Minimisation d'un automate

Lucas Willems

30 novembre 2016

Introduction

Problème

Trouver l'automate avec un nombre minimal d'état reconnaissant un langage L donné.

Definition (Congruence de Nerode)

Soit $\mathcal{A} = (Q, \Sigma, i, F, \delta)$ déterministe. La congruence de Nerode est définie pour tous états q et q' par :

$$q \sim q' \iff \forall w \in \Sigma^* \ \ (q \cdot w \in F \Leftrightarrow q' \cdot w \in F)$$

Proposition

Soit $\mathcal{A}$ un automate déterministe acceptant L. L'automate minimal $\mathcal{A}_{\mathcal{L}}$ est égal à $\mathcal{A}/\sim$ où $\sim$ est la congruence de Nerode de $\mathcal{A}$.

Reformulation du problème

Trouver la congruence de Nerode d'un automate reconnaissant L.

L'algorithme itératif

Définition & Proposition

Soit $\mathcal{A} = (Q, \Sigma, i, F, \delta)$ *déterministe. On définit les relations d'équivalences* $(\sim_i)_{i \in \mathbb{N}}$ *sur* Q *par :*

$$q \sim_0 q' \iff (q \in F \leftrightarrow q' \in F)$$
$$q \sim_{i+1} q' \iff q \sim_i q' \text{ et } \forall a \in \Sigma \;\; q \cdot a \sim_i q' \cdot a$$

Et on a : $\quad \forall i \geq |Q| \;\; \sim_i \; = \; \sim$

Algorithme & Complexité (Itératif)

Etats	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8
$\overline{q}_i$	1	1	1	3	2	3	4	2
$\overline{q \cdot a}_i$	2	2	3	2	2	2	4	2
$\overline{q \cdot b}_i$	3	3	3	1	3	1	4	2
$\overline{q}_{i+1}$	1,2,3	1,2,3	1,3,3	3,2,1	2,2,3	3,2,1	4,4,4	2,2,2

La complexité est en $O(|\Sigma||Q|^2)$.

L'algorithme d'Hopcroft (définitions & lemmes)

Definition (Stabilité et coupure)

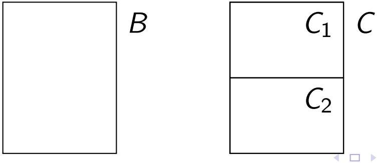
Soit $\mathcal{A} = (Q, \Sigma, i, F, \delta)$ déterministe. Soit $a \in \Sigma$ et $B, C \subset Q$. On pose $B_1 = \{q \in B \mid q \cdot a \in C\}$ et $B_2 = \{q \in B \mid q \cdot a \notin C\}$. On définit :

- Si $B_1 = \emptyset$ ou $B_2 = \emptyset$, alors B est **stable** pour (C, a).
- Sinon, B est **coupé** par (C, a) en B_1 et B_2.

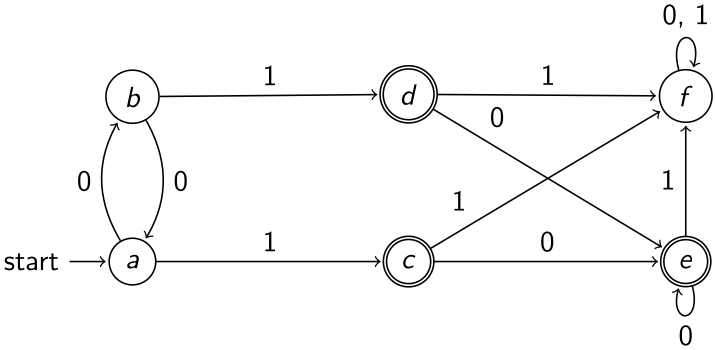
Lemme

Soit $B, C \subset Q$, $C = C_1 \uplus C_2$ et $a \in \Sigma$.

1. *B stable pour (C_1, a) et (C_2, a) $\Rightarrow$ B stable pour (C, a)*
2. *B stable pour (C, a) $\Rightarrow$ (B stable pour (C_1, a) $\Leftrightarrow$ B stable pour (C_2, a))*

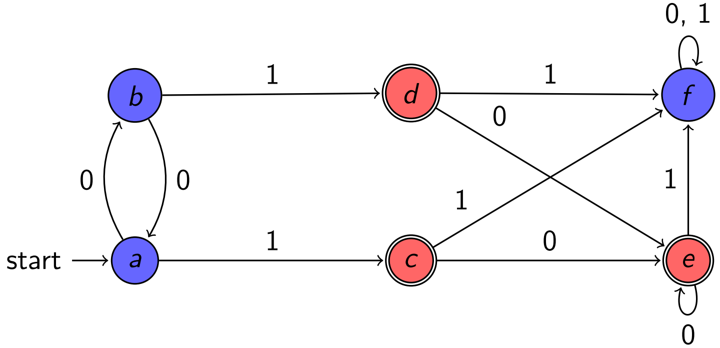


L'algorithme d'Hopcroft (exemple)



