Langages de Programmation, Interprétation, Compilation

Christine Paulin
Christine.Paulin@universite-paris-saclay.fr

Département Informatique, Faculté des Sciences d'Orsay, Université Paris-Saclay

LDD3 Informatique, Mathématiques - Magistère Informatique

2025–26

Analyse lexicale : expressions régulières et automates

Introduction

L'analyse lexicale consiste à découper le texte d'un programme en une séquence de mots.
Deux questions :

1. comment spécifier l'ensemble des mots utilisables dans un langage de programmation ?
2. en partant d'une telle spécification, comment réaliser un programme d'analyse efficace ?

Mots et langages formels : définitions

Un mot est une séquence de caractères, les caractères étant pris dans un ensemble (fini) A appelé alphabet.

$$m = a_1 a_2 \dots a_n$$

- L'ensemble des mots sur l'alphabet A est noté A^*.
- La longueur d'un mot est le nombre de caractères dans la séquence.
- Le mot vide est l'unique mot formé de zéro caractères, on le note ε.
- La concaténation de deux mots $m = a_1 \dots a_n$ et $m' = b_1 \dots b_k$, simplement notée mm', est le mot $a_1 \dots a_n b_1 \dots b_k$ formé en plaçant à la suite les caractères de m et ceux de m'.
- On peut itérer l'opération de concaténation un nombre fini de fois. On note m^n le mot m concaténé n fois avec lui-même :

$$m^0 = \varepsilon \qquad m^{n+1} = mm^n$$

- Dans ce chapitre, on appelle langage un ensemble de mots.
- Un langage L sur l'alphabet A est donc un sous-ensemble de A^*.

Exemples de langages

- mots clés : **if** , **fun**, **while**...
- opérateurs et autres symboles : `+`, `*`, `;`, `{`, `}`...
- des nombres : 123, 0xa9,123.45...
  Plusieurs représentations, longueur arbitraire
- des chaînes de caractères " Il␣dit␣"Salut"!n"
- ...

Opérations sur les langages

En plus des opérations ensemblistes usuelles, on introduit la notion de concaténation de deux langages :

- Si L_1 et L_2 sont 2 langages, alors leur concaténation notée $L_1 L_2$ est définie par
$$L_1 L_2 = \{m_1 m_2 \mid m_1 \in L_1, m_2 \in L_2\}$$
- On peut définir comme précédemment l'itération de cette opération sur un langage L
$$L^0 = \{\varepsilon\} \qquad L^{n+1} = LL^n$$
- L'étoile d'un langage consiste à faire l'union de touts ces itérations
$$L^* = \bigcup_{n \in \mathbb{N}} L^n$$

Langages réguliers

Ensemble de langages décrit de manière inductive par des propriétés de fermetures

- L'ensemble vide est régulier
- Le langage réduit à un mot formé d'un seul caractère $a \in A$ est régulier
- L'union de deux langages réguliers est régulier
- La concaténation de deux langages réguliers est un langage régulier
- L'étoile d'un langage régulier est un langage régulier

Expressions régulières

Une expression régulière est une manière de décrire des langages réguliers à l'aide des éléments que nous venons d'énumérer : caractères isolés, séquences, alternatives et répétitions.

e	$::=$	$\emptyset$	ensemble vide
	$\mid$	ε	mot vide
	$\mid$	a	caractère
	$\mid$	$e_1\ e_2$	séquence
	$\mid$	$e_1 \mid e_2$	alternative
	$\mid$	e^*	répétition

Langage associé à une expression régulière

À chaque expression régulière e on peut associer un langage $L(e)$, qui est l'ensemble des mots décrits par cette expression.

$$\begin{aligned} L(\emptyset) &= \emptyset \\ L(\varepsilon) &= \{\,\varepsilon\,\} \\ L(a) &= \{\,a\,\} \\ L(e_1\,e_2) &= \{\,m_1\,m_2 \mid m_1 \in L(e_1) \wedge m_2 \in L(e_2)\,\} \\ L(e_1|e_2) &= L(e_1) \cup L(e_2) \\ L(e^*) &= \bigcup_{n\in\mathbb{N}} L(e^n) \end{aligned}$$

ne pas confondre le langage vide $\emptyset$ (contenant zéro mot) et le langage $\{\,\varepsilon\,\}$ contenant le mot vide (et donc contenant un mot).

Syntaxe concrète des expressions régulières

- Nous avons décrit la syntaxe abstraite des expressions régulières (représentation arborescente)
- En pratique, on utilise une syntaxe concrète
  - utilisation de parenthèses pour préciser la structure
  - règles de priorités : la priorité la plus forte va à l'opération * puis la concaténation puis l'alternative.

$$ab^* \mid c = (a(b^*)) \mid c$$

- L'expression réguilière utilise les caractères de l'alphabet, plus des caractères spéciaux pour l'étoile, l'alternative, le parenthésage.
- Lorsque ces caractères spéciaux font partie des caractères de l'alphabet (comme pour l'analyse lexicale), on a des conventions pour les distinguer (caractère d'échappement)
- Suivant le contexte, un caractère de l'alphabet $a \in A$ peut représenter un mot $a \in A^*$ de longueur 1 ou une expression régulière qui correspond au langage formé d'un seule mot $\{a\}$

Expressions régulières étendues

- e^+ : répétition stricte (au moins une fois) $e^+ = ee^*$
- $e?$: option $e? = (e|\varepsilon)$
- $[a_1 \dots a_n]$: choix de caractères $[a_1 \dots a_n] = (a_1| \dots |a_n)$
- $[\hat{}\, a_1 \dots a_n]$: exclusion de caractères (n'importe quel caractère de l'alphabet hors ceux énumérés)

Exemples

- $(a|b)(a|b)(a|b)$: mots de 3 lettres
- $(a|b)^*a$: mots terminant par un a
- $(b|\varepsilon)(ab)^*(a|\varepsilon)$: mots alternant a et b

Exemple pour l'analyse lexicale

En Caml, les constantes flottantes décimales ont une partie entière, un partie fractionnaire et un exposant (on ne traite pas le soulignement).

- La partie entière est une suite non vide de chiffres, précédée de manière optionnelle par le signe moins.
- La partie fractionnaire débute par un point et est suivie par une suite de chiffres possiblement vide.
- La partie exposant commence par le caractère e ou E et est suivie de manière optionnelle par le signe plus ou le signe moins puis par une suite non vide de chiffres.
- La partie fractionnaire et la partie exposant sont optionnelles, mais au moins l'une des deux doit être présente.

