Théorie des langages : THL

CM 3

Uli Fahrenberg

EPITA Rennes

S5 2024

Aperçu

Programme du cours

1. Langages rationnels
2. Automates finis
3. Langages algébriques, grammaires hors-contexte
4. Automates à pile
5. Parsage LL
6. Parsage LR
7. `flex` & `bison`

La dernière fois : définitions & algorithmes

- automates finis déterministes complets
- automates finis déterministes
  - complétion par ajout d'un état puits
- automates finis (non-déterministes)
- automates finis avec transitions spontanées
  - suppression des trans. spont. par ε-fermeture arrière
- déterminisation par l'automate des parties
- algo. de Thompson : exp. rat. $\rightsquigarrow$ aut. fini avec trans. spont.
- algo. de Brzozowski-McCluskey : aut. fini $\rightsquigarrow$ exp. rat.

La dernière fois : théorème de Kleene

Théorème (Kleene)

Un langage $L \subseteq \Sigma^*$ est rationnel ssi il est reconnaissable.

syntaxe		sémantique
aut. finis dét. complets		langages reconnaissables
$\subsetneq$		$\parallel$
aut. finis déterministes		langages reconnaissables
$\subsetneq$		$\parallel$
automates finis	$\xrightarrow{L(\cdot)}$	langages reconnaissables
$\subsetneq$		$\parallel$
aut. finis à trans. spontanées		langages reconnaissables
		$\parallel$
expressions rationnelles		langages rationnelles

Dans le poly

La dernière fois :

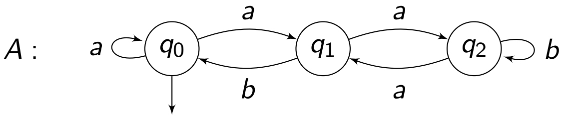
- chapitre 4, moins 4.1.3, 4.2.1, 4.3, 4.4

Aujourd'hui :

- chapitres 5 et 6 (rapidement, avec trous)
- section 9.2.2 (sans démonstration)
- automates à pile (pas dans le poly)

Grammaires régulières et hors-contexte

Automates finis : une autre vue



- soit L_i le langage reconnu par A avec état initial q_i
- alors
$$\begin{aligned} L_0 &= \{a\}L_0 \cup \{a\}L_1 \cup \{\varepsilon\} \\ L_1 &= \{a\}L_2 \cup \{b\}L_0 \\ L_2 &= \{a\}L_1 \cup \{b\}L_2 \end{aligned}$$

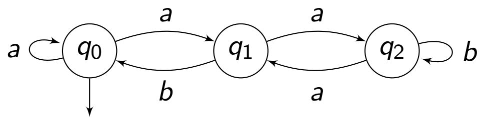
Slogan

Un automate fini est un système des équations linéaires à droite.

- voici pourquoi « langages rationnels »

Automates finis : une autre vue

A :



$$L_0 = \{a\}L_0 \cup \{a\}L_1 \cup \{\varepsilon\} \qquad X_0 \to aX_0 \mid aX_1 \mid \varepsilon$$

$$L_1 = \{a\}L_2 \cup \{b\}L_0 \qquad X_1 \to aX_2 \mid bX_0$$

$$L_2 = \{a\}L_1 \cup \{b\}L_2 \qquad X_2 \to aX_1 \mid bX_2$$

Définition (5.18 variant)

Une grammaire régulière est une structure (N, Σ, P, X_0) où

- Σ est un ensemble fini de terminaux,
- N est un ensemble fini de variables,
- $X_0 \in N$ est le symbole initial et
- $P \subseteq N \times \Sigma N \cup N \times \Sigma \cup N \times \{\varepsilon\}$ est l'ensemble de productions.

Grammaires régulières

Définition (re)

Une grammaire régulière est une structure (N, Σ, P, S) où

- Σ est un ensemble fini de terminaux,
- N est un ensemble fini de variables, avec symbole initial $S \in N$
- $P \subseteq N \times \Sigma N \cup N \times \Sigma \cup N \times \{\varepsilon\}$ est l'ensemble de productions.

- aussi « grammaire linéaire à droite »
- (alors c'est quoi une grammaire linéaire à gauche ?)
- ! grammaires linéaires (mixtes) $\longleftarrow$ pas la même chose !

Définition (5.19)

Un langage est régulier si il est *engendré* par une grammaire régulière.

Théorème (5.20)

Un langage est régulier ssi il est rationnel.

Comment ça marche

C'est quoi « engendré »

Une grammaire régulière : $G = (N, \Sigma, P, S)$:

- N, Σ ensembles finis, $S \in N$,
- $P \subseteq N \times \Sigma N \cup N \times \Sigma \cup N \times \{\varepsilon\}$: l'ensemble de productions

Soit $V = N \cup \Sigma$.

- pour $\alpha, \beta \in V^*$ on note $\alpha \to \beta$ si $(\alpha, \beta) \in P$
- pour $\alpha, \beta, \gamma, \delta \in V^*$ on note $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ si $\alpha \to \beta$

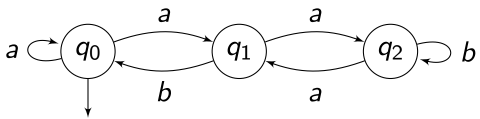
Définition (5.3, 5.4)

- Une dérivation dans G est une séquence $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$.
- On note $\alpha \Rightarrow^* \beta$ si il existe une dérivation $\alpha \Rightarrow \cdots \Rightarrow \beta$.
- Le langage engendré par G est $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

- une sorte de réécriture

Exemple

A :



$$X_0 \to aX_0 \mid aX_1 \mid \varepsilon$$
$$X_1 \to aX_2 \mid bX_0$$
$$X_2 \to aX_1 \mid bX_2$$

- $X_0 \Rightarrow \varepsilon$
- $X_0 \Rightarrow aX_0 \Rightarrow a$
- $X_0 \Rightarrow aX_1 \Rightarrow abX_0 \Rightarrow ab$
- $X_0 \Rightarrow aX_1 \Rightarrow aaX_2 aabX_2 \Rightarrow aabaX_1 \Rightarrow aababX_0 \Rightarrow aabab$
- etc.

