et informatique introduits dans les chapitres précédents. On souhaite de plus étudier l'existence, puis la calculabilité d'une stratégie permettant à un des deux joueurs de gagner.

1 Jeux sur un graphe

1.1 Un exemple : le jeu de la soustraction

Afin de fixer les idées on utilise comme fil rouge dans ce chapitre le jeu de la soustraction (aussi connu comme "le jeu des allumettes"). Dans ce jeu à deux joueurs, on dispose 13 allumettes identiques sur une table, chaque joueur, à son tour, a le droit d'enlever 1, 2 ou 3 allumettes. Le joueur obligé de prendre la dernière allumette a perdu.

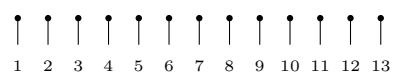
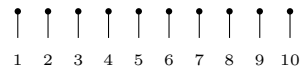
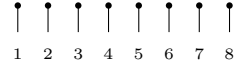



État du jeu	Action
 1 2 3 4 5 6 7 8 9 10 11 12 13	Alice prend 3 allumettes
 1 2 3 4 5 6 7 8 9 10	Bob prend 2 allumettes
 1 2 3 4 5 6 7 8	Alice prend 3 allumettes
 1 2 3 4 5	Bob prend 3 allumettes
 1 2	Alice prend 1 allumette
 1	Bob prend 1 allumette et a perdu

FIGURE 1 – Exemple d'une partie du jeu de la soustraction

On remarque que l'état du jeu au cours de la partie est décrit ici par le nombre d'allumettes restantes et par le joueur qui doit jouer le prochain coup. De plus, les états que l'on peut atteindre en un coup depuis un état (n, j) sont les états (n', \bar{j}) où \bar{j} désigne le joueur opposé au joueur j , et où

$n' \in \{n - 1, n - 2, n - 3\} \cap \mathbb{N}$. Ainsi le jeu est représenté par le graphe orienté biparti de la figure 2 où :

- \odot_n (resp. \heartsuit_n) représente l'état où il reste n allumettes et où c'est à **Alice** (resp. à **Bob**) de jouer ;
- l'état initial est \odot_{13} ;
- l'ensemble des états objectifs pour **Alice** (resp. **Bob**) est le singleton $\{\heartsuit_1\}$ (resp. $\{\odot_1\}$).

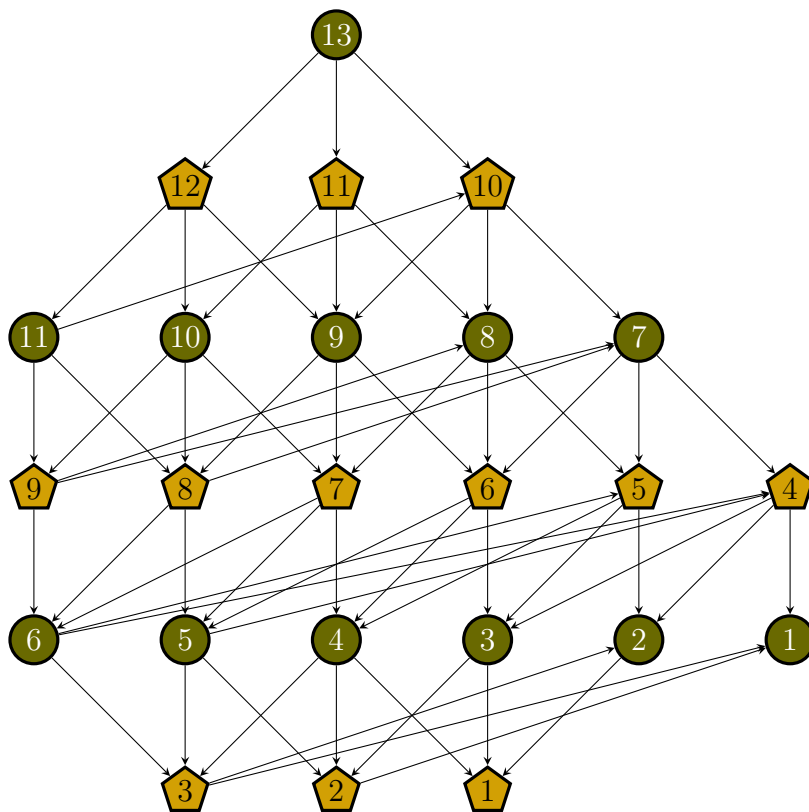


FIGURE 2 – Graphe d'états du jeu de la soustraction

Exercice de cours 1.1

Repérer sur le graphe de la figure 2 le chemin correspondant à la partie de la figure 1.

1.2 Définitions

Définition 1.2

Un **jeu d'accessibilité** est la donnée :

- d'un graphe biparti orienté $G = (V_A \sqcup V_B, E)$ \clubsuit (nous notons $V = V_A \sqcup V_B$ dans la suite) ;
- d'un état initial $s_0 \in V_A$;
- de deux ensembles disjoints d'états sans successeurs \mathcal{O}_A et \mathcal{O}_B ,
i.e. $\mathcal{O}_A \subseteq V$ et $\mathcal{O}_B \subseteq V$ tels que $\mathcal{O}_A \cap \mathcal{O}_B = \emptyset$ et $\forall s \in \mathcal{O}_A \cup \mathcal{O}_B, \forall s' \in V, (s, s') \notin E$.

Vocabulaire 1.3

Un état $u \in V$ d'un jeu est dit **terminal** s'il est sans successeur.

Les états de \mathcal{O}_A (resp. \mathcal{O}_B) sont appelés les états **objectifs** pour **Alice** (resp. pour **Bob**).

\clubsuit . Lorsqu'une partition des états est explicitée dans une telle notation, cela signifie que les arcs ou arêtes ont une extrémité dans chacune des parties, ici l'une dans V_A et l'autre dans V_B .

Remarque 1.4

V_A (resp. V_B) est l'ensemble des états pour lesquels c'est au tour d'**Alice** (resp. de **Bob**) de jouer.

Définition 1.5







Soit $(G = (V, E), s_0, \Theta_A, \Theta_B)$ un jeu d'accessibilité. Soit $s \in V$.

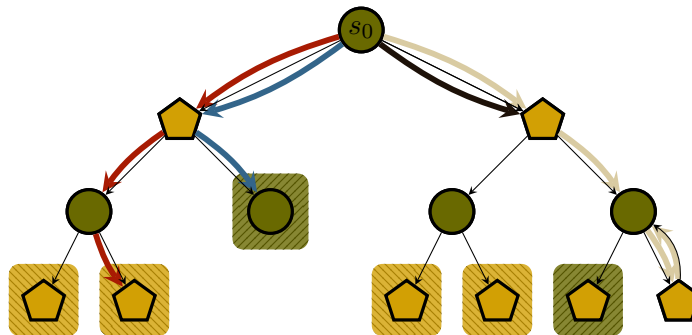
Une **partie depuis l'état** s dans ce jeu est un chemin de G d'origine le sommet s qui aboutit en un état terminal, ou bien un chemin infini de G ♣.

Une **partie** (sans plus de précision), est une partie depuis l'état initial s_0 .

Si une partie termine en un sommet de Θ_A , **Alice** a gagné la partie, si c'est un sommet de Θ_B , **Bob** a gagné la partie, sinon on dit que la partie est un **match nul**.

Exemple 1.6

On considère le jeu d'accessibilité ci-dessous. L'ensemble Θ_A est représenté par , l'ensemble Θ_B est représenté par . Dans cet exemple de jeu d'accessibilité, le chemin  est une partie gagnée pour **Bob**, le chemin  est une partie gagnée pour **Alice**, le chemin  est un match nul, le chemin  n'est pas une partie.



1.3 Représentation en machine des jeux

1.3.1 Représentation au moyen d'un graphe

Il est possible de représenter un jeu au moyen d'un graphe (donné, par exemple, par table de listes d'adjacence) et de la donnée de l'état initial et des états objectifs. Cette représentation nécessite cependant la construction explicite du graphe d'états, ce qui n'est pas possible si le jeu a trop d'états possibles (c'est le cas pour les échecs par exemple).