Exemple pour l'analyse lexicale

- La partie entière correspond à l'expression régulière `-?[0-9]+`
- La partie fractionnaire correspond à l'expression régulière `[0-9]*`
- L'exposant correspond à l'expression régulière : `[eE][+-]?[0-9]+`
- Une constante va commencer par la partie entière puis va être suivie ou bien d'une partie fractionnaire suivie de manière optionnelle par la partie exposant ou bien de juste la partie exposant.
- Le résultat est :
  `-?[0-9]+((.[0-9]*([eE][+-]?[0-9]+)?)|([eE][+-]?[0-9]+))`

Représentation en caml

Type concret pour les expressions régulières

```
type regexp =
  | Vide
  | Epsilon
  | Caractere    of char
  | Sequence     of regexp * regexp
  | Alternative  of regexp * regexp
  | Repetition   of regexp
```

Tester l'appartenance : difficultés

- L'ensemble des mots associé peut être infini : on teste l'appartenance d'un mot au langage
- L'appartenance à la concaténation nécessite de découper le mot en deux : où ?
- L'appartenance à l'étoile nécessite de choisir un nombre d'itérations

Tester l'appartenance : méthode

On raisonne par récurrence sur le mot m.

- $\varepsilon \in L(e)$: quelles expressions acceptent le mot vide ?
- $am \in L(e)$:
  - est-ce que $L(e)$ contient des mots qui commencent par a ?
  - si oui peut-on caractériser l'ensemble de tels mots privés du premier a par une expression régulière ?

Tester acceptance du mot vide

```
let rec accepte_eps : regexp -> bool = function
    | Vide                  -> false
    | Epsilon               -> true
    | Caractere a           -> false
    | Sequence(e1, e2)      -> accepte_eps e1 && accepte_eps
    | Alternative(e1, e2)   -> accepte_eps e1 || accepte_eps
    | Repetition(e)         -> true
```

Calcul du résidu

$$\text{résidu}(e, a) = \{\, m \mid am \in L(e) \,\}$$

```
let rec residu (e:regexp) (a:char) : regexp =
match e with
  | Vide          -> Vide
  | Epsilon       -> Vide
  | Caractere b -> if b=a then Epsilon else Vide
  | Sequence(e1, e2) ->
     let e3 = Sequence(residu e1 a, e2) in
     if accepte_eps e1 then Alternative(e3, residu e2 a)
     else e3
  | Alternative(e1, e2) -> Alternative(residu e1 a, residu e2 a)
  | Repetition(e)         -> Sequence(residu e a, Repetition e)

let rec accepte m e = match m with
  | []     -> accepte_eps e
  | a::m' -> accepte m' (residu e a)
```

Partie 2 : Analyse lexicale et syntaxique

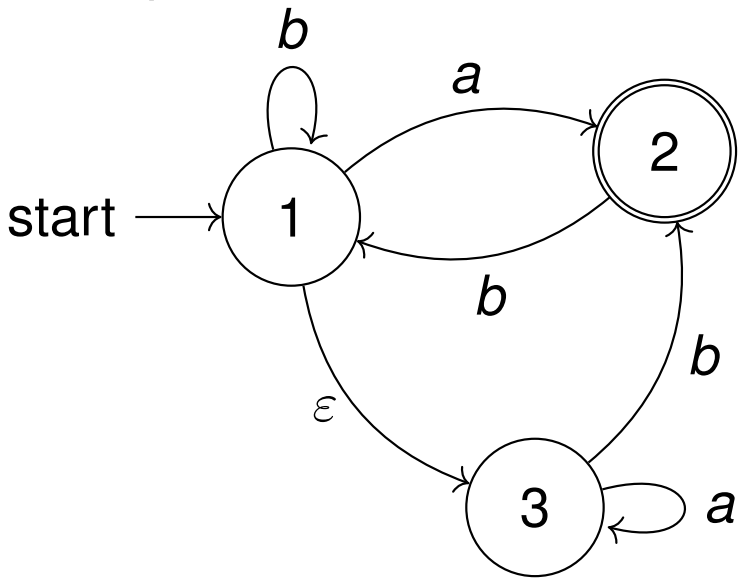
Automates

- Les automates finis sont des dispositifs algorithmiques dédiés à la manipulation de séquences de caractères, et particulièrement adaptés aux expressions régulières.
- Un automate fini sur un alphabet A est un graphe orienté avec un nombre fini de sommets, dont chaque arc est étiqueté par un caractère de A ou par le mot vide ε.
- On s'intéresse aux chemins entre des sommets de départ et des sommets d'arrivée fixés, ou plus précisément aux caractères rencontrés le long de ces chemins.

Automates : représentation graphique

On indique

- le(s) sommet(s) de départ par une flèche entrante extérieure étiquetée par start
- et les sommets d'arrivée par un cercle double.



Automates : définition

Un automate fini sur un alphabet A est décrit par un quadruplet (Q, T, I, F)

- Q est un ensemble fini, ses éléments sont appelés des états,
- T est un ensemble de triplets de $Q \times (A \cup \{\varepsilon\}) \times Q$ appelés transitions,
- I est un ensemble d'états initiaux (en général un seul), et
- F est un ensemble d'états acceptants (terminaux).

Une transition (q, a, q') sera aussi notée $q \xrightarrow{a} q'$.

Automates : langage reconnu

Un mot $m \in A^*$ est reconnu (ou accepté) par un automate (Q, T, I, F) s'il existe un chemin $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \dots q_{n-1} \xrightarrow{a_n} q_n$ avec $a_i \in A \cup \{\varepsilon\}$ qui :

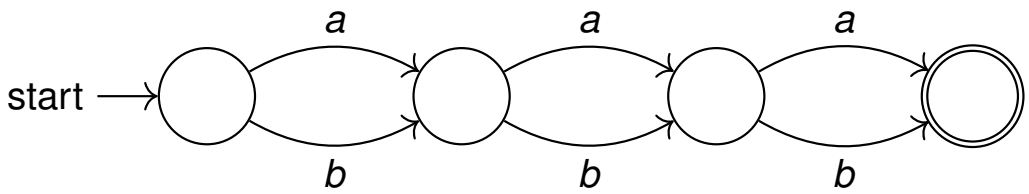
- part d'un des états initiaux : $q_0 \in I$,
- est étiqueté par les caractères de m, pris dans l'ordre : $m = a_1 a_2 \dots a_n$ et
- arrive à l'un des états acceptants : $q_n \in F$.

