IV – Jeux d'accessibilité à deux joueurs

M. Solnon

PT, Lycée Gustave Eiffel, Dijon

13 octobre 2026

Exemple : jeu de Nim



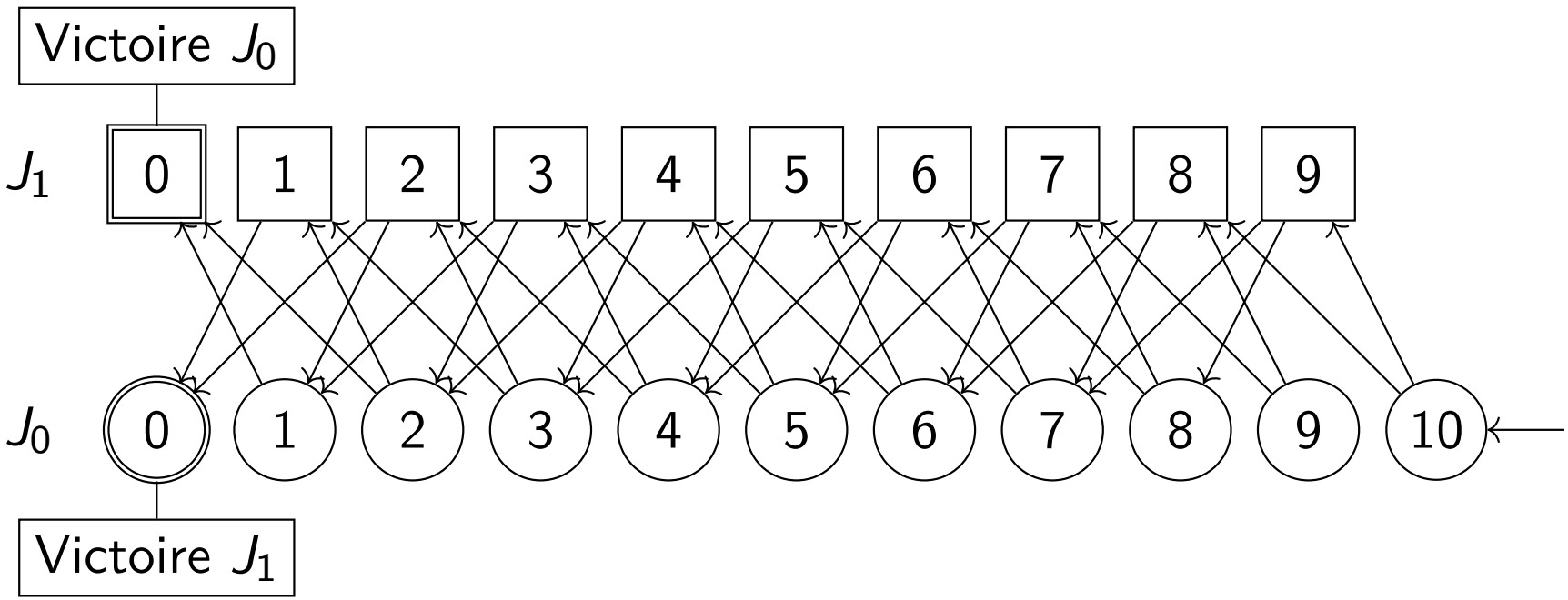
Principe

- n bâtonnets au départ (ici : $n = 10$) ;
- chaque joueur enlève un nombre $i \in R$ de bâtonnets (ici : $R = \{ 1, 2 \}$) ;
- celui qui ne peut plus jouer a perdu.

Autres jeux

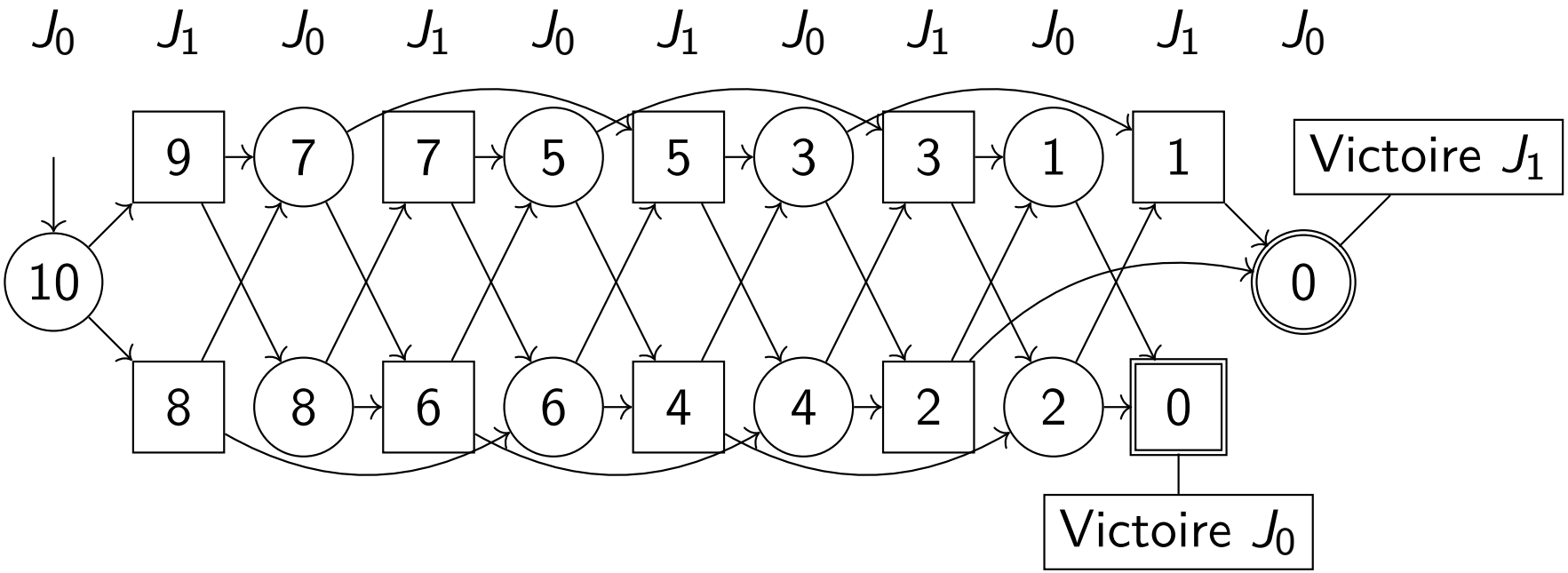
Morpion, puissance 4, hex, dames *etc.*

Exemple du graphe du jeu de Nim



$n = 10,\ R = \{1, 2\}$

On pourrait lui donner une autre forme

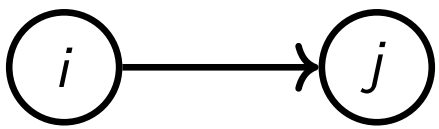


Rappels sur les graphes

Graphe orienté

Couple (S, A) où

- S est l'ensemble des sommets ;
- $A \subset S \times S$ est l'ensemble des arêtes ;
- $(i, j) \in A$ s'il y a une arête qui relie le sommet i au sommet j.



Graphe biparti

- S partitionné en deux parties S_0 et S_1
- Aucune arête ne relie deux sommets de S_0, ou deux sommets de S_1.

Représentation par un graphe

Principe

Jeu représenté par un graphe orienté (S, A) où

- une position du jeu = un sommet du graphe ;
- un coup jouable = une arête du graphe (orienté) ;
- $S = S_0 \sqcup S_1$, où
  - S_0 est l'ensemble des sommets contrôlés par J_0 ;
  - S_1 est l'ensemble des sommets contrôlés par J_1 ;
- condition de victoire : deux parties non vides $V_0, V_1 \subset S$ vérifiant $V_0 \cap V_1 = \varnothing$.