1.3.2 Représentation au moyen d'un graphe implicite

Pour pallier le problème précédent, on peut choisir de représenter le graphe des états du jeu au moyen d'un graphe implicite. En OCAML on peut représenter un jeu par des types, fonctions et valeurs ayant les signatures suivantes.

```
1 (* le type des joueurs *)
2 type joueur
3 (* le type des états du graphe *)
4 type etat
5
6 (* fonction retournant le joueur courant d'un état *)
7 val joueur : etat -> joueur
8 (* valeur donnant l'état initial *)
```

♣. Un chemin infini dans un graphe orienté $G = (S, A)$ est une suite $u \in S^{\mathbb{N}}$ telle que $\forall i \in \mathbb{N}, (u_i, u_{i+1}) \in A$.

```

9 | val init      : etat
10 | (* fonction calculant les états successeurs d'un état *)
11 | val successeurs : etat -> etat list
12 | (* fonction déterminant si un état terminal est un état de victoire *)
13 | val est_gagnant : etat -> joueur option

```

Exemple 1.7

Le jeu de la soustraction défini ci-avant est alors représenté par les définitions suivantes.

```

1 | type joueur = Alice | Bob
2 | type etat   = { allu : int ; joueur : joueur }
3
4 | let joueur (e: etat): joueur = e.joueur
5
6 | let init = {allu = 13 ; joueur = Alice}
7
8 | (** Calcule le joueur adversaire de [j]. *)
9 | let autre (j: joueur) : joueur =
10 |   match j with Alice -> Bob | Bob -> Alice
11
12 | let successeurs (e: etat): etat list =
13 |   (if e.allu > 3 then [{allu = e.allu - 3; joueur = autre e.joueur}] else [])
14 |   @ (if e.allu > 2 then [{allu = e.allu - 2; joueur = autre e.joueur}] else [])
15 |   @ (if e.allu > 1 then [{allu = e.allu - 1; joueur = autre e.joueur}] else [])
16
17 | let est_gagnant (e: etat): joueur option =
18 |   if e.allu = 1 then Some (autre e.joueur) else None

```

1.4 Stratégie et attracteurs

Dans toute cette section on fixe $(G = (V = V_A^* \sqcup V_B, E), s_0, \mathcal{O}_A, \mathcal{O}_B)$ un jeu d'accessibilité. On note de plus V_A^* (resp. V_B^*) les états non terminaux de V_A (resp. V_B).

Définition 1.8

On appelle **stratégie pour Alice** une fonction $f_A : V_A^* \rightarrow V_B$ telle que $\forall s_a \in V_A, (s_a, f(s_a)) \in E$, autrement dit une fonction qui indique à **Alice** ce qu'elle doit jouer dans n'importe quel état non terminal où c'est son tour.

On définit de même une stratégie pour **Bob**.

Définition 1.9

Soit $f_A : V_A^* \rightarrow V_B$ une stratégie pour **Alice**. Soit $s \in V$ un état du jeu.

- Une partie finie (s_0, s_1, \dots, s_n) est dite **jouée selon la stratégie** f_A si à chaque tour de cette partie où **Alice** devait jouer, elle a joué selon f_A , i.e. $\forall i \in \llbracket 0, n-1 \rrbracket, s_i \in V_A \Rightarrow s_{i+1} = f(s_i)$.
- Une partie infinie $(s_i)_{i \in \mathbb{N}}$ est de même dite **jouée selon la stratégie** f_A si $\forall i \in \mathbb{N}, s_i \in V_A \Rightarrow s_{i+1} = f(s_i)$.
- La stratégie f_A est dite **gagnante depuis** s pour **Alice** si toute partie depuis s jouée selon f_A est gagnée par **Alice**.
- Une stratégie est dite **gagnante pour Alice** (sans plus de précision) si elle est gagnante pour **Alice** depuis s_0 .
- Un état du jeu est dit **gagnant pour Alice** dès lors qu'il existe une stratégie gagnante pour **Alice** depuis cet état. On parle aussi de **position gagnante**.

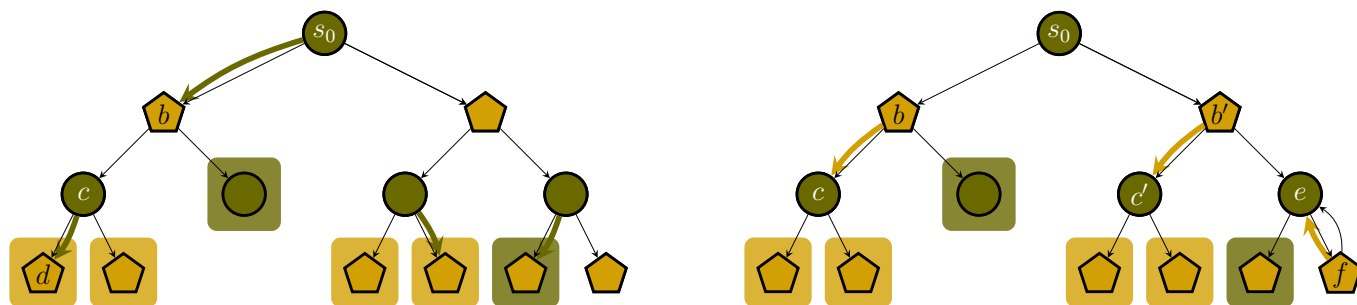
On définit de même les stratégies et états gagnants pour **Bob**.

Exemple 1.10

On reprend le jeu d'accessibilité introduit dans l'exemple précédent.

Les flèches \rightarrow de la figure 3a définissent une stratégie pour **Alice**. On remarque que cette stratégie n'est pas gagnante pour **Alice** pour l'état s_0 : en effet la partie (s_0, b, c, d) est jouée selon la stratégie d'**Alice** mais se conclut par une victoire de **Bob**.

Les flèches \rightarrow de la figure 3b définissent une stratégie pour **Bob**. On remarque que cette stratégie est gagnante pour **Bob** : en effet qu'**Alice** commence par jouer en b ou en b' , **Bob** joue de manière à se ramener à un état où les deux seuls choix d'**Alice** sont de faire gagner **Bob** (i.e. dans un état de V_A dont les deux successeurs sont dans \mathcal{O}_B), à savoir c ou c' . On remarque que le fait que la stratégie de **Bob** lui indique, s'il est dans l'état f , de se ramener à l'état e qui est une position gagnante pour **Alice**, ne gêne en rien puisque l'état n'est jamais atteint dans une partie jouée depuis s_0 suivant cette stratégie.



(a) Une stratégie pour **Alice**.

(b) Une stratégie gagnante pour **Bob**.

FIGURE 3 – Exemples de stratégie

Exercice de cours 1.11

Donner une stratégie gagnante pour **Alice** pour le jeu de la soustraction. On ne demande pas de démontrer que celle-ci est gagnante.

Calcul des positions gagnantes. On s'intéresse dans cette section à la mise en place d'algorithmes permettant le calcul des positions gagnantes et des stratégies gagnantes associées. On remarque qu'**Alice** peut gagner une partie avec certitude :

- si c'est à son tour de jouer et qu'elle se trouve dans un état \mathcal{O}_A ;
- si c'est au tour de **Bob** de jouer et que tous les successeurs de l'état courant sont des positions gagnantes pour **Alice** ;
- si c'est à son tour de jouer et qu'il existe un successeur de l'état courant depuis lequel **Bob** ne peut que perdre (le cas précédent).

De tels états, depuis lesquels **Alice** peut gagner à coup sûr, sont appelés des **attracteurs pour Alice**. Ils nous permettent de construire une stratégie gagnante pour **Alice** : la stratégie qui consiste à jouer de sorte à toujours se ramener sur un attracteur. Dans la suite de cette section on montre qu'une telle stratégie est bien gagnante, et même qu'il existe une stratégie gagnante pour **Alice** depuis un état s seulement si c'est un attracteur.