Le langage reconnu par un automate est l'ensemble de ses mots reconnus.

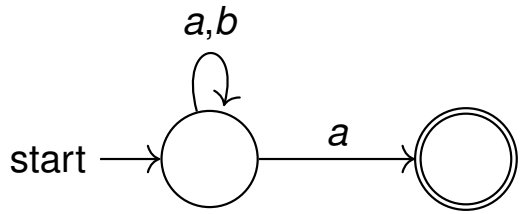
Exemples

sur l'alphabet $\{ a, b \}$

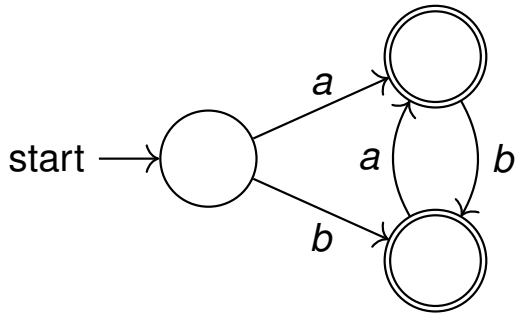
- Mots de 3 lettres :



- Mots terminant par un a.

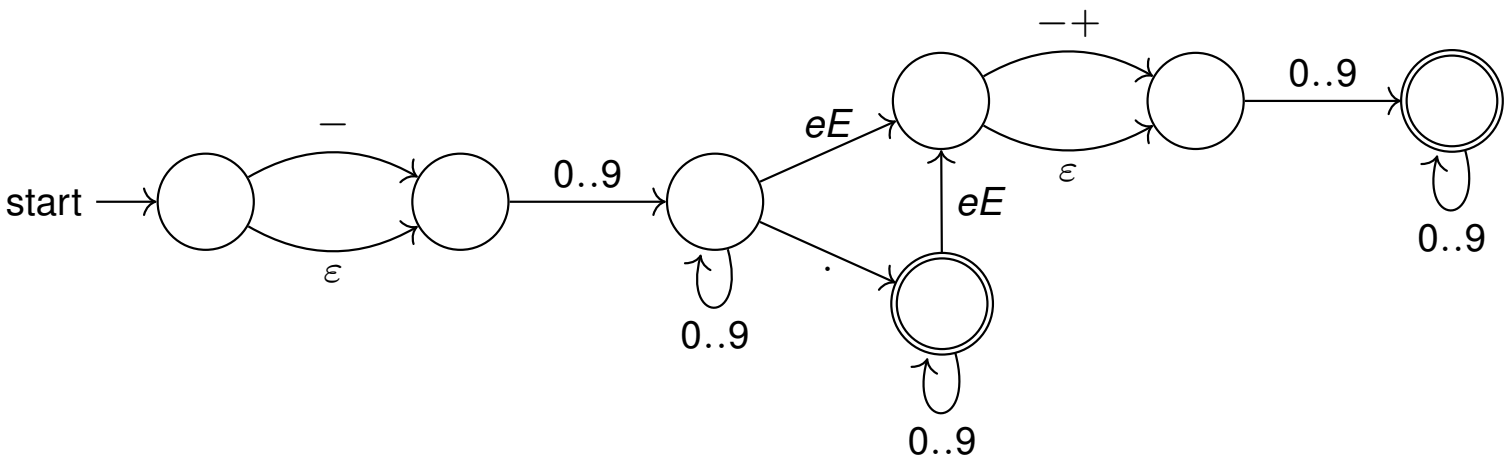


- Mots alternant a et b.



Constantes flottantes

On s'autorise à écrire plusieurs caractères sur une même transition



Automates et algorithmes

Un automate peut être vu comme un algorithme d'un type extrêmement contraint :

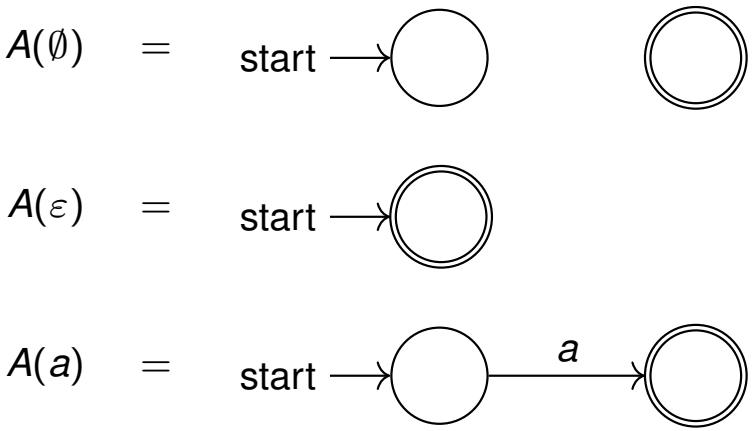
- il prend en entrée un mot,
- il renvoie comme résultat un booléen,
- il lit son entrée de gauche à droite
- les états de l'automate représentent les configurations possibles de la mémoire,
- une transition donne, en fonction d'une configuration de départ de la mémoire et d'un caractère lu dans l'entrée, une nouvelle configuration de la mémoire

Que faire si plusieurs transitions possibles, ou pas de transition ?

Construction d'un automate, première approche

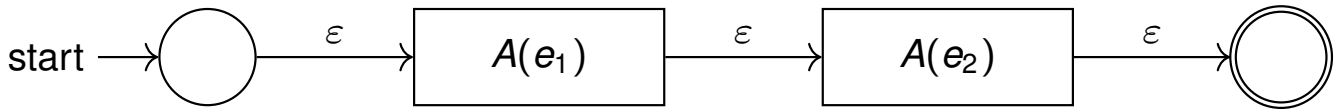
- Pour une expression régulière e, on construit un automate fini qui reconnaît le même langage
- Construction de Thompson : suit la structure récursive de l'expression régulière.
- L'automate construit $A(e)$ a un seul état initial et un seul état final.

