1 Introduction

L'étude des langages est une branche riche de l'informatique qui trouve de nombreuses applications. La famille de langages étudiées dans ce chapitre est la famille des **langages rationnels** qui forme la première classe de la hiérarchie de Chomsky.

1.1 Les langues naturelles

Historiquement, la théorie des langages a été étudiée pour formaliser les langues naturelles, c'est-à-dire les langues utilisées par l'espèce humaine pour communiquer. On peut distinguer différents niveaux de caractérisation pour un traitement automatique :

- Reconnaître un mot d'un langage, éventuellement une conjugaison ou un accord, selon des règles finies. Les langages rationnels peuvent fournir un outil de correction.
- S'assurer qu'une phrase est bien formée : en prenant un ensemble de mots bien formés, toutes les combinaisons ne forment pas nécessairement une phrase cohérente. Les langages rationnels ne sont pas un outil assez puissant pour ce genre d'étude, et il est pour cela nécessaire d'utiliser le deuxième niveau de la hiérarchie de Chomsky : les **langages algébriques**, utilisant la notion de **grammaire hors-contexte**.

1.2 La compilation

L'être humain et l'ordinateur ne sont pas fait pour communiquer. L'être humain comprend des mots et des phrases, le plus souvent dans sa langue natale, alors que l'ordinateur ne comprend que des signaux électriques (présence ou absence de signal). Agissant comme médiateurs, les langages de programmation sont séparés (de manière continue) en langages de haut niveau (par exemple OCaml) et bas niveau (par exemple Assembleur), les premiers étant facilement compréhensibles par un programmeur (mots-clés en anglais, indentation, ...) et les seconds par l'ordinateur (prise en compte des contraintes matérielles du processeur).

Exemple 1.1

Affichage de « Bonjour » en OCaml et Assembleur :

```
print_string "Bonjour";;
```

```
str:
 .ascii "Bonjour"
 .global _start

_start:
movl $4, %eax
movl $1, %ebx
movl $str, %ecx
movl $8, %edx
int $0x80
movl $1, %eax
movl $0, %ebx
int $0x80
```

La compilation est l'outil permettant la « traduction » d'un langage de haut niveau vers un langage de bas niveau. La compilation contient généralement trois analyses successives du code :

- L'analyse lexicale : on s'assure que les mots utilisés dans le code sont corrects, que les mots clés du langage sont bien orthographiés, que les noms de variables vérifient les règles d'écriture (par exemple une variable en OCaml ne doit pas commencer par un chiffre). L'étude des langages rationnels permettent à nouveau de répondre à ce type d'analyse.
- L'analyse syntaxique : on s'assure que les séquences formées par les mots sont correctes. Par exemple, `for begin i = to [||]` ne comporte pas d'erreur de lexique, mais ne passera pas l'analyse syntaxique.

Comme pour l'étude de la grammaire d'une langue naturelle, c'est la classe des langages algébriques qui répond à la question.

– L'analyse sémantique : on s'assure ici que les opérations effectuées sont possibles. Une erreur de typage en OCaml résulte d'un échec de l'analyse sémantique. Par exemple, `let x = [|1; 2; 3|] in x + 5;;` ne présentera pas d'erreur d'analyse syntaxique, mais une erreur d'analyse sémantique.

1.3 La recherche d'expressions

N'importe qui a déjà utilisé un éditeur de texte s'est servi d'une fonction de recherche. Sans besoin d'être sophistiqué, un éditeur peut permettre plus que la simple recherche de mot, c'est-à-dire la recherche d'expressions vérifiant certaines règles. Les systèmes Unix dispose de la commande `grep` (pour Globally search a Regular Expression and Print), permettant la recherche de texte, ou de fichiers et dossiers selon leurs noms.

Exemple 1.2: La commande `grep`

La commande ci-dessous permet de trouver tous les fichiers :

– commençant éventuellement par un numéro suivi de « - »,
– suivi par « exercice » ou « Exercice »,
– contenant le mot « graphe » ou « arbre »,
– terminant par « .pdf » ou « .tex ».

```
find | grep -E '.*/([0-9]+-)?[Ee]xercice[^/]*([Gg]raphe|[Aa]rbre)[^/]*\.(pdf|tex)$'
```

2 Mots et langages

2.1 Mots

2.1.1 Alphabet et mots

Définition 2.1

Un alphabet Σ est un ensemble fini non vide. Les éléments d'un alphabet sont appelés lettres.

Exemple 2.2

En français, l'alphabet courant (sans accentuation) comporte 26 lettres :

$$\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}.$$

Définition 2.3

Un mot u sur un alphabet Σ est soit le mot vide ε, soit une suite finie $a_1 a_2 \dots a_n$ avec, pour tout $k \in [\![1, n]\!]$, $a_k \in \Sigma$, c'est-à-dire, chaque a_k est une lettre de Σ.

La longueur d'un mot u est l'entier, noté $|u|$, qui vaut 0 si $u = \varepsilon$ et n si $u = a_1 a_2 \dots a_n$ avec, pour tout $k \in [\![1, n]\!]$, $a_k \in \Sigma$.

La longueur en $a \in \Sigma$ d'un mot u est l'entier, noté $|u|_a$, qui vaut 0 si $u = \varepsilon$ et qui vaut :

$$\text{card}\{k \in [\![1, n]\!],\ a_k = a\}$$

si $u = a_1 a_2 \dots a_n$ avec, pour tout $k \in [\![1, n]\!]$, $a_k \in \Sigma$. Ainsi, $|u|_a$ est le nombre d'occurrences de a dans u.

Exemple 2.4

Sur l'alphabet courant, *baobab* est un mot de longueur 6 et de longueur 3 en b.

Remarque 2.5: Notation

L'ensemble des mots de longueur n sur Σ est noté Σ^n. L'ensemble de tous les mots sur Σ est noté Σ^*. Par construction,

$$\Sigma^* = \bigcup_{n\in\mathbb{N}} \Sigma^n = \{\varepsilon\} \cup \bigcup_{n\in\mathbb{N}^*} \Sigma^n.$$

L'ensemble de tous les mots non vides sur Σ est noté Σ^+. Par construction,

$$\Sigma^+ = \bigcup_{n\in\mathbb{N}^*} \Sigma^n.$$

Remarque 2.6

On utilise usuellement les lettres du début de l'alphabet (a, b, c, ...) pour désigner des éléments de Σ et celles de la fin de l'alphabet (u, v, w, ...) pour désigner des éléments de Σ^*.

2.1.2 Concaténation

Définition 2.7

La concaténation de deux mots $u = a_1a_2\dots a_n$ et $v = b_1b_2\dots b_p$ est le mot uv (ou $u\cdot v$) de longueur $n+p$ dont le $i^{\text{ième}}$ caractère est a_i si $i \leqslant n$ et b_{i-n} sinon. Si $u \in \Sigma^*$, alors $u\varepsilon = \varepsilon u = u$.

Exemple 2.8

La concaténation de *abaa* et de *abb* est *abaaabb*.

Remarque 2.9

Grâce à l'associativité de la concaténation, on peut simplifier les écritures : on notera a^n pour le mot $aa...a$ où a apparaît n fois et u^n le mot $uu...u$ où u apparaît n fois.

Les éléments sont simplifiables à gauche comme à droite ($uv = uw$ ou $vu = wu$ entraîne $v = w$).

Le triplet $(\Sigma^*, \cdot, \varepsilon)$ forme un monoïde libre, c'est-à-dire un ensemble muni d'une loi de composition interne associative ($\cdot$), d'un élément neutre (ε) et d'une base (Σ).

