Grammaires

Grammaire

- un nouvel outil pour définir et spécifier un langage
- vient s'ajouter aux 2 précédents : expressions rationnelles, automates
- pas équivalent aux 2 précédents (plus « puissant »)

Grammaires

Grammaire : définition

Une grammaire G est un quadruplet $(\Sigma, \mathcal{V}, A, \mathcal{R})$:

- Σ : un ensemble fini non vide de **symboles terminaux** (ou **lettres terminales**)
- $\mathcal{V}$: un ensemble fini non vide de **symboles non terminaux** (ou **variables**), disjoint du précédent
- $A \in \mathcal{V}$: une variable appelée **axiome**
- $\mathcal{R} \subseteq (\mathcal{V} \cup \Sigma)^*\mathcal{V}(\mathcal{V} \cup \Sigma)^* \times (\mathcal{V} \cup \Sigma)^*$: un **ensemble de règles de production**.

  Une règle $(g, d) \in \mathcal{R}$ sera généralement notée

  $$g \to d$$

  g est la **partie gauche** de la règle, d est la **partie droite**.

Grammaires : hiérarchie de Chomsky

Classification (Noam Chomsky, 1957)

Les grammaires sont classifiées selon le type de règles

- Type 0 : cas général $g \in (\mathcal{V} \cup \Sigma)^*\mathcal{V}(\mathcal{V} \cup \Sigma)^*$, $d \in (\mathcal{V} \cup \Sigma)^*$
- Type 1 : **grammaires contextuelles** $uSv \to uwv$
  $u, v \in (\mathcal{V} \cup \Sigma)^* \quad S \in \mathcal{V} \quad w \in (\mathcal{V} \cup \Sigma)^+$
- Type 2 : **grammaires algébriques** $S \to w$
  $S \in \mathcal{V} \quad w \in (\mathcal{V} \cup \Sigma)^*$
- Type 3 : **grammaires régulières** $S \to w$
  $S \in \mathcal{V} \quad w \in \Sigma\mathcal{V} \cup \{\varepsilon\}$

Note : chaque classe est incluse dans la classe de rang inférieur.

Dans le cadre du cours, on s'intéressera aux **grammaires algébriques** (type 2) (et donc aussi aux régulières (type 3))

Grammaires algébriques

Grammaire algébrique
aka « context free » aka « hors contexte »

Une grammaire G est un quadruplet $(\Sigma, \mathcal{V}, A, \mathcal{R})$:

- Σ : un ensemble fini non vide de **symboles terminaux** (ou **lettres terminales**)
- $\mathcal{V}$: un ensemble fini non vide de **symboles non terminaux** (ou **variables**), disjoint du précédent
- $A \in \mathcal{V}$: une variable appelée **axiome**
- $\mathcal{R} \subseteq \mathcal{V} \times (\mathcal{V} \cup \Sigma)^*$: un **ensemble de règles de production**.

  Chaque règle est sous la forme

  $$S \to w$$

  $S \in \mathcal{V} \quad w \in (\mathcal{V} \cup \Sigma)^*$

Grammaires algébriques

Réécriture

Soit G une grammaire, soient u et u' deux mots de $(\mathcal{V} \cup \Sigma)^*$.
u' est **une réécriture** de u
ou encore
u' **dérive directement** de u
si $\exists S \in \mathcal{V},\ \exists (S \to w) \in \mathcal{R},\ \exists w_1, w_2 \in (\mathcal{V} \cup \Sigma)^*$

- $u = w_1.S.w_2$
- $u' = w_1.w.w_2$

$u =$	w_1	S	w_2
$u' =$	w_1	w	w_2

Notation :

$$u \underset{G}{\to} u'$$

ou encore

$$u \to u'$$

on également

$$u \xrightarrow{1} u'$$

Grammaires algébriques

Dérivation

Une **dérivation** d'un mot $u \in (\mathcal{V} \cup \Sigma)^*$ est une suite finie de mots $u_i \in (\mathcal{V} \cup \Sigma)^*, i \in [0, k]$ telle que

$$u_0 = u \quad \text{et} \quad u_i \to u_{i+1} \quad \forall i < k$$

Chaque mot u_i « **dérive de** u ».
k est la **longueur de la dérivation** $u_0 \to u_1 \to \ldots \to u_k$

Notation :

v **dérive de** u **par une dérivation de longueur** k : $\boxed{u \xrightarrow{k} v}$

v **dérive de** u **par une dérivation quelconque** : $\boxed{u \xrightarrow{*} v}$

$\xrightarrow{*}$ est la clôture réflexive et transitive de $\to$
(dérivation de longueur quelconque, même nulle)

Langages algébriques

Le langage engendré par une grammaire $G = (\Sigma, \mathcal{V}, A, \mathcal{R})$

est constitué des tous les mots qui dérivent de l'axiome et qui ne comportent **que de lettres terminales**

$$L(G) = \{w \in \Sigma^*, A \xrightarrow[G]{*} w\}$$

Langage algébrique (définition)

Un langage est **algébrique** s'il existe une grammaire algébrique qui l'engendre

Le langage élargi comporte tous les mots qui dérivent de l'axiome

$$\widehat{L(G)} = \{w \in (\mathcal{V} \cup \Sigma)^*, A \xrightarrow[G]{*} w\}$$

NB : $L(G) \subseteq \Sigma^*$ $\quad \widehat{L(G)} \subseteq (\mathcal{V} \cup \Sigma)^*$

Grammaires algébriques

Lemme

Il existe des langages algébriques non rationnels.