$P = \{\}$
$S = \{\}$

L'algorithme d'Hopcroft (exemple)

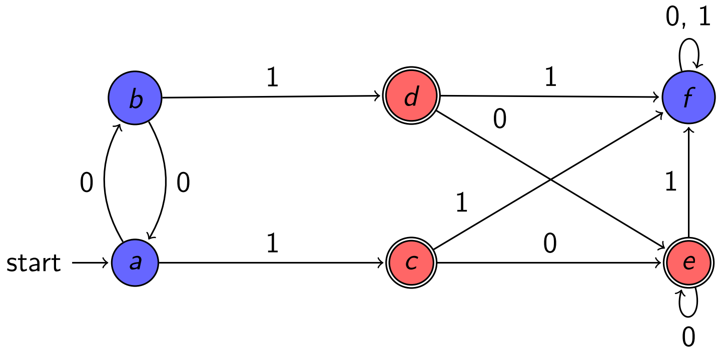


$P = \{\}$
$S = \{\}$

Initialisation

$P = \{ {\color{blue}\bullet}, {\color{red}\bullet} \}$
$S = \{({\color{blue}\bullet}, 0), ({\color{blue}\bullet}, 1), ({\color{red}\bullet}, 0), ({\color{red}\bullet}, 1)\}$
$\xrightarrow[\text{Lemme}]{} \quad S = \{({\color{blue}\bullet}, 0), ({\color{blue}\bullet}, 1), \cancel{({\color{red}\bullet}, 0)}, \cancel{({\color{red}\bullet}, 1)}\}$

L'algorithme d'Hopcroft (exemple)



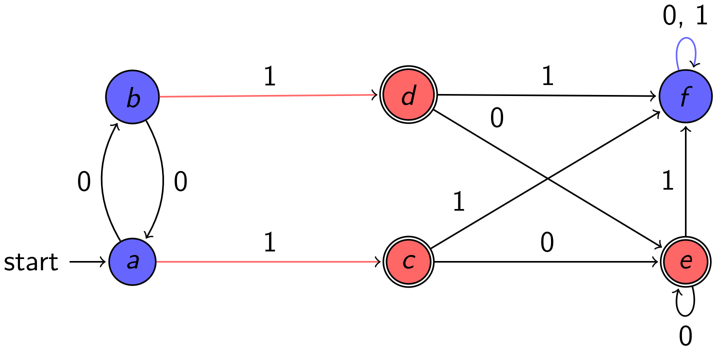
$P = \{$ ● , ● $\}$

$S = \{\cancel{(\text{●}, 1)}, (\text{●}, 0)\}$

Traitement de (● , 1)

$B =$ ●

L'algorithme d'Hopcroft (exemple)



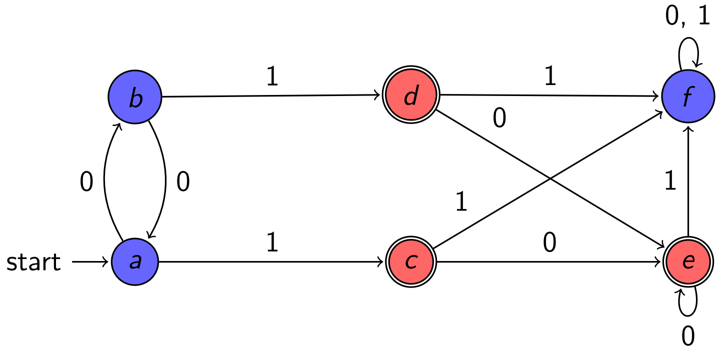
$P = \{$ ● , ● $\}$

$S = \{\cancel{(\bullet, 1)}, (\bullet, 0)\}$

Traitement de (● , 1)

$B =$ ● $\longrightarrow$ B est coupé en {f} et {a, b}

L'algorithme d'Hopcroft (exemple)



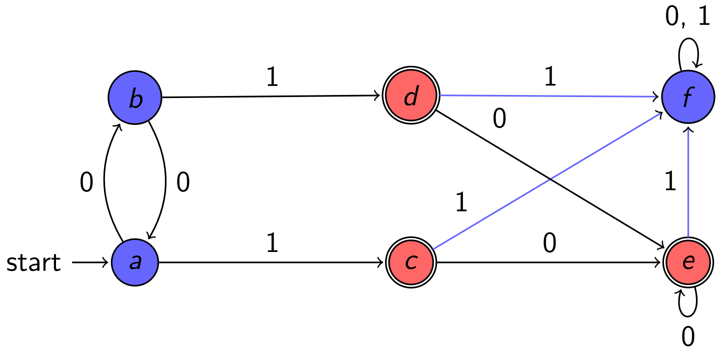
$P = \{$ ● , ● $\}$
$S = \{\cancel{(\text{●}, 1)}, (\text{●}, 0)\}$

Traitement de (● , 1)

$B =$ ● $\longrightarrow$ B est coupé en {f} et {a, b}

$B =$ ●

L'algorithme d'Hopcroft (exemple)

