Automates, Grammaires et Systèmes de Transitions

Matthieu Boyer

Cours TalENS 5 2025-2026

Table des matières

1 Langages, et Générateurs

Dans la suite, on considère un alphabet noté Σ. Il s'agit juste d'un ensemble fini quelconques, dont les éléments sont appelés lettres.

1.1 Langages réguliers

Définition 1.1 Le monoïde libre Σ^* sur l'alphabet Σ est l'ensemble des suites finies d'éléments de Σ. La suite vide est une suite finie notée ε.

Les éléments du monoïde libre sur Σ sont appelés des mots sur Σ. Un langage sur Σ est un ensemble de mots sur Σ, c'est-à-dire une partie de Σ^*.

Définition 1.2 Pour $u \in \Sigma^*$, on note $u_0 \dots u_{n-1}$ ses lettres et $|u| = n$ sa longueur.

On définit la concaténation de deux mots u, u' comme $u \cdot u' = u_0 \dots u_{n-1} u'_0 \dots u'_{n'-1}$.

On écrit notamment $u^k = \underbrace{u \cdot u \cdot \dots \cdot u}_{k \text{ fois}}$ et $u^0 = \varepsilon$.

Définition 1.3 Un monoïde est un ensemble M muni d'une application binaire $+ : M \times M \to M$ ayant un neutre $0 \in M$ tel que $0 + m = m + 0 = m$ pour tout $m \in M$. Un (homo)morphisme de monoïdes est une application $M_1 \to M_2$ préservant le neutre et l'opération.

Théorème 1.4 $|\cdot|$ est un morphisme de monoïdes de $(\Sigma^*, \cdot)$ dans $(\mathbb{N}, +)$.

Démonstration. C'est la définition : $|u \cdot u'| = |u| + |u'|$ et $|\varepsilon| = 0$. ∎

Théorème 1.5 Si $h : \Gamma \to \Sigma$ est une application, h engendre un homomorphisme de monoïdes entre Γ^* et Σ^*.

Démonstration. On définit simplement $h(\varepsilon) = \varepsilon$ et $h(xy) = h(x)h(y)$ pour tous mots x et y. ■

Définition 1.6 Si L_1, L_2 sont deux langages, on définit

$$L_1 \cdot L_2 = \{u_1 \cdot u_2 \,|\, u_1 \in L_1, u_2 \in L_2\},$$

la concaténation de L_1 et L_2. En particulier, on notera $L_1^k = \underbrace{L_1 \cdot L_1 \cdot \cdots \cdot L_1}_{k \text{ fois}}$.

Théorème 1.7 On a l'égalité

$$\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k.$$

Démonstration. Si u est un mot sur Σ, alors $u \in \Sigma^{|u|}$ et $|u| \in \mathbb{N}$. ■

On dit que Σ^* est l'étoile de Kleene de Σ. On définit de même L^* l'étoile de Kleene de L, qui correspond au monoïde libre sur L.

Définition 1.8 L'ensemble des langages rationnels [a] est le plus petit ensemble tel que

- les langages finis sont rationnels ;
- la concaténation de deux langages rationnels est rationnelle ;
- l'union de deux langages rationnels est rationnelle ;
- l'étoile de Kleene de deux langages rationnels est rationnelle.

a. Parfois appelés langages réguliers

Le langage $\{a^{2n} \mid n \in \mathbb{N}\}$ est rationnel.

Théorème 1.9 Il existe des langages non rationnels.

Démonstration. L'ensemble Σ^* est dénombrable, donc l'ensemble $\mathcal{P}(\Sigma^*)$ des langages sur Σ est indénombrable. Puisque l'ensemble des langages régulières est dénombrable, il existe des langages non réguliers. ■

Par exemple, le langage Dyck-1 $\{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas rationnel.

Définition 1.10 Une expression régulière est construite par induction par la grammaire suivante

- les lettres de Σ sont des expressions régulières ;
- si e_1, e_2 sont des expressions régulières, $e_1 + e_2$ et $e_1 \cdot e_2$ sont des expressions régulières ;
- si e est une expression régulière, e^* est une expression régulière.

Le langage $\mathcal{L}(e)$ reconnu par une expression régulière e est défini inductivement par :

- $\mathcal{L}(a) = \{a\}$ pour $a \in \Sigma$;
- $\mathcal{L}(e_1 + e_2) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$;
- $\mathcal{L}(e_1 \cdot e_2) = \mathcal{L}(e_1) \cdot \mathcal{L}(e_2)$;
- $\mathcal{L}(e^*) = \mathcal{L}(e)^*$.

Théorème 1.11 Les langages rationnels sont exactement les langages reconnus par des expressions régulières.

