Informatique
Programme de khôlle 1
MPI(*) Faidherbe

BURGHGRAEVE Marc

23 Octobre 2024

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

On se donne donc notre automate $A = (Q, I, F, \delta)$.
Soit $A' = (\mathscr{P}(Q), I, \{P \subset Q : P \cap F \neq \emptyset\}, \delta')$.
et $\forall P \subset Q, \forall \alpha \in \Sigma$,

$$\delta'(P, \alpha) = \{q \in Q, \exists p \in P, q \in \delta(p, \alpha)\} = \bigcup_{p \in P} \delta(p, \alpha)$$

Montrons par récurrence sur $|m|$ que $\forall m \in \Sigma^*, \forall P \subset Q, \delta'^*(P, m) = \bigcup_{p \in P} \delta^*(p, m)$.
initialisation : Si $|m| = 0, m = \varepsilon$. Soit $P \subset Q, \delta'^*(P, \epsilon) = P$ et $\bigcup_{p \in P} \delta^*(p, \varepsilon) = \bigcup_{p \in P} \{p\} = P$. (La construction ici concerne un algorithme sans ε-transitions).
Hérédité Supposons que la formule soit vraie pour tout $m \in \Sigma^n$. soit n de taille $n + 1$, alors on a $m = ua, u \in \Sigma^n, a \in \Sigma$. Soit $P \subset Q$.

$$\begin{aligned}
\delta'^*(P, ua) &= \delta'(\delta'^*(P, u), a) \\
&\underbrace{=}_{\text{(Par H.R.)}} \delta'(\bigcup_{p \in P} \delta^*(p, u), a) \\
&\underbrace{=}_{\text{(Def de } \delta')} \bigcup_{q \in \bigcup_{p \in P} \delta^*(p,u)} \delta(q, a) \\
&= \bigcup_{p \in P} \bigcup_{q \in \delta^*(p,u)} \delta(q, a) \\
&\underbrace{=}_{\text{(Def de } \delta^* \text{ ND)}} \bigcup_{p \in P} \delta^*(p, ua)
\end{aligned}$$

Conclusion : $m \in \mathscr{L}(A')$

$$\begin{aligned}
&\iff \delta'^*(I, m) \cap F \neq \emptyset \\
&\iff (\bigcup_{i \in I} \delta'^*(i, m)) \cap F \neq \emptyset \\
&\iff \exists i \in I, \delta'^*(i, m) \cap F \neq \emptyset \\
&\iff m \in \mathscr{L}(A)
\end{aligned}$$

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

Soit $A = (Q, q_0, F, \delta)$, $A' = (Q', q'_0, F', \delta')$. On définit l'automate produit cartésien par :

$$A * A' = (Q * Q', (q_0, q'_0), F * F', \Delta)$$

Où Δ est définie - quand c'est possible - par :
$\forall q, q' \in Q * Q', \forall a \in \Sigma, \Delta((q, q'), a) = (\delta(q, a), \delta'(q', a))$. On peut montrer par récurrence que :

$$\forall m \in \Sigma^*, \Delta^*((q, q'), m) = (\delta^*(q, m), \delta'^*(q', m))$$

Ainsi,

$$\begin{aligned}
& m \in L(A * A') \\
& \iff \Delta^*((q_0, q'_0), m) \in F * F' \\
& \iff \delta^*(q_0, m) \in F \text{ et } \delta'^*(q'_0, m) \in F' \\
& \iff m \in L(A) \text{ et } m \in L(A') \\
& \iff m \in L(A) \cap L(A')
\end{aligned}$$

pour l'union, il faut remplacer les états finaux par $F * Q' \cup Q * F'$.

3. Formule des attracteurs à savoir restituer parfaitement.

On va utiliser le principe suivant :

$$T_i \subset A(i).$$
$$v \in S_i, \exists w \in A(i), (v, w) \in A \implies v \in A(i)$$
$$v \in S_{3-i}, \forall w \in S_i, (v, w) \in A, w \in A(i) \implies v \in A(i).$$

On définit :

$$A_o(i) = T_i$$
$$\forall k \geq 0, A_{k+1}(i) = A_k(i) \cup \{v \in S_i, \exists w \in A_k(i), (v_w) \in A\} \cup \{v \in \mathbf{S'_{3-i}}, \forall w, (v, w) \in A \implies w \in A_k(i)\}$$

Les sommets de S'_{3-i} sont ceux de S_{3-i} privé des puits de cet ensemble.

4. Savoir donner le pseudo-code de la fonction minmax avec élagage alpha-beta. Savoir appliquer cet algo avec élagage sur un arbre exemple donné.