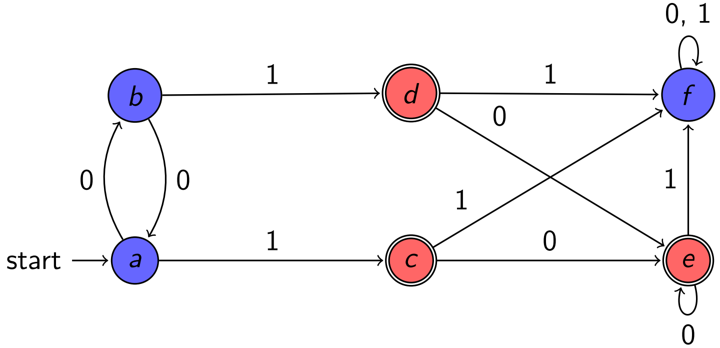


$P = \{ \bullet , \bullet \}$
$S = \{\cancel{(\bullet , 1)}, (\bullet , 0)\}$

Traitement de (• , 1)

$B = \bullet \quad \longrightarrow \quad B$ est coupé en {f} et {a, b}
$B = \bullet \quad \longrightarrow \quad B$ est stable

L'algorithme d'Hopcroft (exemple)



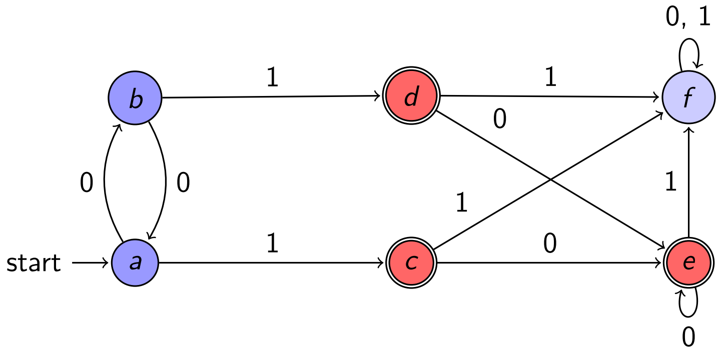
$P = \{ \bullet , \bullet \}$
$S = \{\cancel{(\bullet , 1)}, (\bullet , 0)\}$

Traitement de (● , 1)

$B = \bullet \quad \longrightarrow \quad B$ est coupé en $\{f\}$ et $\{a, b\}$
$B = \bullet \quad \longrightarrow \quad B$ est stable

$$\Longrightarrow \begin{cases} P = \{ \bullet , \bullet , \bullet \} \text{ avec } \bullet = \bullet \uplus \bullet \\ S = \{(\bullet , 0), (\bullet , 0), (\bullet , 0), (\bullet , 1), (\bullet , 1)\} \end{cases}$$

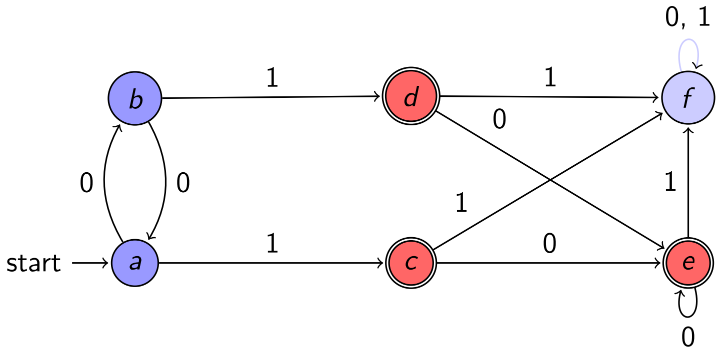
L'algorithme d'Hopcroft (exemple)



$P = \{ ●, ●, ● \}$
$S = \{\cancel{(●, 1)}, (●, 0), (●, 0)\}$

Traitement de (●, 1)
$B = ●$

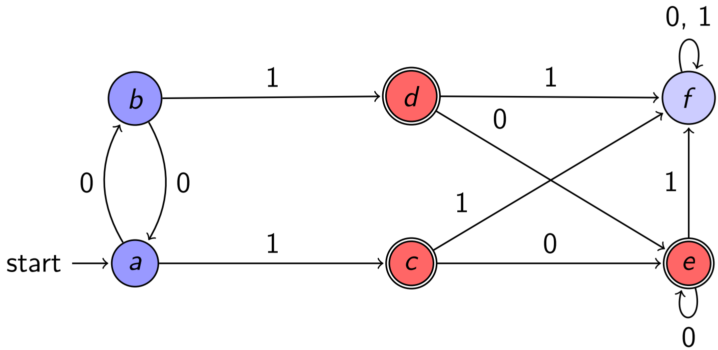
L'algorithme d'Hopcroft (exemple)



$P = \{ \bullet, \bullet, \bullet \}$
$S = \{\cancel{(\bullet, 1)}, (\bullet, 0), (\bullet, 0)\}$

Traitement de ($\bullet$, 1)
$B = \bullet \quad \longrightarrow \quad B$ est stable