Grammaires hors-contexte

- le langage $\{a^n b^n \mid n \geq 0\}$ n'est pas rationnel
- mais bien engendré par une grammaire : $S \to aSb \mid \varepsilon$
- (qui n'est donc pas régulière)

Définition (5.15 variant)

Une grammaire hors-contexte est une structure (N, Σ, P, S) où

- Σ est un ensemble fini de terminaux,
- N est un ensemble fini de variables,
- $S \in N$ est le symbole initial et
- $P \subseteq N \times (N \cup \Sigma)^*$ est l'ensemble de productions.

- aussi « grammaire algébrique »

Comment ça marche

Une grammaire hors-contexte : $G = (N, \Sigma, P, S)$:

- N, Σ ensembles finis, $S \in N$,
- $P \subseteq N \times (N \cup \Sigma)^*$: l'ensemble de productions

Soit $V = N \cup \Sigma$.

- pour $\alpha, \beta \in V^*$ on note $\alpha \to \beta$ si $(\alpha, \beta) \in P$
- pour $\alpha, \beta, \gamma, \delta \in V^*$ on note $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ si $\alpha \to \beta$

Définition (5.3, 5.4)

- Une dérivation dans G est une séquence $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$.
- On note $\alpha \Rightarrow^* \beta$ si il existe une dérivation $\alpha \Rightarrow \cdots \Rightarrow \beta$.
- Le langage engendré par G est $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

- même chose qu'avant !

5 minutes de réflexion

Pour chaque grammaire hors-contexte G si-dessous :

a. G est-elle régulière ?
b. Décrire $L(G)$.

1. $S \to aS \mid b$
2. $S \to 0S1 \mid \#$
3. $S \to Tb\,;\ T \to Ta \mid b$
4. $S \to abSc \mid A\,;\ A \to cAd \mid cd$
5. $S \to (S) \mid SS \mid \varepsilon \qquad (\ \Sigma = \{(,)\}\,!\ )$
6. $S \to S{+}T \mid T\,;\ T \to T{*}F \mid F\,;\ F \to (S) \mid a$

Grammaires syntagmatiques

Un peu de linguistique

Une langue :

- vocabulaire
- grammaire

$\Rightarrow$ phrases valides

Exemple :

`<sentence>` $\to$ `<subject>` `<verb>`

`<subject>` $\to$ he | she

`<verb>` $\to$ sleeps | drinks | drives

Phrases valides :

- he sleeps, she sleeps, he drinks etc.

Un peu de linguistique

Une langue :

- vocabulaire
- grammaire

⇒ phrases valides

Exemple :

<sentence> → <subject> <verb>

<subject> → he | she

<verb> → sleeps | drinks | drives

variables

terminaux

Phrases valides :

- he sleeps, she sleeps, he drinks etc.

Grammaires

Définition (5.1)

Une grammaire syntagmatique est une structure (N, Σ, P, S) où

- Σ est un ensemble fini de terminaux,
- N est un ensemble fini de variables,
- $S \in N$ est le symbole initial et
- $P \subseteq (N \cup \Sigma)^+ \times (N \cup \Sigma)^*$ est l'ensemble de productions.

Soit $V = N \cup \Sigma$. On note $\alpha \to \beta$ si $(\alpha, \beta) \in P$.

- pour $\alpha, \beta, \gamma, \delta \in V^*$ on note $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ si $\alpha \to \beta$

Définition (5.3, 5.4)

- Une dérivation dans G est une séquence $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$.
- On note $\alpha \Rightarrow^* \beta$ si il existe une dérivation $\alpha \Rightarrow \cdots \Rightarrow \beta$.
- Le langage engendré par G est $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

Exemple

Une grammaire G_1 :

$$S \to \texttt{x} \mid LE$$
$$L \to \texttt{x} \mid L\texttt{,\_x}$$
$$\texttt{,\_x}E \to \texttt{\_and\_x}$$

Quelques dérivations :

- $S \Rightarrow \texttt{x}$
- $S \Rightarrow LE \Rightarrow \texttt{x}E \Rightarrow\ ?$
- $S \Rightarrow LE \Rightarrow L\texttt{,\_x}E \Rightarrow \texttt{x,\_x}E \Rightarrow \texttt{x\_and\_x}$
- $S \Rightarrow LE \Rightarrow L\texttt{,\_x}E \Rightarrow L\texttt{,\_x,\_x}E \Rightarrow \texttt{x,\_x,\_x}E \Rightarrow \texttt{x,\_x\_and\_x}$

- Oxford comma !

Problèmes

Théorème

Un langage est engendré par une grammaire ssi il est récursivement énumerable.

Corollaire (sans démonstration ici)

Il n'existe pas d'algorithme qui, en prenant une grammaire G et un mot w comme entrée, décide si $w \in L(G)$.

- la notion est trop libérale, on ne peut rien faire avec

$\Rightarrow$ il nous faut des restrictions

Grammaires monotones

$$G_1: \quad S \to \mathtt{x} \mid LE$$
$$L \to \mathtt{x} \mid L\mathtt{,\_x}$$
$$\mathtt{,\_x}E \to \mathtt{\_and\_x}$$

Définition (5.8)

Une grammaire est monotone si pour chaque production $\alpha \to \beta$, $|\alpha| \leq |\beta|$.

- G_1 est monotone
- il existe un algorithme qui, en prenant une grammaire *monotone* G et un mot w comme entrée, décide si $w \in L(G)$.
- (*pourquoi ? comment ?*)

Théorème (sans démonstration ici)

Il n'existe pas d'algorithme qui, en prenant une grammaire monotone G comme entrée, décide si $L(G) = \emptyset$.

