Compilation

(Version provisoire)

Hicham Bensaid
bensaid@inpt.ac.ma

EMI - INPT

المدرسة المحمدية للمهندسين
Ecole Mohammadia d'Ingénieurs

Première partie I

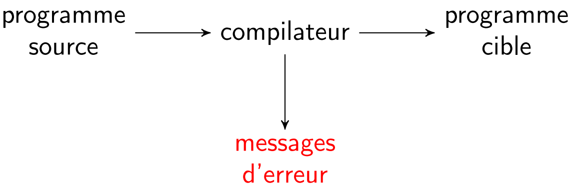
Introduction à la compilation et contexte

Bibliographie

- Compilers Principles, Techniques, and Tools Aho Lam Sethi Ullman Second Edition.
- Introduction to the Theory of Computation, third edition, M. Sipser.
- Théorie des langages, Pierre Berlioux, Mnacho Echenim et Michel Lévy. Année 2017-2018.

Qu'est-ce qu'un compilateur

- Définition : programme qui traduit un langage source en langage cible (souvent langage machine).
- Exemples : C $\rightarrow$ Assembleur/Machine, Java $\rightarrow$ Bytecode.
- Objectif :
  - Assurance qu'il n'y a pas d'erreurs dans le programme source
  - Fidélité sémantique (conserver le sens du programme).
  - Efficacité (code cible optimisé).



Compilateur vs Interpréteur

Compilateur : traduit avant exécution, produit un exécutable.

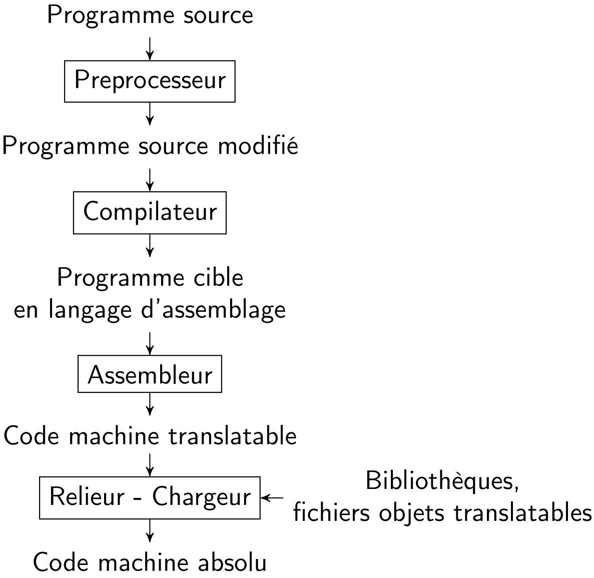
Interpréteur : lit et exécute directement le programme source.

Exemples :

- Compilateur : C $\rightarrow$ exécutable.
- Interpréteur : Python, Ruby.

Hybride : Java (compilation en bytecode + JIT).

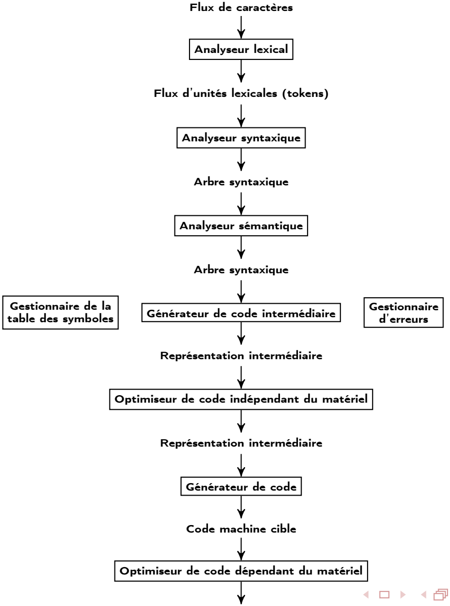
Système de traitement de langage



Les phases principales d'un compilateur

- Analyse (front-end) :
  - Analyse lexicale (scanner) $\rightarrow$ génère les *unités lexicales* (*tokens*).
  - Analyse syntaxique (parser) $\rightarrow$ construit l'*arbre syntaxique*.
  - Analyse sémantique → vérifie les types, portées, . . .
- Synthèse (back-end) :
  - Génération de code intermédiaire.
  - Optimisation.
  - Génération de code cible.

Phases d'un compilateur



Exemple de traduction

- Programme source :

```
position = initial + rate * 60
```

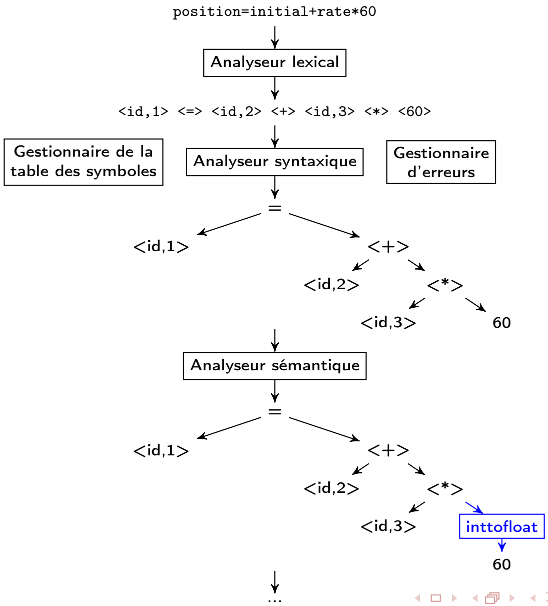
- Traduction en étapes :
  - Analyse lexicale → tokens : <id,1>, <=>, <id,2>, <+>, <id,3>, <*>, <num>
  - Analyse syntaxique → arbre binaire (+, *, =)
  - Analyse sémantique → vérification des types
  - Génération de code intermédiaire →

```
t1 = rate * 60 ;
position = initial + t1;
```

  - Code cible → instructions machines

```
LDF R2, id3
MULF R2, R2, \#60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
```

Exemple de traduction



Exemple de traduction

...

Gestionnaire de la table des symboles

Générateur de code intermédiaire

Gestionnaire d'erreurs

t1 = inttofloat(60)
t2=id3*t1
t3=id2+t2
id1=t3

Optimiseur de code

t1 = id3*inttofloat(60)
id1=id2+t1

Générateur de code

```
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
```

Analyse lexicale

- Regroupe les caractères en *unités lexicales* (*tokens*) : mots-clés, identificateurs, nombres, symboles.
- La suite de caractères qui compose un token est appelée son *lexème*.
- Utilise souvent des *expressions régulières*.
- Exemple :
  - `while (i < n)` → `tokens : while, (, id, <, id, )`
- Le token est généralement consitué d'une paire <token, val>

Analyse syntaxique

- Vérifie la *structure grammaticale* selon la *grammaire* du langage.
- Produit un *arbre syntaxique*.
- Exemple : grammaire pour une expression arithmétique :

$$\begin{array}{lllllll} E & \rightarrow & E+T & \mid & T & & \\ T & \rightarrow & T*F & \mid & F & & \\ F & \rightarrow & (E) & \mid & id & \mid & num \end{array}$$

- On verra celà en détail plus tard dans le cours

Analyse sémantique

- Vérifie la *cohérence logique* :
  - Vérification de type (int + float).
  - Déclarations/portées des variables.
  - Conformité aux règles du langage.
- Exemple : x = true + 3 $\rightarrow$ erreur sémantique.

Optimisation de code

- **But** : améliorer la qualité du code intermédiaire.
- Types :
  - **Optimisation locale** (réduction de calculs inutiles).
  - **Optimisation globale** (réutilisation de résultats, élimination de code mort).
- Exemple : $x = y * 2 * 2 \rightarrow x = y * 4$.

Génération du code cible

- Transforme le code intermédiaire en instructions machine.
- Doit tenir compte de :
  - Registres disponibles.
  - Architecture du processeur.
  - Gestion mémoire.

Applications des techniques de compilation

- Compilateurs classiques (C, Java, Fortran).
- Analyseurs de texte : regex, grep, XML parsers.
- Langages de requêtes : SQL.
- Sécurité : analyse statique.
- Machines virtuelles : JVM, CLR.

Les environnements de langages

Chaîne␣de␣compilation : éditeur → compilateur → assembleur → éditeur de liens → exécutable.

Outils␣associés :

- Débogueurs, éditeurs, gestionnaires de versions.

Importance : un compilateur s'inscrit dans un **écosystème logiciel complet**.

Tendances modernes

- JIT compilation (Java, .NET, JavaScript).
- Compilateurs pour GPU.
- Langages spécialisés (Data Science, IA).
- Optimisation pour mobiles et embarqués.

Deuxième partie II

Introduction et background mathématique

Plan

1. Relations

2. Langages
   - Chaînes et langages

Introduction

- ***Langage :*** désigne les langues naturelles
- désigne aussi les systèmes de notation : mathématiques, chimie, logique ...
- Langage = ensemble d'objets élémentaires ⇒ composition en unités ayant un *sens*.
  - composition = concaténation
- Définir un langage :
  - définir l'ensemble des objets élémentaire : le *vocabulaire*.
  - définir l'ensemble des suites d'objets élémentaires ayant un sens : la *syntaxe*.
  - définir le *sens* de ces suites : la *sémantique*.

Introduction

- ***Langage :*** désigne les langues naturelles
- désigne aussi les systèmes de notation : mathématiques, chimie, logique . . .
- Langage = ensemble d'objets élémentaires ⇒ composition en unités ayant un *sens*.
  - composition = concaténation
- Définir un langage :
  - définir l'ensemble des objets élémentaire : le *vocabulaire*.
  - définir l'ensemble des suites d'objets élémentaires ayant un sens : la *syntaxe*.
  - définir le *sens* de ces suites : la *sémantique*.

Plan

1. Relations

2. Langages
   - Chaînes et langages

Définition 1 (Relation (binaire))

Une *relation (binaire)* sur un ensemble E est un élément de $\mathscr{P}(E \times E)$.
Pour une relation binaire ρ sur E, et deux éléments $x, y \in E$, ρ est dite vraie pour le couple (x, y) ssi $(x, y) \in \rho$.
Par la suite on utilisera une notation infixée pour les relations binaires : $x\rho y$ pour $(x, y) \in \rho$.

- ρ est *réflexive* si $\forall x \in E,\ x\rho x$,
- ρ est *transitive* si $\forall x, y, z \in E,\ (x\rho y \wedge y\rho z) \Rightarrow x\rho z$,
- ρ est *symétrique* si $\forall x, y \in E,\ x\rho y \Rightarrow y\rho x$,
- ρ est *antisymétrique* si $\forall x, y \in E,\ (x\rho y \wedge y\rho x) \Rightarrow (x = y)$,

Définition 2 (Relation (binaire) - suite)

- ρ est une *relation de préordre* si elle est reflexive et transitive.
- ρ est une *relation d'équivalence* si elle est de préordre et symétrique,
- ρ est une *relation d'ordre partiel* si elle de préordre et antisymétrique,
- ρ est une *relation d'ordre total* si elle est une relation d'ordre partiel telle que $\forall x, \in E,\ x\rho y \vee y\rho x$,
- ρ est une *relation fonctionnelle* sur E (ou *fonction partielle* de E dans E) si $\forall x, y, z \in E,\ x\rho y \wedge x\rho z \Rightarrow y = z$,
- ρ est une *fonction totale* (ou *application*) de E dans E si elle est une relation fonctionnelle sur E telle que $\forall x \in E, \exists y, z \in E,\ x\rho y$.

Opérations sur les relations

Opérations ensemblistes

- $(x, y) \in \rho \cup \sigma$ ssi $x\rho y$ ou $x\sigma y$,
- $(x, y) \in \rho \cap \sigma$ ssi $x\rho y$ et $x\sigma y$,
- $x\overline{\rho}y$ ssi $(x, y) \notin \rho$, i.e. $\neg(x\rho y)$.
- Si ρ et σ sont deux relations, alors on dit que ρ *implique* σ (ou que σ *contient* ρ) ssi $\rho \subseteq \sigma$.

Propriété

ρ implique σ ssi $\overline{\rho} \cup \sigma = E \times E$.

