Automate et langage régulier

Romann Szczepaniak

November 26, 2025

1 Introduction

Une expression régulière est une contrainte

1.1 Vocabulaire

1. Un alphabet : un ensemble fini de symboles, par exemple $\Sigma = \{0, 1\}$ ou $\{a, b, c\}$. Σ défini un alphabet quelconque
2. Un mot : une séquence finie de lettres sur un alphabet Σ. On peut le voir comme : $\phi : \{0, \dots, n-1\} \longmapsto \Sigma$ pour un mot de taille n
   Si u est un mot sur Σ de taille $n > 3$. Alors $u[3]$ est sa 4e lettre

1.2 Opération sur les mots

La concaténation

Exemple 1 (Exemple). *En Python, si on écrit "poulet" + "roti", on obtient "pouletroti"*

Soient u, v deux mots sur l'alphabet Σ, on note $u.v$ la concaténation de u et v, et on va appeler $w = u.v$ un mot tel que : $\begin{cases} w[i] = u[i] & \text{si } i < \|u\| \\ w[i] = v[i-n] & \text{sinon} \end{cases}$

Itération

Pour u un mot sur Σ, on note $u^2 = u.u$. Par récurrence, on note : $\begin{cases} u^{k+1} = u^k \cdot u \\ u^0 = \varepsilon \end{cases}$

Avec $\varepsilon : f : \varnothing \longmapsto \Sigma$

Et on a la propriété suivante : $\varepsilon.u = u.\varepsilon = u$ avec u un mot de Σ

En python, on a u+u = u*2, ...

Associativité

Soient u, v, w trois mots sur un alplabet Σ. On a

$$(u.v).w = u.(v.w).$$

Code Python

```python
def f(u : str, v : str, w : str):
  if (u + v) + w != u + (v + w):
    raise ValueError(_)
```

Preuve. Hypothèse de récurrence

On suppose que l'associativité est vérifiée pour tout triplet de mot u, v, w
tel que le mot u est de taille n

Cas de base : $n = 0$

$\varepsilon.(v.w) = (\varepsilon.v).w$, ce qui fonctionne

Hérédité

On suppose que l'hypothèse de récurrence est vraie pour un n. Montrons-le pour $n+1$

On suppose que u est de taille $n+1$. On peut donc écrire

$$u = a.u'.$$

avec u' de taille n

On a donc $(u.v).w = ((u'.a).v).w$

Par l'hypothèse de récurrence, on a $((u'.a).v).w = (u'.(a.v).w) = u'.((a.v).w)$

A finir

□

2 Les langages

Définition 1. *Un langage sur un alphabet Σ est un ensemble de mots sur l'alphabet Σ*

Exemple 2. *Les mots sur $\{a, b\}$ qui débutent par ab :*

$$aba, abab, abaaaab, \ldots.$$

En l'occurrence, cet ensemble de mots est infini, par exemple : $\left\{(ab)^i \mid i \geq 0\right\}$

Cas particulier du langage singleton : C'est un langage qui ne contient qu'un seul mot : $\{poulet\ roti\}$

Exemple 3. *Un langage plus complexe*

Le langage www = $\{$*ensemble des mots qui représente une page web en HTML*$\}$

Un problème de décision : Le mot est-il dans le langage ?

2.1 Opérations sur les langages

i. L, K deux langages. L'union et l'intersection sur le même alphabet

ii. L^c : Le complément : les mots qui sont sur le même alphabet mais pas sur L

iii. La concaténation : Soient L, K deux langages. On note $L.K = \{u.v | u \in L, v \in K\}$

Exemple 4. *Opérations particulières*

$\varnothing.K = \varnothing$

$\{\varepsilon\}.K = K$

$$\begin{cases} L^0 = \{\varepsilon\} \\ L^{i+1} = L.L^i \end{cases}$$

iv. L'étoile de Kleene : $L^* = \bigcup_{i \in \mathbb{N}} L^i$. On peut aussi écrire $\{ab\}^*$

v. $L^+ = L.L^*$

Exemple 5. *L'étoile de Kleene avec un alphabet*

Soit Σ un alphabet, Σ^* est l'ensemble de tous les mots de Σ. $(\Sigma, ., \varepsilon)$ est un monoïde, c'est même le monoïde libre sur Σ