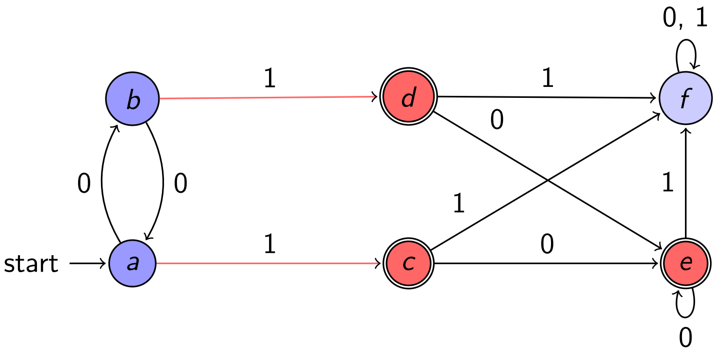
L'algorithme d'Hopcroft (exemple)



$P = \{ \bullet , \bullet , \bullet \}$
$S = \{\cancel{(\bullet , 1)}, (\bullet , 0), (\bullet , 0)\}$

Traitement de ($\bullet$, 1)
$B = \bullet \quad \longrightarrow \quad B$ est stable
$B = \bullet$

L'algorithme d'Hopcroft (exemple)

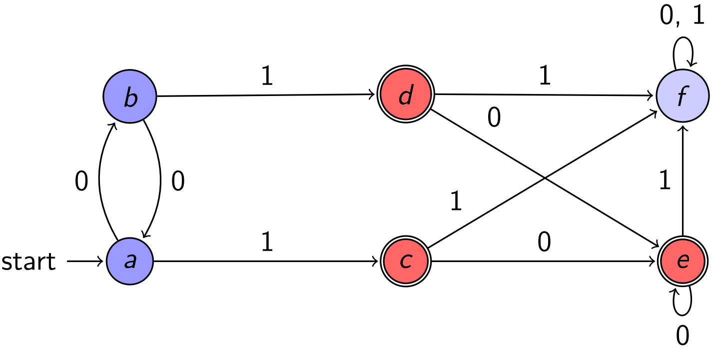


$P = \{ \bullet , \bullet , \bullet \}$
$S = \{\cancel{(\bullet, 1)}, (\bullet, 0), (\bullet, 0)\}$

Traitement de (● , 1)

$B = \bullet \quad \longrightarrow \quad B$ est stable
$B = \bullet \quad \longrightarrow \quad B$ est stable

L'algorithme d'Hopcroft (exemple)



$P = \{ ●, ●, ● \}$

$S = \{\cancel{(●, 1)}, (●, 0), (●, 0)\}$

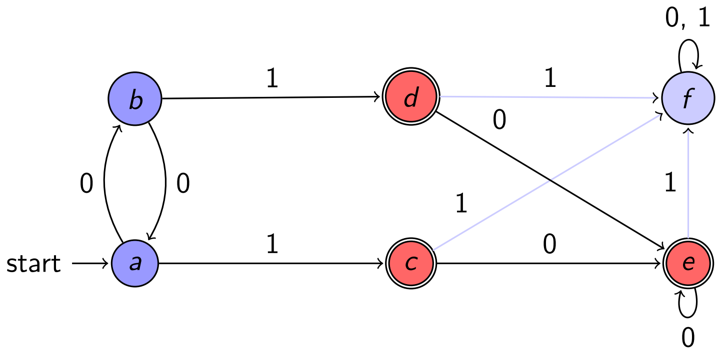
Traitement de (● , 1)

$B =$ ● $\longrightarrow$ B est stable

$B =$ ● $\longrightarrow$ B est stable

$B =$ ●

L'algorithme d'Hopcroft (exemple)



$P = \{\ \bullet\ ,\ \bullet\ ,\ \bullet\ \}$

$S = \{\cancel{(\ \bullet\ , 1)}, (\ \bullet\ , 0), (\ \bullet\ , 0)\}$

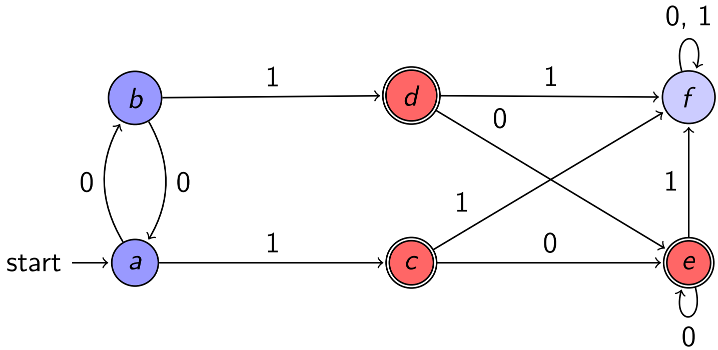
Traitement de (● , 1)