Opérations sur les relations

Définition 3 (Composition des relations)

La *composition* des relations sur E est l'opération définie par :
$\forall \rho, \sigma \in \mathscr{P}(E \times E), \rho.\sigma = \{(x, y) \in E \times E | \exists z \in E, (z, y) \in \sigma \wedge (x, z) \in \rho\}$.

Propriété

La composition des relation est associative et admet comme élément neutre la relation identité $\imath$.

Opérations sur les relations

Définition 4 (Elévation à la puissance des relations)

Soit ρ une relation sur E et $n \in \mathbb{N}$. La relation d'*élévation à la puissance* ρ^n est définie par :

- $\rho^0 = \mathrm{i}$,
- $\rho^{n+1} = \rho.\rho^n$,
- $\rho^* = \bigcup_{n\geq 0} \rho^n$ et $\rho^+ = \bigcup_{n>0} \rho^n$.

Propriété

- $\forall n \in \mathbb{N}^*$, $x\rho^n y$ ssi il existe une suite $x_1, \ldots, x_n$ d'éléments de E tels que $x_1 = x$, $x_{n+1} = y$ et pour tout $i, 1 \leq i \leq n$, $x_i \rho x_{i+1}$,
- $x\rho^* y$ ssi $x = y$ ou $x\rho^+ y$,
- $x\rho^+ y$ ssi : $\exists n \in \mathbb{N}^*$, $x\rho^n y$.

Définition 5 (Fermeture transitive d'une relation. Fermeture transitive et réflexive d'une relation)

Soit ρ une relation sur E. La *fermeture transitive* de ρ est la plus petite relation (au sens de l'inclusion) transitive et contenant ρ.
La *fermeture transitive et réflexive* de ρ est la plus petite relation (au sens de l'inclusion) transitive et réflexive et contenant ρ.

Proposition

1. $\forall \rho$ relation sur E, ρ^+ est la fermeture transitive de ρ.
2. $\forall \rho$ relation sur E, ρ^* est la fermeture transitive et réflexive de ρ.

Définition 5 (Fermeture transitive d'une relation. Fermeture transitive et réflexive d'une relation)

Soit ρ une relation sur E. La *fermeture transitive* de ρ est la plus petite relation (au sens de l'inclusion) transitive et contenant ρ.
La *fermeture transitive et réflexive* de ρ est la plus petite relation (au sens de l'inclusion) transitive et réflexive et contenant ρ.

Proposition

1. $\forall \rho$ relation sur E, ρ^+ est la fermeture transitive de ρ.
2. $\forall \rho$ relation sur E, ρ^* est la fermeture transitive et réflexive de ρ.

Démonstration.

ρ^+ est clairement transitive. Soit σ une relation transitive contenant ρ. Si $x\rho^+y$ alors il existe $x_2,\ldots x_n$ tels que $x\rho x_2\rho\ldots x_n\rho y$. Comme $\rho\subseteq\sigma$ alors on a $x\sigma x_2\sigma\ldots x_n\sigma y$ et comme σ est transitive on $x\sigma y$. Donc $\rho^+\subseteq\sigma$. La deuxième en exercice.

□

Plan

1. Relations

2. Langages
   - Chaînes et langages

Vocabulaire - Chaîne

Définition 6 (Vocabulaire)

Un *vocabulaire* (ou *alphabet*) est un ensemble *fini* quelconque.
Les éléments d'un vocabulaire sont appelés *lettres*, *caractères* ou *symboles*.
$\#V$ désigne le cardinal d'un vocabulaire V.

Définition 7 (Chaîne)

Une *chaîne* (ou *mot*, ou *phrase*) sur un vocabulaire V est une suite finie $a_1, a_2, \ldots, a_n$ d'éléments de V.
Une chaîne est notée en séquence sans séparateur, les lettres la composant.

Exemple 8

abc est une chaîne sur le vocabulaire a, b, c.

Chaîne

Définition 9 (Longueur d'une chaîne)

La *longueur d'une chaîne* est le nombre d'éléments de la suite qui la définit. Pour une chaîne w, sa longueur est notée : $|w|$.
Il existe (par convention) un mot de longueur 0, appelé *chaîne vide*, et noté ϵ.

Exemple 10

$|abc| = 3$, $|a| = 1$, $|aaa| = 3$.

Définition 11 (Mots de longueur n)

Soit $n \in \mathbb{N}$ et V un vocabulaire. V^n est l'*ensemble des chaînes sur V de longueur n*.
En particulier :

- $V^0 = \{\epsilon\}$: la chaîne vide est une chaîne sur tout vocabulaire,
- $V^1 = V$: les chaînes de longueur 1 sont identifiées aux lettres.

Chaîne

Exercice

Soit $n \in \mathbb{N}$ et V un vocabulaire. Montrer que si $V \neq \emptyset$ alors $\#(V^n) = (\#V)^n$.

Chaîne

Exercice

Soit $n \in \mathbb{N}$ et V un vocabulaire. Montrer que si $V \neq \emptyset$ alors $\#(V^n) = (\#V)^n$.

Démonstration.

Par récurrence sur n. $\square$

Ensemble des chaînes, Ensemble des chaînes non vides

Définition 12

Soit V un vocabulaire. V^* désigne l'*ensemble des chaînes* sur V, et V^+ désigne l'*ensemble des chaînes non vides* sur V.
$V^* = \bigcup_{i \geq 0} V^i$ et $V^+ = \bigcup_{i > 0} V^i$.

Propriété

- Si $V \neq \emptyset$ alors V^* est infini.
- Si $V = \emptyset$ alors $V^* = \{\epsilon\}$.

Sous-chaîne

Définition 13

Soit V un vocabulaire et w une chaîne sur V. Une chaîne u est une *sous-chaîne* de w s'il existe $x, y \in V^*$ tels que $w = xuy$.
u est préfixe de w si $x = \epsilon$ et u suffixe de w si $y = \epsilon$.

Exemple 14

- Les sous-chaînes du mot $abba$ sont : ϵ, a, b, ab, bb, ba, abb, bba, $abba$.
- Les préfixes du mot $abba$ sont : ϵ, a, ab, abb, $abba$.
- Les suffixes du mot $abba$ sont : ϵ, a, ba, bba, $abba$.

Sous-chaîne

Exercice

- Soit w une chaîne de longueur n sur un vocabulaire V. Donner le nombre maximal et minimal de sous chaînes, de préfixes et de suffixes de w.
- Montrer que toute chaîne w sur $\{a, b\}$, telle que $|w| \geq 4$, admet deux occurrences consécutives d'une même sous-chaîne non vide.

Exercice

- Soit w une chaîne de longueur n sur un vocabulaire V. Donner le nombre maximal et minimal de sous chaînes, de préfixes et de suffixes de w.
- Montrer que toute chaîne w sur $\{a, b\}$, telle que $|w| \geq 4$, admet deux occurrences consécutives d'une même sous-chaîne non vide.

Démonstration.

- 0 si la chaîne et vide, $2 + \sum_{k=1}^{k=n-1}(n-k+1)$ (si tous les symboles de w sont deux à deux différents), les préfixes et les suffixes : $n+1$
- Par récurrence sur n.

$\square$

Concaténation

Définition 15 (Concaténation)

La *concaténation de deux chaînes* est une opération qui associe à deux chaînes x, y, la chaîne notée $x.y$ (ou xy) définie par :

- si $x = a_1, a_2, \ldots, a_n$ et $y = b_1, b_2, \ldots, b_n$

Il existe (par convention) un mot de longueur 0, appelé *chaîne vide*, et noté ϵ.

Elévation à la puissance

Définition 16 (Elévation à la puissance)

$\forall w \in V^*$.

- $w^0 = \epsilon$.
- $\forall n \in \mathbb{N}^*, \; w^n = w.w^{n-1} = \underbrace{w.w.\ldots.w}_{n \text{ fois}}$.

Propriété

- $\forall n, m, \; x^{n+m} = x^n.x^m$
- $\forall n, m, \; x^{nm} = (x^n)^m$

Définition 17

[Langage] Un *langage* sur un vocabulaire V est tout sous-ensemble de V^*.
Un langage L est fini si $\#L$ est fini, infini sinon.

Exemple 18

$V = \{a, b\}$ et $L = \{a^n b^n | n \in \mathbb{N}\}$

Propriété

Un langage L est infini ssi $\forall n \in \mathbb{N}$, il existe $w \in L$ tel que $|w| > n$.
Soit de façon équivalente : un langage L est fini ssi il existe une borne supérieure à la longueur des chaînes de L.

Opérations sur les langages

Définition 19 (Opérations ensemblistes)

Comme les langages sur V sont des parties de V^*, la *réunion* : $L_1 \cup L_2$, *l'intersection* : $L_1 \cap L_2$, la *différence* : $L_1 - L_2$ et le *complémentaire* $\overline{L} = V^* - L$, sont définis de la manière usuelle.

Définition 20 (Concaténation et Concaténation itérée)

- La *concaténation* sur les chaînes est étendue aux langages en posant pour tous langages L_1, L_2 sur V : $L_1.L_2 = \{w_1.w_2 | w_1 \in L_1, w_2 \in L_2\}$.
- L'*élévation à la puissance* d'un langage L est définie par :
- $L^0 = \{\epsilon\}$
- $L^{n+1} = L.L^n = \underbrace{L \ldots L}_{n \text{ fois}}$
- Les *concaténations itérées* * et $^+$ sont définies par : $L^* = \bigcup_{n \geq 0} L^n$ et $L^+ = \bigcup_{n > 0} L^n$

Troisième partie III

Automates finis et langages réguliers

Plan

3. Automates finis
   - Automates finis
   - Automates finis sans ϵ-transitions
   - Automates finis déterministes

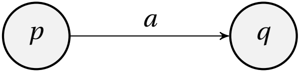
4. Expressions régulières
   - Expressions régulières
   - Equivalence des automates finis et des expressions régulières

5. Propriétés de fermeture des langages réguliers
   - Le lemme de l'étoile
   - Opérations préservant la régularité d'un langage

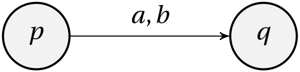
Définition 21 (Automate fini)

- Un *automate fini* est un quintuplet (Q, V, δ, I, F) où :
  - Q est un ensemble fini appelé *ensemble d'états*,
  - V est un vocabulaire,
  - $\delta \subseteq Q \times V \cup \{\epsilon\} \times Q$ appelée la *relation de transition*,
  - $I \subseteq Q$, appelé ensemble des *états initiaux*,
  - $F \subseteq Q$, appelé ensemble des *états finals*.
- Une transition (p, a, q) de δ est appelée a*-transition*, Une transition (p, ϵ, q) de δ est appelée ϵ*-transition*.
  - On peut écrire $q \in \delta(p, a)$ pour $(p, a, q) \in \delta$

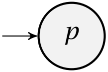
Automate fini : représentation graphique



signifie que $(p, a, q) \in \delta$



signifie que $(p, a, q) \in \delta$ et $(p, b, q) \in \delta$

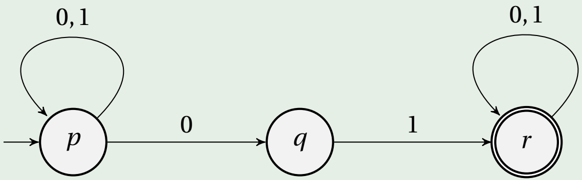


marque un état initial

marque un état final

Automate fini : représentation graphique

Exemple 22



- $Q = \{p, q, r\}$
- $V = \{0, 1\}$
- $I = \{p\}, \ F = \{r\}.$
- $\delta = \{(p,0,p),(p,1,p),(p,0,q),(q,1,r),(r,0,r),(r,1,r)\}$

Définition 23 (Chemin, trace)

Soit $A = (Q, V, \delta, I, F)$ un automate fini.
Un *chemin* de A est une suite de transitions de la forme
$(r_0, a_1, r_1)(r_1, a_2, r_2)\ldots(r_{n-1}, a_n, r_n)$. $(\forall i, 0 \leq i \leq n-1,\ (r_i, a_{i+1}, r_{i+1}) \in \delta)$.
Le *chemin* mène de l'état r_0 à l'état r_n avec la trace $a_1 \ldots a_n$. La *longeur* du chemin est n.
Par convention, il existe un chemin de longueur 0 qui mène de p à p avec la trace ϵ.