- toujours trop libéral

Grammaires contextuelles

Définition (5.9)

Une grammaire est contextuelle si toute production est de la forme $\alpha A \beta \to \alpha \gamma \beta$ pour $\alpha, \beta \in V^*$, $A \in N$ et $\gamma \in V^+$.

$$
\begin{aligned}
G_1 : \quad S &\to \texttt{x} \mid LE \\
L &\to \texttt{x} \mid L\texttt{,\_x} \\
\texttt{,\_x}E &\to \texttt{\_and\_x}
\end{aligned}
$$

- (note $\gamma \neq \varepsilon$)
- α et β donne le contexte dans lequel A peut se dériver en γ
- (motivation linguistique)
- G_1 n'est pas contextuelle

Théorème (5.10, sans démonstration ici)

Chaque grammaire contextuelle est monotone, et pour chaque grammaire monotone G il existe une grammaire contextuelle G' telle que $L(G) = L(G')$.

Grammaires hors-contexte

Définition (re)

Une grammaire est contextuelle si toute production est de la forme $\alpha A\beta \to \alpha\gamma\beta$ pour $\alpha, \beta \in V^*$, $A \in N$ et $\gamma \in V^+$.

Définition (5.15)

Une grammaire est hors-contexte si toute production est de la forme $A \to \gamma$ pour $A \in N$ et $\gamma \in V^+$.

- donc voilà pourquoi « hors-contexte »

Propriétés

Il existe un algorithme qui, en prenant une grammaire hors-contexte G et un mot w comme entrée, décide si $w \in L(G)$.

- le « problème de parsage » est donc décidable
- en fait, en temps non-deterministe polynomiel

Il existe un algorithme qui, en prenant une grammaire hors-contexte G comme entrée, décide si $L(G) = \emptyset$.

- le « problème de vacuité » : aussi décidable

Il n'existe pas d'algorithme qui, en prenant une grammaire hors-contexte G comme entrée, décide si $L(G) = \Sigma^*$.

- le « problème d'universalité » : indécidable
- (mais ce n'est pas trop grave)

Exemple

$$G_1 : \quad \begin{aligned} S &\to \mathtt{x} \mid LE \\ L &\to \mathtt{x} \mid L\mathtt{,\_x} \\ \mathtt{,\_x}E &\to \mathtt{\_and\_x} \end{aligned} \qquad\qquad G_2 : \quad \begin{aligned} S &\to \mathtt{x} \mid A\mathtt{\_and\_x} \\ A &\to \mathtt{x} \mid \mathtt{x,\_}A \end{aligned}$$

Quelques dérivations :

- $S \Rightarrow \mathtt{x}$
- $S \Rightarrow A\mathtt{\_and\_x} \Rightarrow \mathtt{x\_and\_x}$
- $S \Rightarrow A\mathtt{\_and\_x} \Rightarrow \mathtt{x,\_}A\mathtt{\_and\_x} \Rightarrow \mathtt{x,\_x\_and\_x}$

Grammaires régulières

Définition (5.18 variant)

Une grammaire est régulière si toute production est de la forme $A \to aB$ ou $A \to a$ pour $A, B \in N$ et $a \in \Sigma^+$.

Exemple :

$$G_2 : \quad \begin{aligned} S &\to \texttt{x} \mid A\texttt{\_and\_x} \\ A &\to \texttt{x} \mid \texttt{x,\_}A \end{aligned} \qquad\qquad G_3 : \quad \begin{aligned} S &\to \texttt{x} \mid \texttt{x}E \\ E &\to \texttt{,\_x}E \mid \texttt{\_and\_x} \end{aligned}$$

Quelques dérivations :

- $S \Rightarrow \texttt{x}$
- $S \Rightarrow \texttt{x}E \Rightarrow \texttt{x\_and\_x}$
- $S \Rightarrow \texttt{x}E \Rightarrow \texttt{x,\_x}E \Rightarrow \texttt{x,\_x\_and\_x}$

Hiérarchie de Chomsky

type	grammaires	productions	langages	automates
4	finis	$N \to \Sigma^+$	finis	finis acycliques
	$\Downarrow$		$\subsetneq$	
3	régulières	$N \to \Sigma^+ \cup \Sigma^+ N$	réguliers	finis
	$\Downarrow$		$\subsetneq$	
2	hors-contexte	$N \to V^+$	algébriques	à pile
	$\Downarrow$		$\subsetneq$	
1	contextuelles	$\alpha N \beta \to \alpha V^+ \beta$	contextuels	linéairement bornés
	$\Downarrow$		$\subsetneq$	
0	syntagmatiques	$V^+ \to V^+$	récursivement énumerables	de Turing

Hiérarchie de Chomsky

type	grammaires	productions	langages	automates
4	finis	$N \to \Sigma^+$	finis	finis acycliques
	$\Downarrow$		$\subsetneq$	
3	régulières	$N \to \Sigma^+ \cup \Sigma^+ N$	réguliers	finis
	$\Downarrow$		$\subsetneq$	
2	hors-contexte	$N \to V^+$	algébriques	à pile
	$\Downarrow$		$\subsetneq$	
1	contextuelles	$\alpha N \beta \to \alpha V^+ \beta$	contextuels	linéairement bornés
	$\Downarrow$		$\subsetneq$	
0	syntagmatiques	$V^+ \to V^+$	récursivement énumerables	de Turing

Et pourquoi « langages algébriques » ?