Graphe du jeu de Nim

(k, i) : le joueur i joue, avec k bâtonnets placés devant lui

$$S_0 = \{ (k, 0) \mid 0 \leqslant k \leqslant n \},$$
$$S_1 = \{ (k, 1) \mid 0 \leqslant k \leqslant n - 1 \}.$$

si $q \in R$ et si $q \leqslant k \leqslant n$, on a les arêtes

$$(k, 0) \to (k - q, 1),$$
$$(k, 1) \to (k - q, 0).$$

Positions gagnantes :

$$V_0 = \{ (k, 1) \mid 0 \leqslant k < \min(R) \},$$
$$V_1 = \{ (k, 0) \mid 0 \leqslant k < \min(R) \}.$$

Exercice 2.1.10

Consigne

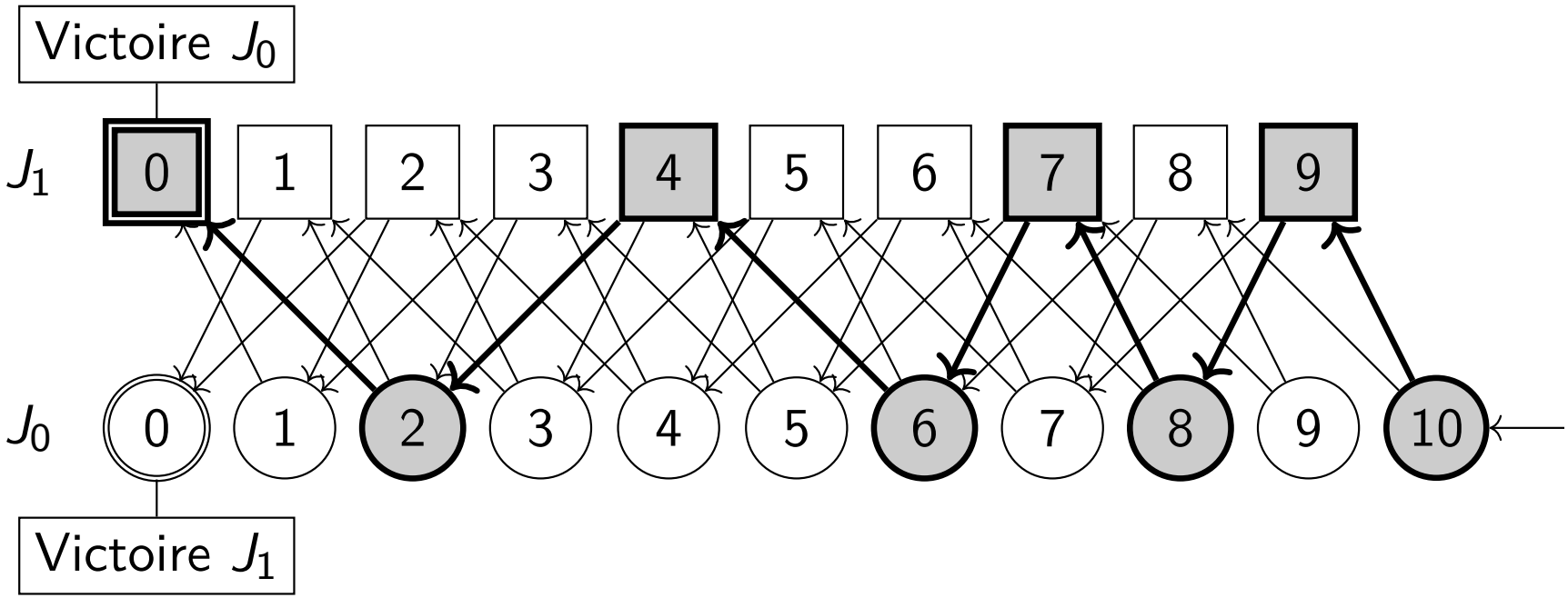
Créer le graphe de ce jeu de Nim en Python et les conditions de victoire ($n = 10$, $R = \{1, 2\}$ pour commencer, puis écrire une fonction prenant en argument n et R).
On utilisera un dictionnaire d'adjacence.

Indication

Utiliser un dictionnaire avec des valeurs `None` pour créer un ensemble.
Exemple pour $\{(0, 0), (1, 2)\}$:

```
d = {(0,0):None,
     (1,2):None}
```

Exemple du jeu de Nim



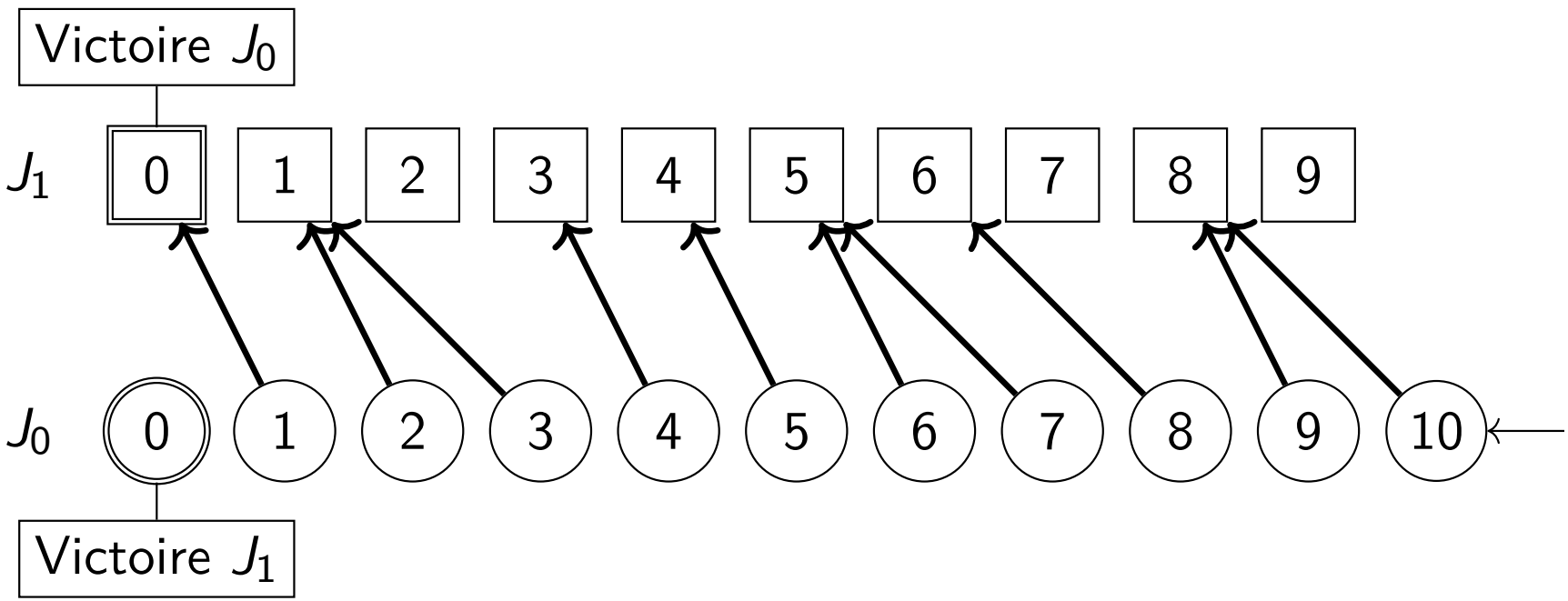
Partie gagnante pour J_0

Stratégie sans mémoire

Définition formelle (pour J_0)

$s : S_0 \to S$ telle que $\forall x \in S_0$, $(x, s(x)) \in A$

Exemple



Stratégies et positions gagnantes

Stratégie gagnantes

Une stratégie est *gagnante* si toutes les parties jouées selon cette stratégie sont gagnantes.