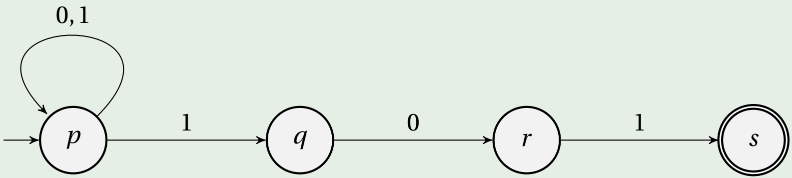
Mots, langage reconnus par un automate fini

Définition 24 (Mots, langage reconnus par un automate fini)

Le mot w est *reconnu* (ou *accepté*) par un automate A ssi A admet un chemin de trace w menant d'un état initial à un état final.
$\mathscr{L}(A)$ désigne l'ensemble des mots (sur V) reconnus par l'automate A. $\mathscr{L}(A)$ est le *langage reconnu* par A.
A, B deux automates sont *équivalents* ssi ils ont le même vocabulaire et $\mathscr{L}(A) = \mathscr{L}(B)$.
Un langage reconnu par un automate fini de vocabulaire V est appelé *langage d'états finis* sur V.

Exemple 25

L'automate $A = (Q, V, \delta, I, F)$:



- $(p,0,p)(p,1,q)(q,0,r)(r,1,s)$ est un chemin qui mène de p vers s avec la trace 0101.
- Le mot 0101 est reconnu par A : $p \in I$, $q \in F$.
- Plus généralement, cet automate reconnaît tous les mots sur le vocabulaire $\{0,1\}$ qui se terminent par le suffixe 101. (on verra plus tard pourquoi).

Propriété

Soit $A = (Q, V, \delta, I, F)$ un automate fini, $x, y \in V^*$ et $p, q \in Q$.
Un chemin mène de p à q avec la trace xy ssi il existe un état $r \in Q$ tel que un chemin mène de p à r avec la trace x et un autre chemin mène de r à q avec la trace y.

Mots, langage reconnus par un automate fini

Propriété

Soit $A = (Q, V, \delta, I, F)$ un automate fini, $x, y \in V^*$ et $p, q \in Q$.
Un chemin mène de p à q avec la trace xy ssi il existe un état $r \in Q$ tel que un chemin mène de p à r avec la trace x et un autre chemin mène de r à q avec la trace y.

Démonstration.

La partie si est immédiate.
Preuve de la partie seulement si : Soit un chemin (r_i, a_{i+1}, r_{i+1}), pour i de 0 à $n-1$, menant de l'état p à l'état q avec la trace xy. Puisque les symboles de chaque transition sont éléments de V ou sont égaux à ϵ, il existe k tel que $0 \leq k \leq n$ et $x = a_1 \ldots a_k$. Par suite, il existe un chemin conduisant de p à r_k avec la trace x et de r_k à q avec la trace y. $\square$

Mots, langage reconnus par un automate fini

Propriété

Soit $A = (Q, V, \delta, I, F)$ un automate fini, $a \in V$ et $p, q \in Q$.
Il y a un chemin de trace a entre p et q ssi il existe deux états $r, s \in Q$ tels que :

- un chemin de trace ϵ mène de p à r,
- $(r, a, s) \in \delta$,
- un chemin de trace ϵ mène de s à q,

Mots, langage reconnus par un automate fini

Propriété

Soit $A = (Q, V, \delta, I, F)$ un automate fini, $a \in V$ et $p, q \in Q$.
Il y a un chemin de trace a entre p et q ssi il existe deux états $r, s \in Q$ tels que :

- un chemin de trace ϵ mène de p à r,
- $(r, a, s) \in \delta$,
- un chemin de trace ϵ mène de s à q,

Démonstration.

En exercice. □

Elimination des ϵ-transitions

Théorème 26

Si un langage est reconnu par un automate fini alors il est reconnu par un automate fini sans ϵ-transitions.

Si $A = (Q, V, \delta, I, F)$ est un automate fini alors on démontre que l'automate $B = (Q, V, \eta, I, G)$ est équivalent à A où :

- $\forall a \in V, p, q \in Q$, $(p, a, q) \in \eta$ ssi $\exists r \in Q$ tel qu'un chemin de trace ϵ conduit de p à r et $(r, a, q) \in \delta$,
- $G = F \cup \{q \in Q |$ il existe un chemin de trace ϵ entre q et un état de $F\}$.

Pour montrer que A et B sont équivalents, on montre la propriété :
$\forall w \in V^*, p, q \in Q$ il existe un chemin entre p et q de trace w dans A ssi il existe un état $r \in Q$ et un chemin de trace w dans B entre p et r et un chemin de trace ϵ dans A entre r et q.

Démonstration.

Par induction sur $|w|$. □

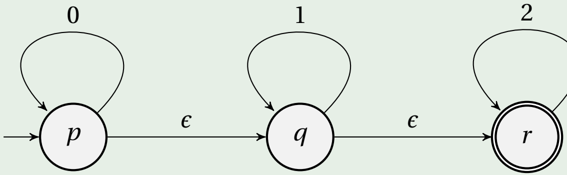
Algorithme de calcul de B

On procède en deux étapes :

1. Calculer pour tout $p \in Q$, l'ensemble d'états $f(p)$ où $f(p) = \{q \in Q |$ il y a un chemin de A de trace ϵ entre p et $q\}$
2. Calculer pour tout $p \in Q$ et tout $a \in V$, l'ensemble $g(p,a)$ où $g(p,a) = \bigcup_{q \in f(p)} \{r \in Q | (q,a,r) \in \delta\}$.

Exemple 27

L'automate $A = (Q, V, \delta, I, F)$:

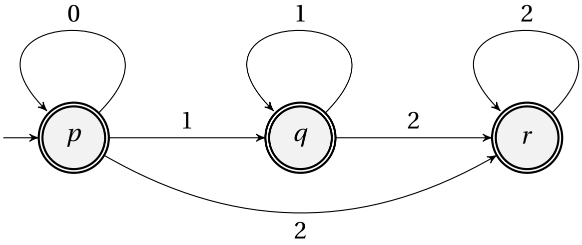


Elimination des ϵ-transitions

1. On calcule $f(s)$pour tout état s : $f(p) = \{p, q, r\}$, $f(q) = \{q, r\}$, $f(r) = \{r\}$.
   $G = \{p, q, r\}$
2. On calcule g :

	0	1	2
p	p	q	r
q	$\emptyset$	q	r
r	$\emptyset$	$\emptyset$	r

On obtient donc l'automate :



Définition 28 (Automates finis déterministes)

Un *automate fini déterministe (AFD)* est un automate fini $A = (Q, V, \delta, I, F)$ où :

- $\#I = 1$ (un seul état initial)
- δ ne contient aucune ϵ-transition
- $\forall p \in Q, a \in V, \exists! q \in Q,\ (p, a, q) \in \delta$.

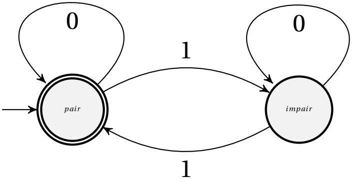
δ dans ce cas est une fonction $Q \times V$ dans Q appelée *fonction de transition*.

- On note $\delta(p, a)$ l'unique q tel que $(p, a, q) \in \delta$.

Exemple 29

A Automate déterministe sur le vocabulaire $\{0, 1\}$: de chaque état part une seule 0-transition et une seule 1-transition.

$\mathscr{L}(A)$ est l'ensemble des chaînes de de 0 et de 1 ayant un nombre pair de 1.



Soit $\delta^* : Q \times V^* \to Q$ la fonction définie par :

- $\forall p \in Q,\ \delta^*(p,\epsilon) = p$,
- $\forall p, q \in Q, x \in V^*, a \in V,\ \delta^*(p, xa) = \delta(\delta^*(p,x), a)$.

$\forall p \in Q, a \in V,\ \delta^*(p,a) = \delta(\delta^*(p,\epsilon), a) = \delta(p,a)$: δ^*est une extension de δ.
Dans la suite on ne fera plus de distinction entre δ et δ^*.

- Soit $A = (Q, V, \delta, q_0, F)$ un AFD,
  - $\forall w \in V^*$, il existe un et un seul calcul de A qui s'arrête après lecture de tous les symboles de w.
  - Tout calcul d'un AFD a la longueur de la chaîne d'entrée
- Un *automate fini déterministe incomplet* est un automate fini $A = (Q, V, \delta, q_0, F)$ sans ϵ-transitions tel que δ est une fonction partielle de $Q \times V$ dans Q : $\forall p \in Q, q \in V$ il existe au plus état $q \in Q$ tel que $(p, a, q) \in \delta$.
  - Le calcul s'arrête ou bien après avoir lu tous les caractères : $\Rightarrow$ acceptation du mot, ou bien avant (faute de transition disponible) : $\Rightarrow$ rejet.

Automates finis déterministes : propriétés

Propriétés

Soit $A = (Q, V, \delta, q_0, F)$ un AFD.

1. $\forall p, q \in Q, w \in V^*$, $\delta(p, w) = q$ ssi il existe un chemin de p à q de trace w.
2. $\mathscr{L}(A) = \left\{w \in V^* | \delta(q_0, w) \in F\right\}$
3. $\forall p \in Q, x, y \in V^*$, $\delta(p, xy) = \delta(\delta(p, x), y)$.

Automates finis déterministes : propriétés

Propriétés

Soit $A = (Q, V, \delta, q_0, F)$ un AFD.

1. $\forall p, q \in Q, w \in V^*$, $\delta(p, w) = q$ ssi il existe un chemin de p à q de trace w.
2. $\mathscr{L}(A) = \{w \in V^* | \delta(q_0, w) \in F\}$
3. $\forall p \in Q, x, y \in V^*$, $\delta(p, xy) = \delta(\delta(p, x), y)$.

Démonstration.

1. Preuve par récurrence sur w.
2. Preuve immédiate en utilisant la propriété 2.
3. Par récurrence sur y.

$\square$

Etats accessibles

Définition 30 (Etat accessible)

Soit $A = (Q, V, \delta, q_0, F)$ un AFD. L'état $p \in Q$ est *accessible* ssi $\exists w \in V^*$ telle que $p = \delta(q_0, w)$.
Un AFD dont tous les états sont accessibles est dit *initialement connecté*.

Définition 30 (Etat accessible)

Soit $A = (Q, V, \delta, q_0, F)$ un AFD. L'état $p \in Q$ est *accessible* ssi $\exists w \in V^*$ telle que $p = \delta(q_0, w)$.
Un AFD dont tous les états sont accessibles est dit *initialement connecté*.

Propriété (Elimination des états inaccessibles)

Soit $A = (Q, V, \delta, q_0, F)$ un AFD et R l'ensemble des états accessibles de A.
L'automate $(R, V, \eta, q_0, R \cap F)$ est un AFD initialement connecté équivalent à A où $\eta = \delta \cap (R \times V \times R)$.

Equivalence des automates finis sans ϵ-transitions et des AFD

Théorème 31 (Equivalence des automates finis sans -transitions et des AFD)

Tout langage reconnu par un automate fini sans ϵ-transitions est reconnu par un AFD.

Equivalence des automates finis sans ϵ-transitions et des AFD

Théorème 31 (Equivalence des automates finis sans -transitions et des AFD)

Tout langage reconnu par un automate fini sans ϵ-transitions est reconnu par un AFD.