grammaire régulière	système d'équations linéaires à droite	langage rationnel
$X_0 \to aX_0 \mid aX_1 \mid \varepsilon$	$L_0 = \{a\}L_0 \cup \{a\}L_1 \cup \{\varepsilon\}$	
$X_1 \to aX_2 \mid bX_0$	$L_1 = \{a\}L_2 \cup \{b\}L_0$	L_0
$X_2 \to aX_1 \mid bX_2$	$L_2 = \{a\}L_1 \cup \{b\}L_2$	

grammaire hors-contexte	système d'équations polynomielles	langage algébrique
$X_0 \to aX_0bX_1a$	$L_0 = \{a\}L_0\{b\}L_1\{a\}$	
$X_1 \to X_1b \mid cX_2d$	$L_1 = L_1\{b\} \cup \{c\}L_2\{d\}$	L_0
$X_2 \to cX_2d \mid \varepsilon$	$L_2 = \{c\}L_1\{d\} \cup \{\varepsilon\}$	

Grammaires hors-contexte

Et le mot vide ?

type	grammaires	productions	langages	automates
4	finis	$N \to \Sigma^+$	finis	finis acycliques
	$\Downarrow$		$\subsetneq$	
3	régulières	$N \to \Sigma^+ \cup \Sigma^+ N$	réguliers	finis
	$\Downarrow$		$\subsetneq$	
2	hors-contexte	$N \to V^+$	algébriques	à pile
	$\Downarrow$		$\subsetneq$	
1	contextuelles	$\alpha N \beta \to \alpha V^+ \beta$	contextuels	linéairement bornés
	$\Downarrow$		$\subsetneq$	
0	syntagmatiques	$V^+ \to V^+$	récursivement énumerables	de Turing

Et le mot vide ?

type	grammaires	productions	langages	automates
4	finis	$N \to \Sigma^*$	finis	finis acycliques
	$\Downarrow$		$\subsetneq$	
3	régulières	$N \to \Sigma^* \cup \Sigma^* N$	réguliers	finis
	$\Downarrow$		$\subsetneq$	
2	hors-contexte	$N \to V^*$	algébriques	à pile
	$\Downarrow$		$\subsetneq$	
1	contextuelles	$\alpha N \beta \to \alpha V^+ \beta$ $S_0 \to S \mid \varepsilon$	contextuels	linéairement bornés
	$\Downarrow$		$\subsetneq$	
0	syntagmatiques	$V^+ \to V^*$	récursivement énumerables	de Turing

Exemple

Un extrait de la grammaire hors-contexte pour C, plus une dérivation :

⟨Stmt⟩ → ⟨Id⟩ = ⟨Expr⟩ ;
⟨Stmt⟩ → { ⟨StmtList⟩ }
⟨Stmt⟩ → if (⟨Expr⟩) ⟨Stmt⟩
⟨StmtList⟩ → ⟨Stmt⟩
⟨StmtList⟩ → ⟨StmtList⟩ ⟨Stmt⟩
⟨Expr⟩ → ⟨Id⟩
⟨Expr⟩ → ⟨Num⟩
⟨Expr⟩ → ⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩
⟨Id⟩ → x
⟨Id⟩ → y
⟨Num⟩ → 0
⟨Num⟩ → 1
⟨Num⟩ → 9
⟨Optr⟩ → >
⟨Optr⟩ → +

⟨Stmt⟩
if (⟨Expr⟩) ⟨Stmt⟩
if (⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩) ⟨Stmt⟩
if (⟨Id⟩ ⟨Optr⟩ ⟨Expr⟩) ⟨Stmt⟩
if (x ⟨Optr⟩ ⟨Expr⟩) ⟨Stmt⟩
if (x > ⟨Expr⟩) ⟨Stmt⟩
if (x > ⟨Num⟩) ⟨Stmt⟩
if (x > 9) ⟨Stmt⟩
if (x > 9) { ⟨StmtList⟩ }
if (x > 9) { ⟨StmtList⟩ ⟨Stmt⟩ }
if (x > 9) { ⟨Stmt⟩ ⟨Stmt⟩ }
if (x > 9) { ⟨Id⟩ = ⟨Expr⟩ ; ⟨Stmt⟩ }
if (x > 9) { x = ⟨Expr⟩ ; ⟨Stmt⟩ }
if (x > 9) { x = ⟨Num⟩ ; ⟨Stmt⟩ }
if (x > 9) { x = 0 ; ⟨Stmt⟩ }
if (x > 9) { x = 0 ; ⟨Id⟩ = ⟨Expr⟩ ; }
if (x > 9) { x = 0 ; y = ⟨Expr⟩ ; }
if (x > 9) { x = 0 ; y = ⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩ ; }
if (x > 9) { x = 0 ; y = ⟨Id⟩ ⟨Optr⟩ ⟨Expr⟩ ; }
if (x > 9) { x = 0 ; y = y ⟨Optr⟩ ⟨Expr⟩ ; }
if (x > 9) { x = 0 ; y = y + ⟨Expr⟩ ; }
if (x > 9) { x = 0 ; y = y + ⟨Num⟩ ; }
if (x > 9) { x = 0 ; y = y + 1 ; }

Exemple

$$
\begin{aligned}
\mathsf{Var} &::= \texttt{[a-zA-Z][a-zA-Z0-9\_]*} \\
\mathsf{Num} &::= \texttt{-?[1-9][0-9]*} \\
\mathsf{Aexp} &::= \mathsf{Num} \mid \mathsf{Var} \mid \mathsf{Aexp} + \mathsf{Aexp} \mid \mathsf{Aexp} - \mathsf{Aexp} \mid \mathsf{Aexp} * \mathsf{Aexp} \\
\mathsf{Bexp} &::= \mathsf{True} \mid \mathsf{False} \mid \mathsf{Aexp} == \mathsf{Aexp} \mid \mathsf{Aexp} < \mathsf{Aexp} \\
&\quad \mid \neg \mathsf{Bexp} \mid \mathsf{Bexp} \wedge \mathsf{Bexp} \mid \mathsf{Bexp} \vee \mathsf{Bexp} \\
\mathsf{Stmt} &::= \mathsf{Var} = \mathsf{Aexp} \mid \mathsf{Stmt} \, ; \mathsf{Stmt} \mid \texttt{while} \ \mathsf{Bexp} \ \mathsf{Stmt} \\
&\quad \mid \texttt{if} \ \mathsf{Bexp} \ \texttt{then} \ \mathsf{Stmt} \ \texttt{else} \ \mathsf{Stmt}
\end{aligned}
$$

- une grammaire hors-contexte en forme Backus-Naur étendue :
- « ::= » au lieu de $\rightarrow$
- productions $N \rightarrow V^* \cup RE(\Sigma)$
- (normalement variables écrits sous forme « $\langle \mathsf{Stmt} \rangle$ » etc.)