$B = \bullet \qquad \longrightarrow \qquad B$ est stable

$B = \bullet \qquad \longrightarrow \qquad B$ est stable

$B = \bullet \qquad \longrightarrow \qquad B$ est stable

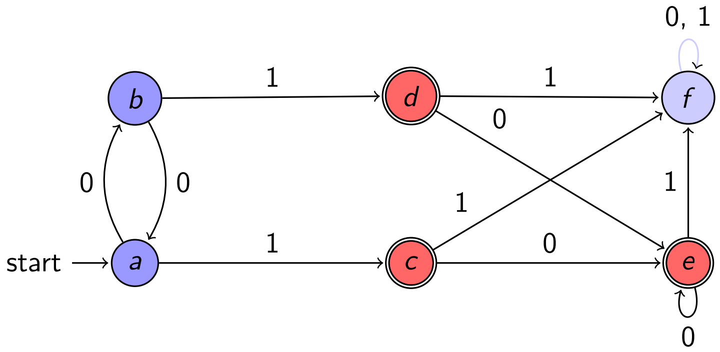
L'algorithme d'Hopcroft (exemple)



$P = \{ \bullet , \bullet , \bullet \}$
$S = \{\cancel{(\bullet , 0)}, (\bullet , 0)\}$

Traitement de (● , 0)
$B = \bullet$

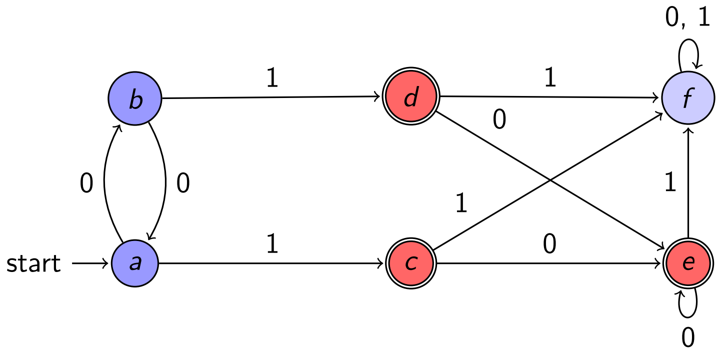
L'algorithme d'Hopcroft (exemple)



$P = \{ \bullet, \bullet, \bullet \}$
$S = \{\cancel{(\bullet, 0)}, (\bullet, 0)\}$

Traitement de ($\bullet$, 0)
$B = \bullet \quad \longrightarrow \quad B$ est stable

L'algorithme d'Hopcroft (exemple)



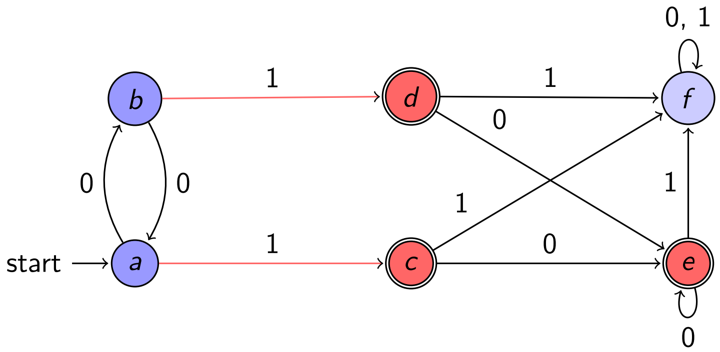
$P = \{ \bullet , \bullet , \bullet \}$
$S = \{\cancel{(\bullet , 0)}, (\bullet , 0)\}$

Traitement de ($\bullet$, 0)

$B = \bullet \quad \longrightarrow \quad B$ est stable

$B = \bullet$

L'algorithme d'Hopcroft (exemple)



$P = \{$ ● , ● , ● $\}$

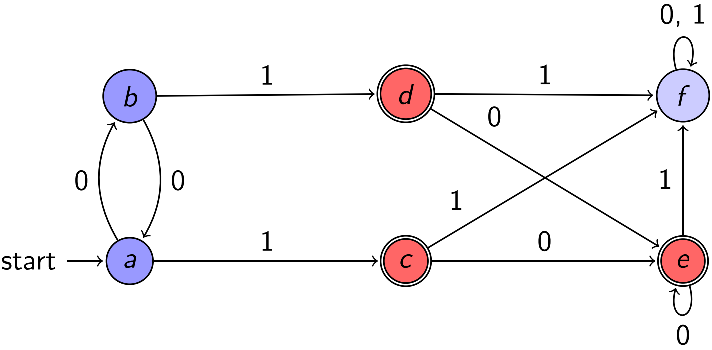
$S = \{$ ~~(● , 0)~~, (● , 0)$\}$

Traitement de (● , 0)

$B =$ ● $\longrightarrow$ B est stable

$B =$ ● $\longrightarrow$ B est stable

L'algorithme d'Hopcroft (exemple)