Des langages intéressants

i. $\varnothing$: le langage vide

ii. $\{\varepsilon\}$: le langage qui contient le mot vide

3 Les expressions rationnelles

La base / les atomes

i. Des lettres d'un alphabet

ii. ε

iii. $\varnothing$

Si e, f sont une expresion logique, alors on a

Les opérations, rangées par ordre de priorité

i. e^* l'étoile de Kleene

ii. $e.f$ la concaténation

iii. $e|f$, l'union

iv. Les parenthèses

Exemple 7. *Exemple d'expression rationnelle*

i. $poulet(\ roti|basquaise)$

ii. $(poulet\ roti)\,|\,(poulet\ basquaise)$

Pour e une expression rationnelle, on note son langage $\mathcal{L}(e)$

Pour a une lettre :

i. $\mathcal{L}(a) = \{a\}$

ii. $\mathcal{L}(\varepsilon) = \{\varepsilon\}$

iii. $\mathcal{L}(\varnothing) = \varnothing$

iv. $\mathcal{L}(e^*) = \mathcal{L}(e)^*$

v. $\mathcal{L}(e.f) = \mathcal{L}(e).\mathcal{L}(f)$

vi. $\mathcal{L}(e|f) = \mathcal{L}(e)\,|\mathcal{L}(f)$

vii. $\mathcal{L}((e)) = \mathcal{L}(e)$

Deux expressions sont équivalentes si elles dénotent le même langage

$e_1 \equiv e_2 \iff \mathcal{L}(e) \equiv \mathcal{L}(f)$

Exemple 8. *Quelques propriétés*

i. $e_1|e_2 \equiv e_2|e_1$ *(commutativité)*

ii. $e_1|e_1 \equiv e_1$ *(idempotence)*

iii. $e_1.(e_2|e_3) \equiv e_1.e_2|e_1.e_3$

Définition 2 (Langage rationnel). *Un langage L est rationnel s'il existe une expression rationnelle e tel que $\mathcal{L}(e) = L$*

4 Les Automates

On a vu ce qu'était un langage rationnel. On va désormais voir comment grep et consors fonctionne

Définition 3 (Automate). *Un automate est un diagramme représentant un programme informatique.*

Un automate est une machine à état fini

Un automate est essentiellement :

i. Des états

ii. Des transitions entre les états qui dépendent d'événements

Chez nous, un automate va lire un mot une lettre à la fois, et mettre à jour son état à la fin de la lecture, il accepte ou rejette le mot

Voir le cours pour un exemple

Formellement, un automate est un tuple $(Q, \Sigma, \delta, i, F)$ où

i. Q est un ensemble fini d'états

ii. Σ un alphabet

iii. $\delta : Q \times E \mapsto Q$ les transitions

iv. i : l'etat initial tel que $i \in Q$

v. $F \subset Q$: Les états finaux

4.1 Les exécutions d'automates

Pour $A = (Q, \Sigma, \delta, i, F)$ on définit $c_A : \Sigma^* \mapsto Q$ la fonction qui accosie un mot a l'état qu'il atteint dans A

- $c_A(e) = i$
- On pose $u = u'.a$ et on suppose par induction que $c_A(u') = q$
- $c_A(u) = \delta(q, a)$

Définition 4. *u est calculé par A si $c_A(u) \in F$*

On note $\mathcal{L}(A)$ le langage des mots calculés par A

Un langage est reconnaissable s'il existe un automate A tel que $\mathcal{L}(A) = L$

Atome avec des automates

- $\varnothing$:

L'union des langages reconnaissables

$L = \mathcal{L}(A), K = \mathcal{L}(B)$

$\exists X$ un automate tel que $\mathcal{L}(C) = L \cup K$

On prend $A = (Q_A, \Sigma, \delta_A, i_a, F_A)$ $B = (Q_B, \Sigma, \delta_B, i_B, F_B)$

On a donc :

- $Q_C = Q_A \times Q_B$
- $i_c = (i_A, i_B)$
- $\delta_C((q, q'), a) = (\delta_A(q, a), \delta_B(q', a))$
- $F_C : \{(q, q') \in Q_C | q \in F_A \vee q' \in F_B\}$