Construction

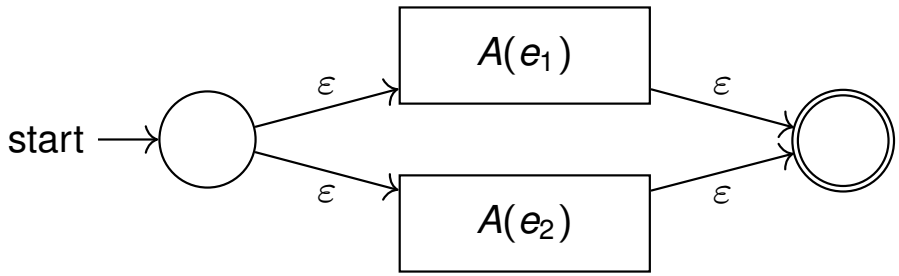


Construction (suite)

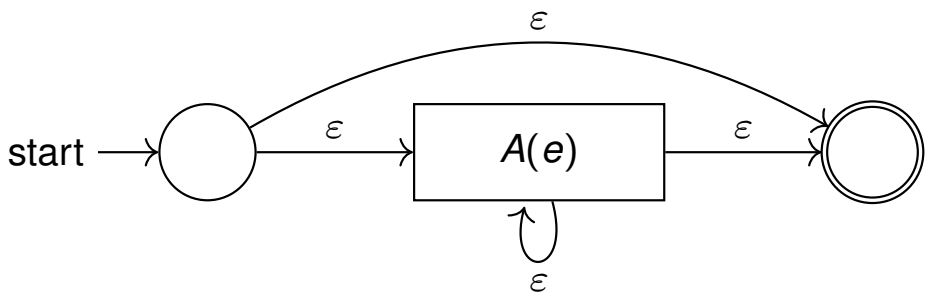
$A(e_1\,e_2)$ =



$A(e_1 | e_2)$ =



$A(e^*)$ =



Automates déterministes

Restrictions sur la forme des automates pour faciliter la reconnaissance d'un mot :

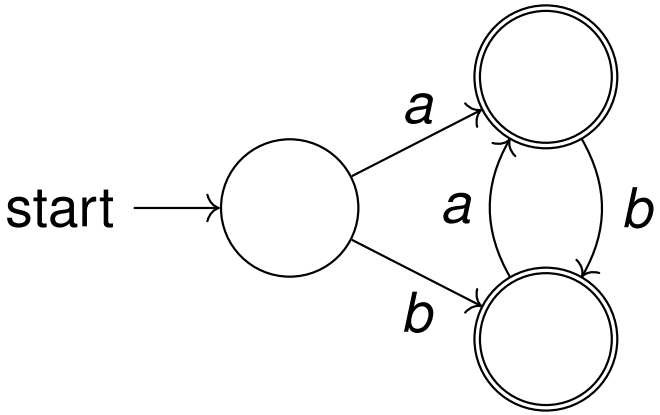
- un seul état initial,
- pas de transitions ε,
- partant de chaque état, au plus une transition par caractère.

Avec ces contraintes, pour un automate et un mot donnés, il y a au plus un chemin partant de l'état initial et correspondant à ce mot :

1. partir de l'état initial,
2. pour chaque caractère *a* de l'entrée, suivre la transition correspondante si elle existe (sinon : échec),
3. une fois la lecture du mot terminée, celui-ci est reconnu si l'état d'arrivée est acceptant.

Le mot est reconnu ssi le chemin existe et arrive sur un état terminal.

Exemple



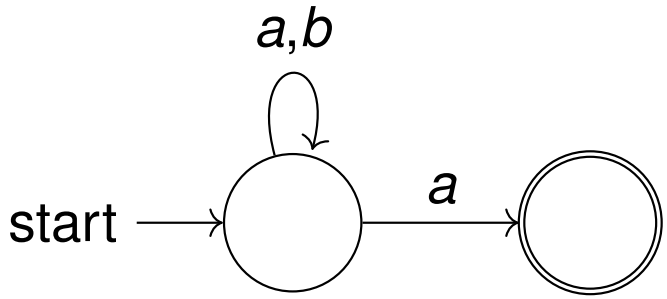
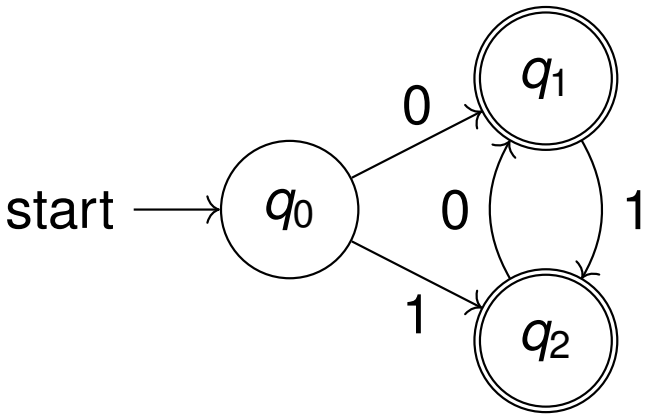


Table de transitions déterministe

- tableau à double entrée
- une ligne par état et une colonne par caractère dans l'alphabet
- s'il existe une transition $q \xrightarrow{a} q'$, on place q' dans la case sur la ligne q et la colonne a.



	0	1
q_0	q_1	q_2
q_1		q_2
q_2	q_1	

Représentation Caml

On suppose que les états et les caractères de l'alphabet sont numérotés. On utilise un tableau d'options :

```
type automate = {
  initial: int;
  transitions: (int option) array array;
  accepting: int list;
}

let a = {
  initial = 0;
  transitions = [| [| Some 1; Some 2 |];
                   [| None;   Some 2 |];
                   [| Some 1; None   |] |];
  accepting = [1; 2];
}
```

Reconnaissance

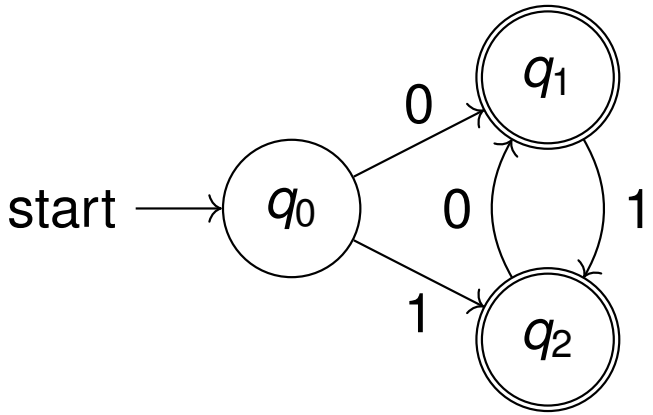
On suppose que les mots sont représentés par des listes d'entiers.

```
let reconnait (m: int list) (a: automate): bool =
  let rec loop m q = match m with
    | []    -> List.mem q a.accepting
    | b::m' -> match a.transitions.(q).(b) with
                 | None    -> false
                 | Some q' -> loop m' q'
  in
  loop m a.initial
```