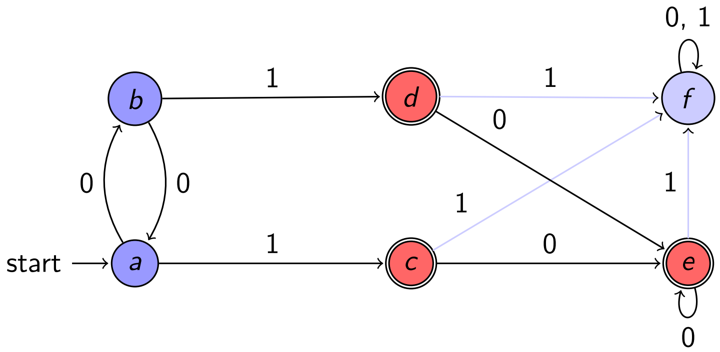


$P = \{ \bullet, \bullet, \bullet \}$
$S = \{\cancel{(\bullet, 0)}, (\bullet, 0)\}$

Traitement de ($\bullet$, 0)

$B = \bullet \longrightarrow B$ est stable
$B = \bullet \longrightarrow B$ est stable
$B = \bullet$

L'algorithme d'Hopcroft (exemple)

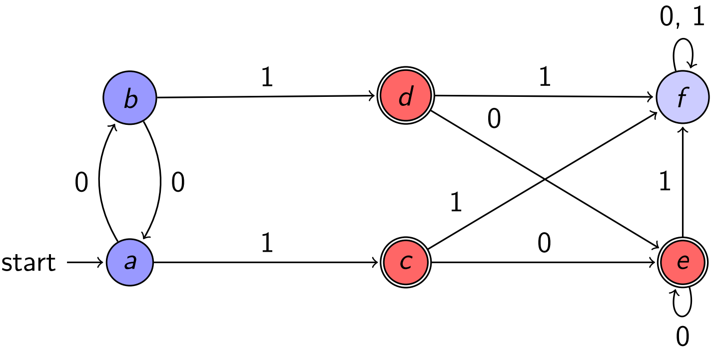


$P = \{ \bullet , \bullet , \bullet \}$
$S = \{\cancel{(\bullet , 0)}, (\bullet , 0)\}$

Traitement de ($\bullet$, 0)

$B = \bullet \longrightarrow B$ est stable
$B = \bullet \longrightarrow B$ est stable
$B = \bullet \longrightarrow B$ est stable

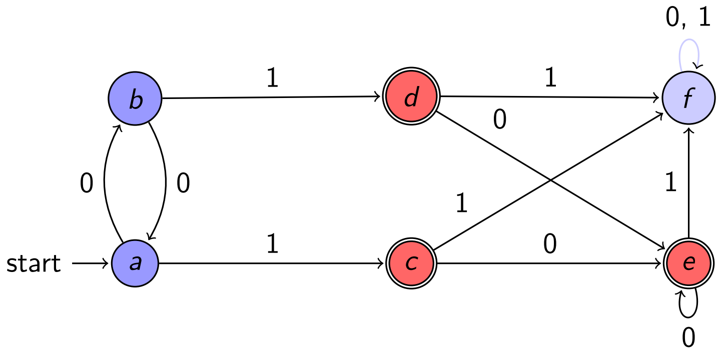
L'algorithme d'Hopcroft (exemple)



$P = \{ \bullet , \bullet , \bullet \}$
$S = \{\cancel{(\bullet , 0)}\}$

Traitement de ($\bullet$, 1)
$B = \bullet$

L'algorithme d'Hopcroft (exemple)



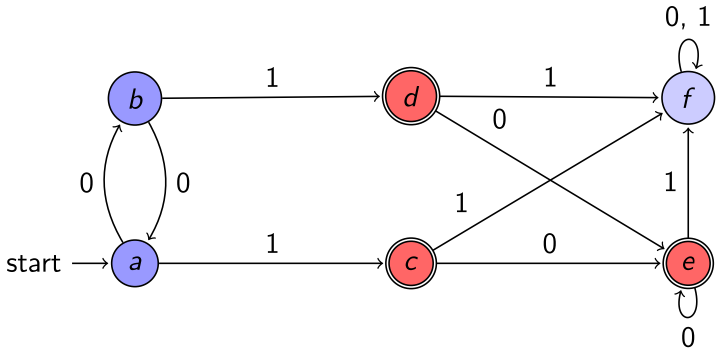
$P = \{ \bullet \, , \bullet \, , \bullet \}$

$S = \{\cancel{(\bullet \, , 0)}\}$

Traitement de ($\bullet$, 1)

$B = \bullet \qquad \longrightarrow \qquad B$ est stable

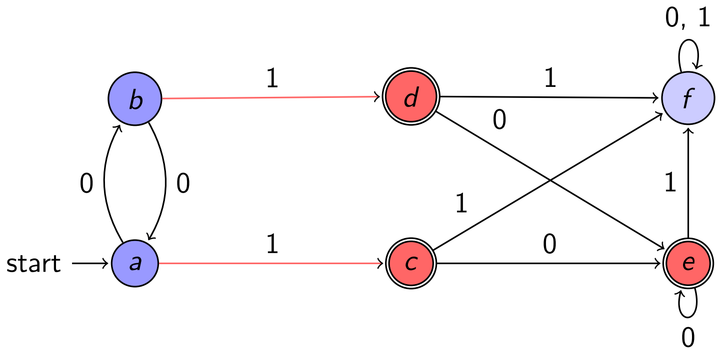
L'algorithme d'Hopcroft (exemple)