Démonstration. Direct par la définition, on prouve d'abord que le langage reconnu par une expression régulière est rationnel, puis on se donne un langage rationnel et une séquence d'étapes permettant de le construire à partir d'ensembles finis et on construit une expression régulière le reconnaissant. ■

Théorème 1.12 Vérifier l'appartenance d'un mot à un langage régulier est un problème de décision en temps linéaire.

Démonstration. On utilise pour cela des automates, qui sont une représentation graphique du processus de transition. On ne rentrera pas dans les détails, mais un automate peut être vu comme un graphe dont les arêtes sont étiquetées par des lettres de Σ, et on appelle calcul d'un mot sur l'automate un parcours du graphe partant d'un état initial et suivant les lettres du mot. ■

1.2 xkcd.com/1090

Définition 1.13 Une grammaire est un quadruplet (N, Σ, P, S) où

- N est un alphabet de symboles dits "non-terminaux" ;
- Σ est un alphabet de symboles dits "terminaux" ;
- P est un ensemble fini de règles de production de la forme $(N \cup \Sigma)^* N (N \cup \Sigma)^* \to (N \cup \Sigma)^*$;
- $S \in N$ est un symbole non-terminal de base.

Une grammaire est dite hors-contexte lorsque les règles dans P ne sont que de la forme $N \to (\Sigma \cup N)^*$.

Définition 1.14 Un alphabet est dit couplé s'il s'écrit $T \cup \bar{T}$ où $|T| = \left|\bar{T}\right|$. On pensera T comme étant des parenthèses ouvrantes et $\bar{T}$ des parenthèses fermantes.

Définition 1.15 Le langage reconnu par une grammaire est défini inductivement par l'application finie de règles :

- Pour deux mots $u, v \in (N \cup \Sigma)^*$, u se dérive v (noté $u \to v$) s'il y a une règle $\alpha \to \beta$ dans P (avec $\alpha \in N$) et $u_1, u_2 \in (N \cup \Sigma)^*$ tels que $u = u_1 \alpha u_2$ et $v = u_1 \beta u_2$.
- Pour deux mots $u, v \in (N \cup \Sigma)^*$, u dérive v (noté $u \xrightarrow{*} v$) quand u peut se dériver en v après un nombre fini d'étapes [a]
- Le langage engendré par G est $\mathcal{L}(G) = \left\{ u \in \Sigma^* \,\middle|\, S \xrightarrow{*} u \right\}$.

Si L est engedré par une grammaire hors-contexte, on dit que L est algébrique ou hors-contexte.

a. On dit que $\xrightarrow{*}$ est la clôture réflexive et transitive de $\to$.

Le langage Dyck-1 est algébrique et engendré par $S \to aSa$.

Théorème 1.16 — Hiérarchie de Chomsky — Non-complétude du Type 2 Il existe des langages non algébriques.

Démonstration. La preuve est la même que pour les langages réguliers. ■

Par exemple, le langage $\{a^i b^i c^i \mid i \geq 0\}$ n'est pas algébrique.

Théorème 1.17 — Hiérarchie de Chomsky — Inclusion du Type 3 dans le Type 2 Les langages réguliers sont algébriques.

Démonstration. Il suffit de voir que les grammaires dont les règles de production contiennent au plus un non-terminal qui se trouve toujours au début ou à la fin de la règle engendrent exactement les langages réguliers. ■

Définition 1.18 Un arbre de dérivation est une représentation visuelle des différentes étapes de la dérivation d'un mot par une grammaire.

Théorème 1.19 Vérifier l'appartenance d'un mot au langage reconnu par une grammaire hors-contexte est un problème de décision en temps polynomial cubique (et linéaire en la taille de la grammaire).

Démonstration. L'algorithme CYK (Cocke-Younger-Kasami) résout ce problème en temps $\mathcal{O}(|u|^3 |G|)$. ■

Théorème 1.20 Les langages algébriques sont clos par concaténation, union, intersection avec un langage régulier.

1.2.1 Bestiaire

On donne ici quelques exemples de grammaires formelles.

Expressions Arithmétiques

Commençons déjà par le langage d^+ des nombres en base 10. C'est un langage régulier, reconnu par l'expression régulière $([0-9]^*[1-9])+0$.

On a ensuite une grammaire hors-contexte avec les règles $S \to S+S$, $S \to S \times S$, $S \to S = S$, $S \to d^+$.

Langages de Dyck

Le k-ème langage de Dyck (Dyck-k) est donné le langage des mots bien parenthésés avec k types de parenthèses. C'est un langage algébrique dont une grammaire est donnée par $S \to \varepsilon$ et $S \to ({}_i S)_i$ pour tout $i \in [\![1, k]\!]$.