Définition 1.12

On définit la suite d'ensemble d'états $(\mathcal{A}_i)_{i \in \mathbb{N}}$ par les relations suivantes.

$$\begin{aligned} \mathcal{A}_0 &= \mathcal{O}_A \\ \forall i \in \mathbb{N}, \mathcal{A}_{i+1} &= \mathcal{A}_i \cup \{s_b \in V_B^* \mid \forall s_a \in \text{succ}(s_b), s_a \in \mathcal{A}_i\} \\ &\quad \cup \{s_a \in V_A^* \mid \exists s_b \in \text{succ}(s_a), s_b \in \mathcal{A}_i\} \end{aligned}$$

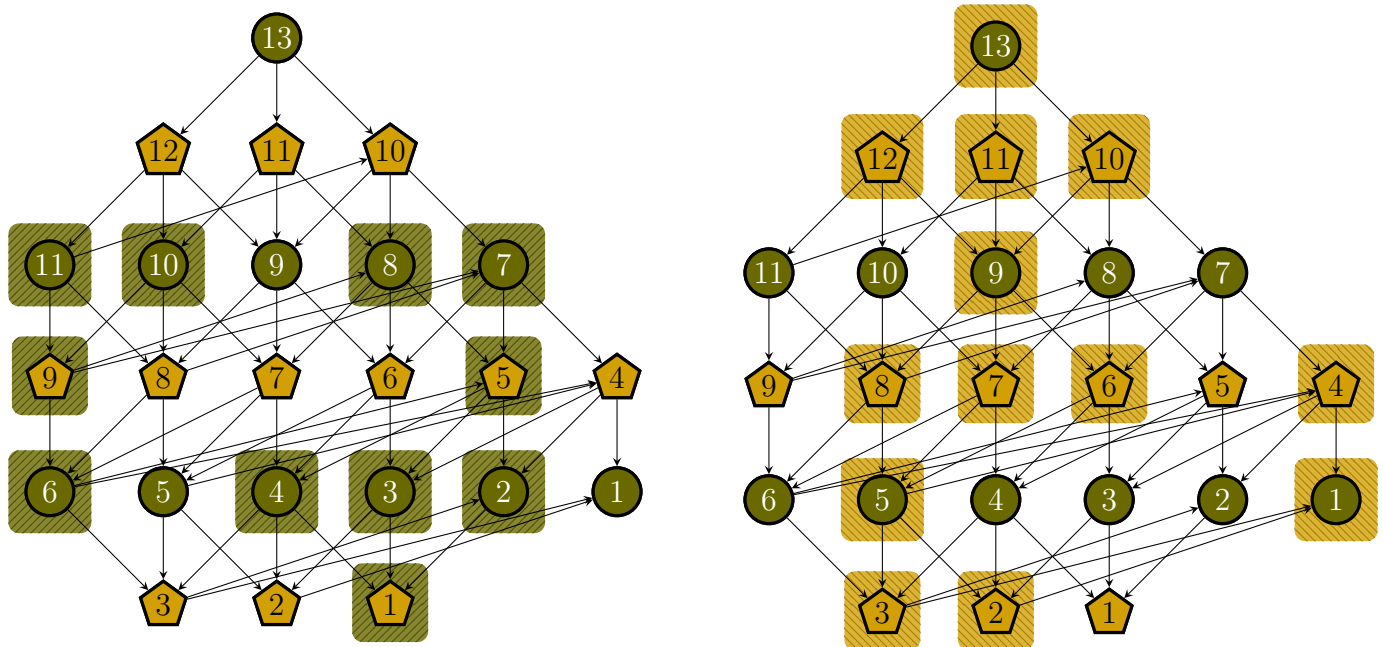
Cette suite d'ensembles d'états est croissante, et puisque V est fini elle est ultimement stationnaire. L'ensemble des **attracteurs pour Alice** est alors $\mathcal{A} = \bigcup_{i \in \mathbb{N}} \mathcal{A}_i$.

Exercice de cours 1.13

Pour chaque joueur, quel est l'ensemble des attracteurs qui sont terminaux ?

Exemple 1.14

Les attracteurs pour **Alice** (resp. **Bob**) pour le jeu de la soustraction sont représentés en figure 4a (resp. figure 4b).



(a) Attracteurs pour **Alice**

(b) Attracteurs pour **Bob**

FIGURE 4 – Attracteurs pour chacun des joueurs pour le jeu de la soustraction

Proposition 1.15

Les attracteurs des deux joueurs ne s'intersectent pas.

Démonstration : Notons $(\mathcal{A}_n)_{n \in \mathbb{N}}$ et $(\mathcal{B}_n)_{n \in \mathbb{N}}$ les suites définies dans la définition 1.12.

Par construction de $(\mathcal{A}_n)_{n \in \mathbb{N}}$, on remarque que :

$$\begin{aligned} \forall n \in \mathbb{N}, \forall s \in \mathcal{A}_{n+1} \setminus \mathcal{A}_n, s \in V_A^* \cup V_B^* & \quad (\star_{\mathcal{A}}) \\ \text{et } (s \in V_A^* \Rightarrow \exists s_b \in \text{succ}(s), s_b \in \mathcal{A}_n) & \\ \text{et } (s \in V_B^* \Rightarrow \forall s_a \in \text{succ}(s), s_a \in \mathcal{A}_n) & \end{aligned}$$

On peut faire une remarque analogue $(\star_{\mathcal{B}})$ pour $(\mathcal{B}_n)_{n \in \mathbb{N}}$.

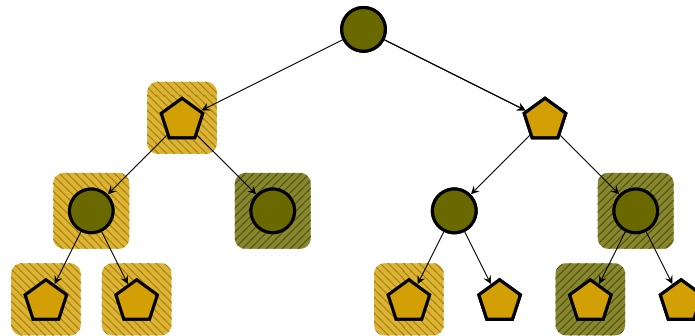
On montre alors par récurrence sur $n \in \mathbb{N}$ que $\mathcal{A}_n \cap \mathcal{B}_n = \emptyset$.

- Par définition d'un jeu, \mathcal{O}_A et \mathcal{O}_B sont disjoints, donc $\mathcal{A}_0 \cap \mathcal{B}_0 = \emptyset$
- Soit $n \in \mathbb{N}$, supposons $\mathcal{A}_n \cap \mathcal{B}_n = \emptyset$, montrons que $\mathcal{A}_{n+1} \cap \mathcal{B}_{n+1} = \emptyset$. Supposons par l'absurde qu'il existe $s \in \mathcal{A}_{n+1} \cap \mathcal{B}_{n+1}$. Puisque $\mathcal{A}_n \cap \mathcal{B}_n = \emptyset$, $s \in \mathcal{A}_{n+1} \setminus \mathcal{A}_n$ ou $s \in \mathcal{B}_{n+1} \setminus \mathcal{B}_n$. On suppose sans perdre en généralité que $s \in \mathcal{A}_{n+1} \setminus \mathcal{A}_n$, d'après $(\star_{\mathcal{A}})$ il y a deux cas possibles :
 - ▷ soit $s \in V_A^*$, auquel cas il existe $s_b \in \text{succ}(s) \cap \mathcal{A}_n$, ce qui contredit le fait que $\forall s' \in \text{succ}(s), s' \in \mathcal{B}_n$ puisque $\mathcal{A}_n \cap \mathcal{B}_n = \emptyset$, et donc contredit $s \in \mathcal{B}_{n+1}$ (d'après $\star_{\mathcal{B}}$);
 - ▷ soit $s \in V_B^*$, auquel cas $\forall s_a \in \text{succ}(s), s_a \in \mathcal{A}_n$, ce qui contredit le fait que $\exists s' \in \text{succ}(s), s' \in \mathcal{B}_n$ puisque $\mathcal{A}_n \cap \mathcal{B}_n = \emptyset$, et donc contredit $s \in \mathcal{B}_{n+1}$ (d'après $\star_{\mathcal{B}}$).

□

Remarque 1.16

Attention, il est possible que les attracteurs de \mathcal{A} et \mathcal{B} ne forment pas une partition de l'espace des états $V_A \cup V_B$. Reprenons par exemple une version modifiée du jeu exemple introduit ci-avant. On a représenté ci-dessous les attracteurs d'**Alice** (●), et de **Bob** (⬠)



Exercice de cours 1.17

Donner un jeu d'accessibilité où :

- les attracteurs des deux joueurs ne couvrent pas tous les états du jeu ;
- les objectifs des deux joueurs couvrent tous les états terminaux du jeu.

