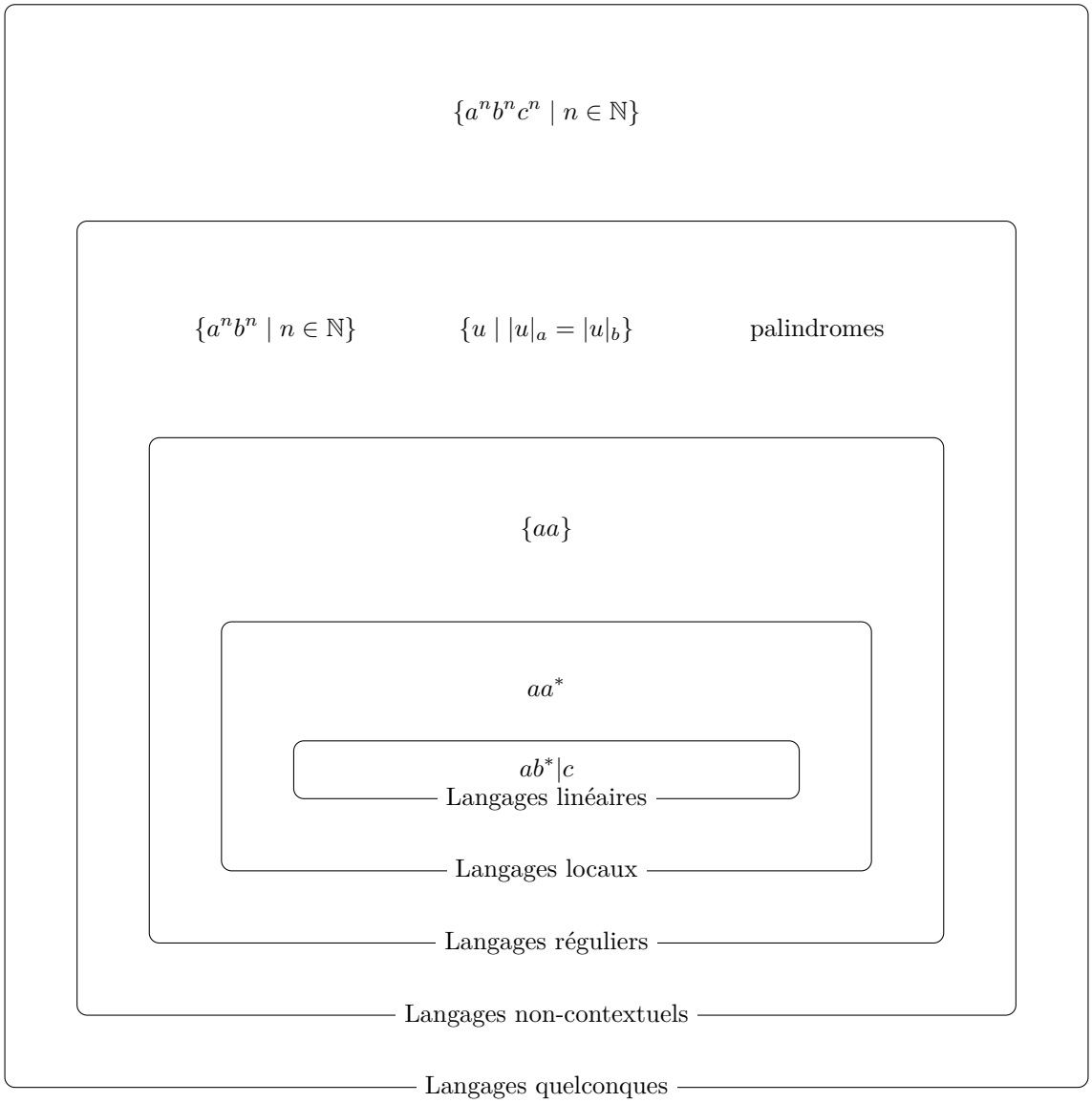


I Définitions

Définition : Grammaire non-contextuelle

Une grammaire non-contextuelle (ou : hors-contexte) est un quadruplet $G = (\Sigma, V, R, S)$ où :
- V est un ensemble fini de variables
- Σ est un alphabet fini de terminaux
- $R \subset V \times (V \cup \Sigma)^*$ est un ensemble fini de règles de production, chaque règle $(X, \alpha) \in R$ étant notée $X \to \alpha$
- $S \in V$ est le symbole initial

