Université de Lille — FACULTÉ DES SCIENCES ET TECHNOLOGIES — Département Informatique

Informatique

L3S5 Miage

Analyse Syntaxique – CTD - Partie 2

Grammaires algébriques

Une **grammaire algébrique** G est définie par un quadruplet (Σ, V, S, P) où :
- Σ est un ensemble fini de symboles **terminaux**,
- V est un ensemble fini de variables (symboles non terminaux),
- S est un élément distingué de V, appelé **axiome**,
- $P \subseteq V \times (\Sigma \cup V)^*$ est un ensemble fini de **règles de production**.

On suppose que $\Sigma \cap V = \emptyset$. $(\Sigma \cup V)^*$ est l'ensemble des mots écrits sur l'alphabet $\Sigma \cup V$.

Une règle de production $(A, aAb) \in P$ sera notée $A \to aAb$.

Un mot u se **dérive en un pas** en un mot v par G s'il existe
- une règle $X \to w$ de P,
- une décomposition $u_1 X u_2$ de u,
- une décomposition $u_1 w u_2$ de v.

On note $u \to_G v$ si u se dérive en un pas en v par G.

Proposition Pour toute grammaire algébrique $G = (\Sigma, V, S, P)$, et pour tout u, v, α, $\beta \in (\Sigma \cup V)^*$, si $u \to_G v$ alors $\alpha u \beta \to_G \alpha v \beta$.

La dérivation d'un mot u est **non déterministe** puuisqu'on a deux sources de choix :
- choix d'un non terminal X dans u,
- choix d'une règle de production de membre gauche X.

Un mot u se **dérive en k pas** en un mot v (noté $u \to_G^k v$) par G si
- $k = 0$ et $u = v$,
- $k > 0$ et il existe un mot u' tel que $u \to_G u'$ et $u' \to_G^{k-1} v$.

Un mot u se dérive en un mot v par G (noté $u \to_G^* v$) si il existe un entier naturel k tel que $u \to_G^k v$. La relation $\to_G^*$ est la clôture réflexive et transitive de la relation $\to_G$.

Proposition Pour toute grammaire algébrique $G = (\Sigma, V, S, P)$, et pour tout u, v, α, $\beta \in (\Sigma \cup V)^*$,
- pour tout entier k, si $u \to_G^k v$ alors $\alpha u \beta \to_G^k \alpha v \beta$.
- si $u \to_G^* v$ alors $\alpha u \beta \to_G^* \alpha v \beta$.

Proposition Pour toute grammaire algébrique $G = (\Sigma, V, S, P)$, et pour tout u_1, u_2, v_1, $v2 \in (\Sigma \cup V)^*$ et k_1, $k_2 \in N$

si $u_1 \to_G^{k_1} v_1$ et $u_2 \to_G^{k_2} v_2$ alors $u_1 u_2 \to_G^k v_1 v_2$ avec $k = k_1 + k_2$.

Le **langage engendré** par la grammaire $G = (\Sigma, V, S, P)$, noté $L(G)$ est l'ensemble des mots w composés uniquement de terminaux que l'on peut dériver à partir de l'axiome de G.

$$L(G) = \{w \ / \ S \to_G^* w, \ w \in \Sigma^*\}$$

Le langage **étendu** engendré par la grammaire $G = (\Sigma, V, S, P)$, noté $\hat{L}(G)$ est l'ensemble de tous les mots w que l'on peut dériver à partir de l'axiome de G.

$$\hat{L}(G) = \{w \ / \ S \rightarrow_G^* w, \ w \in (\Sigma \cup V)^*\}$$

Un langage L est **algébrique** si et seulement si il existe une grammaire algébrique G engendrant L ($L = L(G)$).
Deux grammaires G_1 et G_2 sont **équivalentes** si et seulement si elles engendrent le même langage ($L(G_1) = L(G_2)$).

