Grammaire Hors-Contexte

Quadruplet $G = (V_t, V_n, Z, P)$ tel que :

- V_t : *vocabulaire terminal* (les *lexèmes*)
  ensemble fini de symboles, notés en minuscules
- V_n : *vocabulaire non-terminal*
  ensemble fini de symboles, notés en majuscules ($V_t \cap V_n = \emptyset$)
- $Z \in V_n$: *axiome*
- P : ensemble des *règles* (ou *productions*)
  $P \subseteq V_n \times (V_t \cup V_n)^*$
  Les éléments (X, α) de P seront notés $X \to \alpha$.

Dérivations

$G = (V_t, V_n, Z, P)$ une grammaire hors-contexte, $w, w' \in (V_t \cup V_n)^*$.

- w' est un *dérivé direct* par G de w, noté $w \Rightarrow_G w'$ ssi :
  il existe $X \in V_n$, il existe $u, t, v \in (V_t \cup V_n)^*$ tels que

$$w = uXv \text{ et } w' = utv \text{ et } X \to t \in P.$$

- w' est un *dérivé* par G de w, noté $w \Rightarrow_G^* w'$, ssi :
  il existe $w_0, w_1, \ldots, w_n \in (V_t \cup V_n)^*$ tels que

$$w_0 = w, w_n = w' \text{ et pour tout i } w_{i+1} \text{ est un dérivé direct de } w_i.$$

Remarque : $\Rightarrow_G^*$ est la fermeture transitive de $\Rightarrow_G$.

Langages hors-contextes

$G = (V_t, V_n, Z, P)$ une grammaire hors-contexte.
Le *langage défini* par G (ou *reconnu* par G) est l'ensemble $L(G)$ tel que :

$$L(G) = \{w \in V_t^* \mid Z \Rightarrow_G^* w\}$$

Un langage L est dit *hors-contexte* s'il existe une grammaire hors-contexte qui le reconnaît.

Remarque : Tout langage régulier est un langage hors-contexte, il existe des langages hors-contexte qui ne sont pas réguliers.

Arbres de dérivation

Soit $G = (V_t, V_n, Z, P)$ une grammaire et w un mot de $L(G)$.

On appelle *arbre de dérivation* (ou *arbre syntaxique*) de w dans G tout arbre n-aire A dont les noeuds sont des éléments de $(V_t \cup V_n)$ et tel que :

- la racine de A est Z (l'axiome de G) ;
- les feuilles de A sont des éléments de $V_t \cup \{\varepsilon\}$ et la séquence gauche-droite des feuilles de A est égale à w ;
- les noeuds non feuilles de A sont des éléments de V_n et si un noeud non feuille X a pour fils u_1, u_2, $\ldots u_n$ dans A alors la règle $X \rightarrow u_1.u_2 \ldots u_n$ doit appartenir à P.

A tout mot de $L(G)$ on peut associer un arbre de dérivation.
Si cet arbre est **unique** alors la grammaire est dite **non ambigüe**.

Langages hors-contexte et automates

- langage réguliers $= a^*.b^*$
  $\Rightarrow$ langage reconnaissable par un **automate d'état fini**
- langages hors-contexte $= a^n.b^n, n \geq 0$
  $\Rightarrow$ il faut ajouter une **mémoire non bornée** à l'automate ...
  $\Rightarrow$ il suffit de pouvoir gérer cette mémoire comme une **pile**

Fonctionnement intuitif d'un **automate à pile :**

- un ensemble fini d'etats Q, un vocabulaire de pile Γ
- une *configuration* $= (q, w, \alpha) \in Q \times V^* \times \Gamma^*$
- en fonction de l'état courant q, du sommet de pile α et du symbole d'entrée v
  - lecture éventuelle de v
  - modification éventuelle du sommet de pile (empiler/dépiler) α est remplacé par β (avec $\beta \in \Gamma^*$)

Une séquence ω est reconnue ssi l'état atteint est *accepteur*.

Exemple

Proposer un automate à pile qui reconnait le langage

$$\{a^n.b^n.c \mid n \geq 0\}$$

Quelques propriétés (importantes en pratiques !)

1. tout langage **régulier** est reconnu par un automate **d'état fini** (et réciproquement)
2. tout langage **hors-contexte** est reconnu par un automate **à pile** (et réciproquement)
3. A est un automate d'état fini, il existe un automate A' tq :
   - A' est un automate d'état fini **déterministe**
   - A et A' reconnaissent le **même** langage
4. tout langage régulier est reconnu par un automate **déterministe**
5. la propriété 3 est **fausse** pour les automates à pile
6. il existe des langages hors-contexte **non reconnus** par un automate à pile **déterministe**

Analyse Syntaxique

Soit $G = (V_t, V_n, Z, P)$ une grammaire hors-contexte,
soit ω un séquence de V_t^*.

Analyse syntaxique = déterminer si $\omega \in L(G)$?