Calcul de stratégies gagnantes. La donnée des attracteurs pour un joueur permet de construire une stratégie gagnante pour ce joueur depuis n'importe lequel de ses attracteurs. Dans cette section on fait cette construction pour **Alice**, mais on pourrait faire de même pour **Bob**. On fixe donc $\mathcal{A} = \bigcup_{i \in \mathbb{N}} \mathcal{A}_i$ l'ensemble des attracteurs d'**Alice**.

On l'a déjà dit, l'idée est qu'**Alice** doit à chaque fois jouer pour se ramener dans \mathcal{A} , cependant dans le cas où le jeu admet des circuits, cela n'est pas suffisant car on peut ainsi jouer une partie infinie (ce qui ne fait pas gagner **Alice**).

Afin d'éviter cet écueil, il faut qu'à chacun de ses tours **Alice** joue vers un attracteur "qui la rapproche" de ses objectifs. La notion de rang introduite ci-dessous permet de formaliser cette notion de rapprochement des objectifs, et ainsi de définir une stratégie vraiment gagnante.

Définition 1.18

Le **rang** d'un attracteur d'**Alice** est donné par la fonction $rg : \mathcal{A} \rightarrow \mathbb{N}$ définie comme suit.

$$\forall s \in \mathcal{A}, rg(s) \stackrel{\text{déf}}{=} \min\{i \in \mathbb{N} \mid s \in \mathcal{A}_i\}$$

Exercice de cours 1.19

Montrer que l'ensemble des attracteurs d'**Alice** de rang $i \in \mathbb{N}^*$ est exactement $\mathcal{A}_i \setminus \mathcal{A}_{i-1}$.

Lemme 1.20

Pour tout $s \in \mathcal{A}$ de rang $rg(s) > 0$,

- ▷ si $s \in V_A$, alors il existe $s' \in \text{succ}(s)$ tel que $s' \in \mathcal{A}$ et est de rang $rg(s') < rg(s)$;
- ▷ si $s \in V_B$, alors pour tout $s' \in \text{succ}(s)$, $s' \in \mathcal{A}$ et est de rang $rg(s') < rg(s)$.

Démonstration : ▷ Soit $s \in \mathcal{A} \cap V_A$ tel que $rg(s) > 0$. Notons $r = rg(s)$, ainsi $s \in \mathcal{A}_r \cap V_A$. Avec la remarque $(\star_{\mathcal{A}})$ (faite dans la preuve de la propriété 1.15), on en déduit qu'il existe $s' \in \text{succ}(s)$ tel que $s' \in \mathcal{A}_{r-1}$, ainsi $s' \in \mathcal{A}$ et $rg(s') \leq r - 1$. Un tel s' convient.

▷ Soit $s \in \mathcal{A} \cap V_B$ tel que $\text{rg}(s) > 0$. Notons $r = \text{rg}(s)$, ainsi $s \in \mathcal{A}_r \cap V_B$. Avec la remarque ($\star_{\mathcal{A}}$), on en déduit que pour tout $s' \in \text{succ}(s)$, $s' \in \mathcal{A}_{r-1}$, donc $s' \in \mathcal{A}$ et $\text{rg}(s') \leq r - 1$. □

Définition 1.21

On définit pour **Alice** une **stratégie dérivée de ses attracteurs** de la manière suivante.

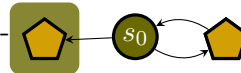
$$f = \left(\begin{array}{l} V_A^* \rightarrow V_B \\ s_a \mapsto \begin{cases} s_b \in \text{succ}(s_a) \cap \mathcal{A} \text{ avec } \text{rg}(s_b) < \text{rg}(s_a) & \text{si } s_a \in \mathcal{A} \\ \text{quelconque} & \text{sinon} \end{cases} \end{array} \right)$$

Remarque 1.22

Une telle stratégie existe d'après le lemme 1.20.

Exercice de cours 1.23

Donner la stratégie dérivée des attracteurs pour **Alice** dans le jeu ci-contre. Justifier l'utilisation du rang dans les définitions ci-dessus.



Remarque 1.24

En général, il n'y a pas unicité de la stratégie dérivées des attracteurs, et ce pour deux raisons.

- D'une part il y a potentiellement de multiples façons de choisir comment **Alice** joue sur les états de $V_A^* \setminus \mathcal{A}$. Cette multiplicité est peu significative, au sens où elle ne s'exprimera pas dans les parties jouées selon une telle stratégie, justement parce que la stratégie nous ramène toujours à un état de \mathcal{A} .
- D'autre part, il y a potentiellement de multiples façons de choisir l'état $s_b \in \text{succ}(s) \cap \mathcal{A}$ à jouer depuis un état $s \in \mathcal{A} \cap V_A$, et ces différents choix ont une incidence sur les parties jouées selon la stratégie.

Exercice de cours 1.25

Pour une stratégie f dérivant de \mathcal{A} fixée, y a-t-il unicité des parties jouées selon f depuis un état $s \in \mathcal{A}$?

Proposition 1.26

Si f est une stratégie dérivée de l'ensemble \mathcal{A} des attracteurs d'**Alice**, alors toute partie jouée suivant f depuis un état de \mathcal{A} est gagnante pour **Alice**.

Démonstration : Soit f une stratégie dérivant de \mathcal{A} . Soit $s \in \mathcal{A}$ un attracteur d'**Alice**. Soit enfin $(s_i)_{i \in I}$ une partie finie ou infinie \clubsuit jouée depuis s en suivant la stratégie f .

Établissons que $\forall i \in I, s_i \in \mathcal{A}$ par récurrence sur $i \in I$.

- $s_0 = s$ puisque la partie est jouée depuis s , et $s \in \mathcal{A}$ par hypothèse.
- Soit $i \in I$. Supposons que $s_i \in \mathcal{A}$, montrons que s_{i+1} aussi.
 - ▷ Si $s_i \in V_A$, $s_{i+1} = f(s_i)$ puisque la partie est jouée par **Alice** selon f , donc $s_{i+1} \in \text{succ}(s_i) \cap \mathcal{A}$ par définition de f dérivant de \mathcal{A} , et donc en particulier $s_{i+1} \in \mathcal{A}$.
 - ▷ Si $s_i \in V_B$, alors par définition de \mathcal{A} tous ses successeurs sont aussi dans \mathcal{A} . Or $s_{i+1} \in \text{succ}(s_i)$ par définition d'une partie, donc $s_{i+1} \in \mathcal{A}$.

Considérons alors la suite $(\text{rg}(s_i))_{i \in I}$. Elle est à valeurs dans \mathbb{N} . Montrons qu'elle est strictement décroissante. Soit $i \in I \setminus \{0\}$.

▷ Si $s_i \in V_A$, $s_{i+1} = f(s_i)$ donc $\text{rg}(s_{i+1}) < \text{rg}(s_i)$ par définition de f dérivant de \mathcal{A} .

\clubsuit . Si la partie est finie I est un intervalle d'entiers de la forme $\llbracket 0, n \rrbracket$, sinon $I = \mathbb{N}$.

▷ Si $s_i \in V_B$, d'après le lemme 1.20, tous les successeurs de s_i sont dans \mathcal{A} et de rang strictement plus petit que s_i , comme $s_{i+1} \in \text{succ}(s_i)$, $\text{rg}(s_{i+1}) < \text{rg}(s_i)$.

L'ensemble ordonné (\mathbb{N}, \leq) étant bien fondé, la suite $(s_i)_{i \in I}$ est donc finie. Notons alors p le rang de son dernier élément. D'une part $s_p \in \mathcal{A}$ et d'autre part s_p est terminal (sans quoi la partie continuerait au delà du rang p), donc $s_p \in \mathcal{O}_A$ ♣, la partie est donc gagnée par **Alice**. □

Exemple 1.27

Sur l'exemple du jeu de la soustraction on a représenté une stratégie gagnante pour **Bob** depuis l'état initial, dérivée du calcul des attracteurs présenté plus haut.