Lemme fondamental Soit $G = (\Sigma, V, S, P)$ une grammaire algébrique. Soient u_1, u_2, $v \in (\Sigma \cup V)^*$ et $k \in N$. Si $u_1 u_2 \rightarrow_G^k v$ alors il existe v_1, $v_2 \in (\Sigma \cup V)^*$ et k_1, $k_2 \in N$ tels que
- $v = v_1 v_2$,
- $u_1 \rightarrow_G^{k_1} v_1$, $u_2 \rightarrow_G^{k_2} v_2$ et $k = k_1 + k_2$.

Exemple Le langage $L = \{a^n b^n / n \in N\}$ n'est pas un langage rationnel, mais c'est un langage algébrique engendré par la grammaire $G = (\{a, b\}, \{S\}, S, P)$ avec $P = \{S \rightarrow aSb, S \rightarrow \epsilon\}$.

Pour montrer que $L(G) \subseteq L$, on montre que si $w \in L(G)$, alors il existe $k \in N$ tel que $S \rightarrow_G^k w$. Montrons par récurrence sur k que, si $S \rightarrow_G^k w$, alors $w = a^{k-1} b^{k-1}$. Si $k = 1$, alors $w = \epsilon$. Supposons la propriété vraie pour k. Pour $k+1$, on a $S \rightarrow_G aSb \rightarrow_G^k w$. On applique le lemme fondamental avec $u_1 = a$ et $u_2 = Sb$: il existe v_1, v_2, k_1, k_2 tels que $u_1 \rightarrow_G^{k_1} w_1$ et $u_2 \rightarrow_G^{k_2} w_2$, $k_1 + k_2 = k$ et $w = w_1 w_2$. Comme $u_1 = a$, $w_1 = a$ et $k_1 = 0$ donc $u_2 = Sb \rightarrow_G^{k_2} w_2$ avec $k_2 = k$. On applique à nouveau le lemme fondamental avec $u'_1 = S$ et $u'_2 = b$: il existe v'_1, v'_2, k'_1, k'_2 tels que $u'_1 \rightarrow_G^{k'_1} w'_1$ et $u'_2 \rightarrow_G^{k'_2} w'_2$, $k'_1 + k'_2 = k$ et $w_2 = w'_1 w'_2$. Comme $u'_2 = b$, on a $w'2 = b$ et $k'_2 = 0$, et donc $S \rightarrow_G^k w'_1$. Par hypothèse de récurrence, on a $w'_1 = a^{k-1} b^{k-1}$ donc $aSb \rightarrow_G^k a^k b^k$ soit $S \rightarrow_G^{k+1} a^k b^k$.
Pour montrer que $L \subseteq L(G)$, on montre par récurrence sur n, que pour tout n, $a^n b^n \in L(G)$. Pour $n = 0$, $S \rightarrow_G \epsilon$ donc $\epsilon (= a^0 b^0) \in L(G)$. Si la propriété est vraie pour n, on a donc $a^n b^n \in L(G)$ soit $S \rightarrow_G^* a^n b^n$. Donc $aSb \rightarrow_G^* aa^n b^n b$ (par la 2ème proposition). Comme $S \rightarrow aSb \in P$, $S \rightarrow_G^* a^{n+1} b^{n+1}$.

Propriétés de clôture Les langages algébriques sont clos par
- **union** : si L et L' sont algébriques, alors $L \cup L'$ est algébrique.
- **concaténation** : si L et L' sont algébriques, alors $L.L'$ est algébrique.
- **étoile** : si L est algébrique, alors $L* = \bigcup_{i \in n} L^i$ est algébrique,

Attention : les langages algébriques **ne sont pas clos** par
- **intersection** : il existe deux langages algébriques L et L' tels que $L \cap L'$ n'est pas algébrique.
- **complémentaire** : il existe un langage algébrique L tel que $\bar{L}$ n'est pas algébrique.

Grammaires régulières Une grammaire $G = (\Sigma, V, S, P)$ est régulière (ou linéaire) si toute règle X de production de P est
- soit de la forme $X \rightarrow \epsilon$,
- soit de la forme $X \rightarrow aY$ avec a un terminal ($a \in \Sigma$) et Y un non-terminal ($Y \in V$).

Un langage est régulier si il peut être engendré par une grammaire régulière. Un langage est régulier si et seulement si il est rationnel. Il existe des grammaires non régulières qui engendrent des langages réguliers [1].