Idée de la démonstration

Soit $A = (Q, V, \delta, I, F)$ un automate fini sans ϵ-transitions.
Soit $B = (\mathscr{P}(Q), V, \eta, I, G)$ l'AFD définit par :

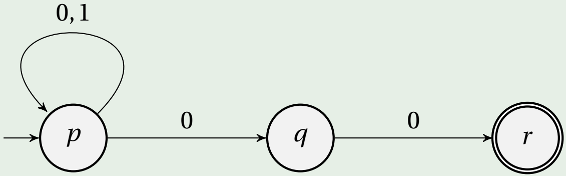
- $\forall P \in \mathscr{P}(Q), a \in V, \eta(P, a) = \bigcup_{p \in P} \{q \in Q | (p, a, q) \in \delta\}$,
- I est le seul état initial,
- $G = \{P \in \mathscr{P}(Q) | P \cap F \neq \emptyset\}$.

On montre que A et B sont équivalents.

Equivalence des automates finis sans ϵ-transitions et des AFD : exemple

Exemple 32

L'automate $A = (Q, V, \delta, I, F)$: $V = \{0, 1\}$



- $\mathscr{L}(A)$: l'ensemble des chaînes se terminant avec la chaîne 00.

Equivalence des automates finis sans ϵ-transitions et des AFD : exemple

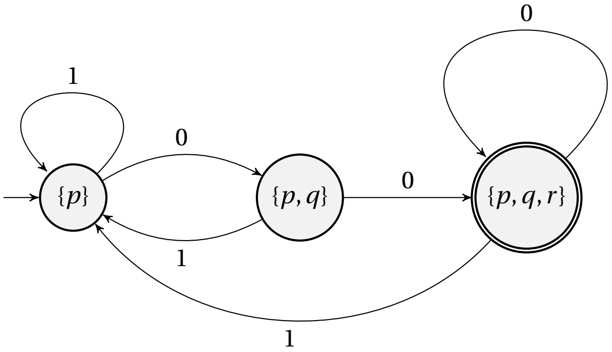
- Soit $B = (\mathscr{P}(Q), V, \eta, I, G)$ l'AFD définit par :
- $\forall P \in \mathscr{P}(Q), a \in V,\ I = \{p\},\ G = \{P \in \mathscr{P}(Q) \mid P \cap F \neq \emptyset\}$. η définie par le tableau :
  -

		$\mathscr{P}(Q)$	0	1
		$\emptyset$	$\emptyset$	$\emptyset$
(a)	initial	$\{p\}$	$\{p,q\}$	$\{p\}$
		$\{q\}$	$\{r\}$	$\emptyset$
	final	$\{r\}$	$\emptyset$	$\emptyset$
(b)		$\{p,q\}$	$\{p,q,r\}$	$\{p\}$
	final	$\{p,r\}$	$\{p,q\}$	$\{p\}$
	final	$\{q,r\}$	$\{r\}$	$\emptyset$
(c)	final	$\{p,q,r\}$	$\{p,q,r\}$	$\{p\}$

  - Les états accessibles sont $\{\{p\},\{p,q\},\{p,q,r\}\}$

Equivalence des automates finis sans ϵ-transitions et des AFD : exemple

On obtient donc l'AFD équivalent :



Equivalence des automates finis des AFD

Corollaire 33 (Equivalence des automates finis des AFD)

Un langage est reconnu par un automate fini si et seulement si il est reconnu par un automate fini déterministe connecté.

Corollaire 33 (Equivalence des automates finis des AFD)

Un langage est reconnu par un automate fini si et seulement si il est reconnu par un automate fini déterministe connecté.

Démonstration.

D'après le théorème 26, un automate fini est équivalent à un automate fini sans ϵ-transition. D'après le théorème 31, un automate fini sans ϵ-transition est équivalent à un automate déterministe. D'après la propriété 59 un automate fini déterministe est équivalent à un automate fini déterministe connecté. Ceci termine la preuve. $\square$

Plan

Définition 34 (Expression régulière)

L'ensemble des *expressions régulières* sur un vocabulaire V, noté $\mathscr{R}(V)$ est le plus petit ensemble (au sens de l'inclusion) tel que :

- $V \subset \mathscr{R}(V)$,
- $\emptyset \subset \mathscr{R}(V)$,
- Si $e, f \in \mathscr{R}(V)$ alors $e + f \in \mathscr{R}(V)$, $e.f \in \mathscr{R}(V)$, $e^* \in \mathscr{R}(V)$.
- Les priorités des opérateurs sont dans l'ordre : $*, ., +$

Une expression régulière définit un langage sur V :

- Chaque lettre $a \in V$ est identifiée au langage $\{a\}$,
- $\emptyset$ est identifié au langage vide,
- $+$ est interprété comme l'union,
- . est interprété comme la concaténation,
- * est interprété comme la concaténation itérée,

Un langage ainsi défini est la *valeur de l'expression régulière*.

Définition 35 (Langage régulier)

Un *langage régulier* est un langage définissable par une expression régulière.

Exemple 36

- Tout langage fini est régulier
- $(0+1+2+3+4+5+6+7+8+9).(0+1+2+3+4+5+6+7+8+9)^*$ définit le langage des nombres entiers
- $\emptyset^*$ est une expression régulière qui définit le langage $\{\epsilon\}$.

Tout langage régulier est reconnu par un automate fini

Définition 37

Soit e une expression régulière sur V. $\lambda(e)$ désigne le langage sur V, valeur de e. $\nu(e)$ désigne le nombre d'occurrences de signes "$+$","$.$","$*$" apparaissant dans e.

Exemple 38

$e = (00)^*(11)^*0+1$ (équivalente à $(0.0)^*.(1.1)^*.0+1$). $\nu(e) = 7$.

Définition 39 (Automate simple)

Un automate fini est *simple* s'il a un seul un état initial et un seul état final et aucune transition ne va vers l'état initial et aucune transition ne sort de l'état final.

Théorème 40

Tout langage régulier est reconnu par un automate fini.

Théorème 40

Tout langage régulier est reconnu par un automate fini.

Démonstration.

On démontre une propriété plus forte : $\forall e \in \mathscr{R}(V), \exists \alpha \in \mathbb{N}, \alpha \geq 2$, il existe un automate fini simple $A = (\{1, 2, \ldots, \alpha\}, V, \delta, \{1\}, \{\alpha\})$ qui reconnaît $\lambda(e)$.
La preuve est par récurrence sur $\nu(e)$. $\square$

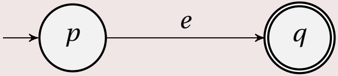
Tout langage régulier est reconnu par un automate fini

Démonstration.

Si $\nu(e) = 0$: ou bien $e \in V \cup \{\epsilon\}$ ou bien $e = \emptyset$.

- Si $e = \emptyset$, alors $\lambda(e) = \emptyset$ et l'automate A est tel que $\mathscr{L}(A) = \lambda(e) = \emptyset$:

- Si $e \in V \cup \{\epsilon\}$ alors l'automate A est tel que $\mathscr{L}(A) = \lambda(e)$:

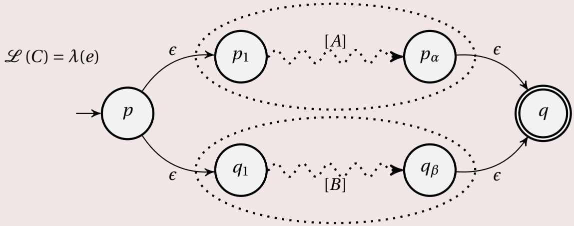


$\square$

Démonstration.

Supposons que c'est vrai jusqu'à n et montrons que c'est vrai pour $n+1$. Soit $e \in \mathscr{R}(V)$ tq $v(e) = n+1$: ou bien $e \in V \cup \{\epsilon\}$ ou bien $e = \emptyset$.

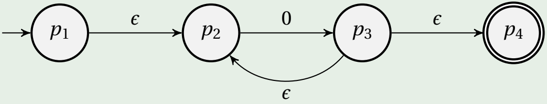
- Si $e = f + g$, alors comme $v(f) \leq n$ et $v(g) \leq n$ alors par H.R. il existe deux automates finis simples $A = \left(\left\{p_1, p_2, \ldots, p_\alpha\right\}, V, \delta, \left\{p_1\right\}, \left\{p_\alpha\right\}\right)$ et $B = \left(\left\{q_1, q_2, \ldots, q_\beta\right\}, V, \mu, \left\{q_1\right\}, \left\{q_\beta\right\}\right)$ tels que $\mathscr{L}(A) = \lambda(f)$ et $\mathscr{L}(B) = \lambda(g)$. Soit l'automate $C = \left(\{p, q\} \cup \left\{p_1, p_2, \ldots, p_\alpha\right\} \cup \left\{q_1, q_2, \ldots, q_\beta\right\}, V, \eta, \{p\}, \{q\}\right)$, tel que $\{p, q\} \cap \left\{p_1, p_2, \ldots, p_\alpha\right\} = \emptyset$ et $\{p, q\} \cap \left\{q_1, q_2, \ldots, q_\beta\right\} = \emptyset$ et $\eta = \delta \cup \mu \cup \left(p, \epsilon, p_1\right) \cup \left(p, \epsilon, q_1\right) \cup \left(p_\alpha, \epsilon, q\right) \cup \left(q_\beta, \epsilon, q\right)$. Alors :



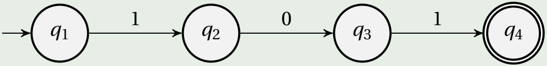
□

Exemple 41

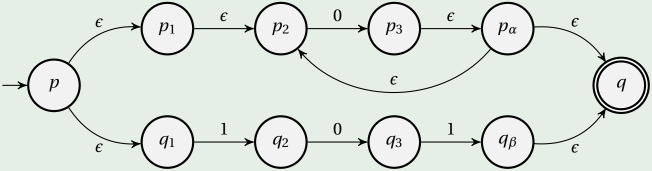
Le langage 0^* est reconnu par l'automate :



Le langage 101 est reconnu par l'automate :



Le langage $0^* + 101$ est reconnu par l'automate :

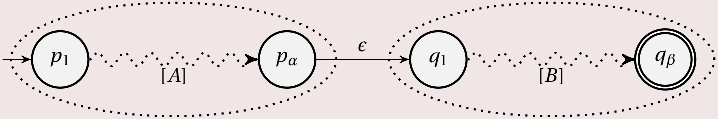


Tout langage régulier est reconnu par un automate fini

Démonstration.

Supposons que c'est vrai jusqu'à n et montrons que c'est vrai pour $n+1$. Soit $e \in \mathscr{R}(V)$ tq $\nu(e) = n+1$: ou bien $e \in V \cup \{\epsilon\}$ ou bien $e = \emptyset$.

- Si $e = f.g$, alors comme $\nu(f) \leq n$ et $\nu(g) \leq n$ alors par H.R. il existe deux automates finis simples $A = \left(\{p_1, p_2, \ldots, p_\alpha\}, V, \delta, \{p_1\}, \{p_\alpha\}\right)$ et $B = \left(\{q_1, q_2, \ldots, q_\beta\}, V, \mu, \{q_1\}, \{q_\beta\}\right)$ tels que $\mathscr{L}(A) = \lambda(f)$ et $\mathscr{L}(B) = \lambda(g)$. Soit l'automate $C = \left(\{p_1, p_2, \ldots, p_\alpha\} \cup \{q_1, q_2, \ldots, q_\beta\}, V, \eta, \{p_1\}, \{q_\beta\}\right)$, tel que $\eta = \delta \cup \mu \cup \left(p_a, \epsilon, q_1\right)$. Alors $\mathscr{L}(C) = \lambda(e)$:

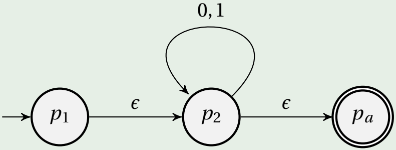


□

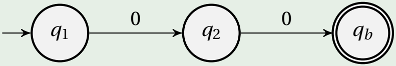
Tout langage régulier est reconnu par un automate fini

Exemple 42

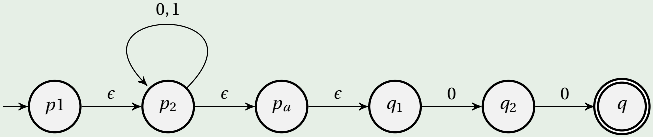
Le langage $(0+1)^*$ est reconnu par l'automate :



Le langage 00 est reconnu par l'automate :



Le langage $(0+1)^*00$ est reconnu par l'automate :

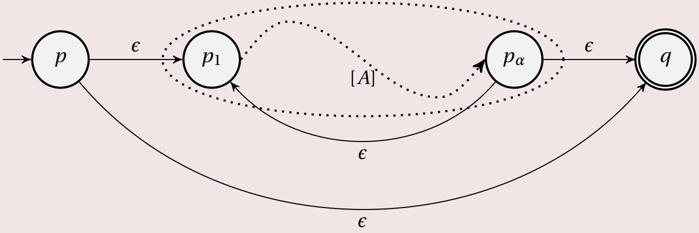


Tout langage régulier est reconnu par un automate fini

Démonstration.

