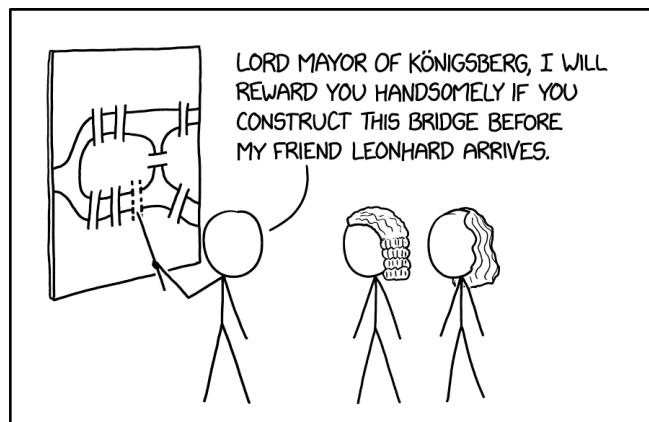


MPI* Info

Programme de khôlles

Édition n°1



I TRIED TO USE A TIME MACHINE TO CHEAT ON MY ALGORITHMS
FINAL BY PREVENTING GRAPH THEORY FROM BEING INVENTED.

Olivier Caffier



Table des matières

1	Preuve complète de la déterminisation d'un automate : description formelle de l'automate des parties, récurrence pour la fonction de transition étendue, conclusion.	1
2	Preuve complète de la construction du produit cartésien de deux automates et application à la reconnaissance de l'union et de l'intersection de deux langages reconnaissables.	2
3	Formule des attracteurs à savoir restituer parfaitement.	3
4	Pseudo-code de la fonction minmax avec élagage alpha-beta. Application à un exemple.	3
5	Code complet en C et en OCAML d'une structure union-find avec les deux optimisations.	6
6	Preuve du lemme de l'étoile.	8
7	Utilisation du lemme de l'étoile pour montrer que $L = \{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas reconnaissable.	9
8	Utilisation de la méthode des résiduels pour montrer que $L = \{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas reconnaissable.	9
9	La classe des langages reconnaissable est stable par complémentaire.	9
10	Dans un graphe non orienté pondéré $G = (S, A, p)$ admettant un unique arbre couvrant de poids minimal, noté $T = (S, A_0)$. Montrer que si $F = (S', A)$ vérifie $S' \subset S$ et que e est une arête sûre de F dans G , alors $e \in A_0$.	10

1 Preuve complète de la déterminisation d'un automate : description formelle de l'automate des parties, récurrence pour la fonction de transition étendue, conclusion.

Théorème - Existence d'un automate déterministe particulier

Soit A un automate non déterministe.

Alors,

il existe \tilde{A} un automate déterministe reconnaissant le même langage.

DÉMONSTRATION.

On suppose disposer d'un AFND $A = (\Sigma, Q, I, F, \delta)$ sans ε -transition (on ajoutera plus tard une preuve qui les prendra en compte) et on construit :

$$\tilde{A} = (\Sigma, \mathcal{P}(Q), I, \{P \subset Q, P \cap F \neq \emptyset\}, \tilde{\delta})$$

$$\text{où } \forall P \subset Q, \forall a \in \Sigma, \tilde{\delta}(P, a) = \bigcup_{p \in P} \delta(p, a).$$

Il faut montrer par récurrence sur $|m|$ (avec $m \in \Sigma^*$) que pour tout $P \subset Q$, $\tilde{\delta}(P, m)$ est l'ensemble des états de Q pour lesquels il existe $p \in P$ tel que cet état soit dans $\delta^*(p, m)$, i.e $\tilde{\delta}^*(P, m) = \bigcup_{p \in P} \delta^*(p, m)$.