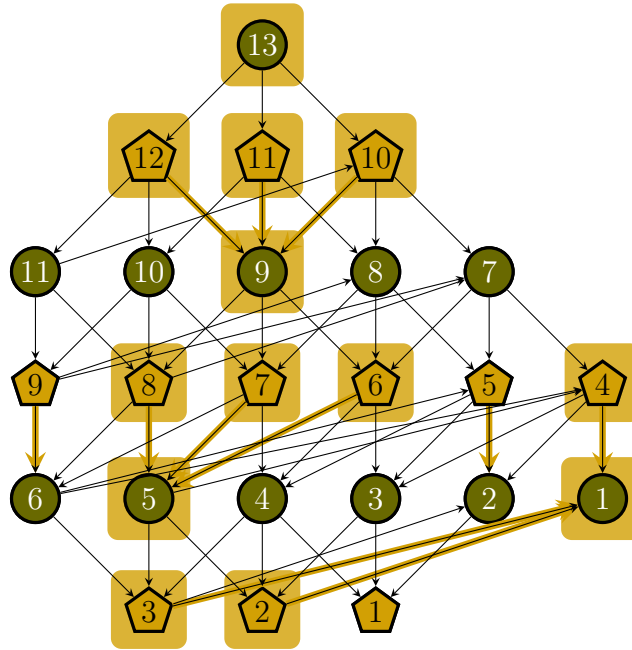


FIGURE 5 – Stratégie gagnante pour **Bob** dérivée de ses attracteurs

La proposition suivante justifie que le calcul des attracteurs permet bien de trouver tous les états du jeu pour lesquels il existe une stratégie gagnante.

Proposition 1.28

Si **Alice** a une stratégie gagnante depuis un sommet s alors s est un de ses attracteurs.

Démonstration : Montrons par récurrence $n \in \mathbb{N}$ la propriété suivante.

$$H_n : \forall s \in V, \left(\begin{array}{l} \text{Alice dispose d'une stratégie } f \text{ gagnante depuis } s \text{ telle que} \\ \text{toute partie jouée depuis } s \text{ selon } f \text{ termine en moins de } n \\ \text{coups} \end{array} \right) \Rightarrow s \in \mathcal{A}_n$$

- Si s est une position gagnante admettant une partie gagnante en moins de 0 coup, alors c'est en 0 coup et donc $s \in \mathcal{O}_A$ or $\mathcal{O}_A = \mathcal{A}_0$, d'où H_0 .
- Soit $n \in \mathbb{N}$. Supposons H_n vraie et montrons que H_{n+1} l'est aussi.
Soit $s \in V$. Supposons qu'il existe f une stratégie gagnante pour **Alice** depuis s telle que toute partie jouée depuis s selon f termine en moins de $n + 1$ coups. Si s est terminal, le fait que f soit gagnante depuis s assure que $s \in \mathcal{O}_A = \mathcal{A}_0$. Traitons donc le cas où s n'est pas terminal.

♣. Revoir l'exercice de cours 1.13

- ▷ Si $s \in V_A^*$, posons $s_1 = f(s)$. Remarquons que f est une stratégie gagnante depuis s_1 , en effet dans le cas contraire, il existerait une partie γ non gagnante jouée selon f depuis s_1 , mais alors $s \cdot \gamma$ serait une partie non gagnante jouée selon f depuis s , ABSURDE. De plus, s'il existait γ une partie jouée selon f depuis s_1 en $n' > n$ coups, $s \cdot \gamma$ serait une partie gagnante jouée selon f depuis s en plus de $n + 1$ coups, ABSURDE. On peut donc appliquer l'hypothèse de récurrence H_n au sommet s_1 et en déduire que $s_1 \in \mathcal{A}_n$. Ainsi s est un état de V_A qui admet un successeur dans \mathcal{A}_n (à savoir s_1), c'est donc un état de \mathcal{A}_{n+1} par définition.
- ▷ Si $s \in V_B^*$, par définition de \mathcal{A}_{n+1} , il suffit de montrer que tous les successeurs de s sont dans \mathcal{A}_n . Soit donc $u \in \text{succ}(s)$. Remarquons que f est une stratégie gagnante depuis u , en effet dans le cas contraire, il existerait une partie γ non gagnante jouée selon f depuis u , mais alors $s \cdot \gamma$ serait une partie non gagnante jouée selon f depuis s , ABSURDE. De plus, comme dans le cas précédent, une partie γ jouée selon f depuis u a moins de n coups puisque $s \cdot \gamma$ a moins de $n + 1$ coups par hypothèse. Donc d'après H_n , $u \in \mathcal{A}_n$.

Ainsi $\forall n \in \mathbb{N}$, H_n est vraie.

Afin de terminer la preuve il nous suffit de montrer que si **Alice** dispose d'une stratégie f gagnante depuis s alors il existe $M \in \mathbb{N}$ tel que toute partie jouée depuis s a au plus M coups, en effet il suffira alors d'utiliser H_M pour conclure.

On dira dans le reste de cette preuve qu'un état s n'admet que des parties bornées jouées selon f s'il existe $M \in \mathbb{N}$ tel que toute partie jouée depuis s a au plus M coups. Supposons par l'absurde que s_0 n'admette pas que des parties bornées jouées selon f , on construit alors par récurrence une partie $(s_n)_{n \in \mathbb{N}}$ jouée selon la stratégie f depuis s telle que tout $n \in \mathbb{N}$, s_n n'admet pas que des parties bornées jouées selon f .

- $s_0 \stackrel{\text{déf}}{=} s$, par supposition s_0 n'admet pas que des parties bornées jouées selon f ;
- en supposant s_0, s_1, \dots, s_n définis, on définit s_{n+1} de la manière suivante.
 - ▷ Si $s_n \in V_A$ alors $s_{n+1} \stackrel{\text{déf}}{=} f(s_n)$. Montrons alors que s_{n+1} n'admet pas que des parties bornées jouées selon f . Soit $M \in \mathbb{N}$, il existe une partie jouée selon f depuis s_n en au moins $M + 1$ coups, notons la $(\gamma_0, \gamma_1, \dots, \gamma_p)$. Puisque $\gamma_0 = s_n$ et que la partie est jouée selon f , $\gamma_1 = f(s_n) = s_{n+1}$ et $(\gamma_1, \dots, \gamma_p)$ est une partie jouée depuis s_{n+1} en au moins M coups.
 - ▷ Sinon $s_n \in V_B$. Puisque s_n n'admet pas que des parties bornées jouées selon f c'est qu'un de ses successeurs n'admet pas que des parties bornées jouées selon f (dans le cas contraire les longueurs des parties jouées depuis s_n serait bornée par le maximum des longueurs des parties jouées depuis ses successeurs augmenté de un, notons qu'il y a un nombre fini de successeurs). Finalement on définit s_{n+1} comme étant ce tel successeur.

La suite $(s_n)_{n \in \mathbb{N}}$ ainsi construite est une partie infinie jouée selon f or f est une stratégie gagnante pour l'état s_0 ce qui est absurde. □

Remarque 1.29

La propriété 1.26 peut être vue comme la correction des attracteurs (un attracteur est bien une position gagnante) et la propriété 1.28 comme leur complétude (une position gagnante a été détectée comme telle par les attracteurs).

Proposition 1.30

*Si le graphe d'états du jeu est sans circuit, et si tout état terminal est un objectif d'**Alice** ou de **Bob** alors chaque état est gagnant pour l'un des joueurs.*

Démonstration : Le graphe d'état du jeu est sans circuit, soit donc $(T_i)_{i \in \llbracket 1, n \rrbracket}$ un tri topologique des états du jeu. Soit \mathcal{N} l'ensemble des états du jeu qui ne sont gagnants ni pour **Alice**, ni pour **Bob**. Supposons par l'absurde que, sous les hypothèses de la propriété, $\mathcal{N} \neq \emptyset$. Soit alors $l = \max\{i \in \llbracket 1, n \rrbracket \mid T_i \in \mathcal{N}\}$. T_l n'est gagnant ni pour **Alice**, ni pour **Bob**, ainsi $T_l \notin \mathcal{O}_A$ et $T_l \notin \mathcal{O}_B$, finalement T_l n'est pas terminal (les terminaux étant exactement $\mathcal{O}_A \cup \mathcal{O}_B$). Supposons alors, sans perdre en généralité que $T_l \in V_A^*$. Par maximalité de l , chaque successeur de T_l est gagnant pour **Alice** ou est gagnant pour **Bob**. Par disjonction de cas :

- ▷ si T_l a un successeur qui est gagnant pour **Alice** alors T_l est gagnant pour **Alice** ;
 - ▷ sinon c'est que tous les successeurs de T_l sont gagnants pour **Bob** auquel cas T_l est gagnant pour **Bob**.
- Ce qui contredit que T_l n'est gagnant ni pour **Alice**, ni pour **Bob**. □

Exercice de cours 1.31

Montrer que les deux hypothèses de la propriété 1.30 sont nécessaires. Autrement dit, donner des contre-exemples à la conclusion de la proposition lorsque le graphe admet des circuits ou des états terminaux qui ne sont objectif d'aucun des deux joueurs.