1. $G = (\{a\}, \{S\}, S, P = \{S \rightarrow aSa \mid \epsilon\})$ par exemple

Classification des grammaires Une grammaire formelle est donnée par (Σ, V, S, P) avec $P \subseteq (\Sigma \cup V)^* \times (\Sigma \cup V)^*$. La hiérarchie de **Chomsky** classe les grammaires suivant ce qu'il est possible de spécifier dans les règles de production $\alpha \to \beta$ de P.

Grammaire arbitraire	$\alpha, \beta \in (\Sigma \cup V)^*$
Grammaire contextuelle	$\alpha, \beta \in (\Sigma \cup V)^*$ et $\lvert\alpha\rvert < \lvert\beta\rvert$
Grammaire algébrique	$\alpha \in V$, $\beta \in (\Sigma \cup V)^*$
Grammaire régulière	$\alpha \in V$, $\beta \in \Sigma V \cup \{\epsilon\}$

Les grammaires algébriques restent insuffisantes pour spécifier tous les aspects des langages de programmation (comme le problème du nombre de paramètres $a^n b^m c^n d^m$).

Arbres de dérivation Soit $G = (\Sigma, V, S, P)$ une grammaire algébrique. Un arbre de dérivation relatif à G est un arbre dont les nœuds sont étiquetés par des symboles de $\Sigma \cup V \cup \{\epsilon\}$ et dont les fils de tout nœud interne sont ordonnés. De plus, il vérifie que :

- la racine est étiquetée par l'axiome de la grammaire,
- les nœuds internes sont étiquetés par des non-terminaux,
- les feuilles sont étiquetées par des terminaux ou ϵ,
- un nœud interne et ses fils correspondent à une règle de production.

Construction d'un arbre de dérivation à partir d'une dérivation :

- On part d'un arbre n'ayant qu'un seul nœud, la racine étiquetée par l'axiome.
- pour une dérivation en un pas $\alpha \to_G \beta$, il existe un non-terminal A dans α et une production $A \to \delta$ dans P tels que la dérivation remplace A par δ dans α (en produisant β). Si A est le $i$ème symbole de α, alors on étend l'arbre en ajoutant à sa $i$ème feuille les fils correspondant à la règle $A \to \delta$.

Exemple : $G = (\{i, +, *, (,)\}, \{E\}, E, P)$ avec $P = \{E \to E + E \mid E \to E * E | (E) | i\}$.
Dériver $i + i$ et $i + i * i$.
Proposition Soit $G = (\Sigma, V, S, P)$ une grammaire algébrique. Un mot w de Σ^* appartient à $L(G)$ si et seulement si il existe un arbre de dérivation dont les feuilles lues de gauche à droite forment le mot w.

Nous avons vu que lorqu'on construit une dérivation d'un mot w, nous pouvons choisir à chaque pas le non-terminal X dans u et la règle de production de membre gauche X ($X \to \ldots$). Il existe donc en général plusieurs dérivations pour un même mot u.
Lorsque l'on construit un arbre de dérivation, seul le choix des règles de production aura un impact sur l'arbre final. L'ordre selon lequel on choisit les non-terminaux n'aura aucun impact sur l'arbre final. Plusieurs dérivations correspondent donc au même arbre de dérivation.
Sur l'exemple $G = (\{i, +, *, (,)\}, \{E\}, E, P)$ avec $P = \{E \to E + E \mid E \to E * E | (E) | i\}$, pour le mot $w = i + i$, on a les 2 dérivations

- $E \to_G E + E \to_G i + E \to_G i + i$
- $E \to_G E + E \to_G E + i \to_G i + i$

Mais il n'y a qu'un arbre de dérivation correspondant à ces deux dérivations. Un arbre de dérivation regroupe ainsi plusieurs dérivations *équivalentes* d'un même mot.