Forme normale de Greibach

Théorème (9.10)

Pour chaque grammaire hors-contexte G il existe une autre G' telle que $L(G') = L(G)$ et toutes les productions sont sous la forme $S \to \varepsilon$ ou $A \to a\alpha$ avec $a \in \Sigma$ et $\alpha \in (N \setminus \{S\})^*$.

- donc $S \to \varepsilon$ ou $A \to aA_1 \dots A_n$
- pas de récursion à gauche
- peu se raffiner en forme normale de Greibach quadratique, ou que des productions $S \to \varepsilon$, $A \to a$, $A \to aB$ et $A \to aBC$ sont permis

Automates à pile

Un algorithme

Notre vieil ami : $G : S \to aSb \mid \varepsilon$ $\quad L(G) = \{a^n b^n \mid n \geq 0\}$

```
def anbn(stream):
    state = 0
    counter = 0
    while x = next(stream):
        if state == 0:
            if x == "a":
                counter += 1
            elif x == "b":
                counter -= 1
                state = 1
        elif state == 1:
            if x == "b":
                counter -= 1
            else: return False
    if counter == 0: return True
    else: return False
```

Un algorithme

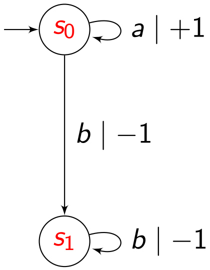
Notre vieil ami : $G : S \to aSb \mid \varepsilon$ $\qquad L(G) = \{a^n b^n \mid n \geq 0\}$

```
def anbn(stream):
    state = 0
    counter = 0
    while x = next(stream):
        if state == 0:
            if x == "a":
                counter += 1
            elif x == "b":
                counter -= 1
                state = 1
        elif state == 1:
            if x == "b":
                counter -= 1
            else: return False
    if counter == 0: return True
    else: return False
```

Un algorithme

Notre vieil ami : $G : S \to aSb \mid \varepsilon$ $\qquad L(G) = \{a^n b^n \mid n \geq 0\}$

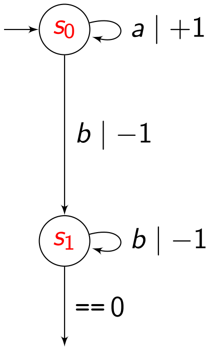
```
def anbn(stream):
    state = 0
    counter = 0
    while x = next(stream):
        if state == 0:
            if x == "a":
                counter += 1
            elif x == "b":
                counter -= 1
                state = 1
        elif state == 1:
            if x == "b":
                counter -= 1
            else: return False
    if counter == 0: return True
    else: return False
```



Un algorithme

Notre vieil ami : $G : S \to aSb \mid \varepsilon$ $\quad L(G) = \{a^n b^n \mid n \geq 0\}$

```
def anbn(stream):
    state = 0
    counter = 0
    while x = next(stream):
        if state == 0:
            if x == "a":
                counter += 1
            elif x == "b":
                counter -= 1
                state = 1
        elif state == 1:
            if x == "b":
                counter -= 1
            else: return False
    if counter == 0: return True
    else: return False
```



Plus compliqué

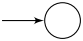
- $G : S \to (S) \mid \{S\} \mid SS \mid \varepsilon$
- $L(G) = \{(\{\}), \{()(\{\})\}, \dots\}$: des mots bien parenthésés

```
def dyck2(stream):
    stack = []
    while x = next(stream):
        if x == "(" or x == "{":
            push(stack, x)
        elif x == ")":
            match = pop(stack)
            if match != "(":
                return False
        elif x == "}":
            match = pop(stack)
            if match != "{":
                return False
    if empty(stack): return True
    else: return False
```

Plus compliqué

- $G : S \to (S) \mid \{S\} \mid SS \mid \varepsilon$
- $L(G) = \{(\{\}), \{()(\{\})\}, \dots\}$: des mots bien parenthésés

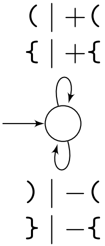
```
def dyck2(stream):
    stack = []
    while x = next(stream):
        if x == "(" or x == "{":
            push(stack, x)
        elif x == ")":
            match = pop(stack)
            if match != "(":
                return False
        elif x == "}":
            match = pop(stack)
            if match != "{":
                return False
    if empty(stack): return True
    else: return False
```



Plus compliqué

- $G : S \to (S) \mid \{S\} \mid SS \mid \varepsilon$
- $L(G) = \{(\{\}), \{()(\{\})\}, \dots\}$: des mots bien parenthésés

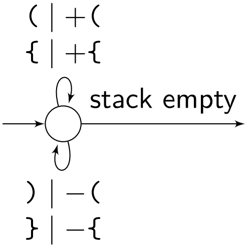
```
def dyck2(stream):
    stack = []
    while x = next(stream):
        if x == "(" or x == "{":
            push(stack, x)
        elif x == ")":
            match = pop(stack)
            if match != "(":
                return False
        elif x == "}":
            match = pop(stack)
            if match != "{":
                return False
    if empty(stack): return True
    else: return False
```