➤ On veut donc montrer que :

$$\forall P \subset Q, \forall m \in \Sigma^*, \tilde{\delta}^*(P, m) = \bigcup_{p \in P} \delta^*(p, m)$$

➤ Si $m = \varepsilon$ alors $\tilde{\delta}^*(P, m) = P = \bigcup_{p \in P} \underbrace{\delta^*(p, m)}_{=\{p\}}$

➤ Soit $n \in \mathbb{N}$ et soit $m = am'$ avec $a \in \Sigma$ et $m' \in \Sigma^n$. Alors :

$$\begin{aligned} \tilde{\delta}^*(P, m) &= \tilde{\delta}^*(P, am') \\ &= \tilde{\delta}^*(\tilde{\delta}(P, a), m') \\ &= \bigcup_{p \in \tilde{\delta}(P, a)} \delta^*(p, m') \quad (\text{par H.R.}) \\ &= \bigcup_{p \in P} \underbrace{\bigcup_{q \in \delta(p, a)} \delta^*(q, m')}_{\delta^*(p, am')} \\ &= \bigcup_{p \in P} \delta^*(p, am') \quad (\text{par définition de } \delta^* \text{ dans le cas non-déterministe}) \end{aligned}$$

d'où l'hérédité.

Ainsi, pour $m \in \Sigma^*$,

$$\begin{aligned} m \in \mathcal{L}(\tilde{A}) &\Leftrightarrow \tilde{\delta}^*(I, m) \cap F \neq \emptyset \\ &\Leftrightarrow \exists i \in I \text{ et un chemin étiqueté par } m \text{ depuis } i \text{ qui mène à un état de } F. \\ &\Leftrightarrow m \in \mathcal{L}(A) \end{aligned}$$

D'où le résultat voulu.

Enfin, si on dispose d'un automate avec des ε -transitions, on déterminise directement en compte les ε -transitions.

2 Preuve complète de la construction du produit cartésien de deux automates et application à la reconnaissance de l'union et de l'intersection de deux langages reconnaissables.

Définition - Automate produit

Soient $A_1 = (\Sigma, Q_1, q_0^1, F_1, \delta_1)$ et $A_2 = (\Sigma, Q_2, q_0^2, F_2, \delta_2)$. On définit l'automate produit $A_1 \otimes A_2$ par :

$$A_1 \otimes A_2 = (\Sigma, Q_1 \times Q_2, (q_0^1, q_0^2), F, \delta)$$

$$\text{avec } \forall a \in \Sigma, \forall q_1 \in Q_1, \forall q_2 \in Q_2, \delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a)).$$

Théorème

La classe des langages reconnaissables est stable par union et par intersection.

DÉMONSTRATION.

Soient $A_1 = (\Sigma, Q_1, q_0^1, F_1, \delta_1)$ et $A_2 = (\Sigma, Q_2, q_0^2, F_2, \delta_2)$. On pose $A_1 \otimes A_2 = (\Sigma, Q_1 \times Q_2, (q_0^1, q_0^2), F, \delta)$.

On peut montrer par récurrence sur $|m|$ que :

$$\forall m \in \Sigma^*, q_1 \in Q_1, q_2 \in Q_2, \delta^*((q_1, q_2), m) = (\delta_1^*(q_1, m), \delta_2^*(q_2, m))$$

On a alors $m \in L(A_1 \otimes A_2) \iff \delta^*((q_0^1, q_0^2), m) = (\delta_1^*(q_0^1, m), \delta_2^*(q_0^2, m)) \in F$.

Ainsi si $F = F_1 \times F_2$ on aura $L(A_1 \otimes A_2) = L(A_1) \cap L(A_2)$ et si $F = (F_1 \times Q_2) \cup (F_2 \times Q_1)$ on aura $L(A_1 \otimes A_2) = L(A_1) \cup L(A_2)$.

Ainsi le langage $L(A_1) \cap L(A_2)$ est reconnaissable car il existe un automate $A_1 \otimes A_2$ tel $L(A_1) \cap L(A_2) = L(A_1 \otimes A_2)$.