Remarques :

- Par convention, on note les variables en majuscules et les terminaux en minuscules.
- On peut noter $X \to \alpha \mid \beta$ au lieu de $X \to \alpha$, $X \to \beta$.
- Il existe des grammaires plus générales (contextuelles) mais nous nous limiterons aux grammaires non-contextuelles, qu'on appellera simplement grammaires.

Définition : Dérivation

Soient $\alpha, \beta \in (V \cup \Sigma)^*$.

- On note $\alpha \Rightarrow \beta$ s'il existe une règle $X \to \gamma$ telle que $\alpha = \alpha_1 X \alpha_2$ et $\beta = \alpha_1 \gamma \alpha_2$ avec $\alpha_1, \alpha_2 \in (V \cup \Sigma)^*$.
  On dit alors qu'on a une dérivation immédiate de α en β.
- On note $\alpha \Rightarrow^n \beta$ s'il existe des mots $\gamma_0 = \alpha, \gamma_1, \ldots, \gamma_n = \beta$ tels que $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n$.
  On dit alors qu'on a une dérivation de longueur n de α en β.
- On note $\alpha \Rightarrow^* \beta$ si $\alpha \Rightarrow^n \beta$ pour un certain $n \in \mathbb{N}$. On parle alors de dérivation de α en β.

Définition : Langage engendré

Soit $G = (\Sigma, V, R, S)$ une grammaire.

- On dit que G génère un mot $w \in \Sigma^*$ si $S \Rightarrow^* w$.
- L'ensemble $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$ est le langage engendré par G.
- Un langage L est dit non-contextuel (ou : algébrique, hors-contexte) s'il existe une grammaire hors-contexte G telle que $L = L(G)$.

Exemples :

- Soit $G = (\Sigma = \{a, b\}, V = \{S\}, R = \{S \to aaS \mid b\}, S)$.
  $L(G) =$ _____
- Soit $G = (\{a, b\}, \{S\}, \{S \to aSb \mid \varepsilon\}, S)$.
  $L(G) =$ _____
- Soit $G = (\{x, y, \top, \bot, \vee, \wedge, \neg\}, \{S\}, R, S)$ avec P contenant les règles suivantes : $S \to \top \mid \bot \mid x \mid y \mid \neg S \mid S \vee S \mid S \wedge S$.
  $L(G)$ est _____

Exercice 1.
Donner des grammaires engendrant les langages suivants sur $\{a, b\}$:

1. $L_1 = ab^*a$.
2. $L_2 =$ ensemble des mots dont la taille est un multiple de 3.
3. $L_3 =$ ensemble des mots ayant bbb comme facteur.
4. $L_4 =$ ensemble des expressions arithmétiques bien formées, comme $4 + 3 \times 2$.
5. $L_5 =$ ensembles des palindromes, c'est-à-dire des mots qui se lisent de la même façon de gauche à droite et de droite à gauche.
6. $L_6 =$ ensembles des mots qui ne sont pas des palindromes.

Méthode : Si G une une grammaire et L un langage, on peut montrer $L(G) = L$ par double inclusion.

- $L(G) \subset L$: montrer que si $S \Rightarrow^n u$ alors $u \in L$, par récurrence sur n.
- $L \subset L(G)$: montrer que si $u \in L$ alors $u \in L(G)$, par récurrence sur $|u|$.