Plus compliqué

- $G : S \to (S) \mid \{S\} \mid SS \mid \varepsilon$
- $L(G) = \{(\{\}), \{()(\{\})\}, \dots\}$: des mots bien parenthésés

```
def dyck2(stream):
    stack = []
    while x = next(stream):
        if x == "(" or x == "{":
            push(stack, x)
        elif x == ")":
            match = pop(stack)
            if match != "(":
                return False
        elif x == "}":
            match = pop(stack)
            if match != "{":
                return False
    if empty(stack): return True
    else: return False
```



Automates à pile

Définition

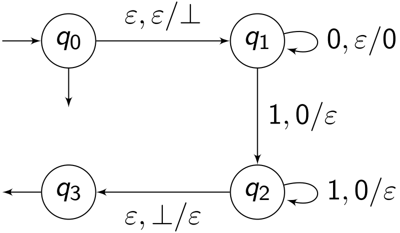
Un automate à pile est une structure $(\Sigma, \Gamma, Q, I, F, \Delta)$ où

- Σ et Γ sont des ensembles finis de symboles,
- Q est un ensemble fini d'états,
- $I \subseteq Q$ est l'ensemble des états initiaux,
- $F \subseteq Q$ est l'ensemble des états finaux, et
- $\Delta \subseteq Q \times (\Gamma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma^*$ est la relation de transition.

- bonne référence pour les automates à pile : M. Sipser, *Introduction to the Theory of Computation*. Cengage Learning, 2012

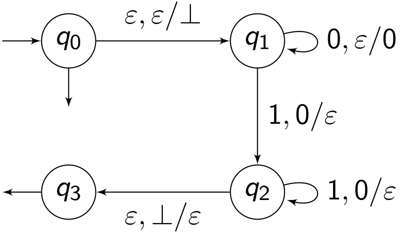
Exemple

$$\Sigma = \{0,1\} \qquad \Gamma = \{0,1,\perp\} \qquad \Delta = \{(q_0,\varepsilon,\varepsilon,q_1,\perp),(q_1,0,\varepsilon,q_1,0),$$
$$Q = \{q_0,q_1,q_2,q_3\} \qquad (q_1,1,0,q_2,\varepsilon),(q_2,1,0,q_2,\varepsilon),$$
$$I = \{q_0\} \qquad F = \{q_0,q_3\} \qquad (q_2,\varepsilon,\perp,q_3,\varepsilon)\}$$



Exemple

$$\Sigma = \{0,1\} \qquad \Gamma = \{0,1,\bot\} \qquad \Delta = \{(q_0,\varepsilon,\varepsilon,q_1,\bot),(q_1,0,\varepsilon,q_1,0),$$
$$Q = \{q_0,q_1,q_2,q_3\} \qquad (q_1,1,0,q_2,\varepsilon),(q_2,1,0,q_2,\varepsilon),$$
$$I = \{q_0\} \qquad F = \{q_0,q_3\} \qquad (q_2,\varepsilon,\bot,q_3,\varepsilon)\}$$



$$L = \{0^n 1^n \mid n \geq 0\}$$

Comment ça marche

Un automate à pile : $A = (\Sigma, \Gamma, Q, I, F, \Delta)$:

- Σ, Γ, Q ensembles finis, $I, F \subseteq Q$,
- $\Delta \subseteq Q \times (\Gamma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma^*$

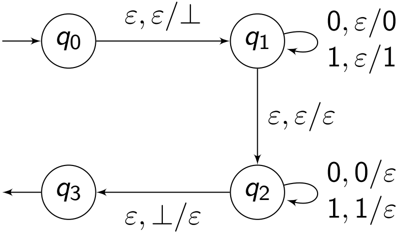
On note $q \xrightarrow{a,\gamma/w} r$ pour $(q, \gamma, a, r, w) \in \Delta$.

Définition

- Une configuration dans A est un tuple $(q, w) \in Q \times \Gamma^*$.
- Un calcul dans A est une séquence $\sigma = (q_1, w_1) \xrightarrow{a_1,\gamma_1/v_1} (q_2, w_2) \xrightarrow{a_2,\gamma_2/v_2} \cdots \to (q_n, w_n)$ telle que pour tout $i = 1, \dots, n-1$, il existe $u_i \in \Gamma^*$ avec $w_i = u_i \gamma_i$ et $w_{i+1} = u_i v_i$.
- L'étiquette d'un calcul σ est $\lambda(\sigma) = a_1 a_2 \dots a_{n-1} \in \Sigma^*$; σ est réussi si $q_1 \in I$ et $q_n \in F$; le langage reconnu par A est $L(A) = \{\lambda(\sigma) \mid \sigma \text{ calcul réussi dans } A\}$.

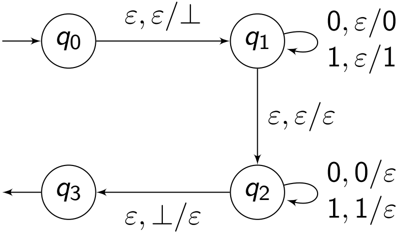
Exercise

Quel est le langage reconnu ?



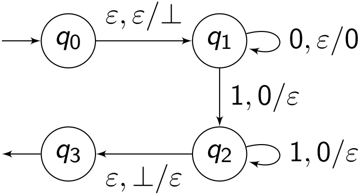
Exercise

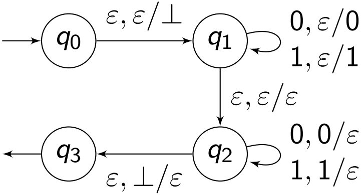
Quel est le langage reconnu ?



- $L = \{ww^{\mathrm{R}} \mid w \in \{0,1\}^*\}$
- tous les mots dans $\{0,1\}^*$ suivi par leur inverse

Vidange de pile





- les deux exemples vide la pile avant d'accepter
- technique : marquer le bas de la pile par symbole spécial $\bot$
- alors si on remplace

Un calcul $(q_1, w_1) \rightsquigarrow (q_n, w_n)$ est réussi si $q_1 \in I$ et $q_n \in F$.

par

Un calcul $(q_1, w_1) \rightsquigarrow (q_n, w_n)$ est réussi si $q_1 \in I$, $q_n \in F$ et $w_n = \varepsilon$.

ou par

Un calcul $(q_1, w_1) \rightsquigarrow (q_n, w_n)$ est réussi si $q_1 \in I$ et $w_n = \varepsilon$.

rien ne change sémantiquement.