3 Formule des attracteurs à savoir restituer parfaitement.

Définition - Suite des attracteurs

Soit $n \in \mathbb{N}$, on définit la suite des attracteurs de i par $A_0(i) = T_i$ et

$$\begin{aligned} A_{n+1}(i) &= A_n(i) \cup \{s \in S_i \mid \exists v \in A_n(i), (s, v) \in A\} \\ &\cup \{s \in S'_{3-i} \mid \forall v \in S_i, (s, v) \in A \Rightarrow v \in A_n(i)\} \end{aligned}$$

4 Pseudo-code de la fonction minmax avec élagage alpha-beta. Application à un exemple.

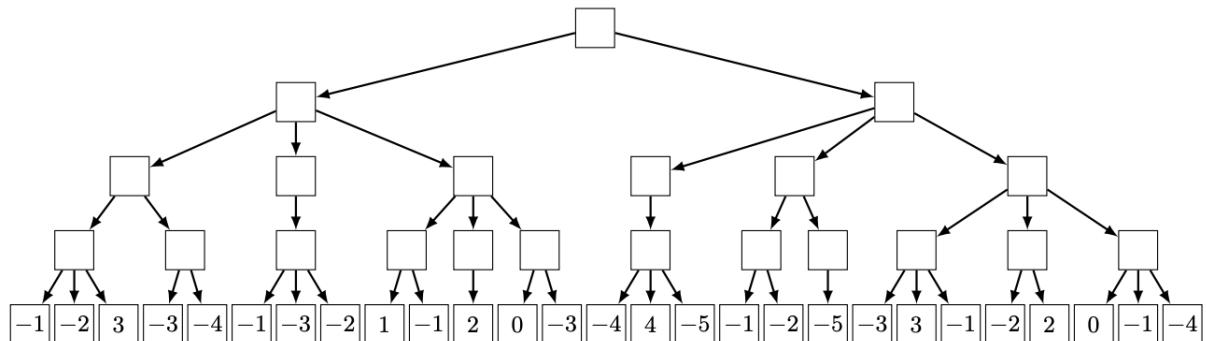
Algorithme - MinMax_ab

```

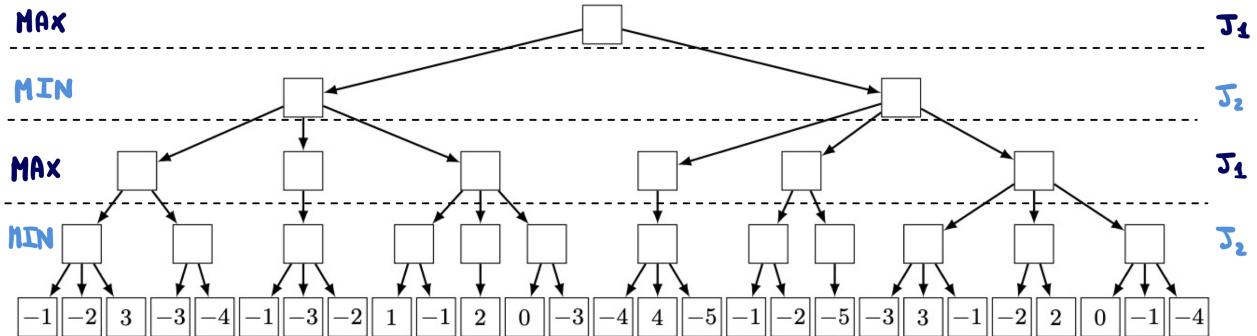
Require:  $(s, p) \in \mathbb{N}^2, (a, b) \in \mathbb{R}^2$ 
if  $s$  est un état final then
    return  $v(s)$ 
else if  $p = 0$  then
    return  $h(s)$ 
else if  $s \in S_1$  then
    for each child  $t$  of  $s$  do
         $x \leftarrow \text{MinMax\_ab}(t, p-1, a, b)$ 
        if  $x \geq b$  then
            return  $b$ 
        else if  $x > a$  then
             $a \leftarrow x$ 
        return  $a$ 
else if  $s \in S_2$  then
    for each child  $t$  of  $s$  do
         $x \leftarrow \text{MinMax\_ab}(t, p-1, a, b)$ 
        if  $x \leq a$  then
            return  $a$ 
        else if  $x < b$  then
             $b \leftarrow x$ 
    return  $b$ 
```

EXAMPLE.