Codage d'un automate par des fonctions récursives

- une fonction par état, prend en argument un mot, renvoie vrai si le mot est reconnu à partir de l'état
- si le mot est vide, renvoie vrai si l'état est acceptant et faux sinon
- sinon le mot s'écrit am, si'il y a une transition avec a vers l'état s', on appelle la fonction associée pour reconnaître m

Exemple



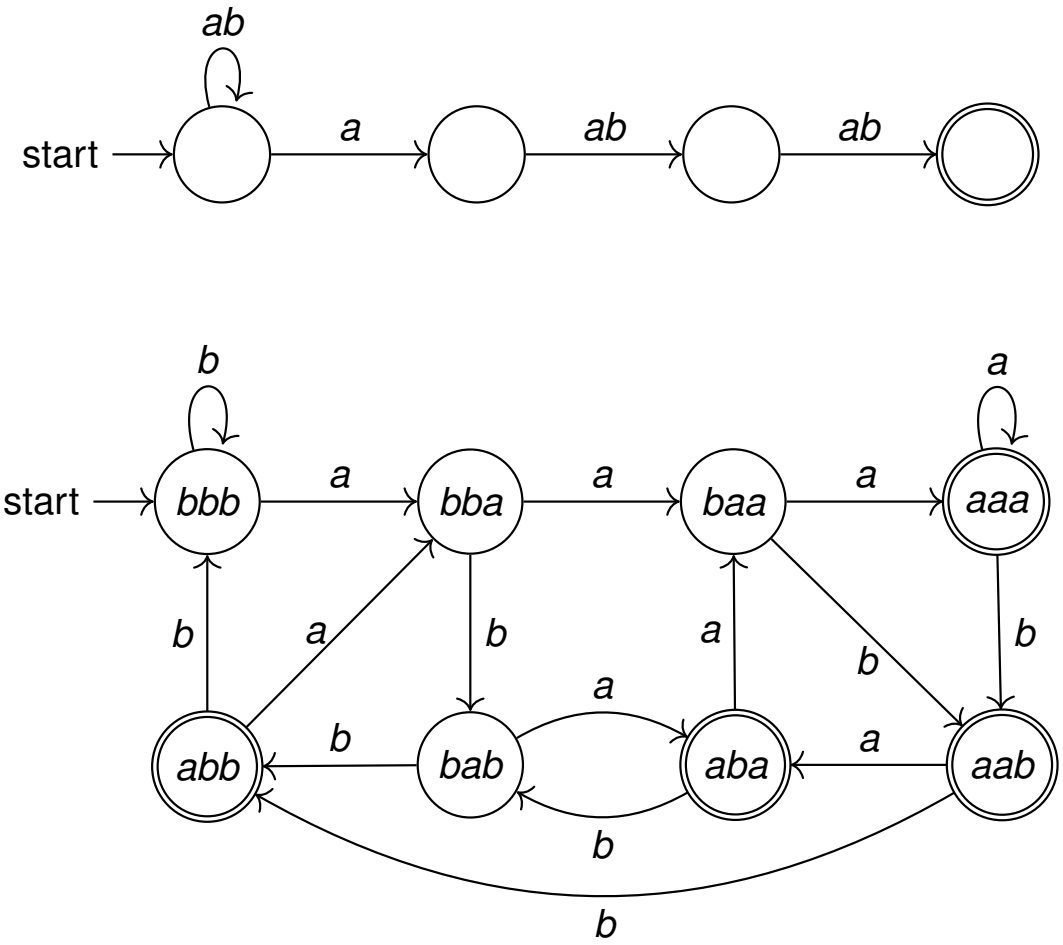

```
let rec q0 = function
  | []    -> false
  | a::m -> if a=0 then q1 m else q2 m

and q1 = function
  | []    -> true
  | a::m -> if a=1 then q2 m else false

and q2 = function
  | []    -> true
  | a::m -> if a=0 then q1 m else false
```

Déterminisation

- Tout automate peut être transformé en un automate déterministe équivalent (reconnait le même langage).
- Ce processus de transformation est appelé déterminisation.
- Les états de l'automate déterministe sont des ensembles d'états de l'automate initial (Q, T, I, F)
- On a une transition $Q \xrightarrow{a} X'$ dans l'automate déterministe, ssi $X' = \{q' \in Q \mid \exists q \in X, q \xrightarrow{\varepsilon}_* \xrightarrow{a} \xrightarrow{\varepsilon}_* q'\}$
- L'état initial est $\{j \in Q \mid \exists i \in I, i \xrightarrow{\varepsilon}_* j\}$ (I plus les états accessible avec ε)
- Les état finaux sont tous les ensembles qui contiennent $q \in F$.

Déterminisation : limite

Tout automate peut être déterminisé mais l'automate résultant peut avoir beaucoup d'états 2^n si n est le nombre d'états de l'automate initial.



Partie 2 : Analyse lexicale et syntaxique

Théorème de Kleene

Les langages qui peuvent être décrits par une expression régulière sont exactement les langages qui peuvent être reconnus par un automate fini.

Automate généralisé

- On étiquette les transitions par des expressions régulières
- Un mot m sur un alphabet A est reconnu par un tel automate s'il existe un chemin $q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} q_2 \dots q_{k-1} \xrightarrow{e_k} q_k$ qui :
  - part d'un des états initiaux $q_0 \in I$ et arrive à l'un des états acceptants $q_k \in F$, et
  - est tel que m puisse être découpé en k morceaux $m_1 m_2 \dots m_k$ de telle sorte que pour chaque i, le mot m_i soit dans le langage de l'expression régulière e_i.
- Toute automate classique, est naturellement un automate généralisé : les étiquettes $a \in A$ et ε sont des expressions régulières.

Algorithme d'élimination des états

élimination des transitions : tant qu'il existe dans T deux transitions distinctes $q \xrightarrow{e_1} q'$ et $q \xrightarrow{e_2} q'$ entre les deux mêmes paires d'état, les supprimer et les remplacer par une unique transition $q \xrightarrow{e_1 \mid e_2} q'$.