Dérivations gauches Une dérivation gauche est une dérivation dans laquelle on choisit à chaque pas de dériver le non-terminal le plus à gauche. Soient $u, v \in (\Sigma \cup V)^*$, on note $u \xrightarrow{g}_G v$ si il existe $l \in \Sigma^*$, $r \in (\Sigma \cup V)^*$ et une règle de production $\alpha \to \beta$ tels que $u = l\alpha r$ et $v = l\beta r$. On notera $\xrightarrow{g}{}^*_G$ la clôture réflexive et transitive de $\xrightarrow{g}_G$.

Le correspondance entre dérivations gauches et arbres de dérivation est *bijective*. Cela vient du fait qu'une dérivation gauche correspond à un parcours dans l'ordre préfixe de l'arbre de dérivation. On utilise la dérivation gauche pour calculer l'arbre de dérivation d'un mot.

Ambiguité

- Un mot $m \in L(G)$ est *ambigu* s'il est le feuillage d'au moins deux arbres de dérivations différents.
- Une grammaire G est *ambiguë* s'il existe un mot m de $L(G)$ tel que m soit ambigu.
- Un langage algébrique L est *ambigu* si toute grammaire G engendrant L est ambiguë.

Exemple : la grammaire $G_1 = (\{-, i\}, \{E\}, E, \{E \to E - E \mid i\})$ est-elle ambiguë ?
Oui, le mot $i - i - i$ a deux arbres de dérivation . Or le langage $L(G_1)$ n'est pas ambigu.
Pour lever l'ambiguité de la grammaire, on peut ajouter des parenthèses :
$G_1' = (\{-, i\,(,)\}, \{E\}, E, \{E \to (E - E) \mid i\})$. G_1' n'est pas ambigüe, mais $L(G_1') \neq L(G_1)$.
Une meilleure solution consiste à introduire une priorité :
$G_2 = (\{-, i\}, \{E\}, E, \{E \to E - i \mid i\})$. G_2 n'est pas ambigüe et $L(G_2) = L(G_1)$.
De même $G_3 = (\{-, i\}, \{E\}, E, \{E \to i - E \mid i\})$ est aussi non-ambiguë et vérifie $L(G_3) = L(G_1)$.
De manière générale, pour rendre non ambigüe une grammaire, on utilise des propriétés sémantiques du langage, comme les priorités sur les opérateurs.
Exemple : expressions arithmétiques

$$G = (\{i, +, *, (,)\}, \{E\}, E, \{E \to E + E \mid E * E \mid (E) \mid i\})$$

La grammaire est ambiguë. Le langage n'est pas ambigu : la multiplication $*$ est prioritaire sur l'addition $+$. On "repousse" les dérivations de l'opérateur $*$ au fond de l'arbre en introduisant des règles termes T et facteurs F supplémentaires.

$$G' = (\{i, +, *, (,)\}, \{E, T, F\}, E, P') \text{ avec } P' = \begin{cases} E \to E + T \mid T & \text{– somme de termes} \\ T \to T * F \mid F & \text{– produit de facteurs} \\ F \to (E) \mid i \end{cases}$$

Il existe cependant des langages algébriques qui sont inhéremment ambigus.
$L = \{a^n b^m c^p \mid n = m \vee m = p\}$ est ambigu (il y a deux arbres de dérivation pour $a^n b^n c^n$) mais est un langage algébrique car engendré par les règles
$\{S \to X \mid Y, X \to AB, Y \to CD, A \to Ac \mid \epsilon, B \to aBb \mid \epsilon, C \to aC \mid \epsilon, D \to bDc \mid \epsilon\}$.

Les langages de programmation ne doivent pas être ambigus. De plus, montrer que :

- une grammaire est ambiguë est en général *facile* : exhiber un mot ayant deux arbres de dérivation différents.
- une grammaire est non-ambiguë est en général *difficile*.
- un langage est non-ambigu est en général *facile* : trouver un grammaire non-ambiguë qui l'engendre.
- un langage est ambigu est en général très *difficile* : montrer que toutes les grammaires l'engendrant sont ambiguës.

Transformation des grammaires

A partir d'une grammaire algébrique G, on construit une grammaire G' telle que

- G' ne comporte pas de "variables inutiles" (grammaire *réduite*).
- G' ne comporte pas de "règles de production inutiles" (grammaire *propre*)
- $L(G') = L(G) \setminus \{\epsilon\}$

De plus, cette transformation peut être *automatisée* (sous la forme d'un programme).