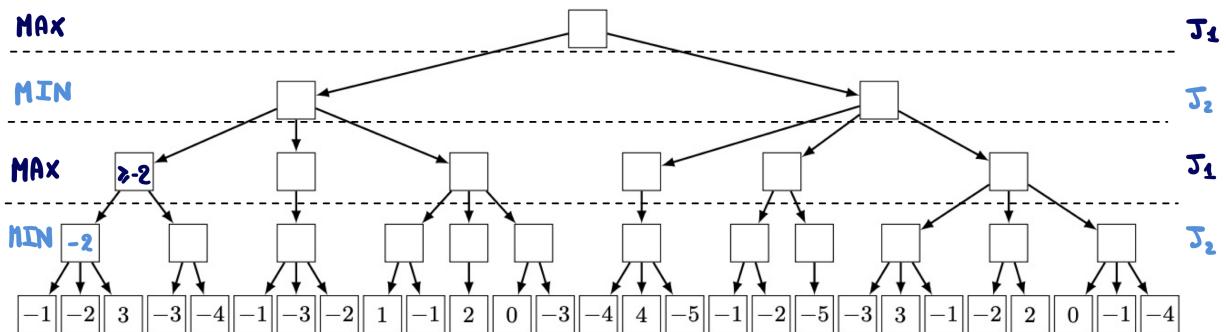
Prenons l'arbre d'appel suivant :



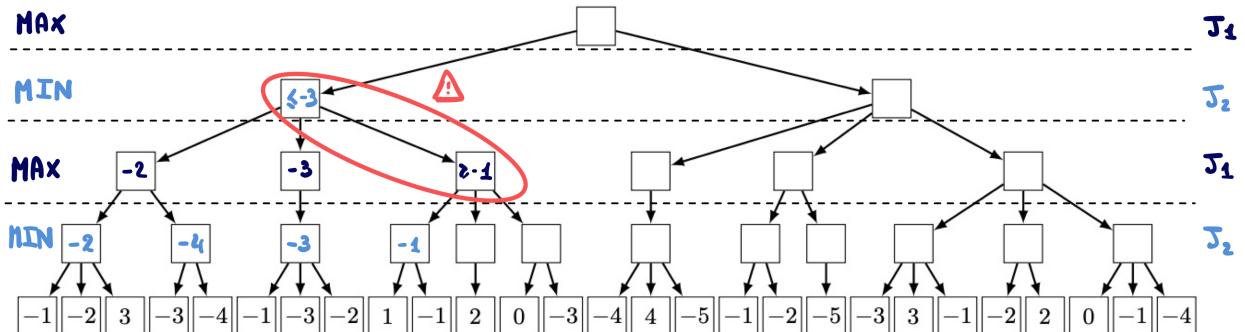
Avant toute manip', identifions (en considérant que le premier appel est issu du joueur 1) les apparteness des différentes profondeurs d'appels (« pour telle profondeur, qui joue ? »), pour ensuite dire s'il faut maximiser ou minimiser à telle profondeur :



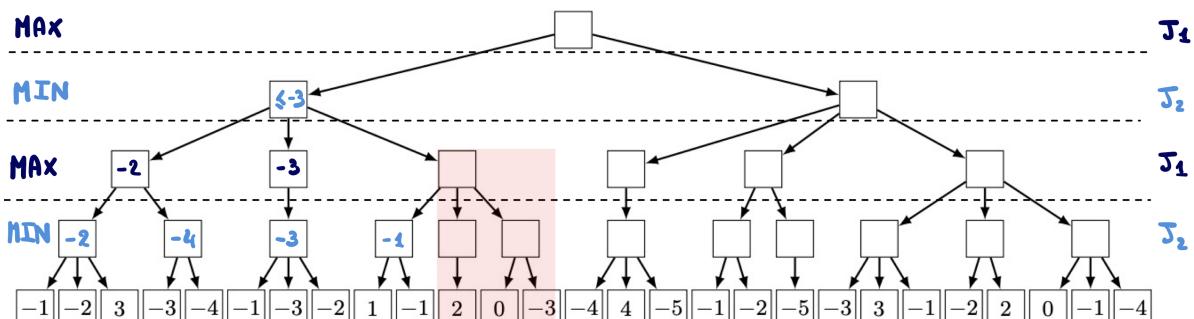
On commence donc par explorer la branche de l'arbre la plus à gauche et on peut déjà poser une première inégalité avant d'aller explorer le sous-arbre suivant :