Qui représente l'automate produit cartésien de A et B

Preuve. On va montrer que cette définition est correcte

$$c_C : \Sigma' \mapsto Q_C.$$

$c_C(u)$ c'est l'état d'arrivé da,s C après avoir lu u

$c_C = (c_A(u), c_B(u))$

Cas de base :

$c_C(\varepsilon) = (c_A, c_B) = (c_A(\varepsilon), c_B(\varepsilon))$

Hérédité :

$c_C(u.a) = \delta_C(q, a)$ avec $q_C = c_C(u)$ par HR

$q_C = (c_A(u), c_B(u)) = (q, q')$

□

Le complémentaire de L est $L^c = \mathcal{L}(B)$ tel que $B = (Q, \Sigma, \delta, i, Q \setminus F)$

5 Les automates non déterministes

Rappel

Si E est un ensemble, $\mathcal{P}(E)$ est l'ensemble des sous ensembles de E

Définition 5. *A un automate non déterministe est un tuple*

$$(Q, \Sigma, \delta, I, F).$$

où

- Q *est un ensemble fini d'états*
- Σ *est un alphabet*
- $\delta : Q \times \Sigma \to \mathcal{P}(E)$
- $I \subset Q$
- $F \subset Q$

Par exemple, on veut représenter ce langage

$$\Sigma^* aba \Sigma^*.$$

On fait ceci :

Cet automate est non déterministe : a permet de transitioner sur 0 ou 1

Formellement, on peut le décrire ainsi

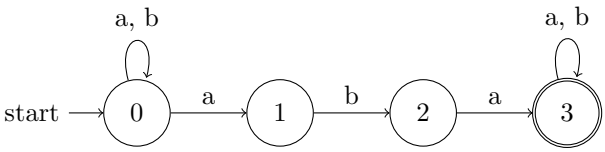


Figure 1: Automate non déterministe

Exemple 9. *Voir pdf sur le fil*

5.1 Le calcul des automates non déterministes

$A = (Q, \Sigma, \delta, I, F)$

On définit $N_A : \Sigma^* \longmapsto \mathcal{P}(Q)$ qui associe à l'ensemble des états qu'il atteint dans A

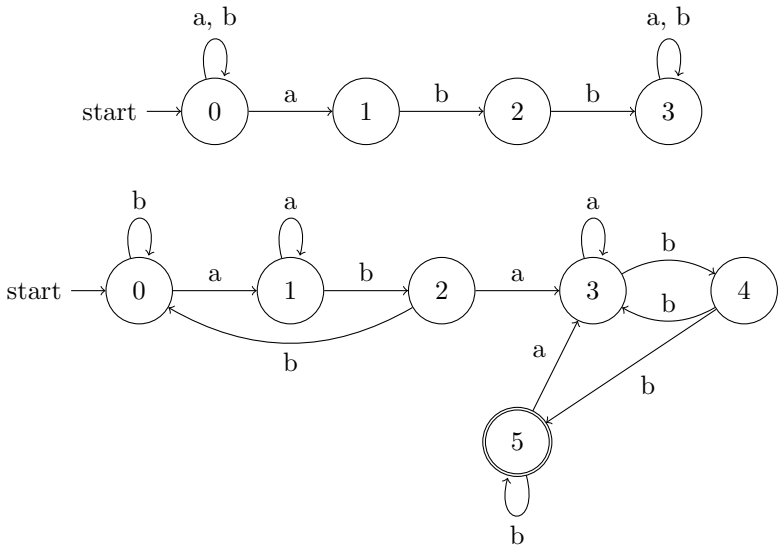
Par induction sur les mots de Σ^*

$$\begin{cases} N_A(\varepsilon) = I \\ N_A(u.a) = \bigcup\limits_{q \in N_A u} \delta(q, a) \end{cases}$$

u est accepté par A si $N_A(u) \cap F \neq \varnothing$

Théorème 1. *Soit $A = (Q, \Sigma, \delta, I, F)$ Un automate non déterministe. Il existe B un automate déterministe tel que $\mathcal{L}(A) = \mathcal{L}(B)$*

Preuve. On va construire B explicitement



Formellement :