élimination d'un état q : pour tous $p \xrightarrow{e_1} q$ et $q \xrightarrow{e_2} s$ dans T,

- ajouter $p \xrightarrow{e_1 e^* e_2} s$ à T si $q \xrightarrow{e} q$ et y ajouter $p \xrightarrow{e_1 e_2} s$ sinon ;
- supprimer $p \xrightarrow{e_1} q$ et $q \xrightarrow{e_2} s$ de T.

Supprimer q de Q.

Algorithme principal

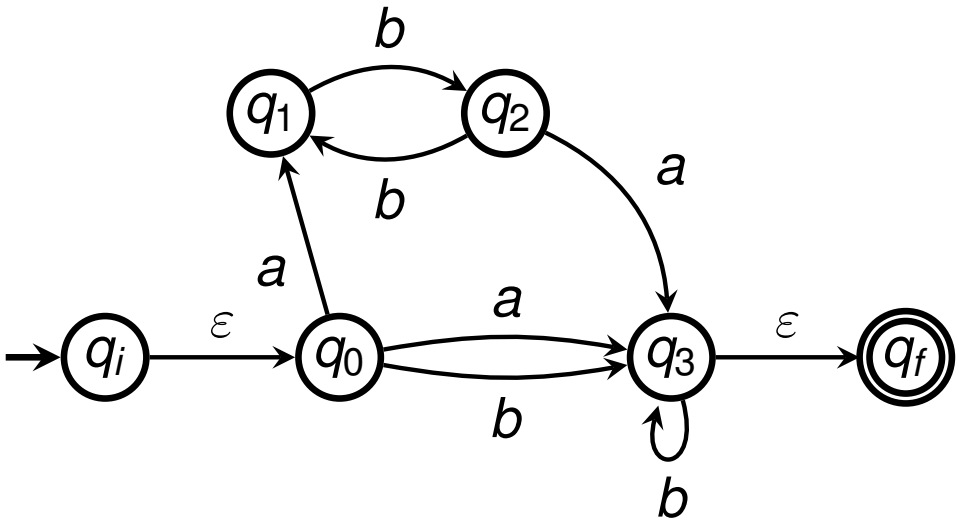
Pour chaque état $q \in Q$,

- éliminer toutes les transitions possibles ;
- éliminer l'état q.

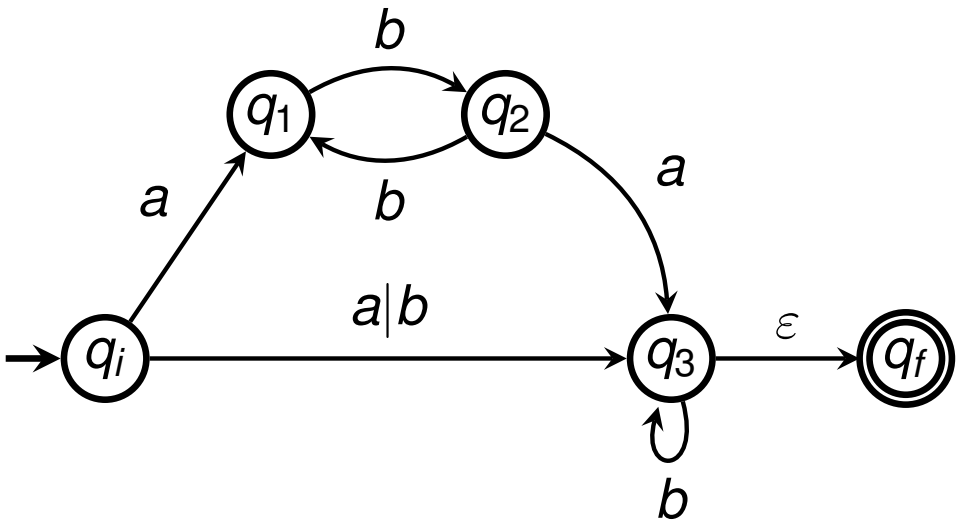
Une fois l'automate réduit à $q_i \xrightarrow{e} q_f$, renvoyer e.

Exemple

Soit l'automate sur l'alphabet $\{a, b\}$:

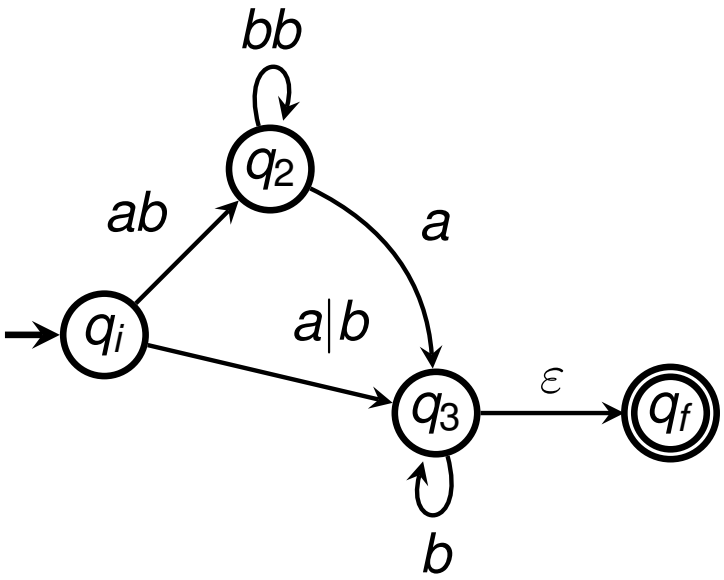


Elimination de q_0



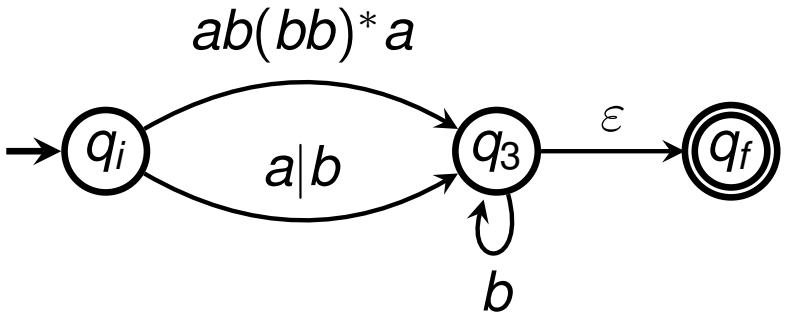
Elimination de q_1

Pas de transitions à éliminer.