Exercice de cours 1.32

Sous les mêmes hypothèses, montrer que l'état initial du jeu est un attracteur d'**Alice** ou de **Bob**.

2 Résolution par heuristiques

Dans cette section, on s'intéresse à des jeux d'accessibilité pour lesquels une exploration exhaustive du graphe d'états n'est pas envisageable en raison de sa taille (on peut à nouveau citer par exemple le graphe du jeu des échecs. L'algorithme de calcul des attracteurs proposé à la section précédente n'est alors plus envisageable, dans cette section on considère des algorithmes à base d'**heuristiques** en espérant obtenir de bonnes stratégies à défaut de stratégies gagnantes.

2.1 Score et heuristique

Score d'un état. On considère à nouveau que l'on cherche à définir une stratégie pour **Alice**, la recherche de stratégie pour **Bob** pouvant être menée de la même manière. Au lieu de savoir pour chaque état $s \in V$ s'il est attracteur ou non pour **Alice**, c'est-à-dire de savoir de manière sûre s'il est avantageux ou non ♣, on aura pour chaque état un **score** à valeur dans $\bar{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\}$, permettant d'évaluer si cet état est avantageux pour **Alice** : plus le score est grand, plus l'état est avantageux. En particulier, le score d'un état de \mathcal{O}_A est $+\infty$, celui d'un état de \mathcal{O}_B est $-\infty$, et on donne le score 0 aux autres états terminaux, c'est-à-dire aux matchs nuls.

Un score peut être représenté en OCAML au moyen du type suivant.

```
1 | type score =
2 |   | Pinf                (* +∞ *)
3 |   | Minf                (* -∞ *)
4 |   | Fini of int         (* n ∈ ℕ *)
```

Calcul naïf des scores. Une fois fixés les scores des états terminaux, on peut imaginer calculer le score de n'importe quel état s de manière récursive comme suit.

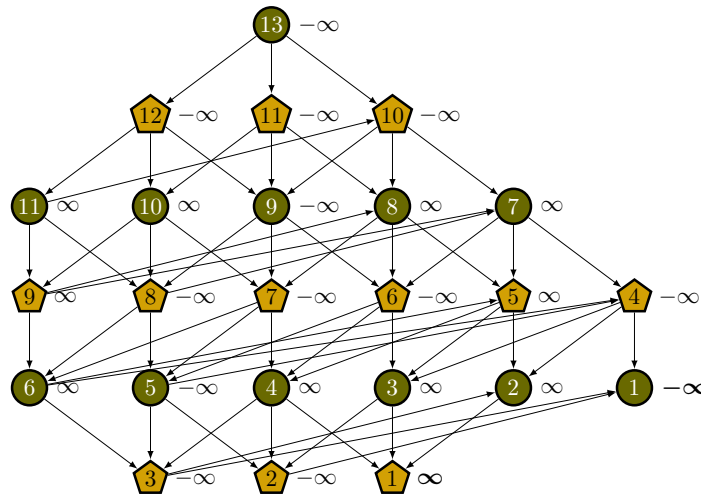
- Si c'est à **Alice** de jouer, c'est dans son intérêt de choisir comme prochain état l'état qui lui est le plus favorable, ainsi le score de e est le maximum des scores de tous les états successeurs de e .
- Si c'est à **Bob** de jouer, c'est dans son intérêt de choisir comme prochain l'état qui est le plus défavorable pour **Alice**, ainsi le score de l'état e est le minimum des scores de tous les états successeurs de e .

Exemple 2.1

Reprenons l'exemple du jeu de soustraction, et calculons les scores pour **Alice**. Les scores des états terminaux

♣. On rappelle qu'on a démontré que, dans un graphe de jeu sans circuit, si **Alice** joue un état $s \in \mathcal{A}$, alors elle est sûre de pouvoir gagner, et qu'au contraire si elle joue un état $s \in V \setminus \mathcal{A}$, alors **Bob** est sûr de pouvoir gagner ou mener à un match nul.

① et ④ sont respectivement fixés $+\infty$ et $-\infty$, car ils sont respectivement gagnant pour **Alice** et gagnant pour **Bob**. Le calcul récursif naïf des scores pour **Bob** conduit aux valeurs suivantes (le score est indiqué à côté de chaque état). On peut vérifier que le score d'un état ⑥ est le maximum des scores de ses successeurs, tandis que le score d'un état ⑤ est le minimum des scores de ses successeurs. On remarque que les états de score $+\infty$ sont exactement les attracteurs d'**Alice**, et ceux de valeur $-\infty$ sont ceux de **Bob**.



Remarque 2.2

L'observation faite sur l'exemple précédent est en fait une remarque générale : le calcul récursif naïf des scores revient au calcul des attracteurs puisque le comportement des opérateurs \forall et \exists sur l'espace $\{\text{vrai, neutre, faux}\}$ est analogue à celui des opérateurs \max et \min sur l'espace $\{\infty, 0, -\infty\}$.

Heuristique. L'algorithme récursif naïf proposé ci-avant conduit à un parcours de l'ensemble du graphe, il est donc exclu. Il nous faut donc une manière d'évaluer **directement** le score d'un état, c'est-à-dire sans faire d'appels récursifs, quitte à donner une évaluation peu précise de la qualité de l'état. Une fonction qui réalise une telle évaluation s'appelle **une heuristique**. En OCAML elle aurait la signature suivante.

```
1 | val heuristique : joueur -> etat -> score
```

Exemple 2.3

Afin d'illustrer la notion d'heuristique, on considère ici le jeu puissance 4 ♣. Ce jeu à deux joueurs se joue sur une grille à 6 lignes et 7 colonnes, dans laquelle chaque joueur essaye d'aligner (horizontalement, verticalement ou en diagonale) 4 de ses jetons. À son tour chaque joueur place un jeton en choisissant une colonne non pleine, le jeton aboutit alors dans la dernière case libre en partant du haut ♥.

Ainsi un état du jeu est entièrement déterminé par la donnée d'un tableau de dimensions 6×7 , dont chaque case indique si la case correspondante dans la grille est vide, ou remplie par un jeton d'**Alice**; ou remplie d'un jeton de **Bob**.

Afin d'évaluer un tel état, on attribue un poids à chaque case de la grille, puis on compte positivement le poids d'une case si l'on possède un jeton dans cette case, et négativement si l'autre joueur possède un jeton dans cette case. On peut par exemple fixer le poids de chaque case de la grille au nombre d'alignements faisables avec cette case, ainsi on accordera plus d'importance au placement de jetons dans les cases centrales, avec l'idée qu'elles offrent plus de possibilités d'alignement.

♣. https://fr.wikipedia.org/wiki/Puissance_4

♥. Lorsqu'on joue avec une grille verticale, le jeton tombe tout seul dans cette case

3	4	5	7	5	4	3
4	6	8	10	8	6	4
5	8	11	13	11	8	5
5	8	11	13	11	8	5
4	6	8	10	8	6	4
3	4	5	7	5	4	3

(a) Poids des cases de la grille de puissance 4

		A				
B	A					
A	B					

(b) Exemple d'état d'heuristique 9 pour Alice

FIGURE 6 – Illustration d'une heuristique pour le puissance 4

Exercice de cours 2.4

Proposer un type etat en OCAML qui permette de représenter un état du jeu puissance 4.

Exercice de cours 2.5

Justifier que le poids 4 dans la case à droite du coin inférieur droit (en gras dans la figure 6a).

Justifier que la valeur de l'heuristique de l'état représenté sur la figure 6b est bien 9.