Supposons que c'est vrai jusqu'à n et montrons que c'est vrai pour $n+1$. Soit $e \in \mathscr{R}(V)$ tq $v(e) = n+1$: ou bien $e \in V \cup \{\epsilon\}$ ou bien $e = \emptyset$.

- Si $e = f^*$, alors comme $v(f) \leq n$ alors par H.R. il existe un automate fini simple $A = \left(\{p_1, p_2, \ldots, p_\alpha\}, V, \delta, \{p_1\}, \{p_\alpha\}\right)$ tel que $\mathscr{L}(A) = \lambda(f)$ Soit l'automate $C = \left(\{p,q\} \cup \{p_1, p_2, \ldots, p_\alpha\} \cup \{q_1, q_2, \ldots, q_\beta\}, V, \eta, \{p\}, \{q\}\right)$, tel que $\{p,q\} \cap \{p_1, p_2, \ldots, p_\alpha\} = \emptyset$ et $\eta = \delta \cup (p,\epsilon,q) \cup (p,\epsilon,p_1) \cup (p_\alpha,\epsilon,q) \cup (p_\alpha,\epsilon,p_1)$. Alors $\mathscr{L}(C) = \lambda(e)$:

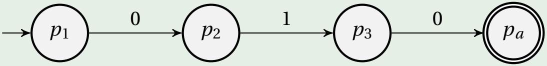


□

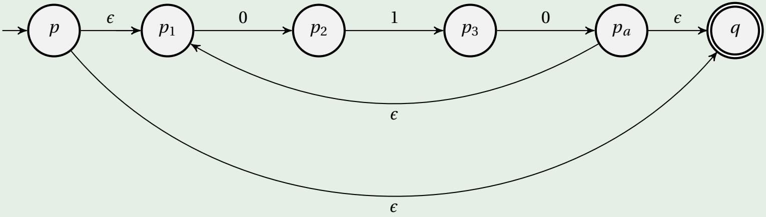
Tout langage régulier est reconnu par un automate fini

Exemple 43

Le langage 010 est reconnu par l'automate :



Le langage $(010)^*$ est reconnu par l'automate :



Définition 44

Soient α et β deux langages sur un vocabulaire V et x une variable non élément de V.
Soit L un langage sur V.
$x = L$ est une *solution* de l'équation $x = \alpha.x + \beta$ si le langage L vérifie $L = \alpha.L + \beta$.

Lemme 45 (Lemme d'Arden)

Soient α et β deux langages sur un vocabulaire V et x une variable non élément de V.

- *Le langage $x = \alpha^*.\beta$ est la plus petite solution (au sens de l'inclusion) de l'équation $x = \alpha.x + \beta$ (i.e. si L est une solution alors $x \subseteq L$).*
- *Si $\epsilon \notin \alpha$ alors $\alpha^*\beta$ est l'unique solution.*

Lemme d'Arden.

1. $x = \alpha^*.\beta$ est trivialement solution de l'équation : $\alpha^*.\beta = \alpha.\alpha^*.\beta + \beta$. (il est facile de démontrer la double inclusion).
2. $\alpha^*.\beta$ est la plus petite solution. Soit $y = L$ une autre solution, on montre (par récurrence) que :
   1. $\forall i \geq 0,\ \alpha^i.\beta \subseteq L$: Pour $i = 0$, $\beta \subseteq L$, en effet, $L = \alpha.L + \beta$ $(L = \alpha.L \cup \beta)$. Supposons que c'est vrai pour i et montrons pour $i+1$. $\alpha^i.\beta \subseteq L$ par HR, donc $\alpha.\alpha^i.\beta \subseteq \alpha.L$ et donc $\alpha^{i+1}.\beta \subseteq \alpha.L \subset L$ (car $(L = \alpha.L \cup \beta)$.
   2. $\alpha^*\beta \subseteq L$. Soit w une chaîne de $\alpha^*\beta$. Il existe $i \geq 0$ tel que $w \in \alpha^i.\beta$ et donc $w \in L$.
3. Si $\epsilon \notin \alpha$ alors $\alpha^*\beta$ est l'unique solution. Soit y une autre solution. $\alpha^*.\beta \subset y$. On a $y = \alpha^{i+1}.y + \alpha^i.\beta + \ldots + \alpha.\beta + \beta$ (la preuve est immédiate par récurrence sur i). Soit w un mot de L de longueur i. $w \in \alpha^{i+1}.y + \alpha^i.\beta + \ldots + \alpha.\beta + \beta$ mais $w \notin \alpha^{i+1}.y$ (la taille de w est i). Donc $w \in \alpha^i.\beta + \ldots + \alpha.\beta + \beta$ et donc $w \in \alpha^*.\beta$.

□

Exemple 46

- Si $V = \{a\}$, alors l'équation $x = x + a$ admet comme solution L tout langage sur V tel que $a \in L$.
- Si $V = \{a, b\}$, alors l'équation $x = (a + \epsilon)x + b$ admet comme solution L tout langage sur V tel que $(a^* b) \subseteq L$.
- Si $V = \{a\}$, alors l'équation $x = ax$ admet une seule solution $L = a^*.\emptyset = \emptyset$.

Définition 47 (Système régulier)

Soit V un vocabulaire et $x_1, \ldots, x_n$ un ensemble de n variables distinctes $\notin V$.
Un *système régulier* sur V est un ensemble de n équations :

$$\begin{cases} x_1 &= & \beta_1 + \sum_{j=1}^{j=n} \alpha_{1,j} x_j \\ x_2 &= & \beta_2 + \sum_{j=1}^{j=n} \alpha_{2,j} x_j \\ \ldots &= & \ldots \\ x_i &= & \beta_i + \sum_{j=1}^{j=n} \alpha_{i,j} x_j \\ \ldots &= & \ldots \\ x_n &= & \beta_n + \sum_{j=1}^{j=n} \alpha_{n,j} x_j \end{cases}$$

où les $\alpha_{i,j}$ et β_i sont des langages sur V.
$x_1 = L_1, \ldots, x_n = L_n$ est une solution sur V de ce système si les langages L_i vérifient : $L_i = \beta_i + \sum_{j=1}^{j=n} \alpha_{i,j} L_j$.

Théorème 48

Tout système régulier V à n équations et n variables $x_1, \ldots, x_n$ admet une plus petite solution $x_1 = L_1, \ldots, x_n = L_n$, i.e. toute autre solution $x_1 = M_1, \ldots, x_n = M_n$ vérifie : $L_i \subseteq M_i$ et L_i est un langage régulier sur V pour tout i.

Théorème 48

Tout système régulier V à n équations et n variables $x_1, \ldots, x_n$ admet une plus petite solution $x_1 = L_1, \ldots, x_n = L_n$, i.e. toute autre solution $x_1 = M_1, \ldots, x_n = M_n$ vérifie : $L_i \subseteq M_i$ et L_i est un langage régulier sur V pour tout i.

Démonstration.

Par récurrence sur n. □

Système d'équations associé à un automate fini

Théorème 49 (Système d'équations associé à un automate fini)

Soit $A = \left(Q = \{q_1, \ldots, q_n\}, V, \delta, I, F\right)$ *un automate fini.*
Soit $\alpha_{i,j} = \left\{a \in V \cup \{\epsilon\} \mid \left(q_i, a, q_j\right) \in \delta\right\} \forall i, j \in [\![1..n]\!]$.
Soit $x_1, \ldots, x_n$ *un ensemble de variables* $\notin V$.
Le système d'équations sur V *défini par :* $\begin{cases} x_i = \epsilon + \sum_{j=1}^{j=n} \alpha_{i,j} x_j & \text{si } q_i \in F \\ x_i = \sum_{j=1}^{j=n} \alpha_{i,j} x_j & \text{si } q_i \notin F \end{cases}$
a pour plus petite solution $x_1 = \mathscr{L}(A_1), \ldots, x_n = \mathscr{L}(A_n)$, *où* $\forall i$,
$A_i = \left(Q =, V, \delta, q_i, F\right)$

Théorème 49 (Système d'équations associé à un automate fini)

Soit $A = (Q = \{q_1, \ldots, q_n\}, V, \delta, I, F)$ *un automate fini.*
Soit $\alpha_{i,j} = \{a \in V \cup \{\epsilon\} \mid (q_i, a, q_j) \in \delta\} \; \forall i, j \in [\![1..n]\!]$.
Soit $x_1, \ldots, x_n$ *un ensemble de variables* $\notin V$.
Le système d'équations sur V *défini par :* $\begin{cases} x_i = \epsilon + \sum_{j=1}^{j=n} \alpha_{i,j} x_j & \text{si } q_i \in F \\ x_i = \sum_{j=1}^{j=n} \alpha_{i,j} x_j & \text{si } q_i \notin F \end{cases}$
a pour plus petite solution $x_1 = \mathscr{L}(A_1), \ldots, x_n = \mathscr{L}(A_n)$, *où* $\forall i$,
$A_i = (Q =, V, \delta, q_i, F)$

Démonstration.

Théorème admis. □

Théorème 50

Tout langage reconnu par un automate fini est un langage régulier

Equivalence des expressions régulières et des automates finis

Théorème 50

Tout langage reconnu par un automate fini est un langage régulier

Démonstration.

Soit L un langage reconnu par un automate fini $\left(Q = \{q_1, \ldots, q_n\}, V, \delta, I, F\right)$. Soient $A_i = \left(Q =, V, \delta, q_i, F\right)$, $\forall i$.
$L = \mathscr{L}(A) = \bigcup_{i \in I} \mathscr{L}(A_i)$. Par le théorème 49, on peut associer à A un système régulier dont les variables $x_1, \ldots, x_n$ correspondent resp. aux états $q_1, \ldots, q_n$ et dont la plus petite solution est $x_i = \mathscr{L}(A_i) \forall i$. D'après le théorème 48, les x_i sont réguliers et donc L l'est aussi. □

Théorème 51 (Equivalence des expressions régulières et des automates finis)

Un langage est reconnu par un automate fini si et seulement si il est défini par une expression régulière.

Plan

Lemme de l'étoile

Définition 52

Soit p un chemin dans un automate fini, tel que l'origine et l'extrémité de p sont identiques (notés q). On définit $p^0 = (q, \epsilon, q)$ et $p^i = p.p^{i-1}$ pour tout $i \geq 1$.

Proposition

Soit p un chemin dans un automate fini A, dont l'origine et l'extrémité sont identiques. Si p est de trace z, alors pour tout $i \in \mathbb{N}$, p^i est un chemin dans A, de trace z^i.

Lemme 53 (Lemme de l'étoile)

Soit L un langage régulier. Alors il existe $n \geq 1$ tel que pour tout $z \in L$, si $|z| \geq n$ alors z est de la forme uvw, où :

- $|v| \geq 1$;
- $|uv| \leq n$;
- $\forall i \geq 0,\ uv^i w \in L$.

Exemple 54

Le langage $L = \{0^n 1^n\}$ n'est pas régulier.

Attention la réciproque n'est pas toujours vraie :

Exemple 55

Soit $M = \left(\{c\}^+ .L\right) \cup \{a, b\}^*$ sur $\{a, b, c\}$. M vérifie la conclusion du lemme de l'étoile, mais il n'est pas régulier (on verra plus tard pourquoi).