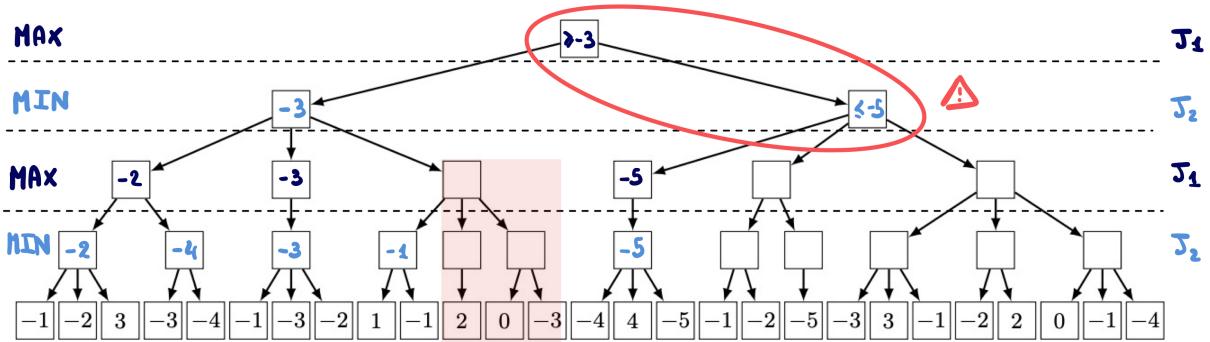
On continue, jusqu'à ce qu'on tombe sur une contradiction :



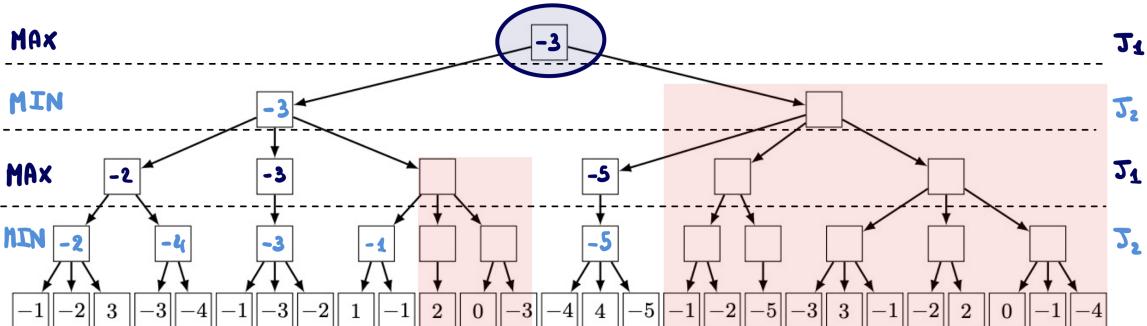
En effet, on ne peut pas en même temps chercher un score inférieur ou égal (ou strict, peu importe ici) à -3 et supérieur ou égal à -1. On sait donc déjà que cette branche ne va pas nous amener à un meilleur score, on peut s'en passer dans notre exploration. Procédons donc à l'élagage :



Continuons, en fixant à -3 la valeur du S.A.G, on part à droite maintenant et on tombe déjà sur une contradiction :