On définit

- $Q' = \{N_A(u) \,|\, u \in \Sigma^*\}$
- Σ
- $\delta' : Q' \times \Sigma \mapsto Q'$ pour $E \in Q', \delta'(E, a) = \bigcup\limits_{q \in E} \delta(q, a) \in Q'$
- $i = I \in Q'$

- $F = (E \in Q' | E \cap F \neq \varnothing)$

□

5.2 Opération sur les unions

Cloture par union (facile)

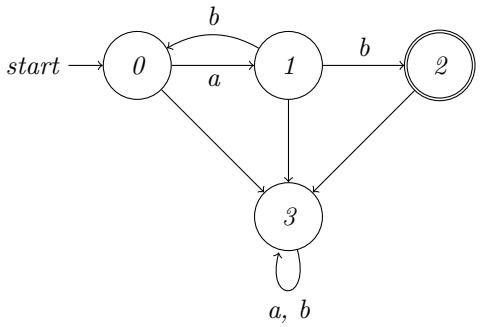
Cloture par L^+

Soit A un automate déterministe sur L et on veut calculer B pour L^+

Exemple 10. *Exemple d'une telle construction*

Le langage : $\{a, b\}$

En rajoutant la transition de 1 à 0, on obtient $(ab)^*$



$A = (Q, \Sigma, \delta, i, F)$ $B = (Q, \Sigma, \delta', \{i\}, F)$

$$\forall q, a \quad \delta(q, a) = \begin{cases} \{\delta(q, a)\} \, si \delta(q, a) \notin F \\ \{i, \delta(q, a)\} \, sinon \end{cases}$$

Cloture par concaténation

$A = (Q_A, \Sigma, \delta_A, i_A, F_A)$

$B = (Q_B, \Sigma, \delta_B, i_B, F_B)$

On veut C tel que $\mathcal{L}(C) = \mathcal{L}(A) . \mathcal{L}(B)$

$Q_C = Q_A \cup Q_B$ (disjoints)

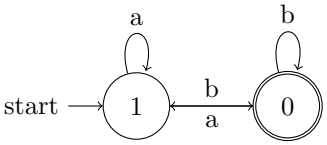
- $\delta_C : (q, a) \mapsto \begin{cases} \text{Si } q \in Q_A \text{ alors} \\ \{\delta_A(q, a)\} \text{ si } \delta_A(q, a) \notin F_A \\ \{\delta_A(q, a), i_B\} \text{ sinon} \\ \text{Si } q \in Q_B, (\delta_B(q, b)) \end{cases}$
- $I_C = \{i_A\}$
- $F_C = F_B$

6 Expressivité des langages rationnels

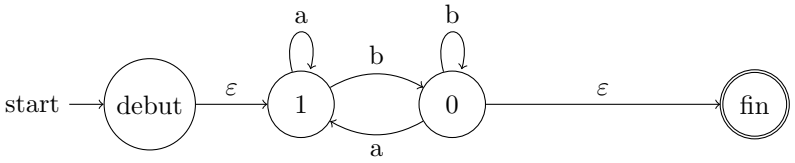
Théorème 2 (De Klenne). *Un langage est rationnel si et seulement si il est reconnaissable*

Pour un langage reconnaissable, comment obtenir une expression rationnelle ?

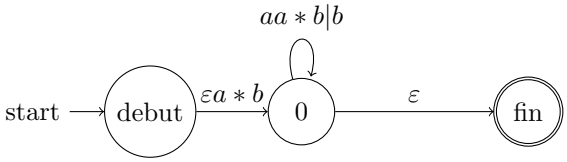
On va réaliser une élimination des états



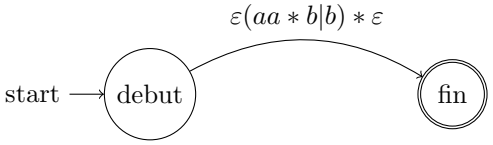
L'automate reconnaissant le langage $\Sigma^* b$