Variables inutiles

Une variable est *inutile* si c'est une variable "puits" ou une variable "inacessible".

— X est une variable *non-puits* si $\{u \in \Sigma^* \mid X \to_G^* u\} \neq \emptyset$.
— X est une variable *accessible* (depuis l'axiome S) si il existe $w_1, w_2 \in (\Sigma \cup V)^*$ tels que $S \to_G^* w_1 X w_2$.

Si X n'est pas une variable "non-puits", alors elle est dite "puits".
Si X n'est pas une variable "accessible", alors elle est dite "inacessible".
Variable puits. On construit l'ensemble NP des variables non-puits d'une grammaire G itérativement de la façon suivante :

— $NP_0 = \emptyset$,
— pour tout $i \in \mathcal{N}$, $NP_{i+1} = NP_i \cup \{X \in V \mid X \to w \in P \text{ et } w \in (\Sigma \cup NP_i)^*\}$.

On définit $NP = \bigcup_{i \in \mathcal{N}} NP_i$.
On peut remarquer que pour tout i, $NP_i \subseteq NP_{i+1} \subseteq V$, et que si $NP_i = NP_{i+1}$ alors pour tout entier k, $NP_i = NP_{i+k}$. On en déduit que l'on fait au plus $Card(V)$ itérations.
On élimine toutes les règles de production contenant une variable puits (en membre droit ou gauche).
Variables accessibles. On construit l'ensemble AC des variables accessibles d'une grammaire G itérativement de la façon suivante :

— $AC_0 = \{S\}$, où S est l'axiome de G.
— pour tout $i \in \mathcal{N}$, $AC_{i+1} = AC_i \cup \{X \in V \mid \exists Y \to w_1 X w_2 \in P \text{ avec } Y \in AC_i\}$.

On définit $AC = \bigcup_{i \in \mathcal{N}} AC_i$. On remarque que pour tout i, $AC_i \subseteq AC_{i+1} \subseteq V$ et que, si $AC_i = AC_{i+1}$, alors pour tout entier k, $AC_i = AC_{i+k}$. On en déduit que l'on fait au plus $Card(V)$ itérations.
On élimine toutes les règles de production contenant une variable inaccessible en membre gauche.
A noter que :

— éliminer les variables inaccessibles ne peut pas créer de variables puits.
— éliminer les variables puits peut créer des variables inaccessibles

Règles inutiles

Les règles de production inutiles sont constituées des règles qui permettent à partir d'un non-terminal X de :

— dériver le mot vide ϵ : $X \to_G^* \epsilon$,
— dériver un non-terminal Y : $X \to_G^+ Y$.

Elimination des règles "$X \to_G^* \epsilon$". On pose $V_\epsilon = \{X \in V \mid X \to_G^* \epsilon\}$, l'ensemble des non-terminaux pouvant se dériver en ϵ. L'ensemble V_ϵ se calcule itérativement de la façon suivante :

— $V_\epsilon^0 = \{X \in V \mid X \to \epsilon \in P\}$
— pour tout $i \in \mathcal{N}$, $V_\epsilon^{i+1} = V_\epsilon^i \cup \{X \in V \mid X \to w \in P \text{ et } w \in (V_\epsilon^i)^+\}$.

On définit $V_\epsilon = \bigcup_{i \in \mathcal{N}} V_\epsilon^i$. On remarque que pour tout i, $V_\epsilon^i \subseteq V_\epsilon^{i+1} \subseteq V$ et que, si $V_\epsilon^i = V_\epsilon^{i+1}$ alors pour tout entier k, $V_\epsilon^i = V_\epsilon^{i+k}$. On en déduit que l'on fait au plus $Card(V)$ itérations.
On considère H, la grammaire donnée par les règles de production $\{X \to \epsilon \mid X \in V_\epsilon\}$. On construit alors à partir des règles de production P un nouvel ensemble de règles de production P' défini par

$$P' = \{X \to v \mid \text{il existe } X \to u \in P,\ u \to_H^* v \text{ et } v \neq \epsilon\}$$

Elimination des règles "$X \to_G^+ Y$" (chain rules). On commence par éliminer les règles $X \to_G^+ X$. Pour tout $X \in V$, on pose $R(X) = \{Y \in V \mid X \to_G^* Y\}$. On peut calculer $R(X)$ itérativement de la façon suivante :

— $R(X)^0 = \{X\}$
— pour tout $i \in \mathcal{N}$, $R(X)^{i+1} = R(X)^i \cup \{Y \in V \mid \exists\, Z \to Y \in P \text{ tq } Z \in R(X)^i\}$.