Automates à pile $\hat{=}$ grammaires hors-contexte

Théorème

Un langage L est algébrique ssi il existe un automate à pile A tel que $L = L(A)$.

Démonstration.

1. $\Longleftarrow$: transformer un automate à pile en grammaire hors-contexte : *it's complicated*
2. $\Longrightarrow$: Soit $G = (N, \Sigma, P, S)$ une grammaire hors-contexte, on construit un automate à pile $A = (\Sigma, \Gamma, Q, I, F, \Delta)$ tel que $L(A) = L(G)$.

Automates à pile $\hat{=}$ grammaires hors-contexte

Théorème

Un langage L est algébrique ssi il existe un automate à pile A tel que $L = L(A)$.

Démonstration.

1. $\Longleftarrow$: transformer un automate à pile en grammaire hors-contexte : *it's complicated*
2. $\Longrightarrow$: Soit $G = (N, \Sigma, P, S)$ une grammaire hors-contexte, on construit un automate à pile $A = (\Sigma, \Gamma, Q, I, F, \Delta)$ tel que $L(A) = L(G)$. $\longleftarrow$ un algorithme de parsage !
3. L'idée est d'utiliser la pile pour simuler des dérivations.

Automates à pile $\hat{=}$ grammaires hors-contexte

Théorème

Un langage L est algébrique ssi il existe un automate à pile A tel que $L = L(A)$.

Démonstration.

1. $\Longleftarrow$: transformer un automate à pile en grammaire hors-contexte : *it's complicated*
2. $\Longrightarrow$: Soit $G = (N, \Sigma, P, S)$ une grammaire hors-contexte, on construit un automate à pile $A = (\Sigma, \Gamma, Q, I, F, \Delta)$ tel que $L(A) = L(G)$. $\longleftarrow$ un algorithme de parsage !
3. L'idée est d'utiliser la pile pour simuler des dérivations.
4. Soit $\Gamma = N \cup \Sigma$, $Q = \{q_i, q_p\}$ et $I = \{q_i\}$. On accepte par pile vide, donc pas besoin de F.

Automates à pile $\hat{=}$ grammaires hors-contexte

Théorème

Un langage L est algébrique ssi il existe un automate à pile A tel que $L = L(A)$.

Démonstration.

2. $\Longrightarrow$: Soit $G = (N, \Sigma, P, S)$ une grammaire hc, on construit un automate à pile $A = (\Sigma, \Gamma, Q, I, F, \Delta)$ tel que $L(A) = L(G)$.
3. L'idée est d'utiliser la pile pour simuler des dérivations.
4. Soit $\Gamma = N \cup \Sigma$, $Q = \{q_i, q_p\}$ et $I = \{q_i\}$. On accepte par pile vide, donc pas besoin de F. état initial, état de parsage

Automates à pile $\hat{=}$ grammaires hors-contexte

Théorème

Un langage L est algébrique ssi il existe un automate à pile A tel que $L = L(A)$.

Démonstration.

2. $\Longrightarrow$: Soit $G = (N, \Sigma, P, S)$ une grammaire hc, on construit un automate à pile $A = (\Sigma, \Gamma, Q, I, F, \Delta)$ tel que $L(A) = L(G)$.
3. L'idée est d'utiliser la pile pour simuler des dérivations.
4. Soit $\Gamma = N \cup \Sigma$, $Q = \{q_i, q_p\}$ et $I = \{q_i\}$. On accepte par pile vide, donc pas besoin de F. état initial, état de parsage
5. Soit $$\Delta = \{(q_i, \varepsilon, \varepsilon, q_p, S)\} \cup \{(q_p, A, \varepsilon, q_p, \gamma) \mid A \in N, (A, \gamma) \in P\} \cup \{(q_p, a, a, q_p, \varepsilon) \mid a \in \Sigma\}.$$

Automates à pile $\hat{=}$ grammaires hors-contexte

Théorème

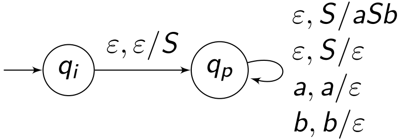
Un langage L est algébrique ssi il existe un automate à pile A tel que $L = L(A)$.

Démonstration.

2. $\Longrightarrow$: Soit $G = (N, \Sigma, P, S)$ une grammaire hc, on construit un automate à pile $A = (\Sigma, \Gamma, Q, I, F, \Delta)$ tel que $L(A) = L(G)$.
3. L'idée est d'utiliser la pile pour simuler des dérivations.
4. Soit $\Gamma = N \cup \Sigma$, $Q = \{q_i, q_p\}$ et $I = \{q_i\}$. On accepte par pile vide, donc pas besoin de F. état initial, état de parsage
5. Soit $$\begin{aligned} \Delta = {} & \{(q_i, \varepsilon, \varepsilon, q_p, S)\} \\ & \cup \{(q_p, A, \varepsilon, q_p, \gamma) \mid A \in N, (A, \gamma) \in P\} \\ & \cup \{(q_p, a, a, q_p, \varepsilon) \mid a \in \Sigma\}. \end{aligned}$$
6. Maintenant il faut montrer que $L(A) = L(G)$.