Autrement dit, on peut définir le langage D_T pour tout alphabet couplé $T \cup \bar{T}$ par $S \to \varepsilon$ et pour des paires $t, \bar{t} \in T \times \bar{T}$, $S \to tS\bar{t}$.

Ce sont les langages algébriques par excellence :

Théorème 1.21 — Théorème de représentation de Chomsky-Schützenberger Un langage L sur Σ est algébrique si et seulement si il existe

1. un alphabet couplet $T \cup \bar{T}$;
2. un langage rationnel R sur $T \cup \bar{T}$;
3. un morphisme de monoïde h de $\left(T \cup \bar{T}\right)^*$ dans Σ^* ;

tels que $L = h(D_T \cap R)$.

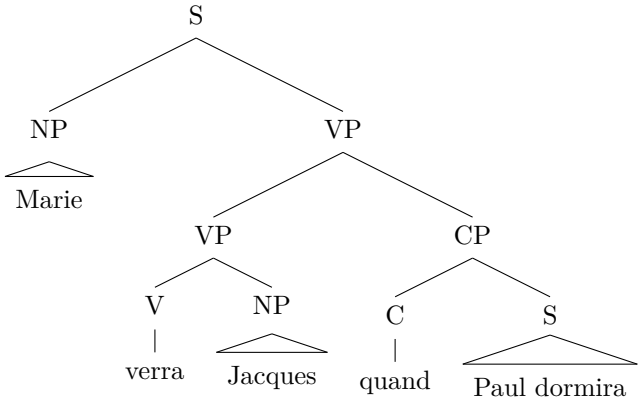
Démonstration. Il suffit de prouver l'existence de tels objets pour un langage algébrique L, l'inclusion réciproque étant directe par les propriétés de cloture des langages algébriques. Soit $G = (V, \Sigma, P, S)$ une grammaire sur Σ. Posons $T = V \cup \Sigma$. On définit h un morphisme de $(T \cup \bar{T})^*$ dans Σ^* par

$$\psi(X) = \begin{cases} 1 & \text{si } X \in V \cup \bar{T} \\ X & \text{si } X \in \Sigma \end{cases}$$

On peut ensuite construire R par le théorème de Shamir, mais on ne rentrera pas dans les détails. Voir par exemple `http://www-igm.univ-mlv.fr/~berstel/Articles/1997CFLPDA.pdf` pour plus de détails. ■

Fragment de Français

S(entence)	→N(oun) P(hrase) + V(erb) P(hrase)	
NP	→	Proper Name
NP	→	Det + N + (Adj)
VP	→	V_{intr}
VP	→	V_{tr} + NP
C(omplementizer) P(hrase)	→	*que* + S
VP	→	$V_{cl(ausal)}$ + CP
V_{tr}	→	voir, entendre,…
V_{intr}	→	dormir, briller,…
V_{cl}	→	penser, croire,…
C_{temp}	→	avant que, quand
CP_{temp}	→	C_{temp} + S
S	→	NP + VP + (CP_{temp})
VP	→	V_{intr} + (CP_{temp})
VP	→	V_{tr} + NP + (CP_{temp})
S	→	S + CP_{temp}
VP	→	VP + CP_{temp}



Syntaxe de l'Anglais

```
CP   ::= DP, VP
       | Cmp, CP
       | CP, CBar

CBar ::= Cor, CP
Dbar ::= Cor, DP
DP   ::= DP, Dbar
       | Dmp, DP
       | Det, NP
       | Gen, TN

Gen  ::= DP, GenD
```

```
NP   ::= AdjP, NP
       | NP, AdjP

AdjP ::= TAdj, DP
       | Deg, AdjP
VP   ::= TV, DP
       | AV, CP
       | VP, AdvP

TV   ::= DV, DP
AdvP ::= TAdv, DP
```

Syntaxe d'un fragment de Pyret

Pyret est un langage de programmation destiné à l'éducation, dont on peut trouver une grammaire [1] exemple ci-dessous. Sans rentrer dans les détails, l'intérêt d'une telle grammaire est l'efficacité de traitement d'un programme dans le langage par l'ordinateur.

1. Cette grammaire a été écrite par Jean-Christophe Filliâtre pour le projet de son (excellent) cours de Compilation des Langages de Programmation, sur l'année 2025-2026.