$a^n b^n$

Le langage $\{a^n b^n, n \geq 0\}$ est un langage algébrique. Il n'est pas rationnel (prouvé lors du cours précédent)

$a^n b^n$

Le langage $\{a^n b^n, n \geq 0\}$ est engendré par la grammaire

- $\Sigma = \{a, b\}$
- $\mathcal{V} = \{A\}$ (variable unique, donc axiome)
- $\mathcal{R} = \{A \to aAb, A \to \varepsilon\}$

Grammaires algébriques

Lemme fondamental

Soient $u_1, u_2, v \in (\Sigma \cup \mathcal{V})^*$ et un entier $k \in \mathbb{N}$ tels que $u_1 u_2 \xrightarrow{k} v$
alors $\exists v_1, \exists v_2 \in (\Sigma \cup \mathcal{V})^*, \exists k_1, k_2 \in \mathbb{N}$

- $u_1 \xrightarrow{k_1} v_1$, $u_2 \xrightarrow{k_2} v_2$
- $v = v_1 v_2$
- $k = k_1 + k_2$

Rationnels et algébriques

RAT* $\subseteq$ *ALG

Tout langage rationnel est algébrique

RAT* $\subseteq$ *REG

Tout langage rationnel peut être engendré par une grammaire régulière.

REG* $\subseteq$ *RAT

Le langage engendré par une grammaire régulière est rationnel

REG* $=$ *RAT

Arbre de dérivation

Définition

Un **arbre de dérivation** pour une grammaire $G = (\Sigma, V, A, \mathcal{R})$ est un arbre étiqueté par $\Sigma \cup V \cup \{\varepsilon\}$

- la **racine** est étiquetée par l'axiome
- un nœud étiqueté par une variable S possède des descendants $e_1, e_2, ...e_n$ où $(S \to e_1 e_2 ... e_n) \in \mathcal{R}$
- un nœud étiqueté par une lettre terminale ou par ε ne possède pas de descendant (c'est une **feuille**)

Feuillage

Le **feuillage** d'un arbre de dérivation est la concaténation des étiquettes portées par les feuilles, lues de la « gauche » vers la « droite ».
Un mot est engendré par G si et seulement si il est le feuillage d'un arbre de dérivation.

Ambiguïté

Mot

Une mot est ambigu pour une grammaire G s'il est le feuillage de deux arbres de dérivations de G différents.

Définition

Une grammaire est ambiguë si et seulement si il existe un mot ambigu pour G.

Langage ambigu

Un langage est ambigu si et seulement si toute grammaire qui l'engendre est ambiguë.

Grammaire réduite

Variable productive
Une variable S d'un grammaire G est dite **productive** si et seulement si $\{w \in \Sigma^*, S \xrightarrow{*} w\} \neq \emptyset$

Variable accessible
Une variable S d'un grammaire G est dite **accessible** si et seulement si $\exists u_1, u_2 \in (\Sigma \cup V)^*, Axiome \xrightarrow{*} u_1 S u_2$

Grammaire réduite
Une grammaire algébrique est dite **réduite** si toutes ses variables sont accessibles et productives.

Réduction de grammaire
Pour toute grammaire algébrique G, il existe une grammaire algébrique réduite qui engendre $\mathcal{L}(G)$

Grammaire propre

Définition
Une grammaire algébrique est dite **propre** si elle ne possède aucune règle sous la forme $S \to \varepsilon$ ni sous la forme $S \to S'$ (avec $S, S' \in V$)

Nettoyage
Pour toute grammaire algébrique G, il existe une grammaire algébrique **réduite** et **propre** qui engendre $\mathcal{L}(G) - \{\varepsilon\}$

Grammaire récursive gauche

Grammaire récursive gauche
Une grammaire est **récursive gauche** si elle comporte une variable récursive gauche

Variable récursive gauche
Une variable S est **récursive gauche** si
$\exists w \in (\Sigma \cup V)^*, \exists n \geq 1, \quad S \xrightarrow{n} Sw$

Tout langage algébrique peut être engendré par une grammaire algébrique non récursive gauche

Récursivité gauche directe
Une variable S est **directement récursive gauche** si
$\exists w \in (\Sigma \cup V)^*, \quad S \to Sw$

Grammaire récursive gauche

Élimination de la récursivité gauche directe
Cas simplifié : $S \to Sw \mid u$, où u ne commence pas par S,est remplacé par

$$S \to uS',\ S' \to wS' \mid \varepsilon$$

Cas général :