Algorithm 1 MinMax élagué avec profondeur maximale et heuristique

```
procedure MinMax_e(s, p, α, β)
    if s est terminal then
        return +∞ (si j1 est gagnant), −∞ (si j2 est gagnant), ou 0 (match nul)
    else if p = h_max then
        return h(s)
    else if s ∈ S1 then                                    ▷ Joueur qui maximise
        for tout voisin s' de s do
            x ← MinMax_e(s', p + 1, α, β)
            if x ≥ β then
                return β
            if x > α then
                α ← x
        return α
    else if s ∈ S2 then                                    ▷ Joueur qui minimise
        for tout voisin s' de s do
            x ← MinMax_e(s', p + 1, α, β)
            if x ≤ α then
                return α
            if x < β then
                β ← x
        return β
```

5. Etre capable de redonner le code complet en C ou en Ocaml d'une structure union find avec optimisation des rangs et compression des chemins

```c
struct UnionFind {
    int *parent;  // Tableau des parents (dans le cours "link")
    int *rank;    // Tableau des rangs
    int size;     // Taille de l'ensemble
};


struct UnionFind* createUnionFind(int n) {
    struct UnionFind *uf =malloc(sizeof(struct UnionFind));
    uf->parent = (int*)malloc(n * sizeof(int));
    uf->rank = (int*)malloc(n * sizeof(int));
    uf->size = n;

    for (int i = 0; i < n; i++) {
        uf->parent[i] = i;  // Chaque element est son propre parent (singleton)
        uf->rank[i] = 0;    // Le rang initial est 0
    }
    return uf;
}

// Trouver le representant (racine) de l'ensemble contenant x avec compression des chemins
int find(struct UnionFind *uf, int x) {
    if (uf->parent[x] != x) {
        // Compression des chemins: faire pointer x directement sur la racine
        uf->parent[x] = find(uf, uf->parent[x]);
    }
    return uf->parent[x];
}

// Union de deux ensembles avec optimisation par rang
void unionSets(struct UnionFind *uf, int x, int y) {
    int rootX = find(uf, x);
    int rootY = find(uf, y);
    if (rootX != rootY) {
        // Union par rang: le plus petit arbre est rattache au plus grand
        if (uf->rank[rootX] > uf->rank[rootY]) {
            uf->parent[rootY] = rootX;
        } else if (uf->rank[rootX] < uf->rank[rootY]) {
            uf->parent[rootX] = rootY;
        } else {
            uf->parent[rootY] = rootX;
            uf->rank[rootX]++;
        }
    }
}
//N'oubliez pas de liberer la memoire !
```

```
(* Structure Union-Find avec rangs et compression des chemins *)
type union_find = {
  parent: int array;
  rank: int array;
}

(* Initialisation *)
let create_union_find n =
  { parent = Array.init n (fun i -> i); (* Chaque element est son propre parent *)
    rank = Array.make n 0 } (* Tous les rangs sont initialises a 0 *)

(* Trouver le representant (racine) de l'ensemble contenant x avec compression des chemins *)
let rec find uf x =
  if uf.parent.(x) <> x then begin
    uf.parent.(x) <- find uf uf.parent.(x); (* Compression des chemins *)
  end;
  uf.parent.(x)

(* Union de deux ensembles avec optimisation par rang *)
let union_sets uf x y =
  let root_x = find uf x in
  let root_y = find uf y in
  if root_x <> root_y then
    if uf.rank.(root_x) > uf.rank.(root_y) then
      uf.parent.(root_y) <- root_x
    else if uf.rank.(root_x) < uf.rank.(root_y) then
      uf.parent.(root_x) <- root_y
    else begin
      uf.parent.(root_y) <- root_x;
      uf.rank.(root_x) <- uf.rank.(root_x) + 1;
    end
```

6. Preuve du lemme de l'étoile.

Soit L un langage reconnaissable, soit $A = (\Sigma, Q, q_0, F, \delta)$ un AFD complet tel que $L(A) = L$.
On pose $n = |Q| \in \mathbb{N}$. Soit $m \in L$ tq $|m| \geq n$.
Notons $m = m_1 \dots m_k$ avec $k \geq n$. $q_i = \delta^*(q_0, m_1 \dots m_i).k \geq n \implies card[q_0, ..., q_k] \geq n + 1$
Donc $\exists i, j$ tq $0 \leq i < j \leq n$ tq $q_i = q_j$ (principe des tiroirs)
On pose

- $u = m_1 \dots m_i \implies q_i = \delta^*(q_0, u)$
- $v = m_{i+1} \dots m_j \implies q_j = \delta^*(q_i, v)$
- $w = m_{j+1} \dots m_k \implies q_k = \delta^*(q_i, w)$

On a bien que :

- m = uvw
- $|uv| \leq n$ car $|uv| = j \leq n$
- $v \neq \varepsilon$ car $|v| = j - i \neq 0 car j > i$

Montrons par récurrence Hn : "$\forall p \in \mathbb{N}, \delta^*(q_i, v^p) = q_i$"

- **p=0** $\delta^*(q_i, \varepsilon) = q_i$
- **p=1** $\delta^*(q_i, v) = q_j = q_i$
- **Hp implique Hp+1** : $\delta^*(q_i, v^{p+1}) = \delta^*(\delta^*(q_i, v^p), v) = \delta^*(q_i, v) = q_j = q_i$.