Théorème 56

Théorème uvwUn langage $L \subseteq V^*$ est régulier si et seulement s'il existe un entier $n \geq 1$ tel que, pour tout $z \in V^*$, si $|z| \geq n$, alors z est de la forme uvw où :

- $|v| \geq 1$;
- $\forall i \geq 0,\ \forall x \in V^*,\ uwx \in L \Leftrightarrow uv^i wx \in L.$

Opérations ensemblistes

Lemme 57

La classe des langages réguliers est fermée par :

- *concaténation,*
- *union,*
- *fermeture de Kleene (l'étoile "*").*

Lemme 58

La classe des langages réguliers est fermée par l'opération de complémentation.

Corollaire 59

Le classe des langages réguliers est fermée par :

- *intersection,*
- *différence.*

Substitutions régulières

Définition 60 (substitution régulière)

Soient deux vocabulaires V et W pas nécessairement disjoints. Une *substitution régulière* est une fonctions $s : V \to \mathscr{P}(W^*)$, qui à chaque lettre $a \in V$ associe un langage régulier $s(a)$ sur le vocabulaire W.
La définition d'une substitution régulière est étendue à V^* par induction, en posant :

- $s(\epsilon) = \epsilon$,
- $s(a.w) = s(a).s(w)$, pour toute lettre $a \in V$ et tout mot $w \in V$.

Enfin, la définition d'une substitution régulière est étendue aux langages de V^*, en posant :

$$\forall L \subseteq V^*, \quad s(L) = \bigcup_{w \in L} s(w)$$

Proposition

Pour tous $x, y \in V^*$ et pour toute substitution régulière s, $s(x.y) = s(x).s(y)$.

Substitutions régulières

Lemme 61

Soient L, L' *des langages réguliers sur* V *et* s *une substitution régulière. Alors :*

1. $s(L.L') = s(L).s(L')$;
2. $s(L \cup L') = s(L) \cup s(L')$;
3. $s(L^*) = [s(L)]^*$.

Lemme 62

Soient E, E' *des expressions régulières sur* V *et* s *une substitution régulière. Alors :*

1. $s(E.E') = s(E).s(E')$;
2. $s(E + E') = s(E) + s(E')$;
3. $s(E)^* = [s(E)]^*$.

Théorème 63

La classe des langages réguliers est fermée par substitution régulière.

Définition 64 (Homomorphisme)

Un *homomorphisme* est une substitution qui à toute lettre $a \in V$ associe un singleton dans W^*. Si h est un homomorphisme, et l'image de $a \in V$ par h est le singleton $\{w\} \in W^*$, alors on note $h(a) = w$.

Corollaire 65

La classe des langages réguliers est fermée par homomorphisme.

Définition 66 (Homomorphisme inverse)

Etant donnés deux vocabulaires V, W, un langage $L \subseteq W^*$ et un homomorphisme h, l'image par homomorphisme inverse de L est l'ensemble
$h^{-1}(L) = \left\{v \in V^* | h(v) \in L\right\}$

Théorème 67

Si $L \subseteq W$ est un langage régulier et h est un homomorphisme, alors $h^{-1}(L)$ est un langage régulier.

Théorème 68

Si $L \subseteq W$ est un langage régulier et h est un homomorphisme, alors $h^{-1}(L)$ est un langage régulier.

Exemple 69

$M = \left(\{c\}^+ .L\right) \cup \{a,b\}^*$ sur $\{a,b,c\}$ n'est pas régulier, où $L = \left\{a^n b^n | n \geq 0\right\}$. Supposons que M est régulier, soit $M' = M \cap \left(\{c\}^+ \{a,b\}^*\right)$. $M' = \{c\}^+ .L$. Or $\{c\}^+ \{a,b\}^*$ est régulier et M l'est aussi (par hypothèse). Donc M' est régulier (intersection de deux langages réguliers). Soit l'homomorphisme $h : V \to V \cup \{\epsilon\}$, tel que $h(a) = a$, $h(b) = b$ et $h(c) = \epsilon$. Comme les langages réguliers sont stables par homomorphisme, $h(M) = L$ doit être régulier ce qui est absurde. M n'est donc pas régulier.

Quatrième partie IV

Grammaires hors contexte

Plan

6 Systèmes de réécriture et grammaires

7 Grammaires hors-contexte

Systèmes de réécriture et grammaires

- AFD = machine capable de reconnaître des éléments du langage.
- De façon duale, nous nous intéressons dans la suite à des systèmes capables d'engendrer les éléments d'un langage.
- $\Rightarrow$ Réécriture de symboles pour engendrer ces éléments
- $\Rightarrow$ Grammaires : permettent d'engendrer les langages formels.

Définition 70 (Système de réécriture (de mots))

On appelle *système de réécriture (de mots)* un couple $S = (V, R)$ où :

- V est un vocabulaire,
- R est un ensemble fini de couples de chaînes sur V (relation sur V^*).

Tout élément (α, β) de R est appelé *règle de réécriture* et est noté $\alpha \rightarrow \beta$;
α est la partie gauche, β la partie droite de la règle $\alpha \rightarrow \beta$.
Des règles $\alpha \rightarrow \beta_1$, $\alpha \rightarrow \beta_2$, ..., $\alpha \rightarrow \beta_k$ ayant même partie gauche sont souvent notées $\alpha \rightarrow \beta_1 | \beta_2 \ldots | \beta_k$.
Soient un système de réécriture $S = (V, R)$ et deux chaînes x et y de V. S'il existe une règle $\alpha \rightarrow \beta$ dans R telle que $x = u\alpha v$, $y = u\beta v$, alors on dit que x se *réécrit (directement) en* y, ce qu'on note $x \Rightarrow_S y$, ou $x \Rightarrow_R y$, ou encore, s'il n'y a pas d'ambiguïté sur le système de réécriture utilisé, $x \Rightarrow y$.

- Si $x \Rightarrow_R y$, plusieurs règles de R peuvent être possibles pour réécrire x en y.
- Par exemple, soit $R = \{ab \rightarrow ba, aab \rightarrow aba\}$: on a $aab \Rightarrow_R aba$, par application de la première ou de la deuxième règle de R.
- Une même règle peut s'appliquer à différentes sous-chaînes de x pour donner y :
- $R = \{a \rightarrow aa\}$: aaa peut être obtenue en réécrivant la première ou la deuxième lettre de la chaîne aa.

Définition 71

- $x \Rightarrow^n y$: x se réécrie en y en appliquant n règles. $x \Rightarrow^0 y$ ssi $x = y$.
- $x \Rightarrow^* y$: $\exists n \in \mathbb{N},\ x \Rightarrow^n y$.
- $x \Rightarrow^+ y$: $\exists n \in \mathbb{N}^*,\ x \Rightarrow^n y$.

Définition 72 (Dérivation dans un système de réécriture)

On appelle *dérivation dans un système de réécriture* toute suite finie $x_1, x_2, \ldots, x_k$ de chaînes telles que $k \geq 1$ $x_1 \Rightarrow x_2 \Rightarrow \ldots \Rightarrow x_k$.
x_k est *dérivée à partir de* x_1 : x_1 est l'*origine*, x_k est l'*extrémité* de la dérivation.
La *longueur* de la dérivation est égale à $k-1$, (i.e. le nombre d'applications de règles de réécriture dans la dérivation).
Les chaînes x_i sont les *éléments* de la dérivation.

Propriété

- $x \Rightarrow^* y$ ssi il existe une dérivation d'origine x et d'extrémité y.
- $x \Rightarrow^+ y$ ssi il existe une dérivation de longueur n d'origine x et d'extrémité y.

Attention

La donnée d'une dérivation ne permet pas toujours de retrouver les règles de réécriture appliquées, ni les sous-chaînes réécrites (car la relation $\Rightarrow$ ne le permet pas toujours).

Définition 73 (Dérivation indicée dans un système de réécriture)

Soit $i \in \mathbb{N}^*$, r une règle . On note $\Rightarrow_{i,r}$ la relation sur V^* définie par : $x \Rightarrow_{i,r} y$ ssi :

- $x = a_1 a_2 \dots a_n$, où $a_1, a_2, \dots, a_n \in V$,
- $r = a_i \dots a_k \rightarrow \omega$, $1 \leq i \leq n+1, i-1 \leq k \leq n$,
- $y = a_1 a_2 \dots a_{i-1} \omega a_{k+1} \dots a_n$.

i désigne le premier caractère de la sous-chaîne réécrite de x, r la règle appliquée. On a $k = i-1$ quand la partie gauche de r est la chaîne vide et $i = n+1$ quand la chaîne vide est réécrite en ω à droite de x.

Une *dérivation indicée (pour un ensemble de règles de réécriture R)* est une suite finie $x_1, (i_1, r_1), x_2, (i_2, r_2), \dots, x_{k-1}, (i_{k-1}, r_{k-1}), x_k$ telle que :

- $k \geq 1$, $x_1, \dots, x_k \in V^*$,
- $r_1, \dots r_{k-1} \in R$,
- $x_1 \Rightarrow_{i_1, r_1} x_2 \dots x_{k-1} \Rightarrow_{i_{k-1}, r_{k-1}} x_k$

Dérivations associées à un système de réécriture

On associe à un système de réécriture quatre relations de dérivation :

- $\Delta_1 : x \Rightarrow^* y$ (il existe une dérivation de x à y),
- $\Delta_2 : x \Rightarrow^n y$ (il existe une dérivation de x à y de longueur n),
- $\Delta_3 : x, x_2, \ldots, x_n, y$ est une dérivation de x à y,
- $\Delta_4 : x, (i_1, r_1), x_2, (i_2, r_2), \ldots, x_n, (i_n, r_n), y$ est une dérivation indicée de x à y.

On a les proriétés suivantes :

- Δ_1 s'écrit : $\exists n$ tel que Δ_2,
- Δ_2 s'écrit : $\exists x_1, \ldots, x_n$ tels que Δ_3,
- Δ_3 s'écrit : $\exists i_1, r_1, \ldots i_n, r_n$ tels que Δ_4.

Exemples de systèmes de réécriture

1. $V_1 = \{a, b, 0, 1, r, s\}$, $R_1 = \{ra \to s, rb \to s, sa \to s, sb \to s, s0 \to s, s1 \to s\}$. On vérifie que pour tout w dans $\{a, b, 0, 1\}^*$, on a $rw \Rightarrow s$ ssi si le premier caractère de w est a ou b. Ce système de réécriture reconnaît ainsi les identificateurs sur le vocabulaire $\{a, b, 0, 1\}$.
2. $V_2 = \{0, 1, +, E\}$, $R_2 = \{E \to 0|1|E + E\}$. Pour tout $w \in \{0, 1, +\}^*$, on a $E \Rightarrow^* w$ ssi w est une expression construite avec les opérandes 0 et 1 et l'opérateur $+$.
3. $V_3 = \{0, 1, +\}$, $R_3 = \{0 + 0 \to 0, 0 + 1 \to 1, 1 + 0 \to 1, 1 + 1 \to 0\}$. Ce système de réécriture calcule la somme modulo 2 d'une suite d'entiers binaires en ce sens que si w appartenant à V^* est expression dérivée à partir de E dans le système de l'exemple précédent, on a dans R_3 la dérivation $w \Rightarrow^* 0$ $w \Rightarrow^* 1$ suivant la valeur de l'expression quand l'opérateur $+$ est interprété comme l'addition modulo 2.
4. $V_4 = \{0, 1\}, R_4 = \{10 \to 01\}$. Soit $x \in V_4^*$ et soit y telle que a) $x \Rightarrow y$ et b) la règle de R_4 ne s'applique pas à y (en d'autres termes : y ne contient pas d'occurrence de 10). Alors y est la permutation des lettres de x où tous les 0 sont avant les 1. Ce système de réécriture trie donc les chaînes de V_4^* par chiffres croissants.

Composition des dérivations

Proposition

Soit $S = (V, R)$ et soient $u_1, \ldots, u_n, v_1, \ldots, v_n$ des chaînes de V^*. $u_1 \Rightarrow^* v_1, u_2 \Rightarrow^* v_2$, ..., $u_n \Rightarrow^* v_n$ alors $u_1 u_2 \ldots u_n \Rightarrow^* v_1 v_2 \ldots v_n$.