2.2 Algorithme MinMax

L'algorithme MinMax est une variante de l'algorithme de calcul récursif naïf des scores où l'on limite la profondeur des appels récursifs. Lorsqu'on a atteint cette profondeur limite, on stoppe les appels récursifs et on évalue le score de l'état à l'aide de l'heuristique. Ainsi un tel algorithme est déterminé par l'heuristique utilisée.

On fixe $((V = V_A \sqcup V_B, E), s_0, \mathcal{O}_A, \mathcal{O}_B)$ un jeu d'accessibilité et $heur : V \rightarrow \overline{\mathbb{Z}}$ une heuristique pour ce jeu. L'algorithme MinMax qui permet de calculer le score d'un état pour Alice est alors le suivant (où succ donne les successeurs d'un sommet dans le graphe orienté (V, E)).

Algorithme 1 : MinMax simple pour Alice

Entrée : un état $s \in V$, un entier $d \in \mathbb{N}$

Sortie : un score pour l'état s calculé avec une profondeur de recherche maximale $d \in \mathbb{N}$

```

1 si succ(e) = ∅ alors // cas de base : l'état est terminal
2   si s ∈ OA alors
3     retourner +∞;
4   sinon si s ∈ OB alors
5     retourner -∞;
6   sinon
7     retourner 0;
8 sinon si d = 0 alors // pseudo cas de base : on a atteint la profondeur maximale de recherche
9   retourner heur(s)
10 sinon
11   si s ∈ VA alors // dans l'état s c'est à Alice de jouer
12     retourner max{MinMax(s', d - 1) | s' ∈ succ(s)}
13   sinon
14     retourner min{MinMax(s', d - 1) | s' ∈ succ(s)}

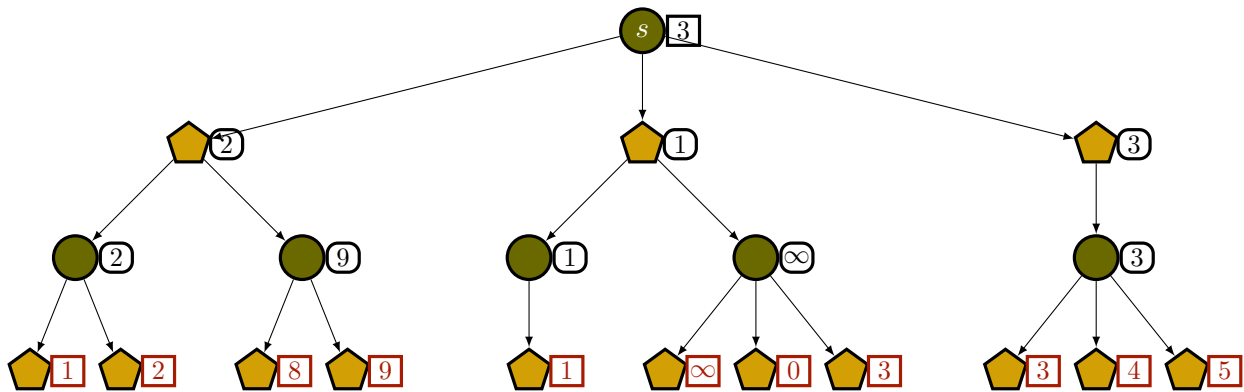
```

Remarque 2.6

On prendra garde pour implémenter l'algorithme MinMax à utiliser de la mémoïsation au besoin. En effet dans des graphes de jeux arborescents une telle mémoïsation n'apporte rien, toutefois dans des graphes non arborescents comme le jeu de la soustraction, une implémentation naïve de l'algorithme ci-dessus peut conduire à des re-calculs inutiles. En effet, la suite de coups où **Alice** retire 1 allumette puis **Bob** retire 2 allumettes conduit exactement dans le même état, disons s , que la suite de coups dans laquelle **Alice** retire 2 allumettes puis **Bob** 1 allumette. Ainsi lorsqu'on appelle l'algorithme MinMax avec une profondeur $d > 2$, on fera deux fois l'appel récursif $\text{MinMax}(s, d - 2)$.

Exemple 2.7

Ci-dessous, on dessine l'**arbre des appels récursifs** d'un appel de la fonction MinMax (Algorithme 1) sur l'état $s \in V_A$ et avec la profondeur de recherche $d = 3$. **ATTENTION** : il ne s'agit pas du graphe du jeu d'accessibilité, il est donc possible que certains nœuds de cet arbre correspondent en fait à un même état du jeu (Cf. remarque 2.6). Un nœud de la forme \bullet (resp. \blacklozenge) représente un appel récursif sur un état où c'est à **Alice** (resp. à **Bob**) de jouer, et on calcule ici les scores pour **Alice**. Les scores calculés par un appel à l'heuristique sont indiqués dans des cases \square , tandis que ceux calculés par appels récursifs sont indiqués dans des cases \circ .



2.3 Élagage $\alpha - \beta$

En inspectant l'exécution de l'algorithme MinMax (Algorithme 1) sur l'exemple en fin de section précédente, on peut remarquer que certains calculs inutiles ont eu lieu. En effet, lorsqu'on calcule le score maximum des successeurs d'un état s où c'est à **Alice** de jouer, la valeur exacte du score d'un de ses successeurs s' n'est utile que si elle est susceptible de réaliser le maximum. En particulier, puisque $s' \in V_B$, le score de s' s'obtient par un calcul de minimum, au cours duquel on a un majorant de ce score : dès lors que le majorant nous assure que le score de s' ne réalisera pas le maximum parmi les successeurs de s , on peut stopper le calcul de ce minimum. On peut alors ignorer s' puisque son score "ne participe pas" au maximum qui définit le score de s .

Exemple 2.8

Le schéma ci-dessous représente les valeurs de scores déjà calculées lors de l'appel récursif sur l'état \star lors de l'exécution de l'algorithme MinMax sur l'exemple précédent. Quel que soit le score pour \star , le score de \blacklozenge (i.e. la valeur de ?) sera nécessairement ≤ 1 puisqu'à cet "étage" on calcule un minimum, et qu'on a déjà trouvé un score valant 1. En particulier la valeur de ? ne réalisera pas le maximum à son étage, puisqu'on y a déjà trouvé un nœud de score 2, soit strictement meilleur. Il est donc inutile de continuer le calcul du score de \star car le maximum qui définit le score de \bullet ne sera pas réalisé par ?.

De plus, si \star avait des frères à droite, il serait aussi inutile de les calculer pour les mêmes raisons. Ainsi on interrompt le calcul du score de \blacklozenge et on lui affecte le score 1 (même si son score au sens de l'algorithme MinMax précédent était peut-être moindre).

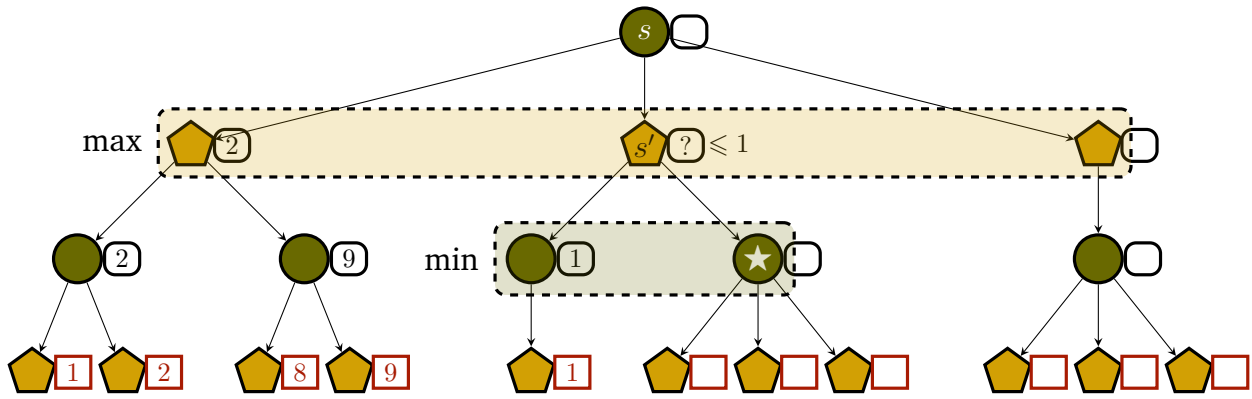


FIGURE 7 – Exemple où l'on interrompt un calcul de minimum

Une telle occurrence de calcul inutile peut aussi avoir lieu dans le cas symétrique. En effet, lorsqu'on calcule le score minimum des successeurs d'un état s où c'est à **Bob** de jouer, la valeur exacte du score d'un de ses successeurs s' n'est utile que si elle est susceptible de réaliser le minimum. En particulier, puisque $s' \in V_A$, le score de s' s'obtient par un calcul de maximum, au cours duquel on a un minorant de ce score : dès lors que le minorant nous assure que le score de s' ne réalisera pas le minimum parmi les successeurs de s , on peut stopper le calcul de ce maximum. On peut alors ignorer s' puisque son score "ne participe pas" au minimum qui définit le score de s .