On a, par définition, l'additivité de la longueur pour la concaténation. On a également :

Proposition 2.10

Soit u, $v \in \Sigma^*$. Alors, pour tout $a \in \Sigma$, $|uv|_a = |u|_a + |v|_a$.

2.1.3 Préfixe, suffixe, facteur

Définition 2.11

Soit $u, x \in \Sigma^*$. On dit que :
- x est un **préfixe** de u s'il existe un mot $y \in \Sigma^*$ tel que $u = xy$;
- x est un **suffixe** de u s'il existe un mot $y \in \Sigma^*$ tel que $u = yx$.

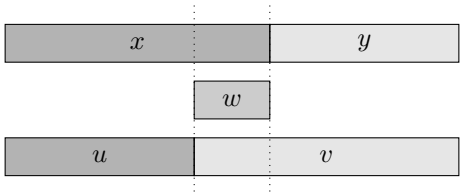
Exemple 2.12

Les préfixes de *baobab* sont ε, b, ba, bao, $baob$, $baoba$ et $baobab$.

Proposition 2.13: Lemme de Levi

Soit x, y, u et $v \in \Sigma^*$ tels que $xy = uv$. Alors, il existe un unique mot w tel que l'une des conditions suivantes est réalisée
- $x = uw$ et $v = wy$;
- $u = xw$ et $y = wv$.

Ce résultat s'illustre par la figure suivante :

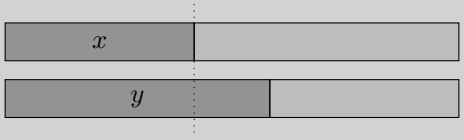


Preuve

On peut faire une preuve formelle, mais le schéma précédent suffit pour comprendre le principe.

Corolaire 2.14

Soit x, y deux préfixes (respectivement suffixes) d'un mot u. Alors, x est un préfixe (respectivement suffixe) de y ou y est un préfixe (respectivement suffixe) de x.



Preuve

Définition 2.15

Soit $x, u \in \Sigma^*$. On dit que x est un **facteur** de u s'il existe des mots y, $z \in \Sigma^*$ tel que $u = yxz$.

Exemple 2.16

Le mot *aoba* est un facteur de *baobab*.

Définition 2.17

On dit que le mot x de longueur k est un **sous-mot** du mot $u = a_1...a_n$, s'il existe des indices $1 \leqslant i_1 < i_2 < ... < i_k \leqslant n$ tels que :

$$x = a_{i_1} a_{i_2} ... a_{i_k}$$

Exemple 2.18

Sur l'alphabet usuel, *baobab* et *chene* sont des sous-mots de *bachobenbaleb*.

Remarque 2.19

Attention à la différence entre facteur et sous-mot. Une confusion est d'autant plus probable en comparaison avec l'anglais où le mot « facteur » est traduit par "*subword*" ou "*substring*" et le mot « sous-mot » est traduit par "*subsequence*".

Exercice 1

Montrer que deux mots qui ont le même ensemble de préfixes sont égaux.

Exercice 2

Soit u et v des mots non vides. Montrer qu'il existe des mots x et y tels que $u = xy$ et $v = yx$ si, et seulement s'il existe un mot w tel que $uw = wv$.

2.2 Langages

Définition 2.20

Un **langage** sur l'alphabet Σ est une partie de Σ^*, c'est-à-dire un ensemble de mots.

Exemple 2.21

Voici quelques exemples de langages sur l'alphabet $\{a, b\}$

- $\{a, ab, aba, abab\}$,
- $\{a, b\}^*$,
- l'ensemble des mots se terminant par *bab*,
- l'ensemble des mots de longueur impaire,
- l'ensemble des palindromes.

Les langages étant des ensembles, ils sont susceptibles d'être soumis aux opérations ensemblistes : réunion, intersection, passage au complémentaire, différence symétrique...

Définition 2.22

Soit L, L' deux langages sur l'alphabet Σ. La concaténation de L et L' est le langage LL' défini par

$$LL' = \{uv,\ u \in L,\ v \in L'\}.$$

Exemple 2.23

Si L est le langage des mots comprenant uniquement la lettre a et si L' est le langage des mots comprenant uniquement la lettre b, alors LL' est le langage des mots s'écrivant comme une succession de a puis une succession de b.

Définition 2.24

Soit L un langage. Pour un entier $n \in \mathbb{N}$, on définit L^n par récurrence par :
- $L^0 = \{\varepsilon\}$;
- $L^{n+1} = LL^n = L^nL$ si $n \in \mathbb{N}$.

Remarque 2.25

Ne pas confondre L^n et le langage plus petit $\{u^n,\ u \in L\}$. Par exemple, pour $L = \{ab, ba\}$,

$$\begin{aligned} L^2 = \{u,\ u \in L\}^2 &= \{abab, abba, baba, baab\}, \\ \{u^2,\ u \in L\} &= \{abab, baba\}. \end{aligned}$$

Définition 2.26

Soit L un langage sur l'alphabet Σ. L'étoile (de Kleene) de L est le langage L^* défini par $L^* = \bigcup_{n\in\mathbb{N}} L^n$.

De même, $L^+ = \bigcup_{n\in\mathbb{N}^*} L^n$.

Exercice 3: Lemme d'Arden

Soit $\Sigma = \{a, b\}$. On considère l'équation $L = \{a\}L \cup \{b\}$ d'inconnue le langage $L \subseteq \Sigma^*$.

1. Montrer que $L = \{a\}^*\{b\}$ est solution de cette équation.
2. Montrer qu'il s'agit de l'unique solution.

On suppose maintenant que A, B sont deux langages sur Σ. On considère l'équation $L = AL \cup B$ d'inconnue L.

3. Montrer que $L = A^*B$ est solution de cette équation.
4. Montrer que si de plus $\varepsilon \notin A$, cette solution est unique.

3 Automates finis

Une machine à état est une machine théorique qui peut se situer dans différents états et agit en fonction d'événements entrants :

- soit en changeant d'état ;
- soit en produisant une sortie.

Par exemple, un distributeur automatique de billet peut être considéré comme une machine à états. Les états peuvent être « en attente de la CB », « en attente du code », ... Chaque action de l'utilisateur peut modifier l'état (ou le laisser identique).

Les automates finis sont des machines à états dont les entrées sont des mots. Ils peuvent être utilisés pour faire de la reconnaissance d'expression dans un texte, ou lors de la compilation pour assurer l'analyse lexicale.

3.1 Calculs sur un automate

3.1.1 Définition

Le modèle théorique présenté ici permet d'effectuer la lecture pas à pas d'un mot.

Définition 3.1

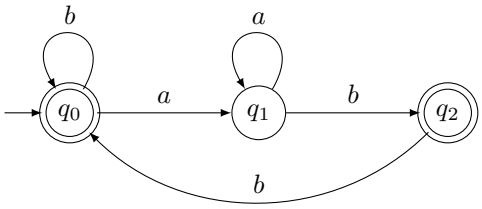
Un **automate fini déterministe** (ou AFD) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ composé de :
- Σ, un alphabet ;
- Q, un ensemble fini non vide dont les éléments sont appelés **états** de l'automate ;
- δ une fonction partielle de $Q \times \Sigma$ dans Q, appelée **fonction de transition** ;
- $q_0 \in Q$, appelé **état initial** ;
- $F \subseteq Q$, appelé ensemble des **états finaux**.