Elimination de 1



Elimination de 0

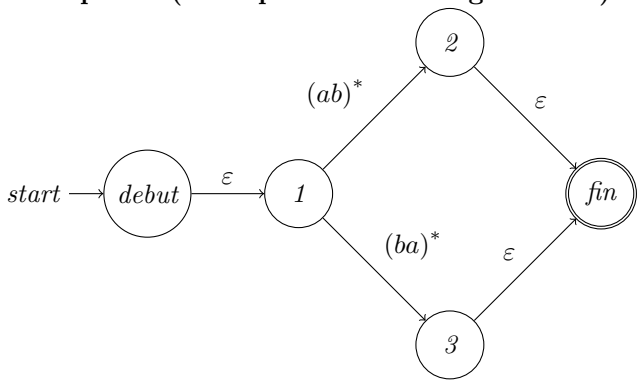


7 Automate généralisé

Un automate généralisé est un tuple de la forme

- Q : Un ensemble fini
- Σ : un alphabet
- $\delta : Q^2 \mapsto \text{Rationnel}(\Sigma)$
- debut $\in Q$
- fin $\in Q$

Exemple 11 (Exemple d'automate généralisé).



Un mot u est accepté par un automate généralisé $A = (Q, \Sigma, \delta, \text{début}, \text{fin})$ s'il existe un suite $q_1, \ldots, q_p \in Q$ tel que $u \in \mathcal{L}(\delta(\text{début}, q_1), \delta(q_1, q_2), \ldots, \delta(q_p, \text{fin}))$

Avec $\mathcal{L}(A)$ les mots acceptés par A

On prend $A = (Q, \Sigma, \delta, i, F)$ un automate déterministe et $B = (Q', \Sigma, \delta, \text{debut}, \text{fin})$

On a donc ces relations entre A et B

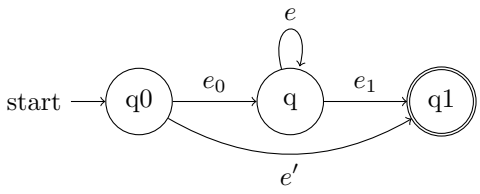
- $Q' = Q \cup \{\text{debut}, \text{fin}\}$
- $\delta' : (q, q') \begin{cases} \varnothing \text{ si } \forall a \in \Sigma \delta(q, a) = q' \\ a_1 | a_2 | ... | a_p \text{ si } \delta(q, q_i) \, \forall i \in ... \text{ pour } q, q' \neq \text{debut}, \text{fin} \end{cases}$
- $\delta'(\text{debut}, q) = \begin{cases} \varepsilon \text{ si } q = i \\ \varnothing \text{ sinon} \end{cases}$
- $\delta'(q, fin) = \begin{cases} \varepsilon \text{ si } q \in F \\ \varnothing \text{ sinon} \end{cases}$

Pour $A = (Q, \Sigma, \delta, \text{ début }, \text{ fin })$ un automate généralisé $Q \in Q \setminus \{ \text{ début }, \text{ fin } \}$

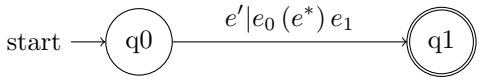
On construit $B = \{Q \setminus \{q\}, \Sigma, \delta', \text{ debut }, \text{ fin } \}$ tel que $\mathcal{L}(A) = \mathcal{L}(B)$

On va chercher :

- Dans A :



- Dans B :



Avec $e' = \delta(q_0, q_1)$

On pose alors :

$$\delta'(q_0, q_1) = \delta(q_0, q_1) | \delta(q_0, q) \delta(q, q)^* \delta(q, q_1) \forall (q_0, q_1) \in Q'.$$

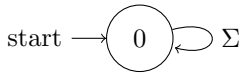
Pour tout automate déterministe, il existe une expression rationnelle qui dénote son langage

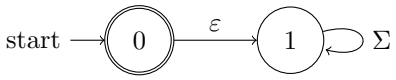
Preuve. i. On prend A déterministe on calcule B généralisé tel que $\mathcal{L}(A) = \mathcal{L}(B)$

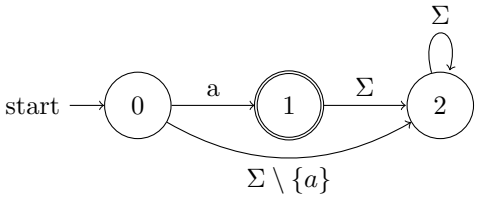
ii. On élimine les états B et on obtient C qui a une unique transition entre le début et la fin

Comme $\mathcal{L}(A) = \mathcal{L}(B) = \mathcal{L}(C)$ □

Soit e une expression rationnelle. Alors il existe un automate déterministe A tel que $\mathcal{L}(e) = \mathcal{L}(A)$
Par induction sur l'expression :