On définit $R(X) = \bigcup_{i \in \mathcal{N}} R(X)^i$. On remarque que pour tout i, $R(X)^i \subseteq R(X)^{i+1} \subseteq V$ et que, si $R(X)^i = R(X)^{i+1}$ alors pour tout entier k, $R(X)^i = R(X)^{i+k}$. On en déduit que l'on fait au

plus $Card(V)$ itérations pour une variable X. Puisqu'il y a $Card(V)$ variables, on fait en tout au plus $(Card(V))^2$ itérations.

Grammaires réduites et propres

Il existe un algorithme, qui donné une grammaire G, calcule une grammaire G' *réduite et propre* telle que $L(G') = L(G) \setminus \{\epsilon\}$. On procède par élimination *dans l'ordre...*

1. ... des variables puits de $G \Rightarrow G_1 : L(G) = L(G_1)$.
2. ... des variables inaccessibles de $G_1 \Rightarrow G_2 : L(G_1) = L(G_2)$.
3. ... des règles pour $X \rightarrow^*_{G_2} \epsilon$ de $G_2 \Rightarrow G_3 : L(G_3) = L(G_2) \setminus \{\epsilon\}$.
4. ... des règles pour $X \rightarrow^+_{G_3} Y$ de $G_3 \Rightarrow G' : L(G_3) = L(G')$.

Proposition Pour toute grammaire G réduite et propre d'axiome S, pour tout mot $w \in \Sigma^*$, pour tout $k \in \mathcal{N}$,

$$\text{si } S \rightarrow^k_G w \text{ alors } k \leq |w|$$

Forme normale de Greibach

Une grammaire est récursive à gauche si elle contient des dérivations de la forme $A \rightarrow^+ A\beta$. On distingue la récursivité à gauche

- immédiate : la grammaire contient un non-terminal A et une règle de production $A \rightarrow A\alpha$ ($\alpha \in (\Sigma \cup V)^+$).
- générale : la grammaire contient un non-terminal A et il existe une dérivation $A \rightarrow^+_G A\alpha$ ($\alpha \in (\Sigma \cup V)^+$).

Elimination de la récursivité à gauche immédiate

On remplace les règles de la grammaire de la forme

$$A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$$

où

- $\alpha_i \in (\Sigma \cup V)^+$ et $\beta_j \in (\Sigma \cup V)^*$
- les β_j ne commencent pas par A

par les règles

$$A \rightarrow \beta_1 A' \mid \ldots \mid \beta_m A' \mid \beta_1 \mid \ldots \mid \beta_m$$
$$A' \rightarrow \alpha_1 A' \mid \ldots \mid \alpha_n A' \mid \alpha_1 \mid \ldots \mid \alpha_n$$

où A' est un nouveau non-terminal.

Elimination de la récursivité à gauche générale

- On ordonne les non-terminaux : $A_1, A_2, \ldots, A_n$
- On applique l'algorithme suivant :

```
Pour i allant de 1 à n faire
    Pour j allant de 1 à i − 1 faire
        on remplace la règle A_i → A_j α où A_j → β_1 | ... | β_k
         par la règle A_i → β_1 α | ... | β_k α.
    FinPour
    On élimine la récursivité à gauche immédiate
     pour toutes les règles de A_i.
FinPour
```

Cette méthode peut également s'exprimer sous forme matricielle et donc se programmer de manière systématique. Si la grammaire est propre, la mise sous forme normale de Greibach (qui permet d'ôter le récursivité gauche) fonctionne toujours.