Contraintes pratiques :

- algorithme **linéaire** en fonction de $|\ \omega\ |$
- accés **séquentiel** à ω, de gauche à droite

$\Rightarrow$ possible uniquement pour un **sous-ensemble strict** des langages hors-contexte

$\Rightarrow$ celui qui est reconnaissable par un automate à pile **déterministe !**

Problèmes posés :

- Est-ce que le **langage** considéré appartient à cette classe ?
- Si oui, comment construire (efficacement) un analyseur (efficace) ?

$\Rightarrow$ réponse en fonction de la **structure** de la **grammaire** considérée !

Dans la suite ...

- on considère d'abord une technique **d'analyse descendante** :
  - produire ω à partir de Z
  - produire une dérivation *gauche* $Z \Longrightarrow_G^* \omega$

- on cherche à caractériser :
  - une classe de grammaire pour laquelle une analyse syntaxique **déterministe** est possible ;
  - une **condition suffisante** pour décider si une grammaire appartient à cette classe

$\Rightarrow$ **grammaires LL**

Exemple 1

L1 défini par la grammaire suivante

$$
\begin{aligned}
Z &\rightarrow aXc && (1)\\
Z &\rightarrow bY && (2)\\
X &\rightarrow bX && (3)\\
X &\rightarrow d && (4)\\
Y &\rightarrow e && (5)\\
Y &\rightarrow bY && (6)\\
Y &\rightarrow cYX && (7)
\end{aligned}
$$

Est-ce que abbbdc appartient a L1 ?
Est-ce que bbae appartient a L1 ?
Est-ce que be appartient a L1 ?

Exemple 2

L2 défini par la grammaire suivante

$$Z \rightarrow aX \quad (8)$$
$$X \rightarrow bY \quad (9)$$
$$X \rightarrow bT \quad (10)$$
$$Y \rightarrow c \quad (11)$$
$$T \rightarrow d \quad (12)$$

Est-ce que abd appartient a L2 ?

Exemple 3

L3 défini par la grammaire suivante

$$Z \rightarrow S\# \quad (13)$$
$$S \rightarrow XaY \quad (14)$$
$$X \rightarrow W \quad (15)$$
$$X \rightarrow T \quad (16)$$
$$W \rightarrow bc \quad (17)$$
$$T \rightarrow ac \quad (18)$$
$$Y \rightarrow eY \quad (19)$$
$$Y \rightarrow f \quad (20)$$
$$Y \rightarrow \varepsilon \quad (21)$$

Est-ce que aceef# appartient a L3 ?

Premiers, Suivants

Soit $G = (V_t, V_n, Z, P)$ une grammaire hors-contexte.

- soit $\alpha \in (V_t \cup V_n)*$, le prédicat **Vide**$(\alpha)$ vaut vrai ssi ε peut être dérivé de α par G :

$$\textbf{Vide}(\alpha) \equiv \alpha \Rightarrow_G^* \varepsilon$$

- soit $\alpha \in (V_t \cup V_n)*$, **Premiers**$(\alpha)$ est l'ensemble des symboles terminaux qui peuvent débuter une dérivation de α par G :

$$\textbf{Premier}(\alpha) = \{a \in V_t \mid \alpha \Rightarrow_G^* a\beta\}$$

- soit $X \in V_n$, **Suivants**(X) est l'ensemble des symboles terminaux qui peuvent suivre une occurrence de X dans une dérivation de Z par G :

$$\textbf{Suivants}(X) = \{a \in V_t \mid Z \Rightarrow_G^* \alpha X a \beta\}$$

Directeurs

soit $X \to u_1.u_2 \dots u_n$ une règle de P, avec $u_i \in (V_t \cup V_n)$.
Directeurs$(X \to u_1.u_2 \dots u_n)$ est l'ensemble des symboles terminaux qui peuvent débuter une dérivation par cette règle :

$$\textbf{Directeurs}(X \to u_1.u_2 \dots u_n) = \left(\bigcup_{i \in V_i} \textbf{Premiers}(u_i)\right) \cup (\text{si } \textbf{Vide}(u_1 \dots u_n) \text{ alors } \textbf{Suivants}(X) \text{ sinon } \emptyset)$$

avec $V_i = \{i \mid \textbf{Vide}(u_1 \dots u_{i-1})\}$.

Grammaire LL(1)

Une grammaire hors-contexte $G = (V_t, V_n, Z, P)$ est dite *LL(1)* si et seulement si l'ensemble des règles de P définissant un même symbole non-terminal ont des directeurs disjoints :

$$\forall X \in V_n.\ \forall X \to \alpha, X \to \beta \in P.$$
$$\textbf{Directeur}(X \to \alpha) \cap \textbf{Directeur}(X \to \beta) = \emptyset$$

Un langage hors-contexte est dit LL(1) si et seulement si il existe une grammaire LL(1) qui le reconnait.