Exemple 3.2

Soit l'automate $A = (\{a, b\}, \{q_0, q_1, q_2\}, \delta, q_0, \{q_0, q_2\})$, où δ est définie par le tableau suivant (appelé **table de transition**)

	a	b
$\to q_0 \to$	q_1	q_0
q_1	q_1	q_2
$q_2 \to$	$-$	q_0

On le représente de la façon suivante (appelée **diagramme de transition**) (remarquons que l'état initial est indiqué avec une flèche entrante et les états finaux par un double cercle) :



3.1.2 Langage reconnu par un automate

Définition 3.3

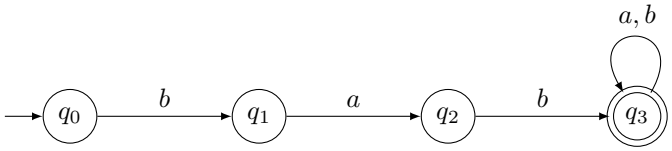
Soit un automate $A = (Q, \Sigma, \delta, q_0, F)$. La **fonction de transition étendue** δ^* est la fonction partielle $\delta^* : Q \times \Sigma^* \to Q$ définie par induction par :
- pour tout $q \in Q$, $\delta^*(q, \varepsilon) = q$;
- pour tous $q \in Q$, $u \in \Sigma^*$ et $a \in \Sigma$, $\delta^*(q, ua) = \delta(\delta^*(q, u), a)$ ou de manière équivalente : $\delta^*(q, au) = \delta^*(\delta(q, a), u)$.

Définition 3.4

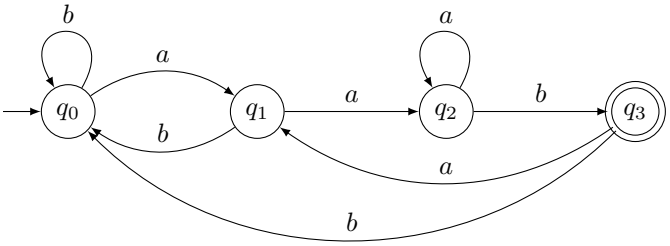
- Un mot $u \in \Sigma^*$ est dit **reconnu** par A si $\delta^*(q_0, u)$ est bien défini et $\delta^*(q_0, u) \in F$.
- Le **langage reconnu** par A, noté $L(A)$, est l'ensemble des mots reconnus par A.
- Un langage L est dit **reconnaissable** s'il existe un AFD A tel que $L = L(A)$. On note $\mathrm{Rec}(\Sigma)$ l'ensemble des langages reconnaissables sur Σ.
- Deux automates A et B sont dits **équivalents** si $L(A) = L(B)$.

Exemple 3.5

L'automate suivant reconnaît le langage $\{bab\}\Sigma^*$, c'est-à-dire l'ensemble des mots commençant par bab.



L'automate suivant reconnaît l'ensemble des mots se terminant par aab.



Remarque 3.6

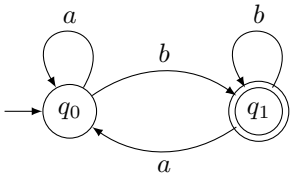
Tous les langages ne sont pas reconnaissables : l'ensemble des automates est dénombrable alors que l'ensemble des langages ne l'est pas.

En effet, Σ^* est un ensemble infini, donc $\mathcal{P}(\Sigma^*)$, qui est l'ensemble des langages est indénombrable.

D'autre part, les ensembles d'états possibles sont dénombrables (isomorphe à l'ensemble des $[\![0, n-1]\!]$, pour $n \in \mathbb{N}^*$) et une fois Q fixé, il n'y a qu'un nombre fini de façons de choisir q_0, F et δ.

Exercice 4

Déterminer le langage reconnu par l'automate suivant :



Exercice 5

Déterminer des diagrammes de transition d'AFD sur $\Sigma = \{a, b\}$ pour les langages suivants. On essaiera de minimiser le nombre d'états.

1. $L = \{a^n b^m | n, m \in \mathbb{N}\}$
2. L'ensemble des mots ayant aab comme sous-mot.
3. L'ensemble des mots ayant aab comme préfixe.
4. L'ensemble des mots ayant aab comme suffixe.
5. L'ensemble des mots ayant aab comme facteur.
6. L'ensemble des mots ne contenant pas aab comme facteur.
7. L'ensemble des mots dont la dernière lettre apparaît au moins deux fois dans le mot.

Exercice 6

Déterminer des diagrammes de transition d'AFD sur $\Sigma = \{0, 1\}$ pour les langages suivants :

1. l'ensemble des mots qui sont la représentation binaire d'un nombre pair ;
2. l'ensemble des mots qui sont la représentation binaire d'un multiple de 3. On prouvera la correction de l'automate ;
3. l'ensemble des mots qui sont la représentation binaire d'un multiple de 6.

On autorisera les représentations binaires à commencer par un nombre quelconque de zéros.

3.2 Caractéristiques des automates

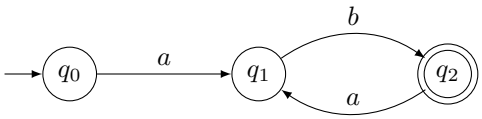
3.2.1 Complétion

Définition 3.7

Un **blocage** de l'automate $(Q, \Sigma, \delta, q_0, F)$ est un couple $(q, a) \in Q \times \Sigma$ pour lequel la fonction de transition n'est pas définie. Un automate sans blocage est dit **complet**.

Exemple 3.8

L'automate suivant n'est pas complet.



En effet, il y a trois blocages : (q_0, b), (q_1, a) et (q_2, b).

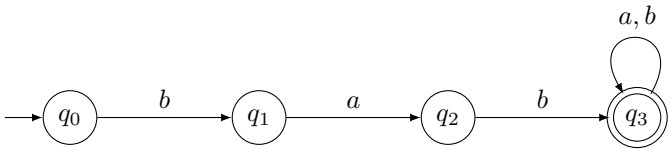
Proposition 3.9

Tout langage reconnaissable est reconnu par un automate complet.

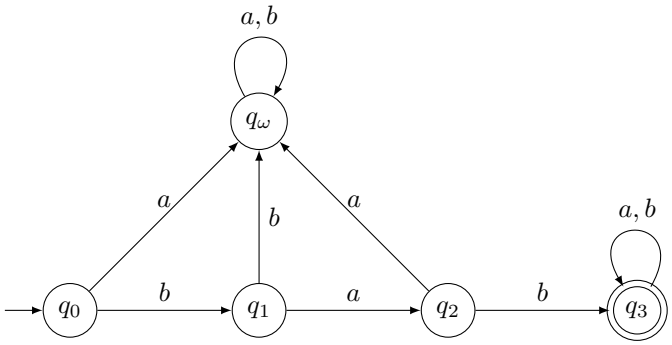
Preuve

Exemple 3.10

Avec ce résultat, on passe de l'automate incomplet



à l'automate complet



3.2.2 Émondage

Définition 3.11

Soit un automate $(Q, \Sigma, \delta, q_0, F)$ et δ^* la fonction de transition étendue.

- Un état $q \in Q$ est dit **accessible** s'il existe un mot $u \in \Sigma^*$ tel que $\delta^*(q_0, u) = q$.
- Un état $q \in Q$ est dit **co-accessible** s'il existe un mot $u \in \Sigma^*$ tel que $\delta^*(q, u) \in F$.
- Un état $q \in Q$ est dit **utile** s'il est accessible et co-accessible.
- L'automate est dit **émondé** si tous ses états sont utiles.

Remarque 3.12

Si un automate reconnaît un langage non vide, alors il existe au moins un état utile et un état final accessible.