- $\varnothing$ calculé par 

- ε calculé par 

- pour $a \subset \Sigma$, 

Cas inductif :
e et f avec A et B tel que $\mathcal{L}(e) = A, \mathcal{L}(f) = B$
$\mathcal{L}(e.f) = \mathcal{L}(e).\mathcal{L}(f)$
calculable par un automate non déterministe, qu'on peut déterminiser :)
De même pour :

- $e|f$
- e^*

Ce qui conclut le théorème de Kleene

8 Lemme de l'étoile

L un langage calculé par un automate A avec n états. Si $u \in L$ avec $\|u\| > n$, alors in existe p, v et s tel que :

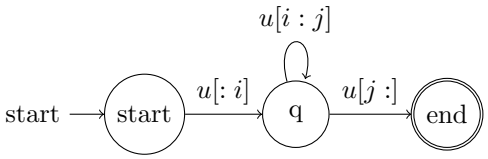
- $u = pvs$
- $pv^*s \subset L$

Preuve. $u \in L$ avec $\|u\| > n$ alors il existe $i < j < \|u\|$ tel que :

$$C_A\left(u\left[:i\right]\right) = C_A\left(u\left[:j\right]\right).$$

par le lemme des tiroirs

Donc on a la situation suivante :



□

Cas d'application

$$L = \left(a^n b^n | n \in \mathbb{N}\right).$$

n'est pas reconnaissable

Par l'absurde :

S'il existe un automate de taille p qui le calcule

$a^p b^p$, il existe p, v, s tel que $a^p b^p = pvs$ et $pv^*s \subset L$ (application du Lemme de l'étoile)

Si le v est un sous mot de a^p, on a $pv^p s = a^{p'} b^p$ avec $p' > p \notin L$

Si v est un sous mot de b^p, on a donc $pv^p s = a^p b^{p'}$ avec $p' > p$

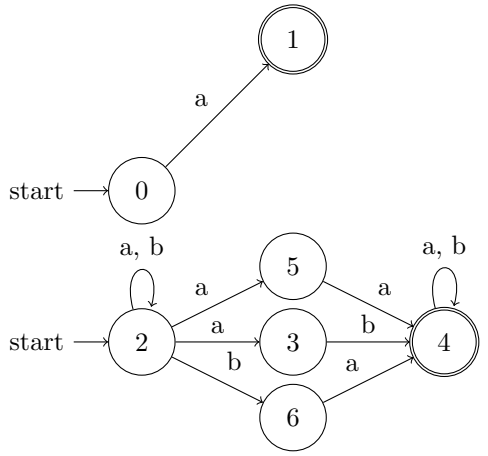
Si v est à cheval, on a $pv^p s = a^* b^*$ et donc pas dans L

Donc, un automate ne peut pas exister

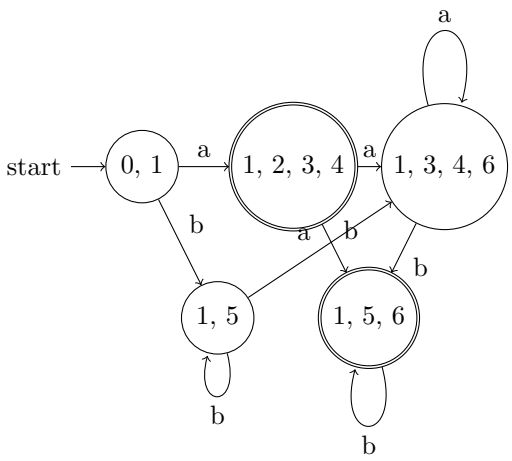
9 Minimisation d'automate déterministe

Prenons l'exemple de l'expression rationnelle : $\Sigma^* \left(ab|ba|aa\right) \Sigma^* | a$

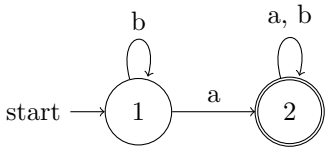
Construisons naïvement l'automate associé



En le déterminisant, on obtient :



Elle peut se simplifier en :