Exemple 2.9

Le schéma ci-dessous représente les valeurs de scores déjà calculées lors de l'appel récursif sur l'état \star lors de l'exécution de l'algorithme MinMax sur l'exemple précédent. Quel que soit le score pour \star , le score de s (i.e. la valeur de $?$) sera nécessairement ≥ 8 puisqu'à cet "étage" on calcule un maximum, et qu'on a déjà trouvé un score valant 8. En particulier la valeur de $?$ ne réalisera pas le minimum à son étage, puisqu'on y a déjà trouvé un nœud de score 2, soit strictement meilleur. Il est donc inutile de continuer le calcul du score de \star car le maximum qui définit le score de s ne sera pas réalisé par $?$.

On interrompt alors le calcul du score de s et on lui affecte le score 8.

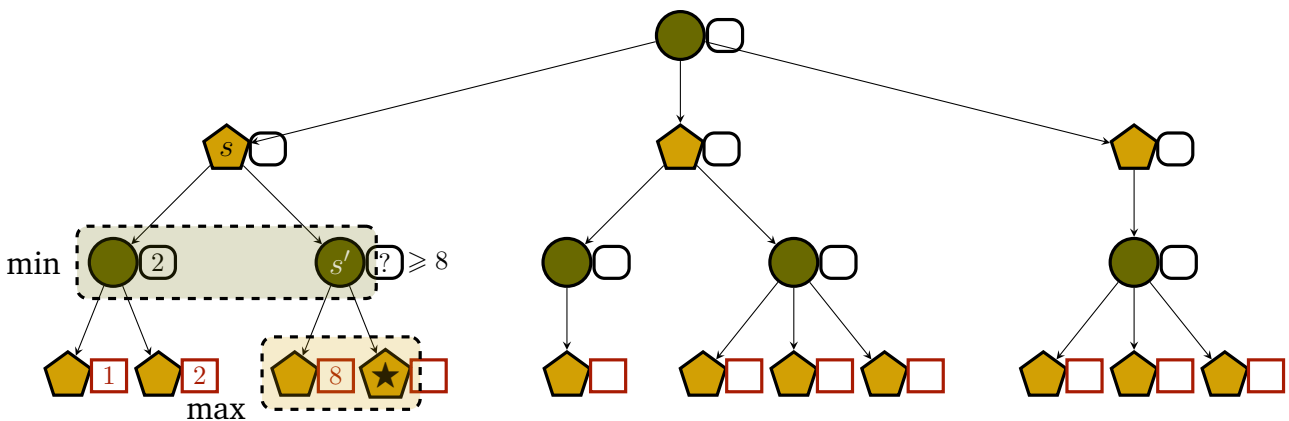


FIGURE 8 – Exemple où l'on interrompt un calcul de maximum

Vocabulaire 2.10

On parle d'**élagage** α - β pour désigner le fait d'interrompre un calcul paresseux de minimum qui sera utilisé dans un maximum ou vice-versa.

Exemple 2.11

La figure 9 résume l'exécution de l'algorithme MinMax avec élagage α - β . Les nœuds correspondant à des

appels récursifs économisés grâce à l'élagage sont encadrés. Pour les autres, le score résultant de l'appel récursif correspondant est indiqué à côté du nœud.

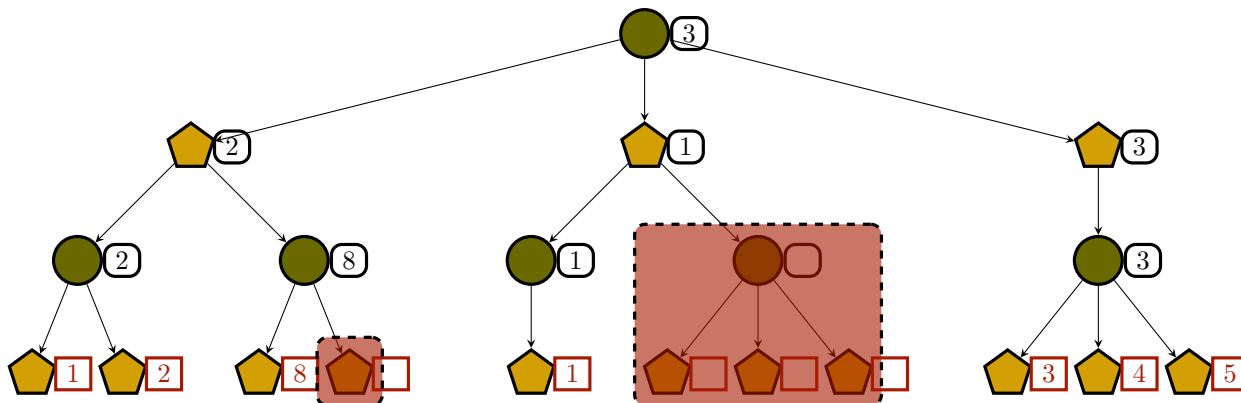


FIGURE 9 – Ensemble des appels récursifs que l'on peut éviter grâce à l'élagage α - β

Remarque 2.12

Les scores affectés à certains états par l'algorithme avec élagage ne correspondent pas aux score calculés lors de l'exécution de l'algorithme initial. Toutefois ce sont des scores dont la valeur n'a, en fait, pas servi au calcul principal, c'est-à-dire le score de l'état donné au premier appel récursif (celui en racine de l'arbre, celui pour lequel on calcule le score afin de décider si l'on joue vers cet état).

Cette amélioration de l'algorithme de Minmax peut être implémentée en ajoutant aux appels récursifs de l'algorithme MinMax (Algorithme 1) deux variables supplémentaires : α et β ♣ contenant, pour α une valeur en dessous de laquelle il n'est pas nécessaire de continuer, et β une valeur au dessus de laquelle il n'est pas nécessaire de continuer. Cet algorithme est initialisé avec les valeurs

♣. du nom de l'algorithme

$\alpha = -\infty$ et $\beta = \infty$.

Algorithme 2 : MinMax avec élagage α - β pour **Alice**

Entrée : un état $s \in V$, un entier $d \in \mathbb{N}$, deux valeurs α et β de $\overline{\mathbb{Z}}$

Sortie : un score pour l'état s pour **Alice** calculé avec une profondeur de recherche maximale d

```
1 si succ( $s$ ) =  $\emptyset$  alors
2   | si  $s \in \mathcal{O}_A$  alors
3   |   | retourner  $+\infty$ ;
4   | sinon si  $s \in \mathcal{O}_B$  alors
5   |   | retourner  $-\infty$ ;
6   | sinon
7   |   | retourner 0;
8 sinon si  $d = 0$  alors
9   | retourner heur( $s$ )
10 sinon
11   | si  $s \in V_A$  alors
12   |   |  $v \leftarrow -\infty$ ;
13   |   | pour tout  $s' \in \text{succ}(s)$  faire
14   |   |   |  $v \leftarrow \max(v, \text{Minmax}(s', d - 1, v, +\infty))$ ;
15   |   |   | si  $v \geq \beta$  alors
16   |   |   |   | retourner  $v$ 
17   | sinon
18   |   |  $v \leftarrow +\infty$ ;
19   |   | pour tout  $s' \in \text{succ}(s)$  faire
20   |   |   |  $v \leftarrow \min(v, \text{Minmax}(s', d - 1, -\infty, v))$ ;
21   |   |   | si  $v \leq \alpha$  alors
22   |   |   |   | retourner  $v$ 
```