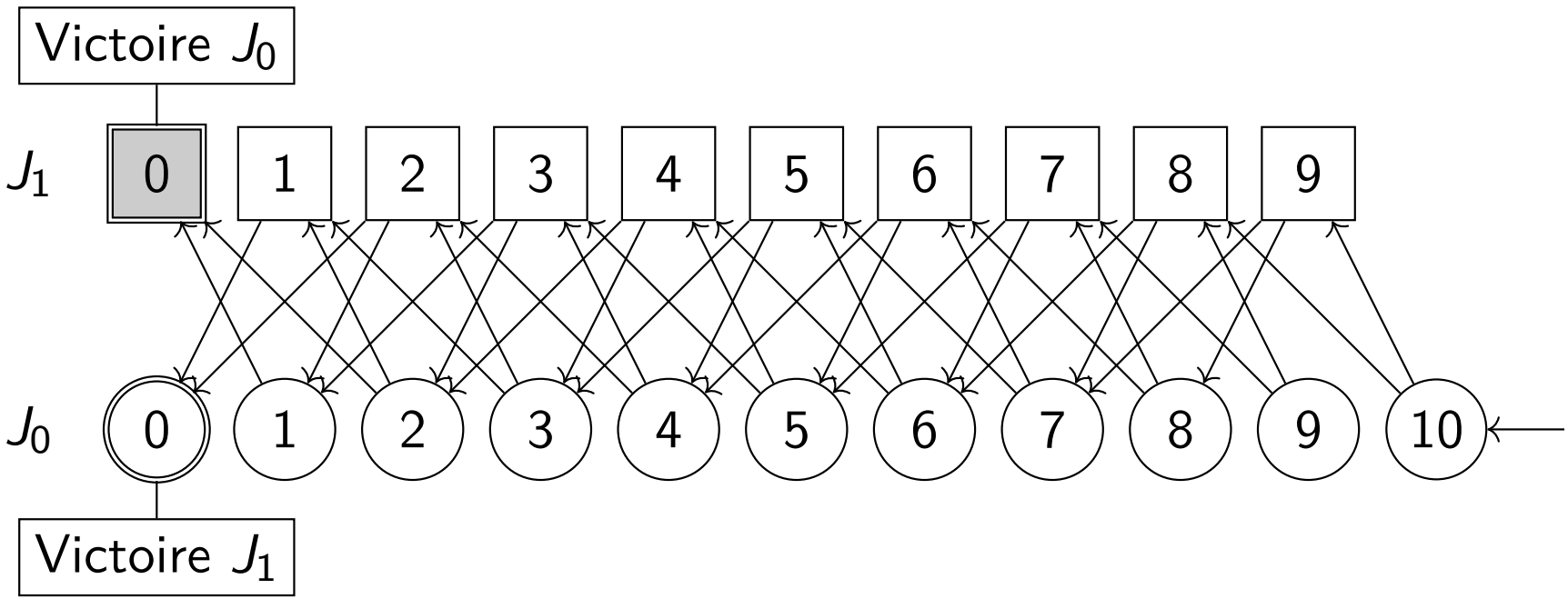
Position gagnante

Une position est *gagnante* s'il existe une stratégie gagnante lorsque la partie débute sur cette position.

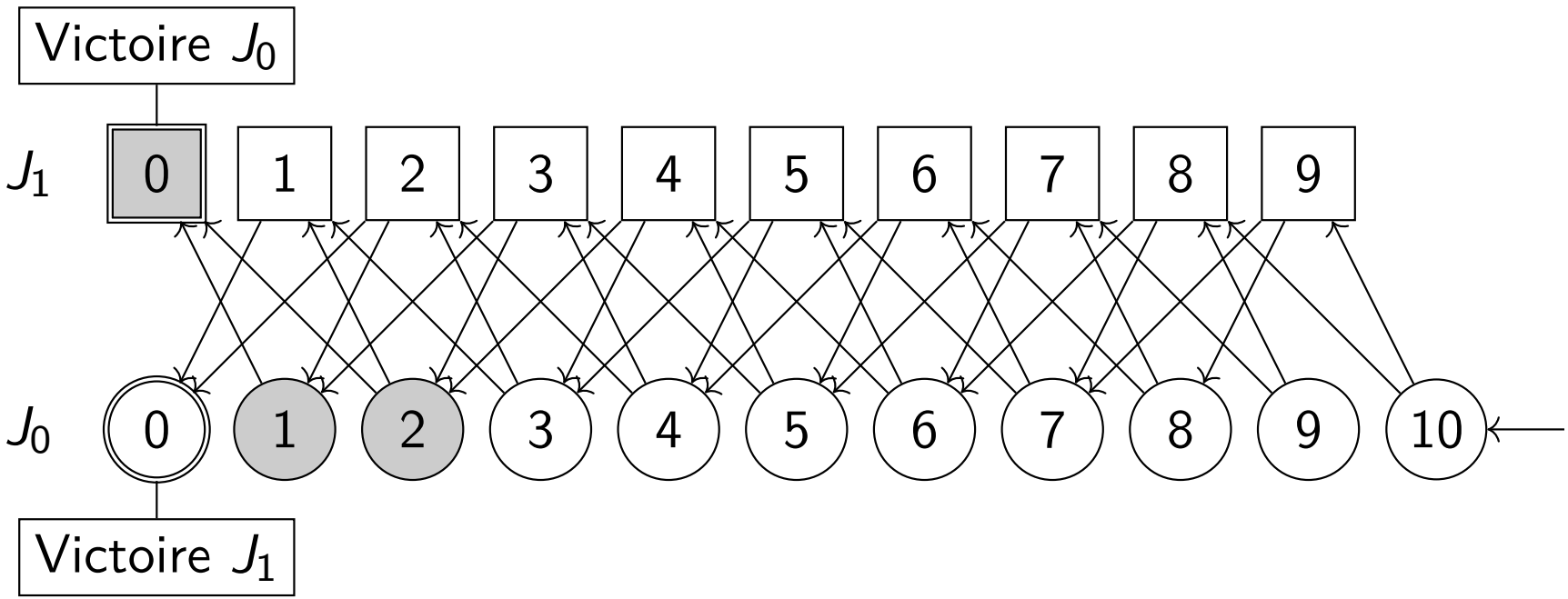
Positions gagnantes évidentes

Les positions de V_0 sont gagnantes pour J_0.
Les positions de V_1 sont gagnantes pour J_1.

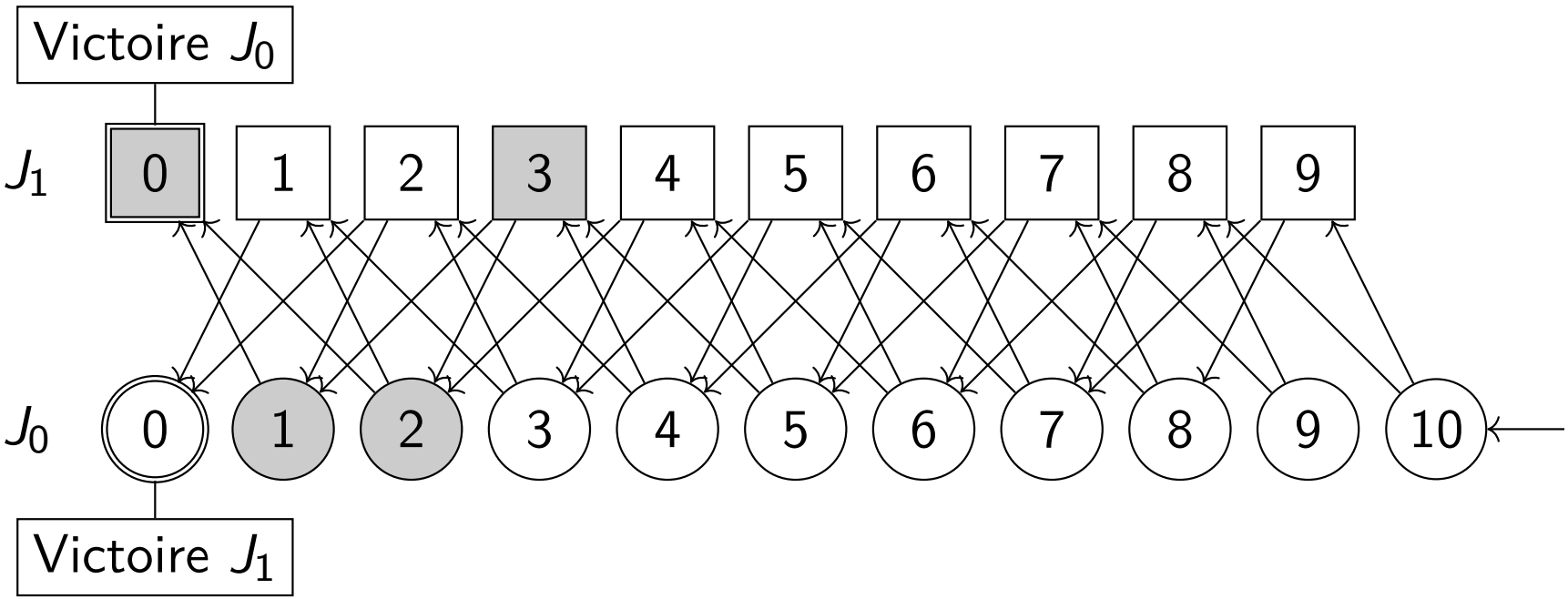
Déterminer deux nouvelles positions gagnantes pour J_0



Déterminer une nouvelle position gagnantes pour J_0



On pourrait continuer...