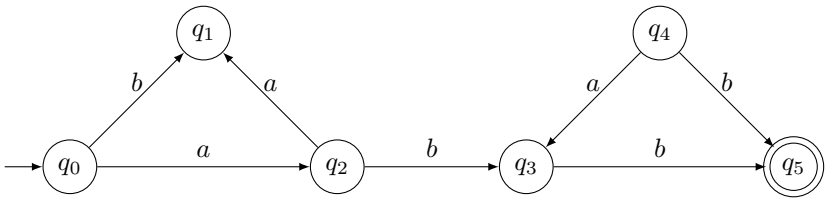
Proposition 3.13

Tout langage reconnaissable non vide est reconnu par un automate émondé.

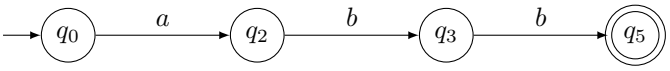
Preuve

Exemple 3.14

Pour l'automate



l'état q_4 est le seul à ne pas être accessible, et q_1 le seul à ne pas être co-accessible. En retirant ces états et les transitions associées, on obtient l'automate émondé suivant (qui reconnaît le même langage)



3.2.3 Standardisation

Définition 3.15

Un automate $(Q, \Sigma, \delta, q_0, F)$ est dit **standard** s'il n'existe aucune transition vers l'état initial, c'est-à-dire s'il n'existe pas de couple $(q, a) \in Q \times \Sigma$ tel que $\delta(q, a) = q_0$.

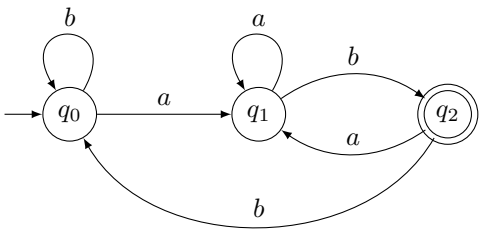
Proposition 3.16

Tout langage reconnaissable est reconnu par un automate standard.

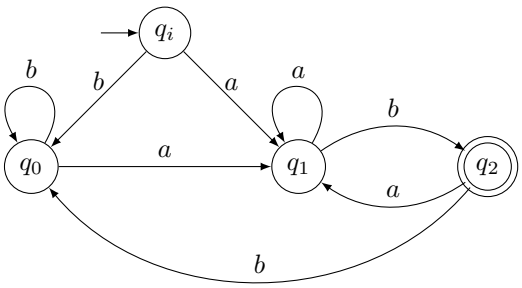
Preuve

Exemple 3.17

L'automate suivant

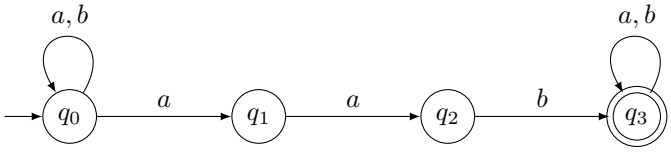


admet la version standardisée



3.3 Automates non déterministes

Une représentation « intuitive » d'un automate reconnaissant les mots ayant pour facteur aab est la suivante :



Cependant, dans cette représentation, $\delta(q_0, a)$ possède deux images possibles (q_0 et q_1). Ainsi, lors du calcul d'un mot par un automate, on souhaiterait se laisser la possibilité de plusieurs choix lors de la lecture d'une lettre. Cette notion est formalisée par les automates non déterministes.

3.3.1 Définition

Définition 3.18

Un **automate fini non déterministe** (ou AFND) est un quintuplet $(Q, \Sigma, \Delta, I, F)$ composé de
- Σ, un alphabet ;
- Q, un ensemble fini dont les éléments sont appelés **états** de l'automate ;
- Δ une application de $Q \times \Sigma$ dans $\mathcal{P}(Q)$, appelée **fonction de transition** ;
- $I \subseteq Q$, appelé ensemble des **états initiaux** ;
- $F \subseteq Q$, appelé ensemble des **états finaux** (ou acceptants).

Remarquons qu'il n'y a plus de « blocage » au sens déjà étudié, mais Δ peut atteindre la valeur $\emptyset$.

3.3.2 Calculs sur un automate non déterministe

Définition 3.19

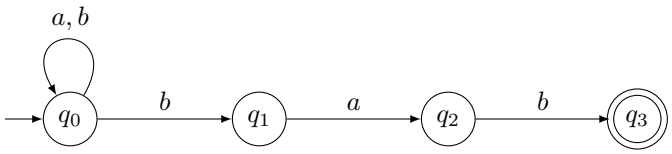
Soit un automate non déterministe $A = (Q, \Sigma, \Delta, I, F)$. La fonction de transition Δ^* étendue aux mots est la fonction $\Delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$ définie récursivement par
- pour tout $q \in Q$, $\Delta^*(q, \varepsilon) = \{q\}$;
- pour tous $q \in Q$, $u \in \Sigma^*$ et $a \in \Sigma$, $\Delta^*(q, ua) = \Delta(\Delta^*(q, u), a)$.

On définit de manière similaire aux automates déterministes la notion de langage reconnu.
- Un mot $u \in \Sigma^*$ est dit **reconnu** par A si $\Delta^*(I, u) \cap F \neq \emptyset$.
- Le **langage reconnu** par A, noté $L(A)$, est l'ensemble des mots reconnus par A.

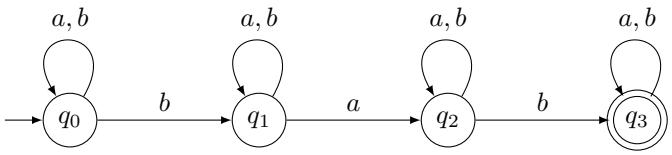
Moins formellement, u est accepté par l'automate si parmi tous les chemins possibles lors de la lecture de u, il en existe un d'un état initial vers un état final.

Exemple 3.20

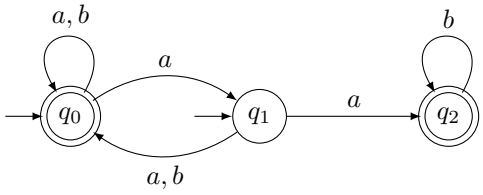
L'automate non déterministe suivant reconnaît le langage $\Sigma^*\{bab\}$, c'est-à-dire l'ensemble des mots se terminant par bab.



L'automate suivant reconnait le langage $\Sigma^*\{b\}\Sigma^*\{a\}\Sigma^*\{b\}\Sigma^*$, c'est-à-dire l'ensemble des mots dont bab est un sous-mot.



L'automate suivant reconnait Σ^*, c'est-à-dire tous les mots.



Exercice 7

Déterminer un AFND reconnaissant les langages suivants sur $\Sigma = \{a, b\}$:

1. L'ensemble des mots commençant par aa **ou** terminant par bb.
2. L'ensemble des mots avec un a en avant-avant-dernière position.

3.3.3 Epsilon-transitions

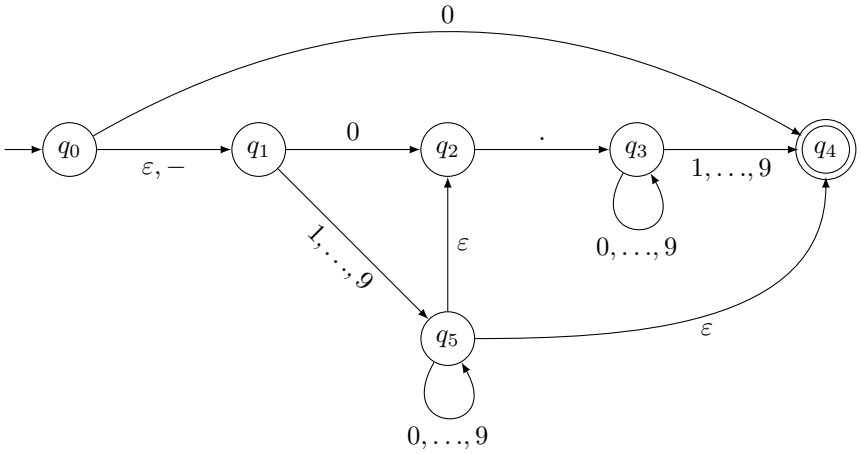
Dans certaines situations, il peut être intéressant de pouvoir passer d'un état à un autre sans lire de lettre. On utilise alors des transitions étiquetées par le mot vide, appelées ε-transitions.