⟨*file*⟩ ::= ⟨*stmt*⟩* EOF

⟨*block*⟩ ::= ⟨*stmt*⟩+

⟨*stmt*⟩ ::= `fun` ⟨*ident*⟩ (⟨*ident*⟩+,)? ⟨*funbody*⟩
| `var`? ⟨*ident*⟩ (:: ⟨*type*⟩)? = ⟨*bexpr*⟩
| ⟨*ident*⟩ := ⟨*bexpr*⟩
| ⟨*bexpr*⟩

⟨*funbody*⟩ ::= (⟨*param*⟩*,) ⟨*rtype*⟩ ⟨*ublock*⟩ ⟨*block*⟩ `end`

⟨*param*⟩ ::= ⟨*ident*⟩ :: ⟨*type*⟩

⟨*rtype*⟩ ::= → ⟨*type*⟩

⟨*ublock*⟩ ::= `:`
| `block :`

⟨*type*⟩ ::= ⟨*ident*⟩ (⟨*type*⟩+)?
| (⟨*type*⟩*, ⟨*rtype*⟩)

⟨*bexpr*⟩ ::= ⟨*expr*⟩ (⟨*binop*⟩ ⟨*expr*⟩)*

⟨*expr*⟩ ::= `true`
| `false`
| ⟨*integer*⟩
| ⟨*string*⟩
| ⟨*ident*⟩
| (⟨*bexpr*⟩)
| `block :` ⟨*block*⟩ `end`
| `if` ⟨*bexpr*⟩ ⟨*ublock*⟩ ⟨*block*⟩ (`else if` ⟨*bexpr*⟩ `:` ⟨*block*⟩)* (`else :` ⟨*block*⟩)? `end`
| ⟨*caller*⟩ (⟨*bexpr*⟩*,)
| `lam` ⟨*funbody*⟩
| `cases` (⟨*type*⟩) ⟨*bexpr*⟩ ⟨*ublock*⟩ ⟨*branch*⟩* `end`
| `for` ⟨*caller*⟩ (⟨*from*⟩*,) ⟨*rtype*⟩ ⟨*ublock*⟩ ⟨*block*⟩ `end`

⟨*caller*⟩ ::= ⟨*ident*⟩
| ⟨*caller*⟩ (⟨*bexpr*⟩*,)

⟨*branch*⟩ ::= ⟨*ident*⟩ ((⟨*ident*⟩*,))? ⇒ ⟨*block*⟩

⟨*from*⟩ ::= ⟨*param*⟩ `from` ⟨*bexpr*⟩

⟨*binop*⟩ ::= == | <> | < | <= | > | >= | + | − | * | / | `and` | `or`

2 Séries et Langages

Dans le cours précédent, nous avons étudié les séries numériques, et avons parlé de la notion de série génératrice d'une suite.

Définition 2.1 La série génératrice associée à la suite de terme général u_n est la suite de terme général $u_n x^n$ pour x un réel, sur son domaine de convergence.

Définition 2.2 La série génératrice d'un langage $\mathcal{L}$ sur Σ est la série génératrice associée à la suite de terme général $\ell_n = |\mathcal{L} \cap \Sigma^n|$.

Ici, ℓ_n est donc le nombre de mots de $\mathcal{L}$ de longueur n.

Démonstration. Cette série converge pour tout $x \in \mathcal{D}(0, \frac{1}{|\Sigma|^n})$. En effet, pour x dans ce disque, la suite $\ell_n x^n$ est strictement bornée par $\frac{|\Sigma^n|}{|\Sigma^n|}$ et on a convergence par le critère de Riemann puis le théorème d'Abel. ■

Théorème 2.3 Soit $\mathcal{L}$ un langage sur Σ, et $f_{\mathcal{L}}$ la fonction qui à x associe $\sum_{n=0}^{\infty} |\mathcal{L} \cap \Sigma^n| \, x^n$. Alors,

- Si $\mathcal{L}$ est rationnel, $f_{\mathcal{L}}$ est une fraction rationnelle [a] ;
- Si $\mathcal{L}$ est algébrique et non ambigu, $f_{\mathcal{L}}$ est une fonction algébrique [b].

a. De la forme $P(x)/Q(x)$ pour P et Q deux polynômes.
b. On ajoute les exposants fractionnels

Démonstration. Pour la partie langages rationnels, on peut procéder par induction, en vérifiant pour $\mathcal{L}$ un langage fini que $f_{\mathcal{L}}$ est un polynôme, puis en étudiant le cas où $\mathcal{L}$ est infini selon les règles pouvant définir un langage rationnel – l'union étant la somme, l'intersection la différence, la concaténation étant le produit et l'étoile de Kleene étant la transformation $u \mapsto 1/(1-u)$ (qui découle de la série de Taylor pour cette fonction). On renvoie vers `https://www.eleves.ens.fr/home/mpboyer/Recherche/Misc/gen_series_languages.pdf` pour une preuve plus détaillée.

Pour la partie langage algébriques, la preuve est légèrement hors du but de ce cours, mais on peut renvoyer vers `https://www.sciencedirect.com/science/article/pii/0304397587900119` pour une preuve détaillée du résultat. ■