Formalisation

Attracteurs

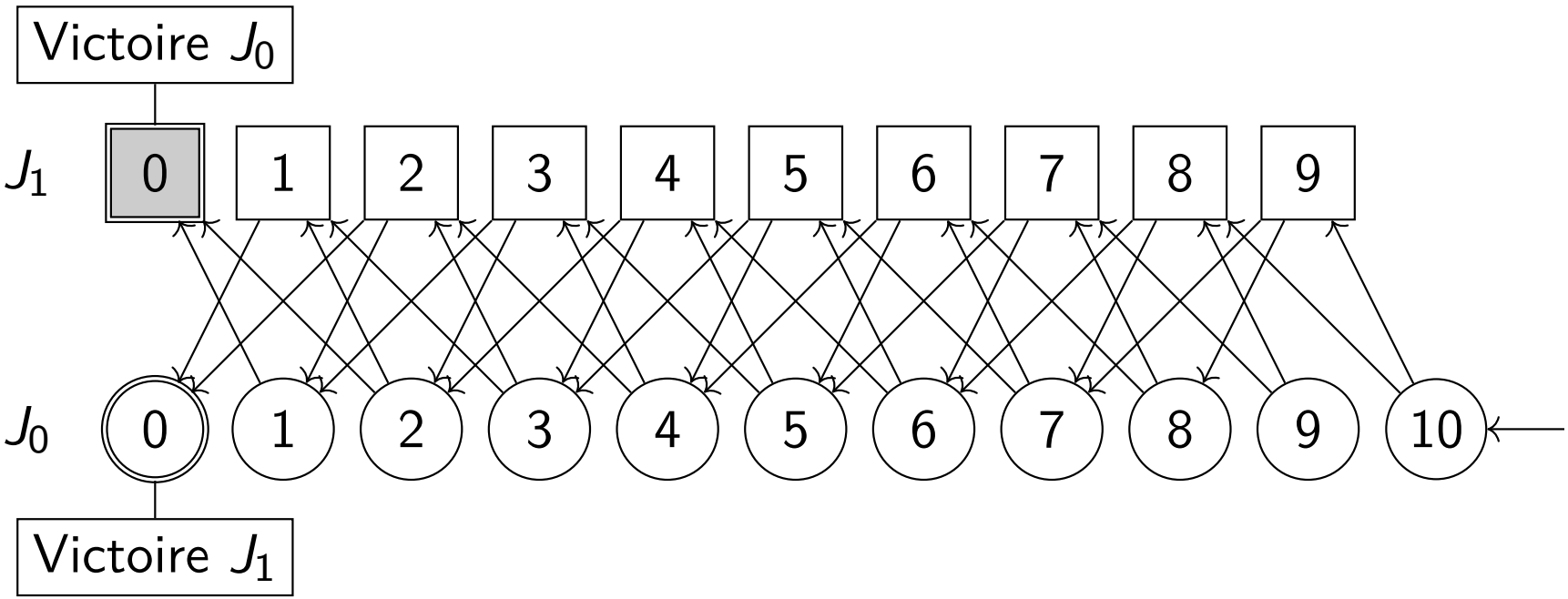
- $\mathscr{A}_n^0$: ensemble des positions gagnantes pour J_0 en moins de n coups
- $\mathscr{A}^0$: ensemble des positions gagnantes pour J_0

Formalisation

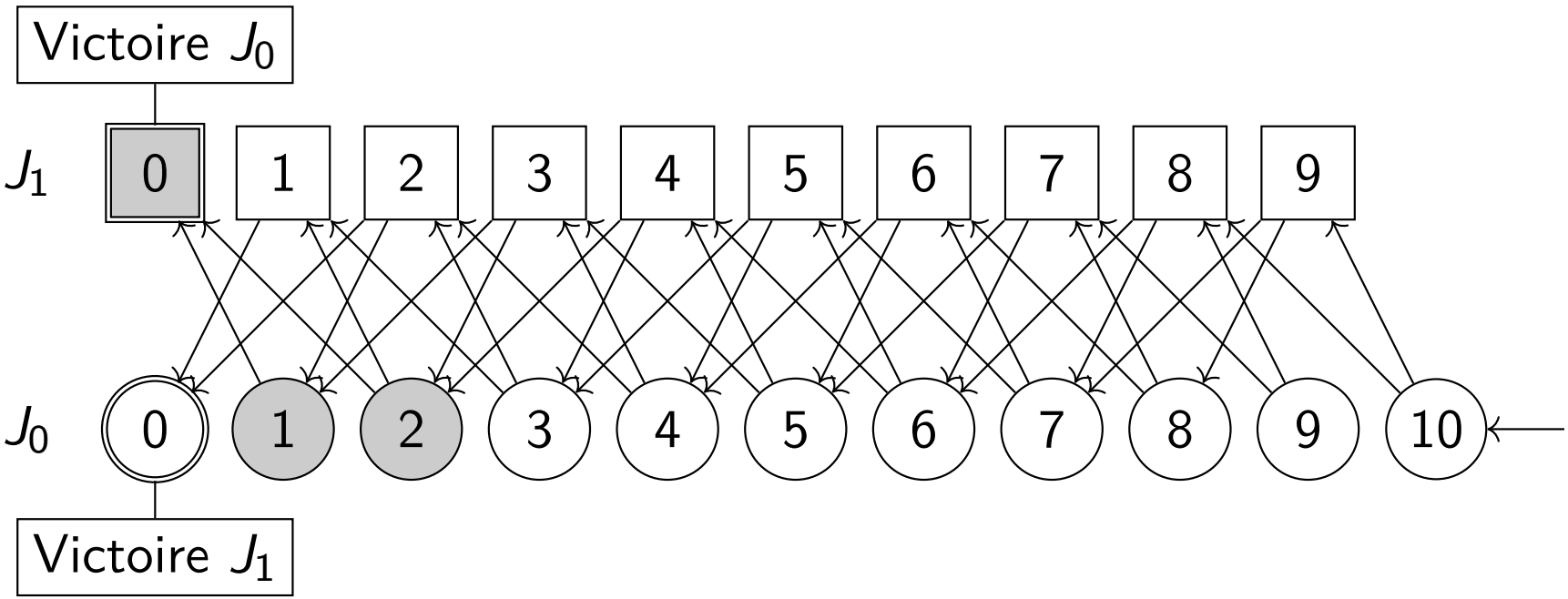
- $\mathscr{A}_0^0 = V_0$, $\mathscr{A}^0 = \bigcup_{n \in \mathbb{N}} \mathscr{A}_n^0$
-

$$\mathscr{A}_{n+1}^0 = \mathscr{A}_n^0 \cup \left\{ x \in S_0 \;\middle|\; \exists y \in \mathscr{A}_n^0,\ (x,y) \in A \right\}$$
$$\cup \Big\{ x \in S_1 \mid \exists y \in S,\ (x,y) \in A$$
$$\text{et} \quad \forall y \in S,\ (x,y) \in A \Rightarrow y \in \mathscr{A}_n^0 \Big\}.$$