On utilise alors souvent, implicitement, le théorème « évident » suivant :

Théorème

Soit $G = (\Sigma, V, R, S)$ une grammaire, $\alpha_1, \alpha_2, \beta \in (V \cup \Sigma)^*$ et $n \in \mathbb{N}$.
Si $\alpha_1\alpha_2 \Rightarrow^n \beta$ alors il existe $\beta_1, \beta_2 \in (V \cup \Sigma)^*$, $k, p \in \mathbb{N}$ tels que :

- $\beta = \beta_1\beta_2$
- $\alpha_1 \Rightarrow^k \beta_1$
- $\alpha_2 \Rightarrow^p \beta_2$
- $n = k + p$

Exercice 2.
Soit G la grammaire définie par les règles $S \to aSbS \mid bSaS \mid \varepsilon$.
Déterminer $L(G)$, en le démontrant.

II Langages non-contextuels et langages réguliers

Théorème

L'ensemble des langages non-contextuels est stable par union, concaténation et étoile.
C'est-à-dire : si L_1 et L_2 sont des langages non-contextuels alors $L_1 \cup L_2$, L_1L_2 et L_1^* sont des langages non-contextuels.

Preuve :

Remarque : les langages non-contextuels ne sont pas stables par intersection, différence, complémentaire.

Théorème

Tout langage régulier est non-contextuel.

Preuve :

Remarque : la réciproque est fausse, car $\{a^nb^n \mid n \in \mathbb{N}\}$ est algébrique mais pas régulier.

Définition : Grammaire régulière (HP)

Une grammaire est dite régulière (à droite) si chaque règle est de la forme $X \to aY$, $X \to a$ ou $X \to \varepsilon$.

Théorème

Un langage est régulier si et seulement s'il est engendré par une grammaire régulière.

Preuve :

III Arbre de dérivation

Définition : Arbre de dérivation

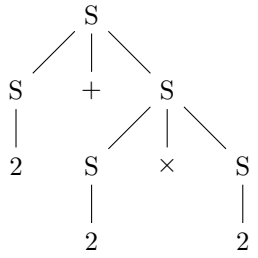
Soient $G = (\Sigma, V, R, S)$ une grammaire et $u \in L(G)$.
Un arbre de dérivation (ou : arbre syntaxique) pour u est un arbre tel que :

- la racine est étiquetée S
- chaque nœud interne est étiqueté par un élément de V
- chaque feuille est étiquetée par un élément de $\Sigma \cup \{\varepsilon\}$
- si un nœud interne est étiqueté X et possède n fils étiquetés $\alpha_1, \ldots, \alpha_n$, alors $X \longrightarrow \alpha_1 \ldots \alpha_n \in R$

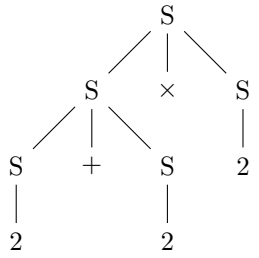
Remarque : les étiquettes des feuilles, lues de gauche à droite, forment le mot u.

Exemple : Soit G la grammaire définie par $S \to S + S \mid S \times S \mid 2$ avec $\Sigma = \{+, \times, 2\}$.
Il y a plusieurs façons d'engendrer $2 + 2 \times 2$, donnant des arbres de dérivation différents :

1. $S \Rightarrow S + S \Rightarrow 2 + S \Rightarrow 2 + S \times S \Rightarrow 2 + 2 \times S \Rightarrow 2 + 2 \times 2$
2. $S \Rightarrow S \times S \Rightarrow S + S \times S \Rightarrow 2 + S \times S \Rightarrow 2 + 2 \times S \Rightarrow 2 + 2 \times 2$



1.



2.

IV Grammaire ambiguë

Définition : Grammaire ambiguë

Soit $G = (\Sigma, V, R, S)$ une grammaire.

- On dit que u est ambigu pour G s'il existe plusieurs arbres de dérivation distincts pour u.
- On dit que la grammaire G est ambiguë s'il existe au moins un mot u ambigu pour G.

Exercice 3.
Montrer que les grammaires suivantes sont ambiguës :

1. $S \to S \mid \varepsilon$
2. $S \to aXb,\ X \to a \mid b \mid \varepsilon \mid XX.$

Attention : Si $S \to SaS \mid b$, bab peut être engendré de deux façons ($S \Rightarrow SaS \Rightarrow baS \Rightarrow bab$ et $S \Rightarrow SaS \Rightarrow Sab \Rightarrow bab$) mais n'est pas ambigü (les deux arbres de dérivation sont les mêmes).