Démonstration.

1. si $u_1 \Rightarrow^* v_1$, on a alors $u_1 u_2 \Rightarrow^* v_1 v_2$ (démonstration par récurrence sur la longueur de la dérivation).
2. si $u_2 \Rightarrow^* v_2$, alors $v_1 u_2 \Rightarrow^* v_1 v_2$.
3. On en conclut que si $u_1 \Rightarrow^* v_1$ et $u_2 \Rightarrow^* v_2$, alors $u_1 u_2 \Rightarrow^* v_1 v_2$
4. La généralisation au cas de n dérivations est immédiate.

□

Proposition

Soient $u_1, \ldots, u_n, v_1, \ldots, v_n$ des chaînes de V^*, Pour tout $q_1, \ldots, q_n \in \mathbb{N}$, si $u_1 \Rightarrow^{q_1} v_1, \ldots, u_n \Rightarrow^{q_n} v_n$ alors $u_1 u_2 \ldots u_n \Rightarrow^{p} v_1 v_2 \ldots v_n$, avec $p = \sum_{i=1}^{i=n} q_i$

Grammaires

Définition 74 (Grammaire)

Une *grammaire* est un quadruplet $G = (V_T, V_N, S, R)$ où

- V_T et V_N sont deux vocabulaires disjoints. V_T est le *vocabulaire terminal* (ses éléments sont appelés *symboles terminaux*), V_N est le *vocabulaire non-terminal* (ses éléments sont les *symboles non-terminaux*). Les chaînes sur V_T sont dites *chaînes terminales*.
  $V = V_T \cup V_N$ est appelé le *vocabulaire* de la grammaire.
- S est un élément de V_N, l'*axiome* de la grammaire.
- (V, R) est un système de réécriture. R est l'ensemble des *règles* de la grammaire.

Exemple

$V_T = \{a, b\},\ V_N = \{S\},\ R = \{S \to aSb|\epsilon\},\ G = (V_T, V_N S, R)$.

Définition 75 (Langage engendré par une grammaire)

On appelle *langage engendré* par une grammaire $G = (V_T, V_N, S, R)$ l'ensemble des chaînes sur le vocabulaire terminal V_T que l'on peut dériver à partir de l'axiome S. On note $\mathscr{L}(G)$ le langage engendré par la grammaire G. On a donc : $\mathscr{L}(G) = \left\{x \in V_T^* | S \Rightarrow^* x\right\}$.

Définition 76 (Equivalence des grammaires)

Deux grammaires sont dites équivalentes si elles engendrent le même langage.

Exemple

$V_T = \{a, b\},\ V_N = \{S\},\ R = \{S \to aSb|\epsilon\},\ G = (V_T, V_N S, R).\ \mathscr{L}(G) = \left\{a^n b^n | n \in \mathbb{N}\right\}$.

Hiérarchie des grammaires

En imposant des restrictions sur la forme des règles, on définit différents types de grammaires :

Grammaires générales (de type 0) Une grammaire est dite *générale* si toutes ses règles sont de la forme $\alpha \to \beta$, avec $\alpha, \beta \in V^*$ et $\alpha \neq \epsilon$.

Grammaires sous-contexte (de type 1) Une grammaire est dite *sous-contexte* si toutes ses règles sont de la forme $\alpha A \beta \to \alpha \omega \beta$, avec $\alpha, \beta \in V^*$, $A \in V_N$ et $\omega \in V^+$. On autorise parfois d'ajouter la règle $S \to \epsilon$, à condition que l'axiome S n'apparaisse dans aucune partie droite de règle. Ceci permet à des grammaires sous-contexte d'engendrer le mot vide.

Grammaires hors-contexte (de type 2) Une grammaire est dite *hors-contexte* si toutes ses règles sont de la forme $A \to \omega$, avec $A \in V_N$, $\omega \in V^*$. On appelle ϵ-*règle* une règle de la forme $A \to \epsilon$, et 1-*règle* une règle de la forme $A \to B$ avec $B \in V_N$. Les grammaires hors-contexte sont souvent appelées *formes de Backus*. Elles sont équivalentes aux cartes syntaxiques couramment utilisées pour définir la syntaxe des langages de programmation.

Grammaires linéaires à droite (de type 3) Une grammaire est dite *linéaire à droite* si toutes ses règles sont de l'une des deux formes suivantes :

- $A \to \omega B,\ A, B \in V_N,\ \omega \in V_T^*$
- $A \to \omega,\ A \in V_N,\ \omega \in V_T^*$

Grammaires linéaires à gauche (de type 3) grammaire est dite *linéaire à gauche* si toutes ses règles sont de l'une des deux formes suivantes :

- $A \to B\omega,\ A, B \in V_N,\ \omega \in V_T^*$
- $A \to \omega,\ A \in V_N,\ \omega \in V_T^*$

Ces grammaires sont également appelées grammaires régulières.

Propriétés

- Toute grammaire linéaire à droite (resp. à gauche) est une grammaire hors-contexte.
- Toute grammaire hors-contexte ne contenant pas d'ϵ-règle est une grammaire sous-contexte.
- Un langage est dit *linéaire à droite* (*hors-contexte*, *sous-contexte*) s'il existe une grammaire linéaire à droite (hors-contexte, sous-contexte) qui l'engendre.
- Pour tout langage régulier, il existe une grammaire linéaire à droite et une grammaire linéaire à gauche qui l'engendrent.
- Tout langage régulier est hors-contexte (puisque toute grammaire linéaire à droite est hors-contexte) ; la réciproque est fausse.
- Tout langage hors-contexte ne contenant pas la chaîne vide est sous-contexte (en effet, tout langage hors-contexte ne contenant pas ϵ peut être engendré par une grammaire hors-contexte sans ϵ-règle. La réciproque est fausse : par exemple le langage $\{a^n b^n c^n\}$ est sous-contexte mais n'est pas hors-contexte.

Plan

Propriété fondamentale (décomposition des dérivations hors-contexte)

- La propriété 102 signifie que l'on peut composer des dérivations d'un système de réécriture ;
- la réciproque de cette propriété, qui est fausse pour les systèmes de réécriture généraux, est vraie pour les grammaires hors-contexte :
  - on peut décomposer une dérivation hors-contexte en sous-dérivations indépendantes.
  - Cette propriété fondamentale des dérivations hors-contexte est très souvent utilisée dans les démonstrations.

Propriété fondamentale (décomposition des dérivations hors-contexte)

Théorème 77

Soit $G = (V_T, V_N, S, R)$ une grammaire hors contexte, $V = V_T \cup V_N$. Soit une dérivation $u_1 u_2 \dots u_n \Rightarrow^p \omega$, avec $u_i \in V^*$ pour $i = 1, \dots, n$ et $\omega \in V^*$.
Pour tout $i = 1, \dots, n$, il existe $v_i \in V^*$ et $q_i \in \mathbb{N}$ tels que :

- $\omega = v_1 v_2 \dots v_n$,
- *pour tout* $i = 1, \dots, n$, $u_i \Rightarrow^{q_i} v_i$,
- $p = \sum_{i=1}^{i=n} q_i$.

Corollaire 78

Si $u_1 \dots u_n \Rightarrow^* \omega$ dans une grammaire hors-contexte, alors il existe $v_1, \dots, v_n$ tels que $\omega = v_1 \dots v_n$ et pour tout $i = 1, \dots n$, $u_i \Rightarrow^* v_i$.

Définition 79 (Dérivation gauche)

Soit $x_1, \ldots, x_k$ une dérivation dans une grammaire hors-contexte. On dit que cette dérivation est une *dérivation gauche* si pour chaque élément x_i de la dérivation, le non-terminal réécrit est le nonterminal le plus à gauche de x_i.
Pour une grammaire hors-contexte donnée, on définit la relation $\Rightarrow_g$ par : $x \Rightarrow_g y$ si et seulement si il existe une chaîne terminale u, un non-terminal A, une chaîne v et une règle $A \rightarrow \alpha$ tels que $x = uAv$ et $y = u\alpha v$.
Une dérivation $x_1, \ldots, x_k$ est une dérivation gauche si et seulement si $x_i \Rightarrow_g x_{i+1}$ pour $i = 1 \ldots k-1$.

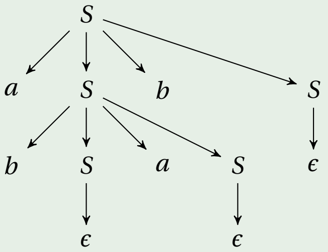
Soit la grammaire suivante avec $V_T = \{a, +, *, (,)\}$, $V_N = \{E, T, F\}$ et les règles :

$$\begin{array}{lcl} E & \rightarrow & E+T|T \\ T & \rightarrow & T*F|F \\ F & \rightarrow & (E)|a \end{array}$$

on a $E \Rightarrow_g E+T \Rightarrow_g T+T \Rightarrow_g F+T \Rightarrow_g a+T \Rightarrow_g a+F \Rightarrow_g a+a$

Arbre de dérivation

Soit la grammaire $S \to aSbS|bSaS|\epsilon$ et la dérivation gauche :
$S \Rightarrow_g aSbS \Rightarrow_g abSaSbS \Rightarrow_g abaSbS \Rightarrow_g ababS \Rightarrow_g abab$
L'arbre de dérivation est représenté par :



Définition 80

Soit $G = (V_T, V_N, S, R)$ une grammaire hors contexte. La grammaire G est *ambigüe* s'il existe une chaîne terminale admettant au moins deux arbres de dérivation de racine S, soit encore de façon équivalente, deux dérivations gauches distinctes d'origine S et d'extrémités identiques. Une chaîne terminale qui est l'extrémité d'au moins deux dérivations gauches d'origine S est une chaîne ambigüe de la grammaire G. Un langage hors-contexte est ambigu si *toutes* les grammaires hors-contextes qui l'engendrent sont ambigües.

Grammaires LL

Définition 81

Une grammaire hors-contexte G est une grammaire *LL(1)* Ssi pour tout mot $\omega \in \mathcal{L}(G)$ il existe une seule dérivation gauche de S vers ω telle que à chaque étape, le caractère courant dans ω suffit pour déterminer la règle de la grammaire à appliquer.

- En pratique, pour qu'une grammaire soit LL(1) il faut :
  - Eliminer l'ambiguité de la grammaire,
  - Eliminer la récursivité à gauche,
  - Factoriser à gauche.

Elimination de l'ambiguité

- Problème indécidable en général
- 2 propriétés pour "enlever" l'ambiguité
  - La précédence :
    - Si plusieurs opérateurs sont utilisés, il faut considérer la précédence sur les opérateurs. Les 3 caractéristiques importantes sont :
    - Le niveau de la règle de production dénote la priorité de l'opérateur utilisé
    - La production dans les niveaux élevés aura des opérateurs avec moins de priorité (dans l'arbre de dérivation)
    - La production dans les niveaux inférieurs aura des opérateurs avec plus de priorité (dans l'arbre de dérivation)
  - L'associativité :
    - Si plusieurs opérateurs avec la même précédence dans une production $\Rightarrow$ considérer l'associativité.
    - Si l'associativité est à gauche, alors il faut priviligier la récursivité à gauche dans la production : $+, -, *, /$ sont des opérateurs associatifs à gauche.
    - Si l'associativité est à droite, alors il faut priviligier la récursivité à droite dans la production : ^ est un opérateur associatif à droite.

Exemple 82

$E \to E - E | id$. Grammaire ambigüe : $id - id - id$? On la transforme en
$E \to E - P | P$, $P \to id$

Exemple 83

$E \to E + E | E * E | id$. Grammaire ambigüe : $id + id * id$. $*$ est plus prioritaire que $+$.