Mais il faut un peu réfléchir pour arriver à ça... Il faut trouver une méthode pour simplifier un automate plus facilement

9.1 Le test du vide

Si on prend deux automates, comment savoir s'ils calculent le même langage

- Tester si un langage reconnaissable est vide Accessible$(A, n) \subset Q$ les états qu'on peut atteindre avec un mot de taille n :
  - cas de base : Accessible $(A, c) = I$
  - Accessible $(A, n+1) = \bigcup_{q \in \text{Accessible}(A,n), a \in \Sigma} \delta(q', a)$

  Accessible $(A) = \bigcup_{n \in \mathbb{N}}$ Accessible $(A, n) = \bigcup_{n \leq \|A\|}$ Accessible (A, n)
- Un mot est accepté si : $F \cap$ Accessible $(A) \neq \varnothing$
- L, K reconnaissable. On peut vérifier algorithmiquement que $L = K$ en vérifiant que ; $\begin{cases} (L \cap K^c) = \varnothing \\ (L^c \cap K) = \varnothing \end{cases}$

Définition 6 (Résiduel). • $L \subset \Sigma^*$ *un langage*

- $u \in \Sigma^*, u^{-1}L = \{v \in \Sigma^* | uv \in L\}$

Exemple 12. $L = (ab)*$

$a^{-1}L = (ab)*$

$(ab)^{-1} L = (ab)* = L$

$b^{-1}L = \varnothing$

Si on prend un automate $A = (Q, \Sigma, \delta, i, F)$ déterministe et $A_q = (Q, \Sigma, \delta, q, F)$ avec $q \in$ Accessible (A)

On a que $\mathcal{L}(A_q) = u^{-1}L$ pour un mot tel que $c_a(u) = q$

$\forall u, v \in \Sigma^*$ tel que $c_A(u) = c_A(w)$, alors $u^{-1}L = v^{-1}L$

Grâce à ça, un automate qui calcule un lagage L a au moins autant d'états que de résidus de L

9.2 L'automate des résidus

On fixe L un langage reconnaissabme et on pose $R = \left\{u^{-1}L | u \in \Sigma^*\right\}$

On a que $\|R\|$ est plus petit que la taille d'un automate déterministe qui calcule L

$A_r = (R, \Sigma, \delta_r, L, F_r)$

$\delta_r : \begin{cases} R \times \Sigma \mapsto R \\ \left(u^{-1}L, a\right) \mapsto (ua)^{-1} L = a^{-1}\left(u^{-1}L\right) (admis) \end{cases}$

$F_R = \{K \in R^n | \varepsilon \in R\}$

Par construction, l'automate des résidus reconnait L. C'est donc <u>un</u> automate minimal.

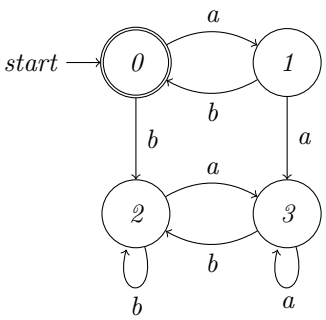
Tous les automates de cette taille sont en fait "égaux" (à isomorphisme près), c'est l'automate canonique

Comment construire l'automate minimal ?

Il faut identifier les états qui calculent le même résidus et les regrouper

Exemple 13. *Minimisation d'un automate*

Prenons l'automate suivant :



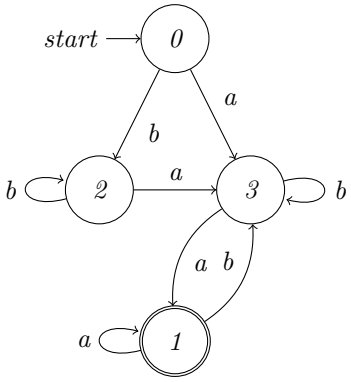
On a $L = (ab)*$

- $0 \mapsto L$
- $1 \mapsto a^{-1}L = b(ab)*$

- $2 \mapsto b^{-1}L = \varnothing$
- $3 \mapsto (ab) * L = \varnothing$

Exemple 14. *Un autre exemple*

b



- $0 \mapsto \varepsilon^{-1}L = L$
- $a^{-1}L = \Sigma^*$
- $b^{-1}L = L$
- $(aa)^{-1}\, L = \Sigma^*$