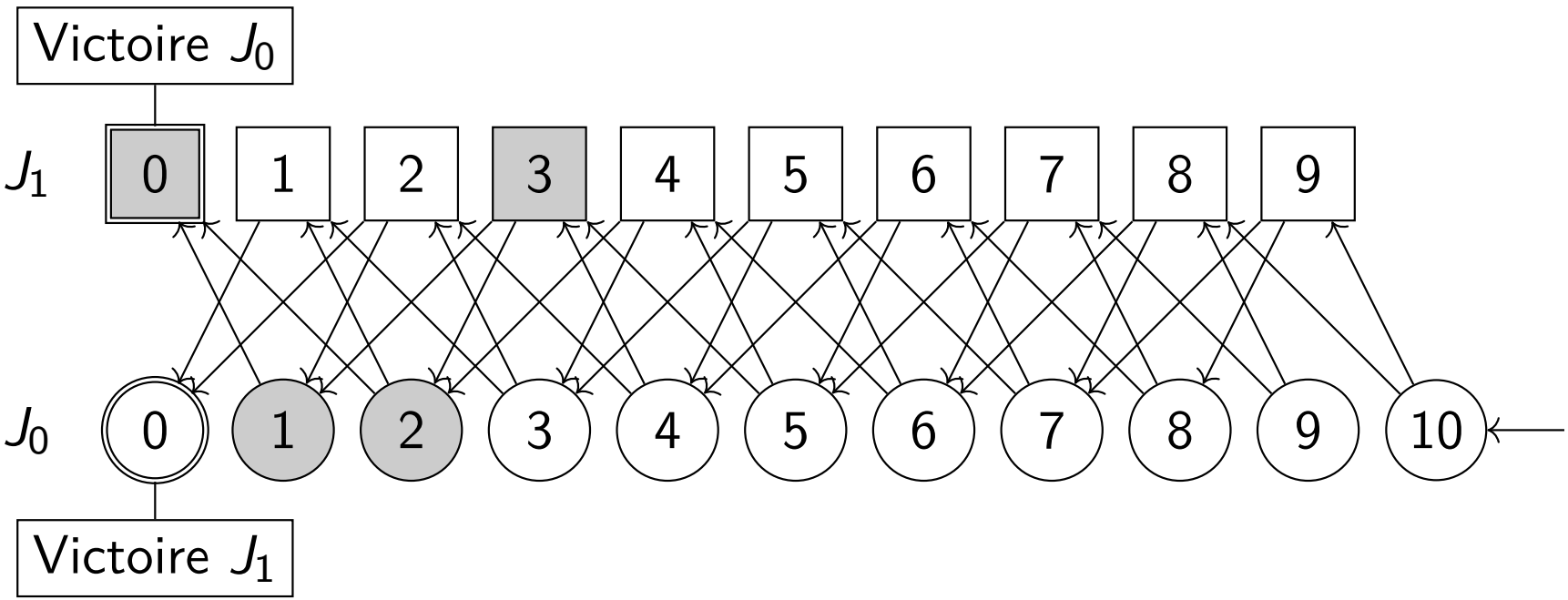
$\mathscr{A}_0^0$



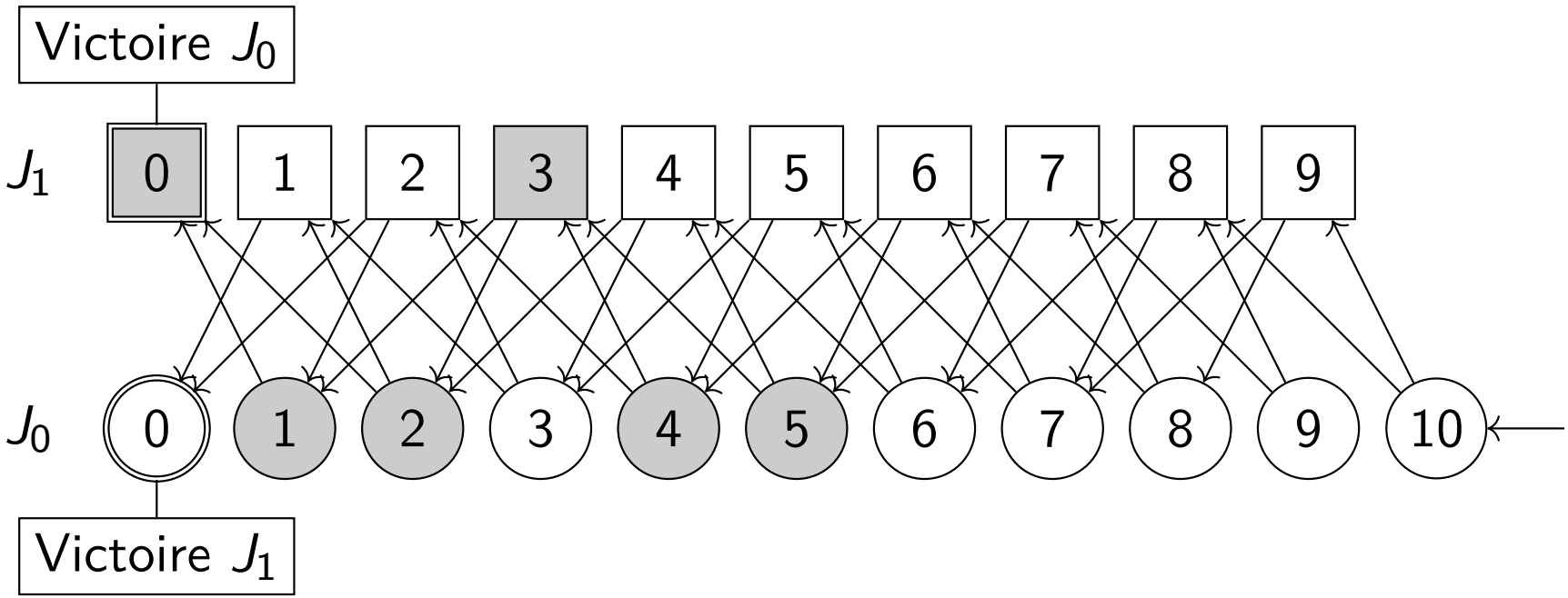
$\mathscr{A}_0^1$



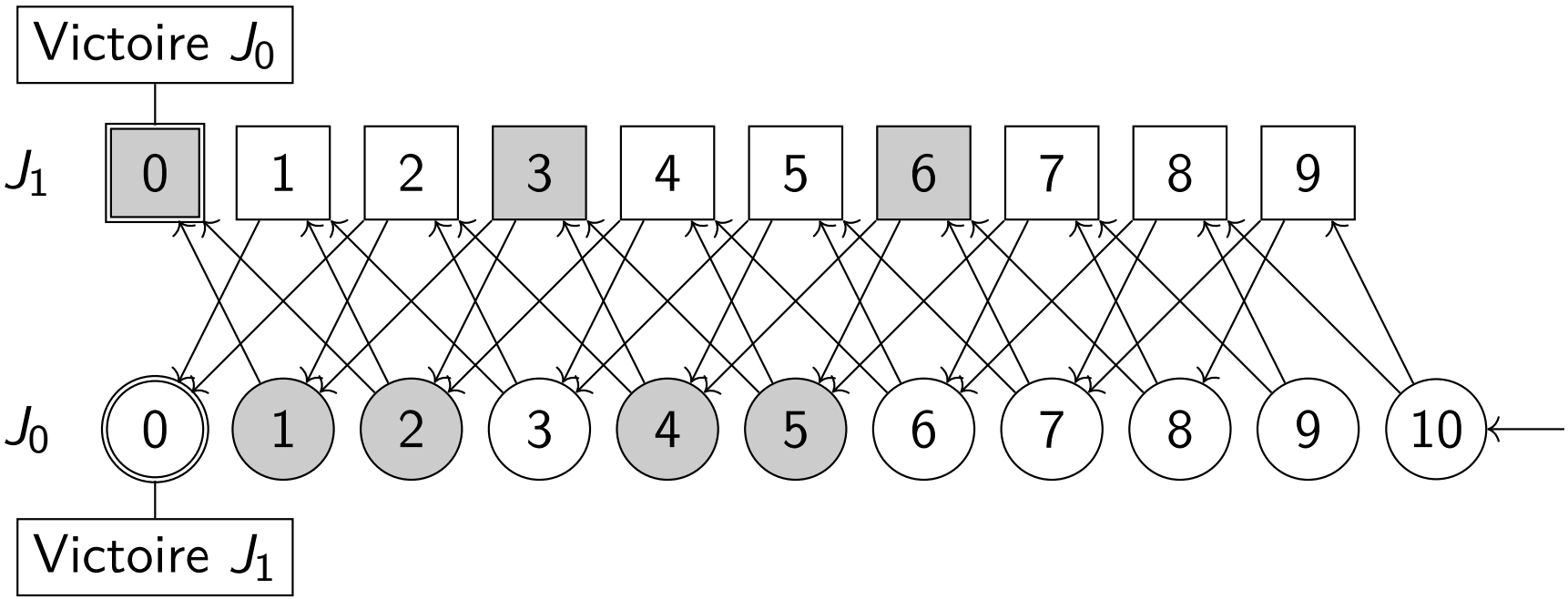
$\mathscr{A}_0^2$



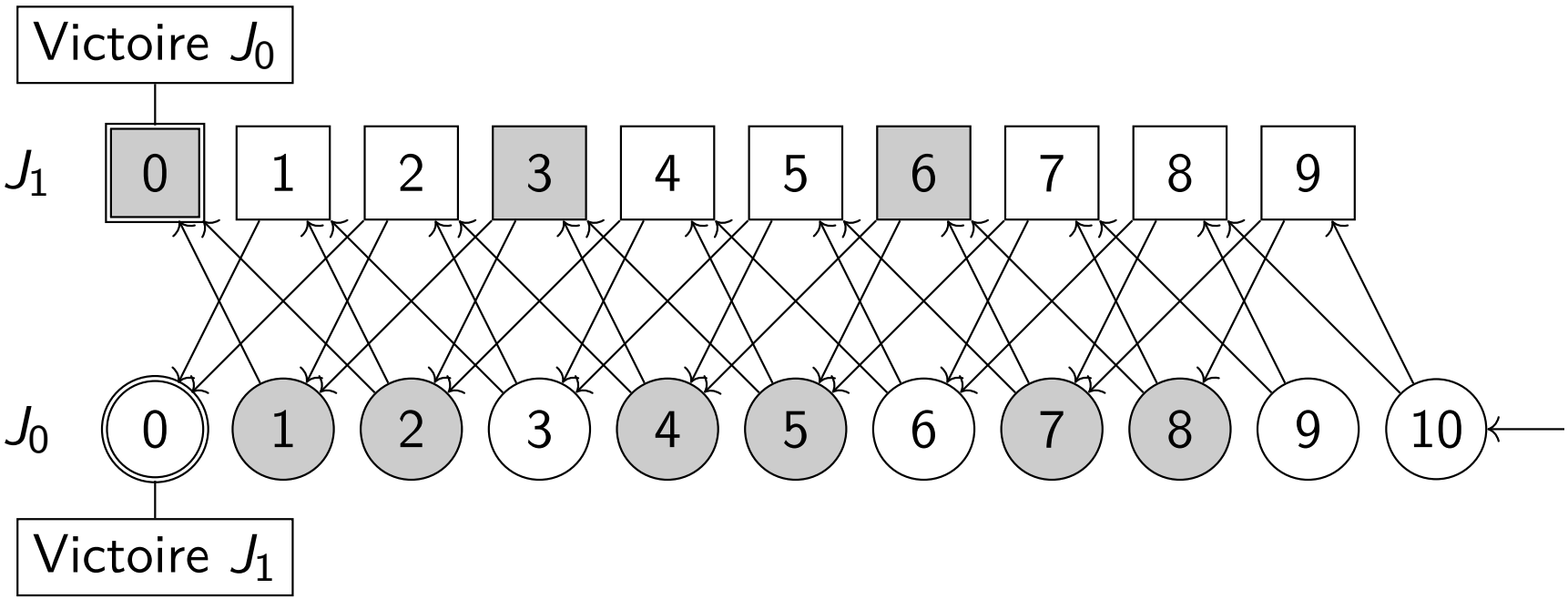
$\mathscr{A}_0^3$



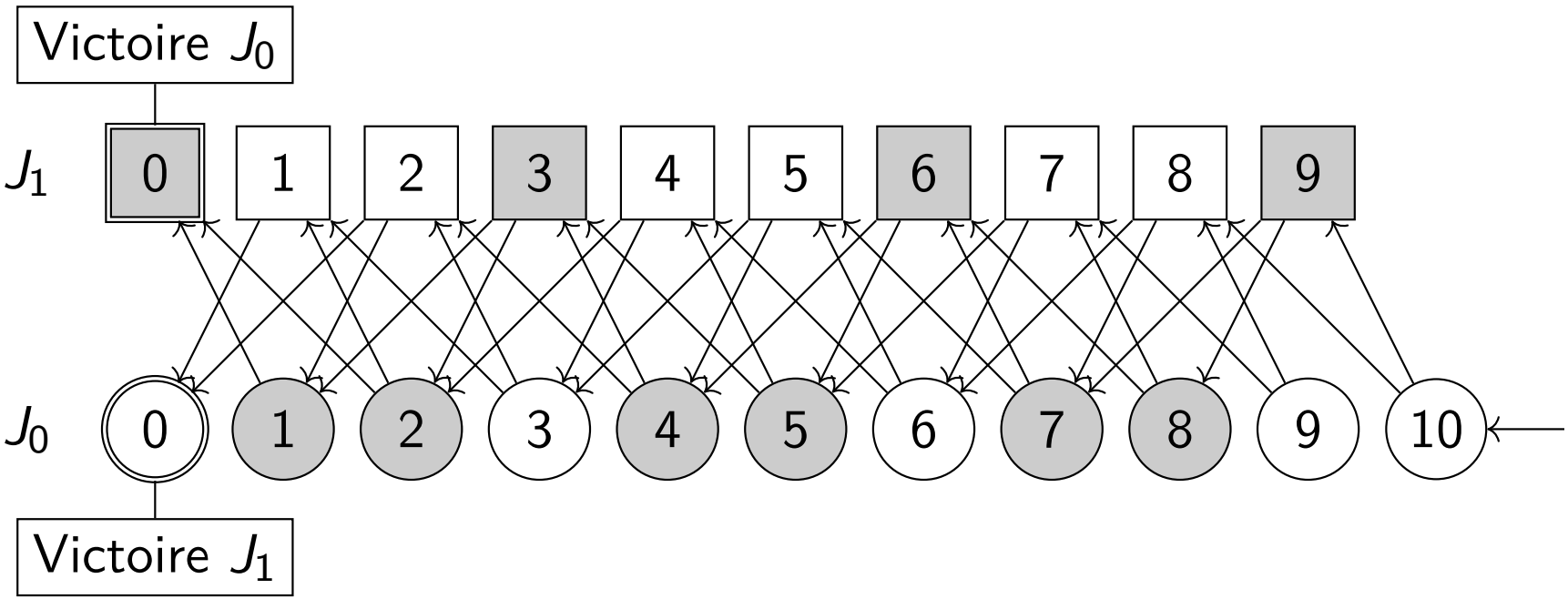
$\mathscr{A}_0^4$



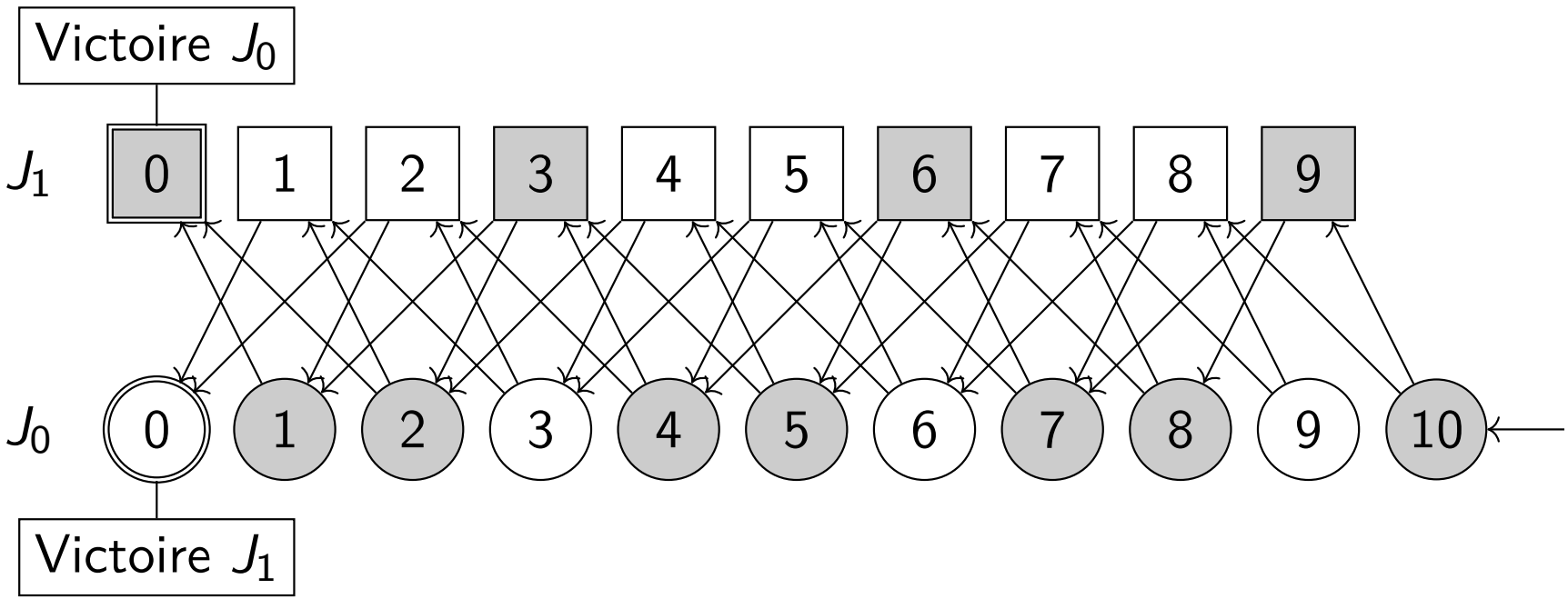
$\mathscr{A}_0^5$



$\mathscr{A}_0^6$



$$\mathscr{A}_0^7 = \mathscr{A}_0$$



Stratégie gagnante pour J_0

Objectif

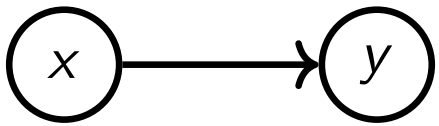
Si $x \in S_0$, construire $s(x) \in A$ de manière gagnante (si possible).

Idée

- Si $x \in \mathscr{A}_0^n$, trouver $y \in \mathscr{A}_0^{n-1}$ tel que $(x, y) \in A$, poser $y = s(x)$;
- si $x \in \mathscr{A}_1$, trouver $y \in S$ tel que $(x, y) \in A$ (quelconque), poser $y = s(x)$;
- sinon, trouver $y \in S \setminus \mathscr{A}_1$ tel que $(x, y) \in A$, poser $y = s(x)$.

Rappels

Prédécesseur / successeur



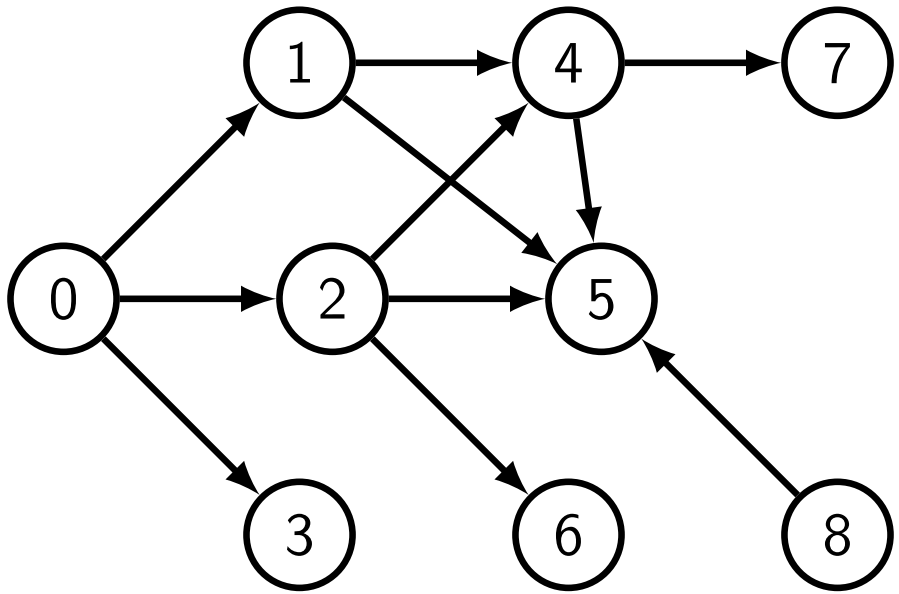
- x est un prédécesseur (ou un antécédent) de y
- y est un successeur de x

Degrés