$P = \{ \bullet , \bullet , \bullet \}$
$S = \{\cancel{(\bullet , 0)}\}$

Traitement de (• , 1)
$B = \bullet \quad \longrightarrow \quad B$ est stable
$B = \bullet$

L'algorithme d'Hopcroft (exemple)



$P = \{$ ● , ● , ● $\}$

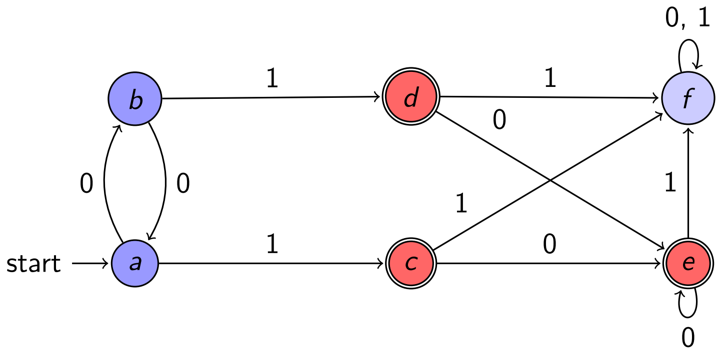
$S = \{$~~(● , 0)~~$\}$

Traitement de (● , 1)

$B =$ ● $\longrightarrow$ B est stable

$B =$ ● $\longrightarrow$ B est stable

L'algorithme d'Hopcroft (exemple)

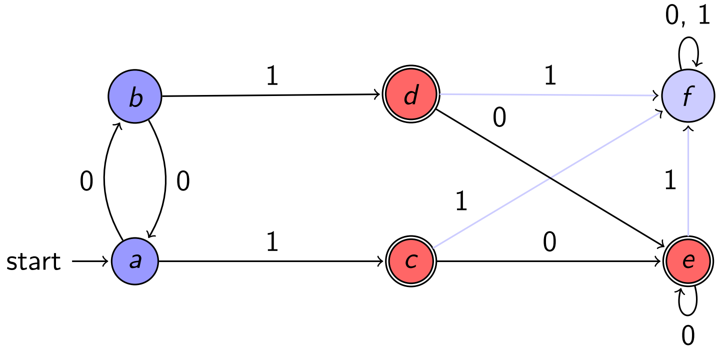


$P = \{ \bullet , \bullet , \bullet \}$
$S = \{\cancel{(\bullet , 0)}\}$

Traitement de (● , 1)

$B = \bullet \quad \longrightarrow \quad B$ est stable
$B = \bullet \quad \longrightarrow \quad B$ est stable
$B = \bullet$

L'algorithme d'Hopcroft (exemple)



$P = \{ \bullet , \bullet , \bullet \}$

$S = \{\cancel{(\bullet , 0)}\}$

Traitement de ($\bullet$, 1)

$B = \bullet \longrightarrow B$ est stable

$B = \bullet \longrightarrow B$ est stable

$B = \bullet \longrightarrow B$ est stable

L'algorithme d'Hopcroft (algorithme)

Algorithme (Hopcroft)

Data: $\mathcal{A} = (Q, \Sigma, i, F, \delta)$ *déterministe*
$P = \{F,\ Q\backslash F\}$
$S = \{(\min(F, Q\backslash F), a) \mid a \in \Sigma\}$
while $S \neq \emptyset$ **do**
    Prendre et supprimer un élément (C, a) *de* S
    for $B \in P$ *coupé en* B_1 *et* B_2 *par* (C, a) **do**
        Remplacer B *par* B_1 *et* B_2 *dans* P
        for $b \in \Sigma$ **do**
            if $(B, b) \in S$ **then**
                Remplacer (B, b) *par* (B_1, b) *et* (B_2, b) *dans* S
            else
                Ajouter $(\min(B_1, B_2), b)$ *dans* S
            end
        end
    end
end

L'algorithme d'Hopcroft (terminaison)

A chaque passage dans la boucle *while* :

- Soit la partition est raffinée
- Soit S perd un élément

Comme la partition ne peut être raffinée qu'un nombre fini de fois, S va finir par être vide. Ainsi, le programme termine.

L'algorithme d'Hopcroft (correction)

A chaque passage dans la boucle *while*, l'invariant suivant est respecté :

1. Tout (B, a) de $(P \times \Sigma) \backslash S$ laisse stable tous les éléments de P
2. $\forall (q, q') \in Q^2, L_q = L_{q'} \Rightarrow \overline{q} = \overline{q'}$ où
   - $\overline{q}$ est l'unique $B \in P$ tel que $q \in B$
   - $L_q = \{w \in \Sigma^* \mid q \cdot w \in F\}$

En sortant de la boucle *while*, on a alors :

- Tout (B, a) de $(P \times \Sigma)$ laisse stable tous les éléments de P
- $\forall (q, q') \in Q^2, L_q = L_{q'} \Rightarrow \overline{q} = \overline{q'}$

Ce qui donne :

$$\forall (q, q') \in Q^2, \overline{q} = \overline{q'} \Leftrightarrow L_q = L_{q'}$$

On obtient bien la congruence de Nerode.