Exemple 3.21

On considère $\Sigma = \{0, 1, \ldots, 9, -, .\}$, et on pose L le langage des nombres réels x bien formés avec les conditions :

- un signe $-$ peut éventuellement être présent devant le nombre ;
- si $|x| < 1$, alors le nombre $|x|$ commence par un 0, suivi par un point (on utilise la représentation anglophone des nombres), et une suite non vide de chiffres sauf éventuellement si $x = 0$;
- si $|x| > 1$, alors $|x|$ commence par un chiffre différent de 0, suivi par une éventuelle suite de chiffre, et une éventuelle partie décimale ;
- la partie décimale ne termine pas par un 0.

Une représentation possible est la suivante :



Définition 3.22

On définit un **automate fini non déterministe avec ε-transitions** (ou transitions **spontanées** ou transitions **instantanées**), noté ε-AFND, de la même manière qu'un AFND, à l'exception de la fonction de transition $\Delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$: une transition peut être étiquetée par ε.

Définition 3.23

Soit $A = (Q, \Sigma, \Delta, I, F)$ un ε-AFND et $q \in Q$. On définit l'**ε-clôture** de q, notée $\mathcal{E}(q)$, comme l'ensemble des états accessibles depuis q par le mot vide. C'est le plus petit ensemble vérifiant :

- $q \in \mathcal{E}(q)$;
- si $p \in \mathcal{E}(q)$ et $r \in \Delta(p, \varepsilon)$, alors $r \in \mathcal{E}(q)$.

Exemple 3.24

Sur l'exemple précédent, on a :

- $\mathcal{E}(q_0) = \{q_0, q_1\}$;
- $\mathcal{E}(q_1) = \{q_1\}$;
- $\mathcal{E}(q_5) = \{q_5, q_2, q_4\}$.

Définition 3.25

Soit un ε-AFND $A = (Q, \Sigma, \Delta, I, F)$. La fonction de transition Δ^* étendue aux mots est la fonction $\Delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$ définie récursivement par

- pour tout $q \in Q$, $\Delta^*(q, \varepsilon) = \mathcal{E}(q)$;
- pour tous $q \in Q$, $u \in \Sigma^*$ et $a \in \Sigma$, $\Delta^*(q, ua) = \mathcal{E}(\Delta(\Delta^*(q, u), a))$.

On définit de manière similaire aux automates non déterministes la notion de langage reconnu.

Exercice 8

Décrire des ε-AFND qui reconnaîssent les langages suivants. On ne représentera qu'un seul état initial et qu'un seul état final.

1. Le langage des mots sur $\{a, b\}$ formé d'un mot de taille multiple de 3, suivi de plusieurs répétitions de ba (potentiellement aucune).
2. L'ensemble des mots commençant par des a, suivi par des b, suivi par des c. Le nombre de a, b ou c peut être nul.

3.3.4 Équivalence entre les modèles d'automates

Proposition 3.26

Pour tout automate non déterministe, il existe un automate déterministe reconnaissant le même langage.

Preuve

Remarque 3.27

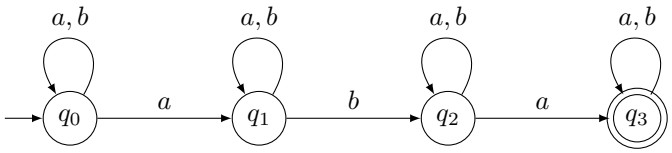
Pour déterminiser un automate, on peut procéder comme suit :

- on écrit un tableau dont les colonnes sont indexées par les lettres et la première ligne est indexée par l'état initial de l'automate des parties de la preuve précédente ;
- on indique dans cette première ligne, à quel(s) état(s) mène chaque lettre puis on ajoute des lignes correspondant aux ensembles d'états atteints qui n'ont pas encore été considérés ;
- on recommence jusqu'à ce que le procédé s'arrête.

On n'obtient ainsi que les états accessibles (mais pas forcément co-accessibles) de l'automate des parties.

Exemple 3.28

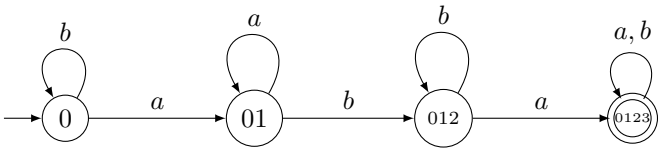
L'automate suivant reconnait les mots dont aba est un sous-mot,



le tableau est

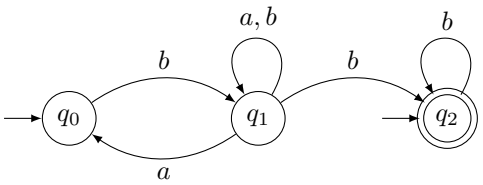
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$

Donc l'automate des parties est le suivant (en notant $i_1 i_2 ... i_k$ l'état $\{q_{i_1}, q_{i_2}, ..., q_{i_k}\}$) :



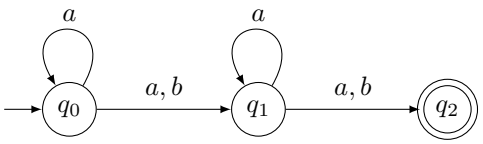
Exercice 9

Déterminiser l'automate suivant en construisant l'automate des parties :



Exercice 10

Déterminiser l'automate suivant en construisant l'automate des parties :



Définition 3.29

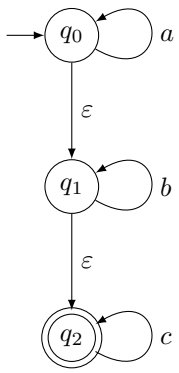
Soit $A = (Q, \Sigma, \Delta, I, F)$ un ε-AFND. On définit les AFND suivants :

- la clôture **avant** (*forward*) de A, $A_F = (Q, \Sigma, \Delta', I', F)$, avec $I' = \mathcal{E}(I)$, et pour $(q, a) \in Q \times \Sigma$, $\Delta'(q, a) = \mathcal{E}(\Delta(q, a))$;
- la clôture **arrière** (*backward*) de A, $A_B = (Q, \Sigma, \Delta'', I, F'')$ avec $F'' = \{q \in Q \mid \mathcal{E}(q) \cap F \neq \emptyset\}$, et pour $(q, a) \in Q \times \Sigma$, $\Delta''(q, a) = \Delta(\mathcal{E}(q), a)$.

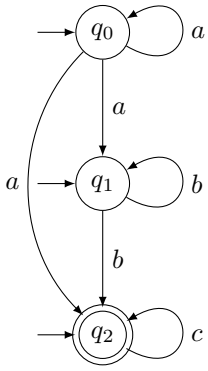
Autrement dit, dans la clôture avant, on applique des ε-transitions après chaque transition normale, et dans la clôture arrière, on applique des ε-transitions avant chaque transition normale.