- Degré entrant de x ($d^-(x)$) : nombre de prédécesseurs de x
- Degré sortant de x ($d^+(x)$) : nombre de successeurs de x

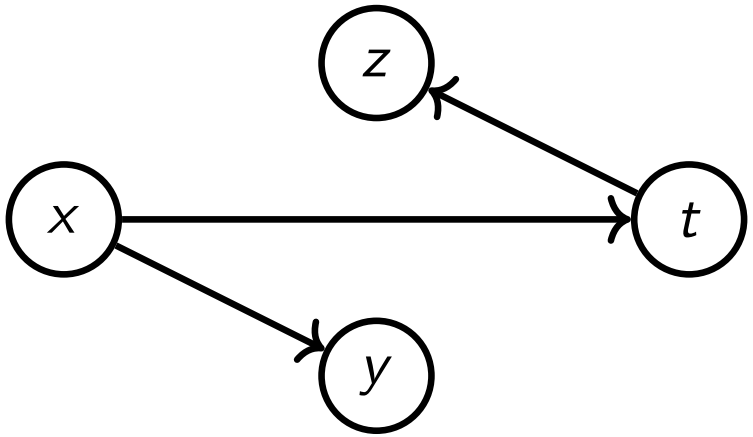
Exercice 3.1.3

Calculer les degrés entrants et sortants des sommets du graphe suivant.
Écrire en python sa représentation par dictionnaire de listes d'adjacence.

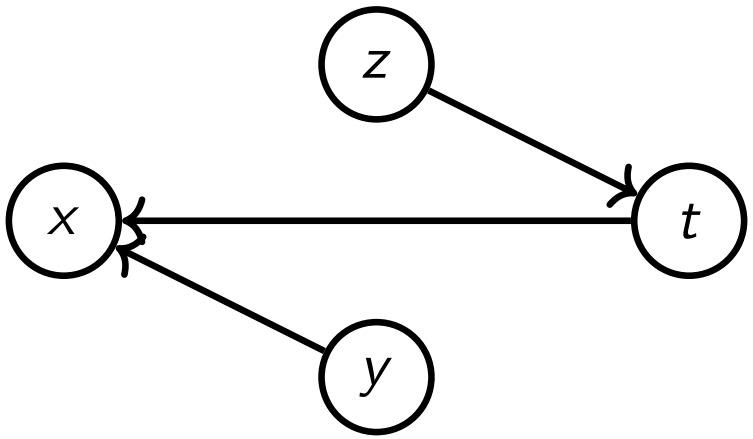


Graphe des prédécesseurs

Le graphe



est renversé en



Représentation des données

Graphes

- Représentés par des dictionnaires de listes d'adjacence
- Clefs : sommets/positions
- Valeur associée à une clef : liste des successeurs

Ensembles

- Représentés par des dictionnaires
- Clefs : éléments
- Valeur associée à une clef : `None`

(on contourne le fait que la structure `set` n'est pas au programme)

Principe du calcul des attracteurs (1)

- Calculer le degré sortant de chaque position.
- `A` : dictionnaire d'adjacence du graphe.

Principe du calcul des attracteurs (1)

- Calculer le degré sortant de chaque position.
- `A` : dictionnaire d'adjacence du graphe.

```
Fonction dplus( A ) :
    d ← dictionnaire vide
    Pour chaque x De A, Faire
        d[x] ← longueur de A[x]
    Renvoyer d
Fin Fonction
```

Principe du calcul des attracteurs (2)

- Calculer les prédécesseurs de chaque sommet (*i.e.* on renverse le graphe).
- `A` : dictionnaire d'adjacence du graphe.

Principe du calcul des attracteurs (2)

```
Fonction renverse_graphe( A ):
    B ← dictionnaire vide
    Pour chaque x De A, Faire
        B[x] ← liste vide
    Pour chaque x De A, Faire
        Pour chaque y De A[x], Faire
            Ajouter x à B[y]
    Renvoyer B
Fin Fonction
```

Principe du calcul des attracteurs (3)

- Attracteur : ensemble.
- Pour chaque sommet, on a calculé le nombre de successeurs non gagnants (initialisé au degré sortant).
- Pour chaque (nouveau) sommet que l'on sait être gagnant :
  - on l'ajoute à l'attracteur en cours de construction ;
  - on diminue de 1 le compteur de successeurs non gagnants de chaque prédécesseur du sommet ;
  - pour chacun de ces prédécesseurs, s'il est dans S_0 ou si son compteur est nul, c'est un sommet gagnant : on le rajoute récursivement à l'attracteur.