L'algorithme d'Hopcroft (complexité)

La complexité des instructions dans la boucle *while* pour un séparateur (C, a) est en $O(|a^{-1}C|)$ où $a^{-1}C = \{q \in Q \mid q \cdot a \in C\}$ car on peut :

- Obtenir et parcourir les éléments de $a^{-1}q$ en $O(|a^{-1}q|)$.
- Obtenir et parcourir les éléments de $a^{-1}C$ en $O(|a^{-1}C|)$.
- Obtenir $\overline{q}$ et $|\overline{q}|$ en temps constant pour $q \in Q$.
- Obtenir la liste des éléments de P coupés par (C, a) en $O(|a^{-1}C|)$.

Pour $a \in \Sigma$ et $q \in Q$, notons $C_1, \ldots, C_k$ toutes les parties contenant q traitées par l'algorithme, classées par ordre de traitement. Alors, par construction, $\forall i \in [\![1, k-1]\!], |C_{i+1}| \leq \frac{|C_i|}{2}$ donc $k \leq \log_2(|Q|)$.

Ainsi, la complexité de l'algorithme est en $O(|\Sigma||Q|\log_2(|Q|))$ car :

$$\sum_{(C,a)\text{ traitée}} O(|a^{-1}C|) = \sum_{\substack{q \in Q \\ a \in \Sigma}} O(|\{(C, a) \mid (C, a) \text{ traitée et } q \cdot a \in C\}|)$$

$$= \sum_{\substack{q \in Q \\ a \in \Sigma}} O(\log_2(|Q|)) = O(|\Sigma||Q|\log_2(|Q|))$$

Lien avec l'algorithme de Brzozowski (algorithme 1)

Lemme

Soit $\mathcal{A} = (Q, \Sigma, i, F, \delta)$ *déterministe reconnaissant* L*. Alors,* $D(R(A))$ *est l'automate minimal reconnaissant* L^r*, où* D *donne le déterminisé,* R *le miroir et* L^r *le langage miroir de* L*.*

Algorithme & Complexité (Brzozowski naïf)

Data: $\mathcal{A} = (Q, \Sigma, i, F, \delta)$ *déterministe*
$\mathcal{A}' = D(R(D(R(A))))$

La complexité est en $O(|\Sigma| \exp(|Q|))$.

Lien avec l'algorithme de Brzozowski (algorithme 2)

L'algorithme précédent est nettement améliorable ! Un algorithme présenté dans DFA minimization : from Brzozowski to Hopcroft est polynomial. Essayons de le comprendre intuitivement.

Prenons $\mathcal{A} = (Q, \Sigma, i, F, \delta)$ déterministe. L'idée est la suivante :

1. Considérer que l'automate minimal a 2 états : $P = \{F, Q \backslash F\}$. Initialiser une liste d'état à tester : $S = \{(F, a) \,|\, a \in \Sigma\}$.
2. Pour $(F, a) \in S$, regarder si les états de l'automate $D(R(A))$ obtenus en lisant une seule lettre a depuis l'état initial I se trouvent dans les états déjà présents dans l'automate minimal (i.e. regarder si (F, a) coupe F et $Q \backslash F$). Ainsi, 2 cas :
   - Soit les états s'y trouvent, alors l'automate est bien minimal.
   - Soit ils ne s'y trouvent pas tous, alors l'automate n'est pas encore minimal. Du coup, on raffine les états et on ajoute dans S les états qui ne se trouvaient pas dans l'automate.
3. Recommencer 2 en prenant un autre élément de S.

Lien avec l'algorithme de Brzozowski (algorithme 2)

Algorithme & Complexité (Brzozowski amélioré)

Data: $\mathcal{A} = (Q, \Sigma, i, F, \delta)$ *déterministe*
$P = \{F, Q \backslash F\}$
$S = \{(\min(F, Q\backslash F), a) \mid a \in \Sigma\}$
while $S \neq \emptyset$ **do**
    Prendre et supprimer un élément (C, a) *de* S
    $P' = P$
    for $B \in P$ *coupé en* B_1 *et* B_2 *par* (C, a) **do**
        Remplacer B *par* B_1 *et* B_2 *dans* P
    end
    if $P \neq P'$ **then**
        for $b \in \Sigma$ **do**
            Ajouter $(a^{-1}C, b)$ *dans* S
        end
    end
end

La complexité est en $O(|\Sigma||Q|^2)$.

Conclusion

- Le meilleur algorithme conçu est celui d'Hopcroft avec une complexité en $O(|\Sigma||Q|\log_2(|Q|))$.
- Les algorithmes d'Hopcroft et de Brzozowski a priori différents se ressemblent beaucoup au niveau de leur implémentation.