$S \to Sw_1 \mid ... \mid Sw_n \mid u_1 \mid ... \mid u_k$
où aucun u_i ne commence par S, est remplacé par
$S \to u_1 S' \mid ... \mid u_k S'$
$S' \to w_1 S' \mid ... \mid w_n S' \mid \varepsilon$

Analyse syntaxique

Analyse syntaxique
- processus consistant à vérifier si un mot (ou texte) peut être engendré par une grammaire
- trouve la **structuration** logique du texte, en constituant l'arbre de dérivation qui permet de l'engendrer.
- la grammaire est la **spécification** du texte attendu
- l'analyseur répond VRAI **si et seulement si** le texte correspond aux spécifications (i.e. peut être engendré par la grammaire)

Algorithmes d'analyse
- on recherche des algorithmes d'analyse syntaxique **efficaces**, de préférence nécessitant une seule lecture du texte.
- les algorithmes efficaces ne fonctionnent pas pour n'importe quelle grammaire.

Analyse syntaxique

Analyse syntaxique
- peut être vue comme la construction d'un arbre dont on connaît au départ les « extrémités »
  - la racine (l'axiome)
  - les feuilles (le texte à analyser)

Descendante ou ascendante ?
Deux grandes catégories d'analyseurs :
- analyse **descendante** : construction de l'arbre depuis la racine vers les feuilles.
  types d'analyse : **LL(1), LL(k)**
- analyse **ascendante** : construction de l'arbre depuis les feuilles vers la racine.
  types d'analyse : **LR(0), LR(1), SLR(1), LALR(1)**....

Dans tous les cas, dissymétrie gauche/droite dans l'ordre de construction, en raison du sens de lecture du texte.

Analyse LL(1)

Principe

- on dérive toujours d'abord la variable la plus à gauche.
- le texte est parcouru par un « tête de lecture » qui progresse de gauche à droite (jamais de retour)
- toutes les lettres situées à gauche de la tête de lecture ont été intégrées à l'arbre de dérivation
- les lettres situées à droite de la tête de lecture (incluse) n'ont pas encore été intégrées à l'arbre de dérivation
- une table indique l'unique règle applicable en tenant compte uniquement de la variable non encore dérivée la plus à gauche et de la lettre située
- si, à une étape de l'algorithme, aucune règle n'est applicable, c'est que le texte n'est pas conforme à la grammaire. sous la tête de lecture.

Exemple LL(1)

Grammaire

$$S \to AB \mid Da$$
$$A \to aAb \mid \varepsilon$$
$$B \to bB \mid \varepsilon$$
$$D \to dD \mid e$$

Table LL(1)

	S	A	B	D
a	$S \to AB$	$A \to aAb$	/	/
b	$S \to AB$	$A \to \varepsilon$	$B \to bB$	/
d	$S \to Da$	/	/	$D \to dD$
e	$S \to Da$	/	/	$D \to e$
#	$S \to AB$	$A \to \varepsilon$	$B \to \varepsilon$	/

Le # est le symbole marqueur de fin

Analyse LL(1)

```
empiler(axiome) ; courant ← 1ère lettre du texte ;
while pile non vide do
  S ← dépiler() ;
  if S est une variable then
    if TableLL1[S,courant] contient S → x1x2...xn then
      empiler(x1x2...xn) ;
    else
      return false ;
    end if
  else
    if S == courant then
      courant ← lettre suivante ;
    else
      return false ;
    end if
  end if
end while
return courant == marqueur de fin ;
```


Calcul de la table LL(1) : les 3 étapes

1 - Calcul des variables et des règles ε-productives

- Une variable V est ε-productive si $V \xrightarrow{*} \varepsilon$
- Une régle $V \to w$ est ε-productive si $V \to w \xrightarrow{*} \varepsilon$

2- Calcul des ensembles « Premier »

- À partir des règles, établir le système d'équations liant les ensembles « Premier ».
- Résolution par plus petit point fixe.

cf : plus loin

3- Calcul des ensembles « Suivant »

- À partir des règles, établir le système d'équations liant les ensembles « Suivant ».
- Résolution par plus petit point fixe.

cf : plus loin

Calcul de la table LL(1) : ensembles « Premier »

Premier : définition

$Premier(\varepsilon) = \emptyset$

$w \in (\mathcal{V} \cup \Sigma)^+,\ Premier(w) = \{x \in \Sigma, w \xrightarrow{*} x.u\}$

Premier : propriété de décomposition

- Si $w = x.w',\ x \in \Sigma$
  $Premier(w) = \{x\}$
- Si $w = S.w',\ S \in \mathcal{V}$ et $\neg(S \xrightarrow{*} \varepsilon)$
  $Premier(w) = Premier(S)$
- Si $w = S.w',\ S \in \mathcal{V}$ et $S \xrightarrow{*} \varepsilon$
  $Premier(w) = Premier(S) \cup Premier(w')$;

Premier d'une variable

$$S \in \mathcal{V}, \quad Premier(S) = \bigcup_{w,\ (S \to w) \in \mathcal{R}} Premier(w)$$