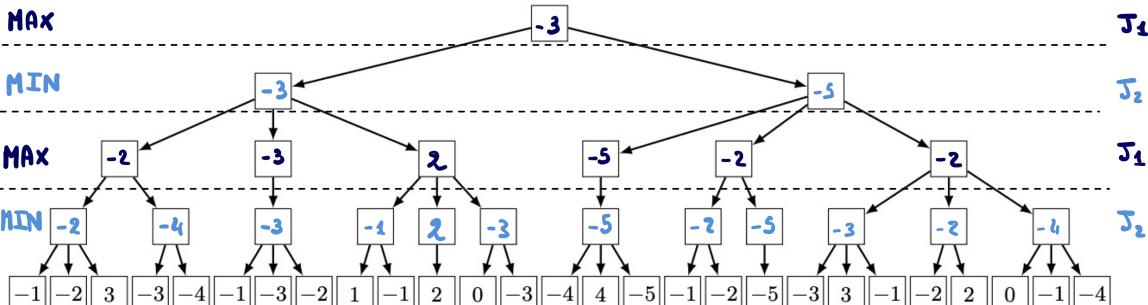


Vous connaissez la manœuvre maintenant :



et on trouve **-3!**

Cela nous aura bien sauvé des appels sur un arbre qui pourtant, n'est pas si grand que ça :



Maintenant, on peut toujours dérouler avec les valeurs de α et β , mais ça revient au même, seulement en un peu + brouillon.

5 Code complet en C et en OCAML d'une structure union-find avec les deux optimisations.

```
1 type uf = {
2   link : int array;
3   rank : int array;
4 }
5
6 let create n =
7   { link = Array.init n (fun i-> i) ; rank = Array.make n 0}
8
9 let rec find uf i =
10  let p = uf.link.(i) in
11  if (p=i) then i
12  else begin
13    let r = find uf p in
14    uf.link.(i) <- r;
15    r
16  end
17
18 let union uf i j =
19  let ri = find uf i in
20  let rj = find uf j in
21  if (ri <> rj) then
22    begin
23      if uf.rank.(ri) < uf.rank.(rj) then uf.link.(ri) <- rj
24      else if uf.rank.(ri) < uf.rank.(rj) then uf.link.(rj) <- ri
25      else
26        begin uf.rank.(ri) <- uf.rank.(ri) + 1 ; uf.link.(rj) <- ri end
27    end
```

Code 1 - Implémentation UF en OCAML

```

1 struct UnionFind{
2     int size; int* link; int* rank;
3 };
4 typedef struct UnionFind uf;
5
6 uf* create(int size){
7     uf* res = malloc(sizeof(uf)); res->rank = malloc(size* sizeof(int)); res->link = malloc(size * sizeof(
8         int));
9     for (int i=0; i<size; i++){
10         (res->link)[i] = i;
11         (res->rank)[i] = 0;
12     }
13     res->size = size;
14     return res;
15 }
16
17 int uf_find(uf* uf, int i){
18     if ((uf->link)[i] == i){
19         return i;
20     }
21     else{
22         int p = (uf->link)[i];
23         int r = find(uf, p);
24         (uf->link)[i] = r;
25         return r;
26     }
27 }
28
29 void uf_union(uf* uf, int i, int j){
30     int ri = uf_find(uf,i); int rj = uf_find(uf,j);
31     if (ri != rj){
32         int rank_ri = (uf->rank)[ri]; int rank_rj = (uf->rank)[rj];
33         if (rank_ri > rank_rj){
34             (uf->link)[rj] = ri;
35         }
36         else if (rank_rj > rank_ri){
37             (uf->link)[ri] = rj;
38         }
39         else{
40             (uf->link)[rj] = ri; (uf->rank)[ri] += 1;
41         }
42     }
43 }

```

Code 2 - Implémentation UF en C

6 Preuve du lemme de l'étoile.

Théorème - Lemme de l'étoile

Si L est reconnaissable alors :

$$\exists N \in \mathbb{N} \text{ tq. } \forall m \in L, |m| \geq N, \exists u, v, w \in \Sigma^* \text{ tq. } \begin{cases} m = uvw \\ |uv| \leq N \\ v \neq \epsilon \end{cases} \text{ et } \forall k \in \mathbb{N}, uv^k a \in L$$

En pratique on utilisera la contraposée : Si

$$\forall N \in \mathbb{N}, \exists m \in L, |m| \geq N, \forall u, v, w \in \Sigma^* \text{ tq. } \begin{cases} m = uvw \\ |uv| \leq N \\ v \neq \epsilon \end{cases} \text{ et } \exists k \in \mathbb{N}, uv^k a \notin L$$

alors L n'est pas reconnaissable.