Elimination de q_2 et q_3

- Elimination de q_2 :



-
- Élimination de q_3.
  - On remplace $q_i \xrightarrow{ab(bb)^*a} q_3$ et $q_i \xrightarrow{a|b} q_3$ par $q_i \xrightarrow{ab(bb)^*a|a|b} q_3$.
  - On considère cette transition et $q_3 \xrightarrow{\varepsilon} q_f$ pour obtenir $q_i \xrightarrow{(ab(bb)^*a|a|b)b^*\varepsilon} q_f$.
- L'automate est réduit aux états q_i et q_f, l'expression recherchée est $(ab(bb)^*a|a|b)b^*$.

A propos de l'algorithme

- La taille des expressions régulières renvoyées par cet algorithme peut, dans le pire cas, être exponentielle en le nombre d'états de l'automate de départ
- Le choix de l'ordre des sommets influence la taille finale de l'expression.
- On justifie la correction de la construction en montrant que l'élimination d'une transition et d'un état préserve le langage.

Propriétés des langages reconnaissables

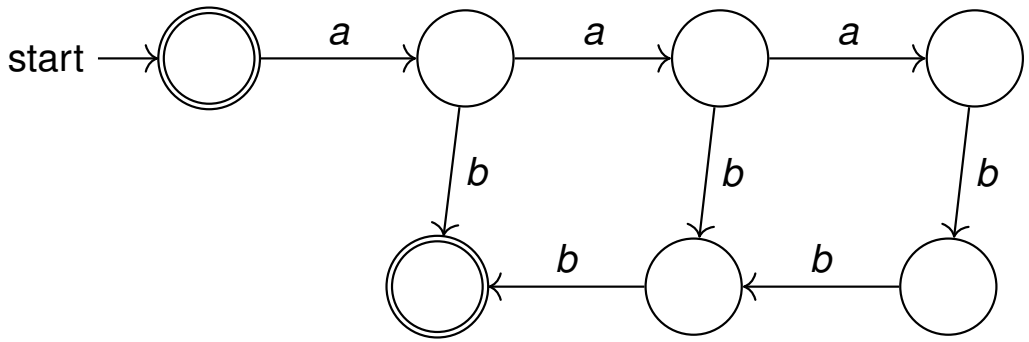
Les langages reconnaissables ont de bonnes propriétés.

- L'union de langages reconnaissables L_1 et L_2 est un langage reconnaissable (par définition).
- L'intersection de langages reconnaissables L_1 et L_2 est un langage reconnaissable
  - construction de l'automate produit qui fait une reconnaissance "en parallèle" dans les deux automates.
- Le complémentaire d'un langage reconnaissable L est reconnaissable.
  - On complète l'automate avec un état "puits" où aboutissent les mots non reconnus et des transitions pour chaque caractère.
  - On inverse état acceptant et état non-acceptant
- Il est possible de décider si un langage reconnaissable est vide, il suffit de démontrer qu'il n'existe pas de chemin d'un état initial à un état final. (recherche de chemin dans un graphe)

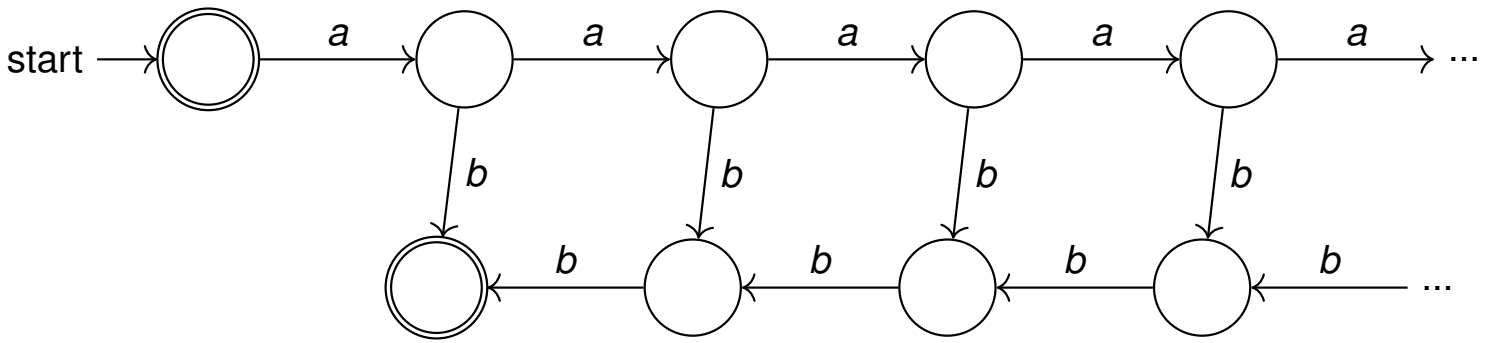
Partie 2 : Analyse lexicale et syntaxique

Langages non reconnaissables

- Les automates modélisent des algorithmes avec une mémoire bornée
- Certains langages ne seront pas reconnaissables par ces méthodes
- $L_1 = \{ a^n b^m \mid n \in \mathbb{N} \wedge m \in \mathbb{N} \}$
  Expression régulière $a^* b^*$
- $L_2 = \{ a^n b^n \mid n \leqslant 3 \}$
  Expression régulière $\varepsilon|ab|aabb|aaabbb$reconnu par l'automate

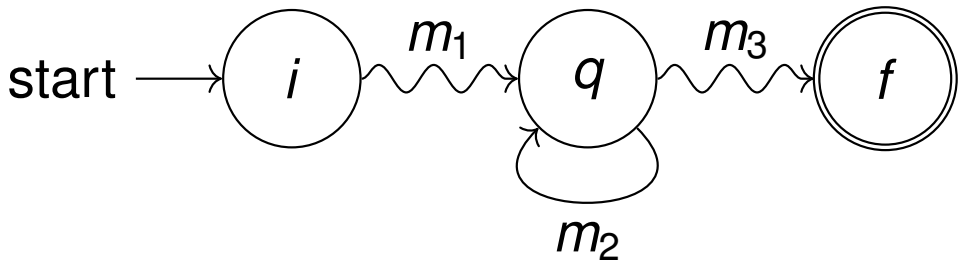


- $L_3 = \{ a^n b^n \mid n \in \mathbb{N} \}$



Conséquences de la finitude d'un automate

- automate (Q, T, I, F) à N états
- mot m de longueur $l \geqslant N$ reconnu.
- Un chemin étiqueté par ce mot passe par $l + 1$ états. Comme $l + 1 > N$, par le lemme des tiroirs il existe un état qui est vu au moins deux fois dans ce chemin.
- on a $m = m_1 m_2 m_3$.