L'ensemble des langages LL(1) est **strictement inclus** dans l'ensemble des langages hors-contexte reconnus par un automate à pile **déterministe**

Exemple

$$
\begin{aligned}
Z &\rightarrow S\# && (22)\\
S &\rightarrow Xa && (23)\\
S &\rightarrow \varepsilon && (24)\\
X &\rightarrow bX && (25)\\
& && (26)
\end{aligned}
$$

$$
\begin{aligned}
Premiers(S) &= Premiers(X) = \{b\}\\
Vide(S) &= vrai\\
Suivants(S) &= \{\#\}\\
Suivant(X) &= Suivants(X) \cup \{a\} = \{a\}\\
Directeurs(Z \rightarrow S\#) &= Premiers(S) \cup \{\#\} \ \ (\text{car } Vide(S))\\
Directeurs(S \rightarrow Xa) &= Premiers(X) = \{b\}\\
Directeurs(S \rightarrow \varepsilon) = Suivants(S) &= \{\#\}
\end{aligned}
$$

Rendre une grammaire LL(1) ?

Etant donnée une grammaire hors-contexte G, construire G' telle que :

- $L(G') = L(G)$
- G' est LL(1)

Problème :

- ce n'est pas toujours possible !
  (langages LL(1) $\subset$ langages hors-contexte)
- savoir si cela est possible est **indécidable** ...

On propose donc des "heuristiques" pour construire G', mais :

- il n'est pas certain que la grammaire obtenue soit LL(1) ...
- si elle ne l'est pas il faut peut-être continuer à chercher ...

Heuristique 1 : factorisation

Si G contient deux règles de la forme :

$$\begin{aligned} X &\rightarrow a\alpha \\ X &\rightarrow a\beta \end{aligned}$$

Alors on peut construire G' :

$$\begin{aligned} X &\rightarrow aY \\ Y &\rightarrow \alpha \\ Y &\rightarrow \beta \end{aligned}$$

$\Rightarrow G'$ est LL(1) si Premier(α) $\cap$ Premier(β) $= \emptyset$

Heuristique 2 : élimination de la récursivité à gauche

Si G contient deux règles de la forme :

$$\begin{aligned} X &\rightarrow X\alpha \\ X &\rightarrow \beta \end{aligned}$$

Alors on peut construire G' :

$$\begin{aligned} X &\rightarrow \beta Y \\ Y &\rightarrow \alpha Y \\ Y &\rightarrow \varepsilon \end{aligned}$$

$\Rightarrow G'$ est LL(1) si Premier(α) $\cap$ Suivants(X) $= \emptyset$

Analyseur LL(1) basé sur une “table de prédiction”

$$
\begin{array}{l|l}
Z \to S\$ \quad \{\mathbf{1},\mathbf{0}\} & B \to 0Y \quad \{\mathbf{0}\} \\
S \to AB \quad \{\mathbf{1},\mathbf{0}\} & Y \to B \quad \{\mathbf{0}\} \\
A \to 1A0 \quad \{\mathbf{1}\} & Y \to \varepsilon \quad \{\$\} \\
A \to \varepsilon \quad \{\mathbf{0}\} &
\end{array}
$$

$$L(G) = \{1^n 0^n 0^m \mid 0 \leq n \text{ et } 0 < m\}$$

L'automate de l'analyseur :

	0	1	$
Z	$Z \to S\$$	$Z \to S\$$	
S	$S \to AB$	$S \to AB$	
A	$A \to \varepsilon$	$A \to 1A0$	
B	$B \to 0Y$		
Y	$Y \to B$		$Y \to \varepsilon$

Analyseur LL(1) basé sur des procédures récursives

$Z \rightarrow S\$ \quad \{\mathbf{1}, \mathbf{0}\}$

```
Rec_Z :
          init_sequence() ; si symbole_courant() = "1" ou symbole_courant() = "0" alors Rec_S sinon erreur() ;
          Rec_terminal($)
```

$S \rightarrow AB \quad \{\mathbf{1}, \mathbf{0}\}$

```
          Rec_S :
           si symbole_courant() = "1" ou symbole_courant() = "0" alors Rec_A ; Rec_B ; sinon erreur()
```

$A \rightarrow 1A0 \quad \{\mathbf{1}\} \quad | A \rightarrow \varepsilon \quad \{\mathbf{0}\}$

```
          Rec_A :
           si symbole_courant() = "1" alors
              Rec_terminal("1") ; Rec_A ; Rec_terminal("0") ;
           sinon si symbole_courant() = "0" alors
              rien ;
           sinon erreur()
```

$B \rightarrow 0Y \quad \{0\}$

```
          Rec_B :
           si symbole_courant() = "0" alors Rec_terminal("0") ; Rec_Y ;
           sinon erreur()
```

$Y \rightarrow B \quad \{0\} \quad | Y \rightarrow \varepsilon \quad \{\$\}$

```
          Rec_Y :
           si symbole_courant() = "0" alors
              Rec_B ;
           sinon si symbole_courant() = "$" alors
              rien ;
           sinon erreur()
```