DÉMONSTRATION.

Soit L un langage reconnaissable et soit $A = (\Sigma, Q, q_0, F, \delta)$ un AFD complet tq. $\mathcal{L}(A) = L$. Soit $N \in \mathbb{N}$ le nombre d'états de A .

Soit $m \in L$ tq. $|m| \geq N$. Notons $m = m_1 \dots m_k$ et $\forall i \in \llbracket 0; k \rrbracket$, posons $q_i = \delta^*(q_0, m_1 \dots m_i)$.

Par principe des tiroirs ($k \geq N$) : $\exists i, j$ tq. $0 \leq i < j \leq k$ et $q_i = q_j$.

On pose alors $\underbrace{u = m_1 \dots m_i}_{q_0 \xrightarrow{u} q_i}, \underbrace{v = m_{i+1} \dots m_j}_{q_i \xrightarrow{v} q_j = q_i}$ et $\underbrace{w = m_{j+1} \dots m_k}_{q_i = q_j \xrightarrow{w} q_f}$.

On a bien sûr $m = uvw$ mais aussi $v \neq \epsilon$ (car $i < j$) et $|uv| = j \leq N$.

Ensuite, prouvons par récurrence sur k que $\delta^*(q_i, v^k) = q_i$:

- $k = 0$: on a $v^k = \epsilon$ et donc on a bien $\delta^*(q_i, v^k) = \delta^*(q_i, \epsilon) = q_i$
- $k = 1$: on a (par construction) $\delta^*(q_i, v^k) = \delta^*(q_i, v) = q_j = q_i$
- Hérédité : Soit $k \geq 0$, on a

$$\begin{aligned} \delta^*(q_i, v^{k+1}) &= \delta^*(\underbrace{\delta^*(q_i, v^k)}_{= q_i \text{ par H.R}}, v) \\ &= \delta^*(q_i, v) \\ &= q_j = q_i \end{aligned}$$

Enfin, soit $k \in \mathbb{N}$,

$$\begin{aligned} \delta^*(q_0, uv^k w) &= \delta^*(\delta^*(q_0, u), v^k w) \\ &= \delta^*(q_0, v^k w) \\ &= \delta^*(\delta^*(q_i, v^k), w) \\ &= \delta^*(q_i, w) \\ &= q_f \in F \end{aligned}$$

7 Utilisation du lemme de l'étoile pour montrer que $L = \{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas reconnaissable.

Prenons donc le langage $\{a^n b^n \mid n \in \mathbb{N}\}$. Soit $N \in \mathbb{N}$.

Soit $m \in L$ tq. $m = a^N b^N$.

$$\text{Soient } u, v, w \text{ quelconques tq. } \begin{cases} a^N b^N &= uvw \\ |uv| &\leq N \\ v &\neq \epsilon \end{cases}$$

Alors $\exists i \in \llbracket 0; N-1 \rrbracket$, $j > 0$ avec $i + j \leq N$ tq. $u = a^i$, $v = a^j$, $w = a^{N-i-j} b^N$.

Enfin, pour $k = 2$, on a

$$uv^k w = a^i a^{2j} a^{N-i-j} b^N = a^{N+j} b^N \notin L \text{ (car } j > 0\text{)}$$

Donc d'après la contraposée du lemme de l'étoile, L n'est pas reconnaissable. D'où le résultat voulu.

8 Utilisation de la méthode des résiduels pour montrer que $L = \{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas reconnaissable.

Prenons $L = \{a^n b^n \mid n \in \mathbb{N}\}$.