Exemple 3.30

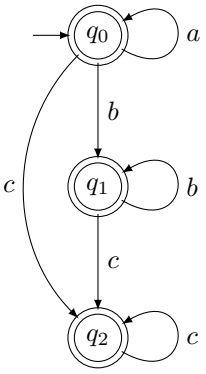
Ci-après, un ε-AFND et ses clôtures avant et arrière.



ε-AFND A



clôture avant A_F



clôture arrière A_B

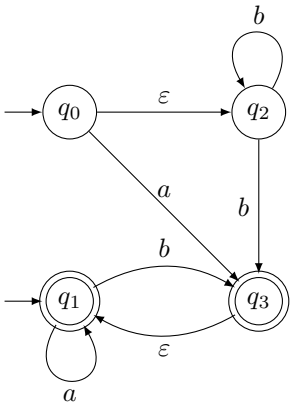
Proposition 3.31

Pour tout ε-AFND, les clôtures avant et arrière reconnaissent le même langage.

Preuve

Exercice 11

Déterminer les clôtures avant et arrière de l'automate suivant.



L'automate des parties obtenu à partir d'un automate non déterministe à n états compte 2^n états : il est donc de taille exponentielle. La proposition suivante montre que dans certains cas, on ne peut significativement faire mieux.

Proposition 3.32

Pour $n \in \mathbb{N}^*$, il existe un langage L reconnu par un AFND à $n+1$ états et non reconnu par un AFD à moins de 2^n états.

Preuve

3.4 Implémentation

On peut représenter en machine un automate de différentes façons dont certaines ne sont pas sans rappeler les représentations des graphes. Généralement, l'ensemble des états Q est assimilé à $[\![0, n-1]\!]$ où $n = |Q|$.

3.4.1 Automates non déterministes

Pour le cas le plus général des automates non déterministes, on peut représenter, pour un AFND $A = (Q, \Sigma, \Delta, I, F)$:

- l'ensemble des états initiaux (ou finaux) :
  - soit par un tableau de booléens : un tableau T de taille $n = |Q|$ tel que $T[q]$ vaut `true` si et seulement si $q \in I$ (ou $q \in F$) ;
  - soit par une liste chaînée des états initiaux (ou finaux).

  La première représentation permet un test d'appartenance en temps constant, tandis que la seconde facilite le parcours de tous les états concernés ;
- la fonction de transition Δ :
  - par un tableau de listes d'adjacence, de type `('a * int) list array` par exemple en Caml, où si `delta` est un tel tableau et $q \in Q$, alors `delta.(q)` contient tous les couples (a, p) tels que $p \in \Delta(q, a)$;
  - si on assimile Σ à $[\![0, |\Sigma| - 1]\!]$, par un tableau à trois dimensions, de type `bool array array array` par exemple en Caml, où `delta.(q).(a).(p)` vaut `true` si et seulement si $p \in \Delta(q, a)$.

Exemple 3.33

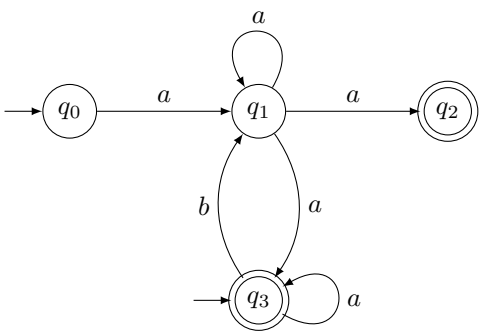
On peut utiliser le type suivant :

```
type afnd = {initiaux : int list;
             finaux : bool array;
             delta : (char * int) list array};;
```

Avec ce type, l'automate codé par :

```
let aut = {initiaux = [0; 3];
           finaux = [|false; false; true; true|];
           delta = [| [('a', 1)];
                      [('a', 1); ('a', 2); ('a', 3)];
                      [];
                      [('b', 1); ('a', 3)] |]};;
```

représente l'automate suivant :



Remarque 3.34

Attention, il n'existe pas de commande pour créer un tableau à trois dimensions en évitant la liaison de données (contrairement à un tableau à deux dimensions avec `Array.make_matrix`). La création d'un tel tableau doit se faire en utilisant des boucles.

3.4.2 Automates déterministes

Si $A = (Q, \Sigma, \delta, q_0, F)$ est un AFD, on peut utiliser les représentations précédentes pour encoder A. On peut également choisir de représenter :

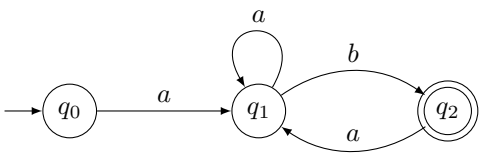
- l'état initial par l'état 0 ;
- la fonction de transition, si on assimile Σ à $[\![0, |\Sigma|-1]\!]$, par un tableau de tableaux, de type `int array array` par exemple en Caml, où si `delta` est un tel tableau, $q \in Q$ et $a \in \Sigma$, alors `delta.(q).(a)` est égal à $\delta(q, a)$ s'il est défini et -1 sinon.

Exemple 3.35

Avec le type suivant :

```
type afd = {finaux : bool array; delta : int array array};;
```

l'automate représenté par :



admet pour représentation en Caml :

```
let aut = {finaux = [|false; false; true|];
           delta = [| [|1; -1|]; [|1; 2|]; [|1; -1|] |]};;
```

Exercice 12

On considère des automates déterministes représentés par le type de l'exemple précédent. On représente un mot comme une chaîne de caractères de type 'string', et on suppose disposer d'une fonction `num (a: char) : int` qui associe, à un caractère a, un entier entre 0 et $|\Sigma| - 1$ (on peut par exemple écrire `let num a = Char.code a - Char.code 'a'`).

1. Écrire une fonction `delta_etoile (aut: afd) (q: int) (u: string) : int` qui calcule $\delta^*(q, u)$. La fonction renverra -1 s'il y a un blocage dans le calcul.
2. En déduire une fonction `reconnu (aut: afd) (u: string) : bool` qui détermine si un mot est reconnu par un automate.

4 Langages rationnels

4.1 Définition inductive

Définition 4.1

Soit Σ un alphabet. On appelle **ensemble des langages rationnels sur** Σ, noté $\mathrm{Rat}(\Sigma)$, le sous-ensemble de $\mathcal{P}(\Sigma)$ défini par induction par :

- $\emptyset \in \mathrm{Rat}(\Sigma)$;
- $\forall a \in \Sigma$, $\{a\} \in \mathrm{Rat}(\Sigma)$;
- Si $L, L' \in \mathrm{Rat}(\Sigma)$, alors LL' (concaténation), $L \cup L'$ (union) et L^* (étoile) sont des langages de $\mathrm{Rat}(\Sigma)$.

Exemple 4.2

Les langages suivants sont rationnels :

- $L_1 = \{\varepsilon\}$, car $L_1 = \emptyset^*$;
- $L_2 = \Sigma$, car $\Sigma = \bigcup_{a \in \Sigma} \{a\}$;
- L_3 l'ensemble des mots de longueur impaire, car il s'écrit $L_2(L_2L_2)^*$;
- L_4 l'ensemble des mots commençant par un a et terminant par un b, car il s'écrit $\{a\}L_2^*\{b\}$.

Les expressions régulières permettent de simplifier la notation et la manipulation de langages rationnels.

Définition 4.3

On définit l'ensemble des **expressions régulières** sur Σ par induction par :

- $\varnothing$ est une expression régulière ;
- pour tout $a \in \Sigma$, a est une expression régulière ;
- si e et f sont des expressions régulières, alors ef, $e|f$ et $e^\star$ sont des expressions régulières.