- L'automate reconnaît également
  - $m_1 m_3$
  - $m_1 m_2 m_3$
  - $m_1 m_2 m_2 m_3$
  - $m_1 m_2 m_2 m_2 m_3$
  - ...

Lemmes de l'étoile

Version faible. *Soit L un langage reconnaissable. Il existe un entier N tel que tout mot $m \in L$ de longueur $l \geqslant N$ puisse être décomposé en $m_1 m_2 m_3$ avec $m_2 \neq \varepsilon$ et $L(m_1 m_2^* m_3) \subseteq L$.*

Version forte. *Soit L un langage reconnaissable. Il existe un entier N tel que pour tout mot $m_0 m m_4 \in L$ avec m de longueur $l \geqslant N$, le mot m peut être décomposé en $m_1 m_2 m_3$ avec $m_2 \neq \varepsilon$ et $L(m_0 m_1 m_2^* m_3 m_4) \subseteq L$.*

- les lemmes de l'étoile indiquent que dans un langage reconnaissable, tout mot long contient une partie répétable à l'infini.
- la version forte permet de maîtriser approximativement l'endroit où placer cette boucle.
- lemmes utilisés pour montrer que certains langages (comme $\{ a^n b^n \mid n \in \mathbb{N} \}$) ne sont pas reconnaissables.

Partie 2 : Analyse lexicale et syntaxique

Expressions régulières et automates

- Une expression régulière permet de décrire de manière précise un langage spécifique (possiblement infini)
- A chaque expression régulière, on peut associer un automate fini (déterministe, minimal) qui permet de décider efficacement si le mot appartient au langage

Analyseur lexical

- Décomposer une suite de caractères en une suite de lexèmes
- Plusieurs catégories : mots clés, identifiants, constantes...
- Chaque catégorie peut être décrite par une expression régulière e_i
- On reconnaît essentiellement que l'entrée est dans le langage $(e_1 \mid \ldots \mid e_n)^*$
- Spécificités de l'analyse lexicale
  - on ne veut pas juste l'information ok ou non, mais la suite des lexèmes (catégorie et possiblement valeur)
  - il peut y avoir plusieurs découpage possibles, lequel choisir ?
    - Exemple ifthen34

  Stratégie raisonnable (gloutonne) : choisir le plus long mot possible

Exemple

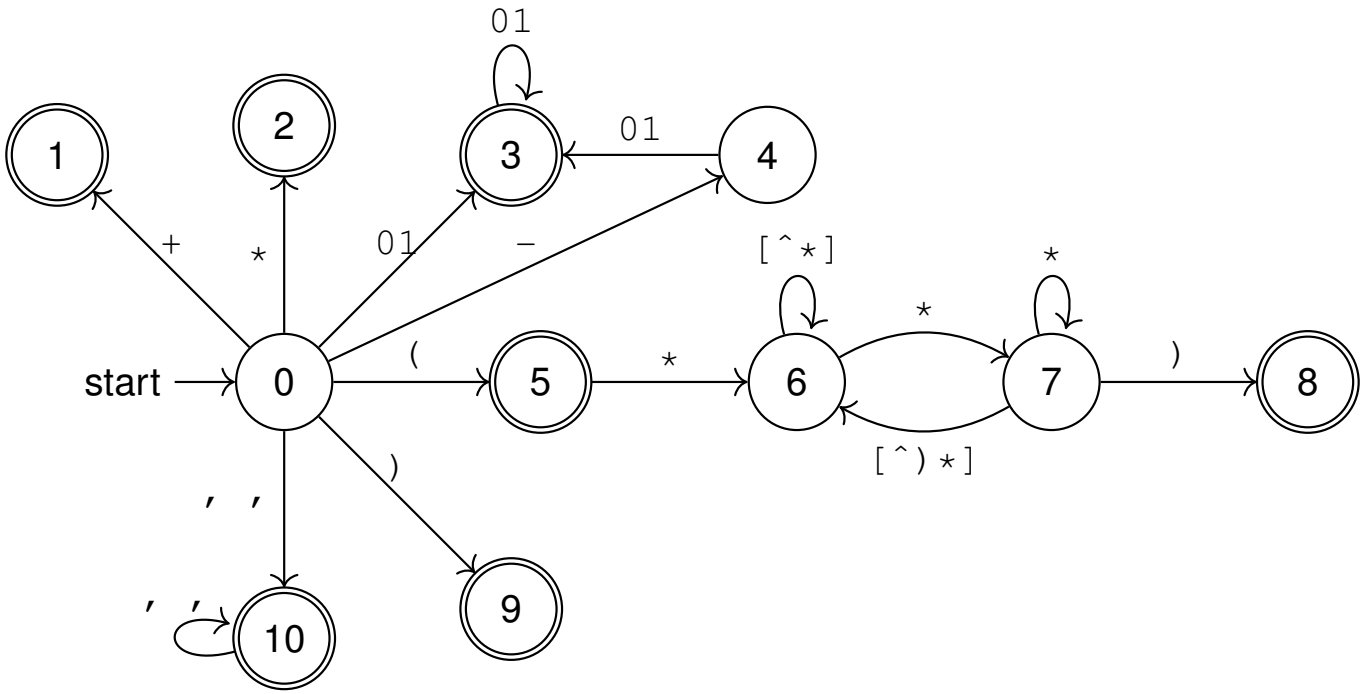
Automate reconnaissant :

- les symboles arithmétiques + et $*$ et les parenthèses,
- les nombres binaires positifs ou négatifs,
- les séquences d'espaces,
- les commentaires délimités par $(*$ et $*)$, sans imbrication.

Voir fichier lexer.ml

Exemple

Automate qui reconnaît un léxème



Synthèse

A retenir

- La syntaxe et le sens des expressions régulières étendues
- La définition d'un automate fini, d'un automate automate fini déterministe
- L'équivalence entre langage régulier et langage reconnaissable
- Exemples de langages non reconnaissables
- Le rôle des automates dans l'analyse lexicale

Savoir faire

- Savoir justifier qu'un mot appartient ou non à un langage régulier
- Décrire par des expressions régulières les principales classes d'unités lexicales à partir d'une description en langue naturelle