Principe du calcul des attracteurs (3)

- `s` : nouvelle position gagnante
- `d` : dictionnaire des compteurs de successeurs non gagnants (initialisé par la fonction `dplus`)
- `B` : dictionnaire d'adjacence du graphe des antécédents (renversement de `A`).

Principe du calcul des attracteurs (3)

```
Fonction marque_propage(s,d,B,S0,attracteur):
    Si s ∉ attracteur Alors
        attracteur[s] ← valeur arbitraire
        Pour chaque x De B[s], Faire
            d[x] diminué de 1
            Si x ∈ S0 ou d[x]=0 Alors
                marque_propage(x,d,B,S0,attracteur)
Fin Fonction
```

Principe du calcul des attracteurs (fin)

```
Fonction calcule_attracteur( A,S0,V0 ):
    d ← dplus(A)
    B ← renverse_graphe(A)
    attracteur ← dictionnaire vide
    Pour chaque s De V0, Faire
        marque_propage(s,d,B,S0,attracteur)
    Renvoyer attracteur
Fin Fonction
```

Principe du calcul d'une stratégie gagnante (1)

- Ajouter aux attracteurs un indicateur du nombre de coups permettant de gagner la partie.
- Si `attracteur[position]` vaut `n`, le joueur peut gagner en moins de `n` coups.

Principe du calcul d'une stratégie gagnante (2)

```
Fonction
marque_propage2(s,d,B,S0,attracteur,n):
    Si s ∉ attracteur Alors
        attracteur[s] ← n
        Pour chaque x De B[s], Faire
            d[x] diminué de 1
            Si x ∈ S0 ou d[x]=0 Alors
                marque_propage2(x,d,B,S0,attracteur,n+1)
Fin Fonction
```

Principe du calcul d'une stratégie gagnante (3)

```
Fonction calcule_attracteur2(A,S0,V0):
    d ← dplus(A)
    B ← renverse_graphe(A)
    attracteur ← dictionnaire vide
    Pour chaque s De V0, Faire
        marque_propage2(s,d,B,S0,attracteur,0)
    Renvoyer attracteur
Fin Fonction
```

Principe du calcul d'une stratégie gagnante (4)

Rappel du principe pour J_0 sur une position s

- Si $s \in \mathscr{A}_0$, J_0 joue sur un successeur dans $\mathscr{A}_0$ « plus proche » de la victoire.
- Si $s \in \mathscr{A}_0$, J_0 sur n'importe quel successeur.
- Sinon, J_0 joue sur un successeur qui n'est pas dans $\mathscr{A}_1$.

Arguments

- `A` : dictionnaire d'adjacence du graphe
- `attr0`, `attr1` : attracteurs de J_0 et de J_1 (avec indicateurs de distance à la victoire)
- `s` : sommet/position pour lequel on calcule la stratégie

Principe du calcul d'une stratégie gagnante (fin)

```
Fonction strategie(A,attr0,attr1,s):
    Si s∈attr0 Alors
        Pour chaque x De A[s], Faire
            Si x∈attr0 et attr0[x]<attr0[s] Alors
                Renvoyer x
```

Principe du calcul d'une stratégie gagnante (fin)

```
    Sinon si s∈attr1 Alors
        Renvoyer un élément quelconque de A[s]
    Sinon
        Pour chaque x De A[s], Faire
            Si x∉ attr1 Alors
                Renvoyer x
Fin Fonction
```

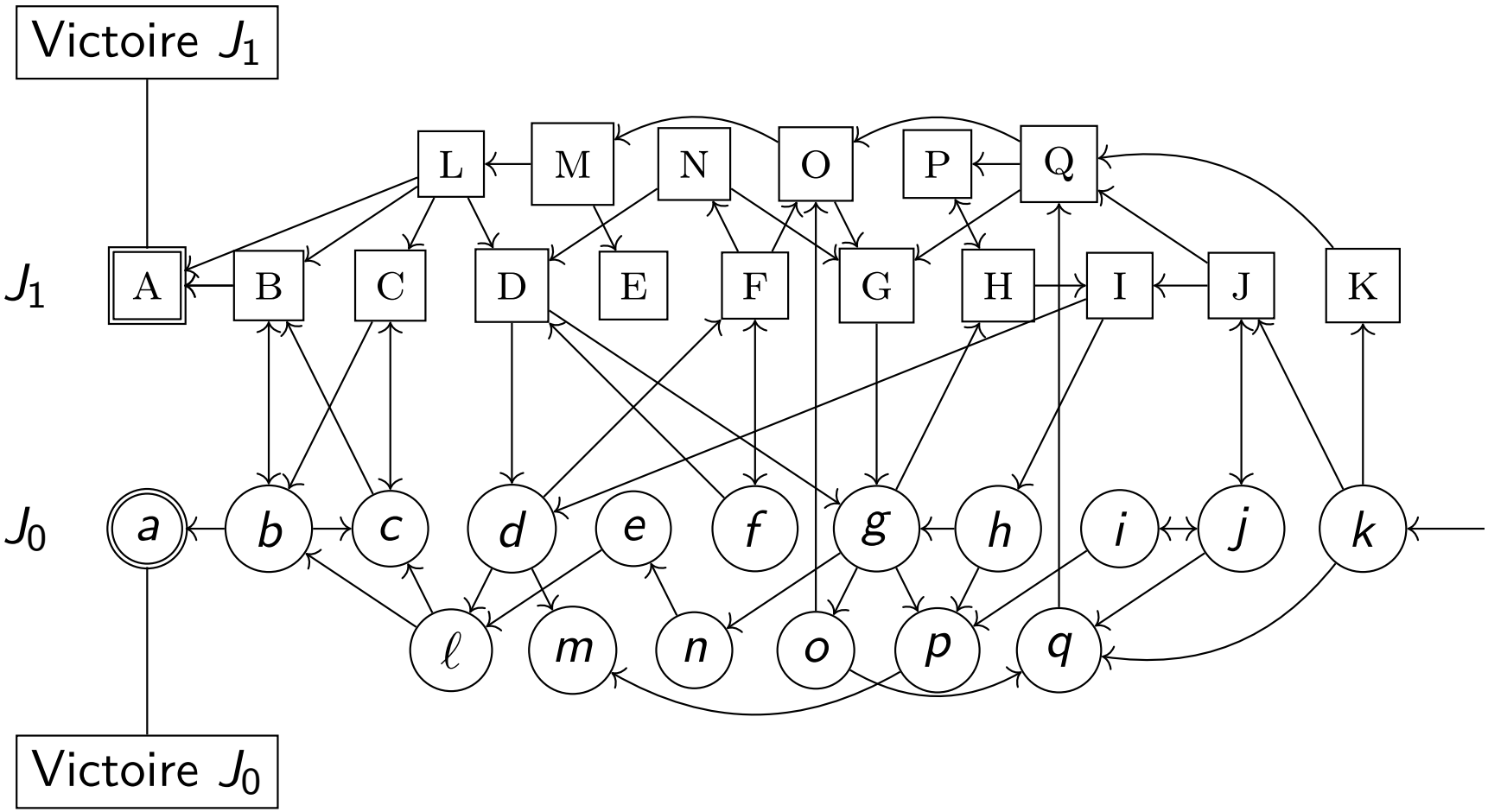
Exercice 4.1

Dresser le graphe du jeu et les positions gagnantes pour chaque joueur pour le jeu de Nim où :

- on commence avec 15 bâtonnets ;
- à chaque tour, on peut enlever 2, 3 ou 5 bâtonnets (la règle est $\{2, 3, 5\}$).

Représenter une stratégie gagnante pour le joueur concerné, en surlignant les arêtes de la stratégie.

Exercice 4.2



Déterminer les attracteurs de J_0 et de J_1. Est-ce qu'un joueur est certain de gagner ? Déterminer une stratégie gagnante pour ce joueur.