En l'absence de parenthèses, la priorité des opérations va d'abord à l'étoile puis la concaténation et enfin l'union.

La **taille** d'une expression régulière est le nombre de symboles y apparaissant parmi $\varnothing$, $a \in \Sigma$, $|$ et *. Les parenthèses ne sont pas comptées dans la taille.

L'opérateur d'évaluation permet d'associer un langage à une expression régulière.

Définition 4.4

On définit l'opération d'interprétation par $\mathcal{L}$ qui, à une expression régulière sur Σ, associe un langage sur Σ de manière récursive avec les règles naturelles suivantes :

- $\mathcal{L}(\varnothing) = \emptyset$;
- pour tout $a \in \Sigma$, $\mathcal{L}(a) = \{a\}$;
- si e et f sont deux expressions régulières, $\mathcal{L}(ef) = \mathcal{L}(e)\mathcal{L}(f)$;
- si e et f sont deux expressions régulières, $\mathcal{L}(e|f) = \mathcal{L}(e) \cup \mathcal{L}(f)$;
- si e est une expression régulière, $\mathcal{L}(e^\star) = \mathcal{L}(e)^*$.

Remarque 4.5

Deux expressions régulières distinctes peuvent être évaluées en le même langage. On dit alors qu'elles sont équivalentes. Par exemple, $(a|b)^\star$ et $(a^\star b^\star)^\star$ sont équivalentes.

S'il n'y a pas ambiguïté, on associera une expression régulière et son interprétation.

La définition des expressions régulières et des langages rationnels amène directement à la proposition suivante.

Proposition 4.6

Un langage est rationnel si et seulement s'il est l'interprétation d'une expression régulière.

Remarque 4.7

On ajoute souvent les expressions régulières ε et Σ telles que $\mathcal{L}(\varepsilon) = \{\varepsilon\}$ et $\mathcal{L}(\Sigma) = \Sigma$ (l'ensemble des mots d'une seule lettre).

Exercice 13

Décrire en français les langages qui sont l'évaluation des expressions régulières suivantes sur $\Sigma = \{a, b\}$.

1. $(a|b)^\star ab$;
2. $(a|ba)^\star b^\star$.

Exercice 14

Déterminer des expressions régulières sur $\Sigma = \{a, b\}$ pour l'ensemble des mots :

1. commençant par bb ;
2. contenant aba comme facteur ;
3. contenant exactement 3 b ;
4. contenant un nombre impair de a ;
5. ne contenant pas bb comme facteur ;
6. ne contenant pas aba comme sous-mot ;
7. qui ont autant de a que de b et tels que pour tout préfixe u, $-1 \leqslant |u|_a - |u|_b \leqslant 1$;
8. $L = \{a^n b^m \mid (n + m) = 0 \mod 2\}$.

4.2 Implémentation

De par leur définition inductive, les expressions régulières peuvent être implémentée par une structure arborescente, c'est-à-dire un type somme récursif. Par exemple, on peut utiliser le type suivant en Caml :

```
type regex =
     | Vide
     | Lettre of char
     | Concat of regex * regex
     | Union of regex * regex
     | Etoile of regex;;
```

5 Théorème de Kleene

Le théorème de Kleene permet de relier la puissance de calcul des automates à la définition des expressions régulières.

Théorème 5.1: Kleene

Soit Σ un alphabet. Alors $\mathrm{Rec}(\Sigma) = \mathrm{Rat}(\Sigma)$.

Les parties suivantes ont pour objectif de prouver le théorème.

5.1 Langages locaux

Définition 5.2

Soit L un langage quelconque. On définit :
- $P(L)$ l'ensemble des premières lettres de L : $P(L) = \{a \in \Sigma | \exists u \in \Sigma^*, au \in L\}$;
- $S(L)$ l'ensemble des dernières lettres de L : $S(L) = \{a \in \Sigma | \exists u \in \Sigma^*, ua \in L\}$;
- $F(L)$ l'ensemble des facteurs de L de taille 2 : $F(L) = \{ab \in \Sigma^2 | \exists u, v \in \Sigma^*, uabv \in L\}$;
- $N(L)$ l'ensemble des facteurs de taille 2 n'apparaissant pas dans L : $N(L) = \Sigma^2 \setminus F(L)$.

Proposition 5.3

Soit L un langage. Alors un mot non vide de L commence par l'une de ses premières lettres, termine par l'une de ses dernières lettres et ne contient pas de facteur de taille 2 absent. Autrement dit :

$$L \setminus \{\varepsilon\} \subseteq (P(L)\Sigma^* \cap \Sigma^* S(L)) \setminus \Sigma^* N(L) \Sigma^*$$

Définition 5.4

Un langage est dit **local** si et seulement si l'inclusion précédente est une égalité.

Remarque 5.5

Même si ce n'est pas une notation standard, on notera $\mathrm{loc}(L) = (P(L)\Sigma^* \cap \Sigma^* S(L)) \setminus \Sigma^* N(L) \Sigma^*$ pour la suite.

Exercice 15

1. Montrer que sur $\Sigma = \{a, b, c\}$, $(abc)^\star$ est un langage local.
2. Montrer que sur $\Sigma = \{a, b\}$, $a^\star | (ab)^\star$ n'est pas un langage local.
3. Montrer que sur $\Sigma = \{a, b\}$, $a^\star ba$ n'est pas un langage local.

Proposition 5.6

Un langage est local si et seulement s'il existe $P \subseteq \Sigma$, $S \subseteq \Sigma$ et $N \subseteq \Sigma^2$ tels que

$$L \setminus \{\varepsilon\} = (P\Sigma^* \cap \Sigma^* S) \setminus \Sigma^* N \Sigma^*$$

Preuve

Proposition 5.7

L'intersection de deux langages locaux et l'étoile de Kleene d'un langage local sont des langages locaux.

Preuve

Exercice 16

Soit $\Sigma = \{a, b\}$ et $L_1 = ab$ et $L_2 = a^\star$.

1. Montrer que L_1 et L_2 sont des langages locaux.
2. Montrer que $L_1 L_2$ n'est pas local.
3. Montrer que $L_1 \cup L_2$ n'est pas local.

Proposition 5.8

Soient L et L' deux langages locaux sur des alphabets disjoints. Alors LL' et $L \cup L'$ sont des langages locaux.

Preuve

Définition 5.9

Une expression régulière sur Σ est dite **linéaire** si chaque lettre de Σ y apparaît au plus une fois.

Proposition 5.10

L'interprétation d'une expression régulière linéaire est un langage local.

5.2 Algorithme de Berry-Sethi et automate de Glushkov

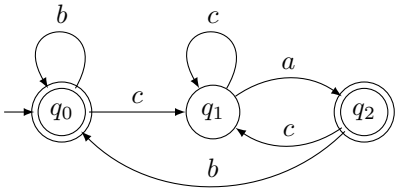
L'algorithme de Berry-Sethi a pour objectif de transformer une expression régulière quelconque en un automate non déterministe correspondant au même langage.

Définition 5.11

Un AFD $(Q, \Sigma, \delta, q_0, F)$ est dit **local** si et seulement si pour tout $a \in \Sigma$, toutes les transitions étiquetées par a arrivent dans un unique état, c'est-à-dire $\exists q \in Q, \forall p \in Q$, $\delta(p, a) = q$ ou (p, a) est un blocage.

Exemple 5.12

L'automate suivant est un automate local.