On transforme la grammaire en :

$$\begin{array}{lcll} E & \to & E+P & | \quad P \\ P & \to & P*Q & | \quad Q \\ Q & \to & id & \end{array}$$

Elimination de la récursivité à gauche (directe)

- Une grammaire G hors-contexte est récursive à gauche ssi il existe une règle $A \rightarrow A\alpha$ (récursivité à gauche directe) où plus généralement $A \Rightarrow B\beta$ telle que $B \Rightarrow A\gamma$.
- On remplace des règles de la forme $A \rightarrow A\alpha_1|A\alpha_2|\ldots|A\alpha_n|\beta_1|\ldots|\beta_m$ par :

$$\begin{array}{lcl} A & \rightarrow & \beta_1 T|\beta_2 T|\ldots|\beta_m T \\ T & \rightarrow & \alpha_1 T|\alpha_2 T|\ldots|\alpha_n T|\epsilon \end{array}$$

- Ce qui est équivalent à

$$\begin{array}{lcl} A & \rightarrow & \beta_1 T|\beta_2 T|\ldots|\beta_m T|\beta_1|\ldots|\beta_m \\ T & \rightarrow & \alpha_1 T|\alpha_2 T|\ldots|\alpha_n T \end{array}$$

Exemple 84

La grammaire $E \rightarrow E+E|I|N$ peut être remplacée par $\begin{array}{lcl} E & \rightarrow & IT|NT \\ T & \rightarrow & +ET|\epsilon \end{array}$

Elimination de la récursivité à gauche (directe)

- Une grammaire G hors-contexte est récursive à gauche ssi il existe une règle $A \rightarrow A\alpha$ (récursivité à gauche directe) où plus généralement $A \Rightarrow B\beta$ telle que $B \Rightarrow A\gamma$.
- On remplace des règles de la forme $A \rightarrow A\alpha_1|A\alpha_2|\ldots|A\alpha_n|\beta_1|\ldots|\beta_m$ par :
$$\begin{array}{lcl} A & \rightarrow & \beta_1 T|\beta_2 T|\ldots|\beta_m T \\ T & \rightarrow & \alpha_1 T|\alpha_2 T|\ldots|\alpha_n T|\epsilon \end{array}$$
- Ce qui est équivalent à $\begin{array}{lcl} A & \rightarrow & \beta_1 T|\beta_2 T|\ldots|\beta_m T|\beta_1|\ldots|\beta_m \\ T & \rightarrow & \alpha_1 T|\alpha_2 T|\ldots|\alpha_n T \end{array}$

Exemple 85

La grammaire $\begin{array}{lclcl} E & \rightarrow & E+T & | & T \\ T & \rightarrow & T*F & | & F \\ F & \rightarrow & (E) & | & a \end{array}$ peut être remplacée par

$$\begin{array}{lclcl} E & \rightarrow & TR & & \\ R & \rightarrow & +TR & | & \epsilon \\ T & \rightarrow & FU & & \\ U & \rightarrow & *FU & | & \epsilon \\ F & \rightarrow & (E) & | & a \end{array}$$

Elimination de la récursivité à gauche (directe)

- Parfois ce n'est pas suffisant pour éliminer la récursivité à gauche (indirecte)

Exemple 86

La grammaire $\begin{array}{lllllll} S & \rightarrow & Aa & \mid & b & & \\ A & \rightarrow & Ac & \mid & Sd & \mid & c \end{array}$ devient $\begin{array}{lllll} S & \rightarrow & Aa & \mid & b \\ A & \rightarrow & SdT & \mid & cT \\ T & \rightarrow & cT & \mid & \epsilon \end{array}$

mais on a $S \rightarrow Aa \rightarrow SdT$ donc $S \Rightarrow SdT$

Elimination de la récursivité à gauche (indirecte)

- Idée de l'algorithme : pour les paires de règles du type $A \rightarrow A'\delta$ et $A' \rightarrow A\eta$, on « anticipe » les dérivations problématiques : $A \rightarrow A\eta\delta$

Entrée : Grammaire hors-contexte G
Resultat : Grammaire hors-contexte G' sans récursivité à gauche telle que $\mathscr{L}(G) = \mathscr{L}(G')$

```
procedure eliminerRecGauche(G)
Soit (A_1 ... A_n) la liste ordonnée des éléments de V_N
for i = 1 to n do
    for j = 1 to i − 1 do
        if A_i → A_j α existe then
            la remplacer par : A_i → δ_1 α | δ_2 α | ... | δ_h α
            où A_j → δ_1 | δ_2 | ... | δ_h
    Eliminer la récursivité directe des A_i-productions.
```

Algorithme 1 : Suppression de toutes les récursivités

Elimination de la récursivité à gauche (directe)

Exemple 87

La grammaire
$$\begin{array}{lllllll} S & \rightarrow & Aa & \mid & b & & \\ A & \rightarrow & Ac & \mid & Sd & \mid & c \end{array}$$

Ordre des non terminaux (arbitraire) : $\{A_1 = S, A_2 = A\}$

- $i = 1$: $\emptyset$
- $i = 2$; $j = 1$:
  - La règle $A \rightarrow Sd$ doit être traitée ($A = A_2$ et $S = A_1$)
  - On la remplace par $A \rightarrow Aad|bd$
  - On supprime les récursivités à gauche : $A \rightarrow Ac|Aad|bd|c$ devient
    $$\begin{array}{lllllll} A & \rightarrow & bdT & \mid & cT & & \\ T & \rightarrow & cT & \mid & adT & \mid & \epsilon \end{array}$$
  - Soit le résultat : $$\begin{array}{lllllll} S & \rightarrow & Aa & \mid & b & & \\ A & \rightarrow & bdT & \mid & cT & & \\ T & \rightarrow & cT & \mid & adT & \mid & \epsilon \end{array}$$

- On remplace des règles de la forme $A \to \alpha\beta_1|\alpha\beta_2|\ldots|\alpha\beta_n$ par :

$$\begin{array}{lcl} A & \to & \alpha T \\ T & \to & \beta_1|\beta_1|\ldots|\beta_n \end{array}$$

Exemple 88

La grammaire $\begin{array}{lcl} P & \to & E \\ E & \to & id \\ E & \to & id[E] \\ E & \to & id(E) \end{array}$ peut être remplacée par $\begin{array}{lcl} P & \to & E \\ E & \to & idT \\ T & \to & \epsilon \\ T & \to & [E] \\ T & \to & (E) \end{array}$

Les ensembles premiers et suivants

- De manière informelle, PREMIERS(α) désigne l'ensemble des terminaux (y compris ϵ) qui peuvent éventuellement apparaitre au début de toute derivation de α.
- SUIVANTS(A) désigne l'ensemble des terminaux (y compris \$) qui peuvent éventuellement apparaitre après toute derivation du symbole non terminal A.
- Un analyseur lexical LL(1) "connaît" exactement à chaque symbole lu quelle règle appliquer grâce à ces deux ensembles.

Calcul de l'ensemble premiers

- Pour les terminaux :

for *chaque terminal* $a \in V_T$ **do**
  premiers(a) $= \{a\}$

- Pour les non terminaux :

repeat
  for *chaque regle* $X \to Y_1 Y_2 \dots Y_n \in G$ **do**
    if $a \in$ premiers(Y_1) **then**
      premiers(X) = premiers(X) $\cup \{a\}$
    if $a \in$ premiers(Y_k) **and** $Y_1 \dots Y_{k-1} \Rightarrow \epsilon$ **then**
      premiers(X) = premiers(X) $\cup \{a\}$
    if $Y_1 \dots Y_n \Rightarrow \epsilon$ **then**
      premiers(X) = premiers(X) $\cup \{\epsilon\}$
until *aucun changement n'est plus possible*;

Calcul de l'ensemble premiers

- Pour les mots :

premiers(ϵ) = $\emptyset$
for *chaque symbole* $Y_1 Y_2 \ldots Y_k$ *dans un mot* α **do**
    if $a \in$ premiers(Y_1) **then**
        premiers(α) = premiers(α) $\cup \{a\}$
    if $a \in$ premiers(Y_k) **and** $Y_1 \ldots Y_{k-1} \Rightarrow \epsilon$ **then**
        premiers(α) = premiers(α) $\cup \{a\}$
    if $Y_1 \ldots Y_n \Rightarrow \epsilon$ **then**
        premiers(α) = premiers(α) $\cup \{\epsilon\}$

- Pour les mots :

```
suivants(S) = {$}
repeat
    if A → αBβ then
        suivants(B) = (premiers(β) − ϵ) ∪ suivants(B)
    if A → αB or ϵ ∈ premiers(β) then
        suivants(B) = suivants(A) ∪ suivants(B)
until aucun changement n'est plus possible;
```

Définition 89

directeur : $R \to (V_T \cup \{\$\})$ avec
directeur$(A \to \omega) = \left\{x \in V_T \cup \{\$\} \,|\, \exists \omega_1, \omega_2, \omega_3 S\$ \Rightarrow^* \omega_1 A \omega_2 \Rightarrow^* \omega_1 \omega \omega_2 \Rightarrow^* \omega_1 x \omega_3\right\}$.

Définition 90

Une grammaire $G = (V_T, V_N, S, R)$ est LL(1) ssi elle vérifie la condition :

$$\forall A \to \omega_1 \forall A \to \omega_2$$
$$\omega_1 \neq \omega_2 \Rightarrow \text{directeur}(A \to \omega_1) \cap \text{directeur}(A \to \omega_2) = \emptyset$$

Définition 91

$$\text{directeur}(A \to \omega)$$
$$=$$
$$\text{premiers}(\omega) \cup (\text{si } \omega \Rightarrow^* \epsilon \text{ alors suivants(A) sinon } \emptyset)$$

Analyseur LL(1)

- Si G est LL(1) et si on veut montrer $\omega \in \mathscr{L}(G)$, On commence de la configuration $(S\$,\omega)$ et on doit atteindre (ϵ,ϵ).
- 2 opérations :
  - effacement : $(x.\omega, x.s) \rightarrow (\omega, s)$,
  - expansion : $(A\omega, x.s) \rightarrow (B\omega, x.s)$ où $A \Rightarrow B \in R$ telle que directeur$(A \Rightarrow B) = x$ (LL(1) garantit un choix au plus)

Principe de l'algorithme de parsing :

- On écrit une procédure par non terminal :
  - l'appel A a pour effet de reconnaitre $\mathscr{L}(A)$
- Soit $\omega \in (V_T \cup V_N)^*$, La procédure produire génère le code reconnaissant (ω)
  - produire(ϵ) = null ; (on ne fait rien)
  - produire($x\omega$) = si mot $\neq x$ alors soulever erreur ; sinon mot=lire_mot() ; produire(ω) ; ($x \in V_T$)
- procedure ($A\omega$) = $A()$; produire(ω) ;

- Ecriture des procédures : Soit A tel que $A \rightarrow \omega_1 | \ldots | \omega_n$.

```
procedure A
Switch mot :
    Case = directeur(A → ω1) :
        produire(ω1)
    Case ... :
        ...
    Case = directeur(A → ωn) :
        produire(ωn)
    Other :
        soulever erreur
```

```
procedure reconnaitre
mot : Token;
mot = lire_mot;
S(); /* appel de la procédure associée à l'axiome */
if mot ≠ $ then
    soulever erreur
```

Exemple

$$\begin{array}{llll} E & \rightarrow & numT & num \\ T & \rightarrow & EopT & num \\ & \rightarrow & \epsilon & \{+,-,*,\$\} \\ op & \rightarrow & +|-|* & \{+,-,*,\$\} \end{array}$$

```
procedure E
Switch mot :
    Case = num :
        T();
    Other :
        soulever erreur
```

```
procedure T
Switch mot :
    Case = num :
        E(); op(); T();
    Case +|-|*|$ :
        ;
    Other :
        soulever erreur
```

```
procedure op
Switch mot :
    Case +|-|*|$ :
        mot = lire_mot;
    Other :
        soulever erreur
```

```
procedure reconnaitre
mot : Token;
mot = lire_mot;
E();
if mot ≠ $ then
    soulever erreur
```

Conclusion sur l'analyse descendante

- Simple et facile à mettre en oeuvre
- Bien adapté au rattrapage d'erreurs
  - Repositionnement sur les caractères attendus.
- Bien adapté à l'évaluation
- Assez restrictif sur la manière de décrire les langages
  - Ex : les expressions postfixées.
- On aurait préféré utiliser la définition $E \rightarrow E\,E\,op$