Supposons par l'absurde que L soit reconnaissable. Soit donc $A = (\Sigma, Q, q_0, F, \delta)$ un AFD complet à N états.

Soit $m = a^N b^N$. Pour tout $i \in \llbracket 0; N \rrbracket$, notons $q_i = \delta^*(q_0, a^i)$.

Par principe des tiroirs, il existe j tq. $i < j \leq N$ et $q_i = q_j$.

On en déduit que

$$\delta^*(q_i, a^{N-j} b^N) = \delta^*(q_j, a^{N-j} b^N)$$

Ainsi,

$$\underbrace{\delta^*(q_0, a^i a^{N-j} b^N)}_{\notin F \text{ car } N+i-j \neq N} = \underbrace{\delta^*(q_0, a^j a^{N-j} b^N)}_{\in F}$$

ce qui est bel et bien absurde.

9 La classe des langages reconnaissables est stable par complémentaire.

Corollaire

La classe des langages reconnaissables est stable par complémentaire.

DÉMONSTRATION.

Soit $A = (\Sigma, Q, q_0, F, \delta)$. On pose $L = \mathcal{L}(A)$ et $L' = \Sigma^* \setminus \mathcal{L}(A)$.

On suppose A complet.

Ainsi $\forall m \in \Sigma^*$, $\delta^*(q_0, m)$ est bien défini et $(m \in \mathcal{L}(A) \Leftrightarrow \delta^*(q_0, m) \in F)$ et $(m \notin \mathcal{L}(A) \Leftrightarrow \delta^*(q_0, m) \notin F)$.

On pose $A' = (\Sigma, Q, q_0, Q \setminus F, \delta)$ et on a alors $m \in L' \Leftrightarrow m \notin L = \mathcal{L}(A) \Leftrightarrow \delta^*(q_0, m) \in Q \setminus F \Leftrightarrow m \in \mathcal{L}(A')$

D'où $L' = \mathcal{L}(A')$ avec A' un automate fini déterministe.
D'où L' reconnaissable.

10 Dans un graphe non orienté pondéré $G = (S, A, p)$ admettant un unique arbre couvrant de poids minimal, noté $T = (S, A_0)$. Montrer que si $F = (S', A)$ vérifie $S' \subset S$ et que e est une arête sûre de F dans G , alors $e \in A_0$.

Définition - Arête sûre

Soient $G = (S, A, p)$ et $S' \subset S (S' \neq S)$.

Une arête sûre pour S' dans G est une arête de poids minimal ayant exactement une extrémité dans S'

Théorème

Soient $G = (S, A, p)$ un graphe non orienté pondéré et connexe, et $S' \subset S (S' \neq S)$, alors :

Toute arête sûre de S' pour G appartient à l'arbre couvrant de poids minimal de G .

DÉMONSTRATION.

Soit $e = (x, y)$ une arête sûre pour S' dans G et soit $\alpha = (S, A')$ l'arbre couvrant de poids minimal de G .

Supposons par l'absurde que $e \notin A'$.

Dans α , il existe un chemin entre x et y qui n'utilise pas e . Notons ce chemin :

$$\underbrace{x}_{\in S'} = x_0 \rightarrow x_1 \rightarrow \cdots \rightarrow x_p = \underbrace{y}_{\notin S'}$$

Notons i_0 le plus petit indice tq. $x_{i_0} \notin S'$ (qui existe bien car $x_p \notin S'$), alors $0 < i_0 \leq p$.

On a alors $(x_{i_0-1}, x_{i_0}) = e'$ une arête avec exactement une extrémité dans S' . On en déduit (sinon il s'agirait d'une arête sûre) :

$$p(e') > p(e)$$

Ainsi, $\alpha' = (S, (A' \setminus \{e'\}) \cup \{e\})$ est un arbre couvrant (car connexe et possède le bon nombre d'arêtes) et

$$\begin{aligned} p(\alpha') &= p(\alpha) - p(e') + p(e) \\ &< p(\alpha) \end{aligned}$$

ce qui est absurde car α est minimal.