Proposition 5.13

Soit L un langage. Les propositions suivantes sont équivalentes :

1. L est un langage local.
2. L est reconnu par un automate local et standard.
3. L est reconnu par un automate local.

Preuve

Théorème 5.14

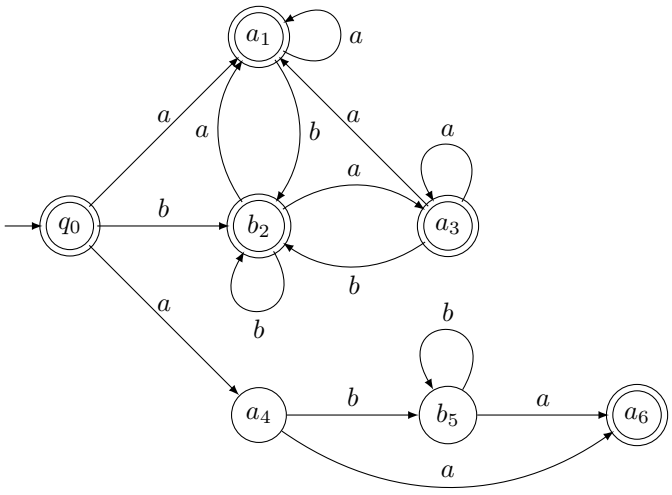
$\mathrm{Rat}(\Sigma) \subseteq \mathrm{Rec}(\Sigma)$

Preuve

Exemple 5.15

Soit $e = (a|ba^\star)^\star|ab^\star a$. On construit $f = (a_1|b_2a_3^\star)^\star|a_4b_5^\star a_6$. On a :
- $P(f) = \{a_1, b_2, a_4\}$;
- $S(f) = \{a_1, b_2, a_3, a_6\}$;
- $F(f) = \{a_1a_1, a_1b_2, b_2a_1, b_2b_2, b_2a_3, a_3b_2, a_3a_3, a_4b_5, a_4a_6, b_5b_5, b_5b_6\}$.

On en déduit un automate reconnaissant e (remarquons par ailleurs que $\mathcal{L}(e) = \{a, b\}^*$) :



Exercice 17

Construire un automate reconnaissant l'expression régulière $a(a|b)^\star b$.

Remarque 5.16

Dans l'algorithme de Berry-Sethi, on peut calculer inductivement les ensembles P, S et F de la façon suivante : en notant $V(L) = L \cap \{\varepsilon\}$, on a, si e et e' sont des expressions régulières :

- $V(\varnothing) = \emptyset$;
- $\forall a \in \Sigma$, $V(a) = \emptyset$;
- $V(e|e') = V(e) \cup V(e')$;
- $V(ee') = V(e) \cap V(e')$;
- $V(e^\star) = \{\varepsilon\}$.

- $P(\varnothing) = \emptyset$;
- $\forall a \in \Sigma$, $P(a) = \{a\}$;
- $P(e|e') = P(e) \cup P(e')$;
- $P(ee') = P(e) \cup V(e)P(e')$;
- $P(e^\star) = P(e)$.

- $S(\varnothing) = \emptyset$;
- $\forall a \in \Sigma,\ S(a) = a$;
- $S(e|e') = S(e) \cup S(e')$;
- $S(ee') = S(e') \cup V(e')S(e)$;
- $S(e^\star) = S(e)$.

- $F(\varnothing) = \emptyset$;
- $\forall a \in \Sigma,\ F(a) = \emptyset$;
- $F(e|e') = F(e) \cup F(e')$;
- $F(ee') = F(e) \cup F(e') \cup S(e)P(e')$;
- $F(e^\star) = F(e) \cup S(e)P(e)$.

5.3 Algorithme de Brzozowski et McCluskey

L'algorithme de Brzozowski et McCluskey, appelé aussi algorithme d'élimination des états, consiste à travailler sur un automate dont les transitions sont étiquetées par des expressions régulières, et à supprimer ses états jusqu'à ce qu'il n'en reste qu'un ou deux.

Définition 5.17

On appelle **automate généralisé** un automate fini (déterministe ou non) dont les transitions sont étiquetées par des expressions régulières. Sachant que ε est une expression régulière, on peut supposer que l'automate n'a qu'un seul état initial et qu'un seul état final.

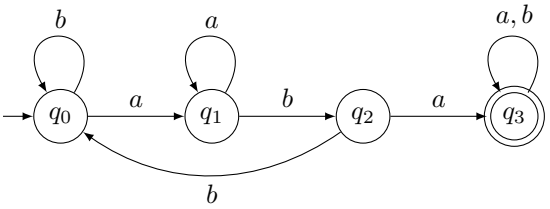
Théorème 5.18

$\mathrm{Rec}(\Sigma) \subseteq \mathrm{Rat}(\Sigma)$

Preuve

Exercice 18

En utilisant la méthode d'élimination des états, déterminer une expression régulière dont le langage est celui de l'automate suivant :



6 Propriétés des langages rationnels

Maintenant que nous connaissons l'équivalence entre automates et expressions régulières, nous pouvons établir une série de propriétés de clôture (ou de stabilité).

6.1 Propriétés de clôture

Proposition 6.1

Le complémentaire d'un langage rationnel est rationnel.

Preuve

Corolaire 6.2

L'intersection de deux langages rationnels est rationnelle.

Preuve

Exercice 19

On sait que $\mathrm{Rat}(\Sigma)$ est clos par concaténation (par définition). Soit $A = (Q, \Sigma, \delta, q_0, F)$ et $A' = (Q', \Sigma, \delta', q'_0, F')$ deux AFD. Construire un ε-AFND reconnaissant $L(A)L(A')$.

6.2 Lemme de pompage

S'il est possible de montrer la rationalité d'un langage en exhibant un automate le reconnaissant, il peut être plus ardu de montrer qu'un langage n'est pas rationnel. Le lemme de pompage (ou lemme de l'étoile) est un outil permettant de le faire. Sans être explicitement inscrit au programme, il est nécessaire de savoir l'utiliser et le redémontrer.

Théorème 6.3: Lemme de pompage

Soit $L \in \mathrm{Rat}(\Sigma)$. Alors il existe un entier $n \in \mathbb{N}^*$, appelé **longueur de pompage**, tel que pour tout mot $u \in L$ tel que $|u| \geqslant n$ admet une décomposition de la forme xyz telle que :

- $y \neq \varepsilon$;
- $|xy| \leqslant n$;
- $\forall k \in \mathbb{N}, xy^k z \in L$ (on « pompe » le facteur y).

Preuve

Exemple 6.4: Application

Le langage $L = \{a^i b^i | i \in \mathbb{N}\}$ n'est pas rationnel. En effet, si c'était le cas, posons n la constante du lemme de pompage. Posons de plus $u = a^n b^n \in L$. On peut alors écrire $u = xyz$ avec x, y et z vérifiant les conditions du lemme. Sachant que $|xy| \leqslant n$, on en déduit que $y \in a^+$. Dès lors, $|xz|_a < |xz|_b$, donc $xz \notin L$, ce qui est absurde par hypothèse.

Exercice 20

En utilisant le lemme de pompage, montrer que les langages suivants ne sont pas rationnels.

1. $L_1 = \{a^i b^j c^i | i, j \in \mathbb{N}\}$;
2. $L_2 = \{a^i b^j | i \leqslant j\}$;
3. $L_3 = \{uu | u \in \{a, b\}^*\}$.

Remarque 6.5

Attention, le lemme de pompage ne donne qu'une condition nécessaire pour être un langage rationnel, mais n'est pas une condition suffisante : il existe des langages non rationnels qui vérifie la conclusion du lemme.