Enfin, pour $p \in \mathbb{N}$.
$\delta^*(q_0, uv^p w) = \delta^*(\delta^*(q_0, u), v^p w) = \delta^*(q_i, v^p w) = \delta^*(\delta^*(q_i, v^p), w) = \delta^*(q_i, w) = q_k \in F$.
$\implies uv^p w \in L(A) = L$

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

Rappel de la contraposée du Lemme :
Si $\forall N \in \mathbb{N}, \exists m \in L, |m| \geq N$,
$\forall u, v, w$ tq

- $m = uvw$
- $|uv| \leq N$
- $v \neq \varepsilon$

et $\exists k \in \mathbb{N}$ tq $uv^k w \notin L \implies$ **L non reconnaissable**
Soit donc $N \in \mathbb{N}$, on choisit $m = a^N b^N$ (on a bien $|m| \geq N$).
Soient donc u,v,w vérifiant les hypothèses du Lemme. Soient $0 \leq i \leq N-1, j \neq 0$ tels que $u = a^i, v = a^j, w = a^{N-i-j} b^N$.
Alors en choisissant $k = 0$, on a $uv^k w = a^i a^{N-i-j} b^N = a^{N-j} b^N$.
Or, $N - j \neq N$ car $j \neq 0$ donc $uv^k w \notin L$:
D'après la contraposée du Lemme, L n'est pas reconnaissable.

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

Montrons que si $A = (Q, q_0, F, \delta)$ est un AFD qui reconnaît L alors $\forall \in \mathbb{N}, card(Q) \geq n$, donc A n'existe pas et L n'est pas reconnaissable.
Soit $n \in N\mathbb{N}$. Posons :

$$u_0 = \varepsilon$$
$$u_1 = a$$
$$\dots$$
$$u_{n-1} = a^{n-1}$$

Soit $i \neq j$. On a $u_i b^i \in L$, mais $u_j b^i \notin L$. Donc si $card(Q) < n$, alors $\exists i \neq j$ tels que $\delta^*(q_0, u_i) = \delta^*(q_0, u_j)$ (principe des tiroirs).
Dès lors,

$$\underbrace{\delta^*(q_0, u_i b^i)}_{\in F}$$
$$= \delta^*(\delta^*(q_0, u_i), b^i)$$
$$= \delta^*(\delta^*(q_0, u_j), b^i)$$
$$\underbrace{\delta^*(q_0, u_j b^i)}_{\notin F}$$

D'où l'absurdité : $card(Q) \geq n$.

9. Preuve du fait que le complémentaire d'un langage reconnaissable est reconnaissable.

Soit L reconnu par $A = (Q, q_0, F, \delta)$. Posons $A' = (Q, q_0, Q \setminus F, \delta)$. A' convient car $m \in L(A') \iff \delta^*(q_0, m) \in Q \setminus F \iff m \notin L(A)$. Surement la preuve la plus dure du programme de colle..

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

On se donne un graphe $G = (S, A, p)$ connexe et $T = (S, A_0)$ un ACPM. On veut donc montrer que :

$\forall S' \subset S, (S' \neq \emptyset),$ si (x, y) est une arête sure pour S' dans G alors $(x, y) \in A_0$.

preuve : Soit $S' \neq \emptyset$ et $S' \neq S$ et (x, y) sure pour S' dans G. Supposons par l'absurde que $(x, y) \notin A_0$. Dans (S, A_0), on sait qu'il existe un chemin entre x et y par connexité, notons le $c = x = x_0 \to \dots x_k = y$. En considérant $i = max_{0 \leq j \leq k-1}\{\forall p \leq j, x_p \in S'\}$. Autrement dit, $i+1 = min_{1 \leq j \leq k}\{x_j \notin S'\}$. Par définition, on aura $x_i \in S'$ et $x_{i+1} \notin S'$ et $(x_i, x_{i+1}) \in A_0$. Par def d'une arête sure on a alors $p(\{x, y\}) < p(\{x_i, x_{i+1}\})$. Soit $T' = (S, (A_0 \setminus \{x_i, x_{i+1}\}) \cup \{x, y\})$ On appelle A'_0 le deuxième ensemble défini et son cardinal est égal à celui de A_0. T' est connexe, car il existe un chemin entre x_i et x_{i+1} dans T'. ainsi T' est un arbre couvrant de G et $p(T') = P(T) - p(x_i, x_{i+1}) + p(x, y) < p(T)$ ce qui contredit la minimalité de T.