Exemple

$$G : S \to aSb \mid \varepsilon$$

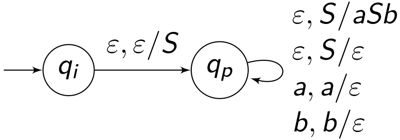


pour reconnaitre *aabb* :

état	pile	reste du mot
q_i	ε	$aabb$

Exemple

$G : S \to aSb \mid \varepsilon$

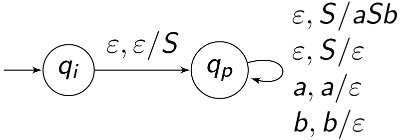


pour reconnaitre *aabb* :

état	pile	reste du mot
q_i	ε	$aabb$
q_p	S	$aabb$

Exemple

$G : S \to aSb \mid \varepsilon$

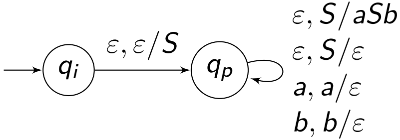


pour reconnaitre *aabb* :

état	pile	reste du mot
q_i	ε	$aabb$
q_p	S	$aabb$
q_p	aSb	$aabb$

Exemple

$G : S \to aSb \mid \varepsilon$

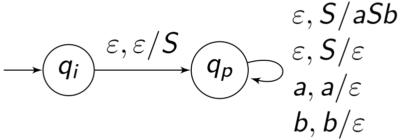


pour reconnaitre *aabb* :

état	pile	reste du mot
q_i	ε	$aabb$
q_p	S	$aabb$
q_p	aSb	$aabb$
q_p	Sb	abb

Exemple

$G : S \to aSb \mid \varepsilon$



pour reconnaitre *aabb* :

état	pile	reste du mot
q_i	ε	$aabb$
q_p	S	$aabb$
q_p	aSb	$aabb$
q_p	Sb	abb
q_p	$aSbb$	abb
q_p	Sbb	bb
q_p	bb	bb
q_p	b	b
q_p	ε	ε

- beaucoup de non-déterminisme !

Fin à la spontanéité

Théorème

Pour chaque automate à pile A il existe un autre A' sans transitions $q \xrightarrow{\varepsilon,\gamma/w} r$ tel que $L(A') = L(A)$.

Esquisse de preuve.

1. Transformer A en grammaire hors-contexte G.
2. Transformer G en forme normale de Greibach G'.
   - donc toutes productions sous forme $A \to aw$
3. Transformer G' en automate à pile avec une construction adaptée :
   - transitions parsage $\{(q_p, A, a, q_p, w) \mid (A, aw) \in P\}$
   - éliminer la transition $(q_i, \varepsilon, \varepsilon, q_p, S)$ par fermeture

Automates à pile déterministes

Définition

Un automate à pile $(\Sigma, \Gamma, Q, I, F, \Delta)$ est déterministe si $|I| = 1$, $a \neq \varepsilon$ pour tous $(q, \gamma, a, r, w) \in \Delta$, et pour chaque (q, γ, a) il y a au maximum un (r, w) tel que $(q, \gamma, a, r, w) \in \Delta$.

- pas de choix du tout
- bien utile pour l'implémentation !

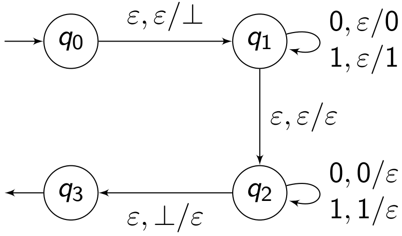
Théorème

Il existe des langages algébriques qui ne sont pas reconnus par un automate à pile déterministe.

- pas de déterminisation !

Exemple (déjà vu)

A :



- $L(A) = \{ww^{\mathrm{R}} \mid w \in \{0,1\}^*\}$
- il n'existe pas d'automate à pile déterministe A' tel que $L(A') = L(A)$
- intuition : on ne peut pas deviner la fin du w (et le début du w^{R}) sans connaitre la longueur de w

Conclusion

Hiérarchie de Chomsky

type	grammaires	productions	langages	automates
4	finis	$N \to \Sigma^+$	finis	finis acycliques
	$\Downarrow$		$\subsetneq$	
3	régulières	$N \to \Sigma^+ \cup \Sigma^+ N$	réguliers	finis
	$\Downarrow$		$\subsetneq$	
2	hors-contexte	$N \to V^+$	algébriques	à pile
	$\Downarrow$		$\subsetneq$	
1	contextuelles	$\alpha N \beta \to \alpha V^+ \beta$	contextuels	linéairement bornés
	$\Downarrow$		$\subsetneq$	
0	syntagmatiques	$V^+ \to V^+$	récursivement énumerables	de Turing

Conclusion

- grammaires hors-contexte : bon compromis entre utilité et efficacité
- pour le parsage de langages de programmation
- automates à pile pour la reconnaissance
- mais non-déterminisables $\Longrightarrow$ peu utile pour le parsage
- la prochaine fois : grammaires et algorithmes de parsage déterministes

Références pour aujourd'hui :

- poly section 5
- poly 6.1, 6.3.3
- sipser-pda.pdf

Derniere remarque : automates à pile visible

Définition

Un automate à pile visible est un automate à pile $(\Sigma, \Gamma, Q, I, F, \Delta)$ où $\Sigma = \Sigma_\mathrm{c} \cup \Sigma_\mathrm{r} \cup \Sigma_\mathrm{i}$ et tel que pour tous $(q, \gamma, a, r, w) \in \Delta$,

- soit $a \in \Sigma_\mathrm{c}$ et $\gamma = \varepsilon$,
- soit $a \in \Sigma_\mathrm{r}$ et $w = \varepsilon$,
- soit $a \in \Sigma_\mathrm{i}$ et $\gamma = w = \varepsilon$.

- $\Sigma = \text{call} \cup \text{return} \cup \text{internal}$
- les types d'opérations sur la pile sont déterminés par le mot
- peuvent être déterminisés et minimisés
- utiles pour le parsage de langages de documents structurisés (XML etc.)
- pas assez expressifs pour les langages de programmation généraux


That's all Folks!