Définition : Dérivation gauche

Soit $G = (\Sigma, V, R, S)$ une grammaire.
Une dérivation gauche pour u est une dérivation où, à chaque étape, on remplace la variable la plus à gauche.

Exemple : $S \Rightarrow S + S \Rightarrow 2 + S \Rightarrow 2 + S \times S \Rightarrow 2 + 2 \times S \Rightarrow 2 + 2 \times 2$ est une dérivation gauche pour $2 + 2 \times 2$.

Théorème

Soient G une grammaire et $u \in L(G)$. Il y a une bijection entre les dérivations gauches de u et les arbres de dérivation de u.

Preuve :

D'où :

Théorème

Soit G une grammaire et $u \in L(G)$.
u est ambigu pour G si et seulement si u possède plusieurs dérivations gauches.

Définition : Faible équivalence

Deux grammaires G_1 et G_2 sont dites faiblement équivalentes si $L(G_1) = L(G_2)$.

Remarque : le terme « faiblement » vient du fait qu'on ne demande que l'égalité sur les langages et pas sur les arbres de dérivation. Ainsi, une grammaire ambiguë peut être faiblement équivalente à une grammaire non ambiguë.

Exemple : la grammaire définie par $S \to S + S \mid S \times S \mid (S) \mid 2$ est ambiguë mais est faiblement équivalente à la grammaire non-ambiguë suivante :

$$\begin{aligned} S &\to S + T \mid T \\ T &\to T \times F \mid F \\ F &\to (S) \mid 2 \end{aligned}$$

On force ici la priorité des opérations.

Remarques :

- Il est indécidable de savoir si une grammaire est ambiguë, c'est-à-dire qu'il n'existe pas d'algorithme qui, étant donnée une grammaire, détermine si elle est ambiguë.
- Un langage non-contextuel qui ne peut être engendré que par une grammaire ambiguë est dit intrinsèquement ambigu.

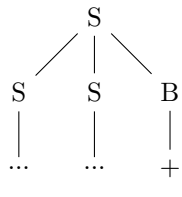
Exemple (*dangling else*) : la grammaire $S \to$ `if` S `else` S |`if` S |... est ambiguë car `if` ... `if` ... `else` ... peut être dérivé de deux façons.

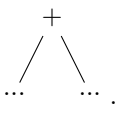
V Analyse syntaxique

La compilation d'un code source permet de passer d'un langage à un autre et comporte deux grandes étapes :

1. Analyse lexicale : découpe le code source en une liste de lexèmes (mots-clés, identifiants, opérateurs...) en vérifiant que le code est bien formé.
   Utilise un automate fini déterministe.
2. Analyse syntaxique (*parsing*) : construit l'arbre de dérivation du code source.
   Utilise un algorithme *top-down* ou *bottom-up*.

Exemple : Une expression arithmique postfixe (ou : polonaise inverse) consiste à écrire les opérateurs après les opérandes : $34+5\times$ au lieu $(3+4)\times 5$. Grammaire (non ambiguë) :

On simplifiera l'écriture d'un arbre syntaxique 

par 

.

On utilise le type suivant :

```
type token = I of int | B of char
type arbre = F of int | N of token * arbre * arbre
```

Ainsi, $34+5\times$ est représenté par :

- la liste de tokens `[I 3; I 4; B '+'; I 5; B '*']`
- l'arbre `N (B '*', N (B '+', F 3, F 4), F 5)`.

On parcourt la liste de tokens en utilisant une pile pour stocker les arbres en cours de construction.

- Si on rencontre un entier `k`, on ajoute un arbre feuille `F k` à la pile.
- Si on rencontre un opérateur `+`, on extrait les deux arbres `a1` et `a2` de la pile et on ajoute l'arbre `N(+, a1, a2)`.
- Quand on a fini de parcourir la liste, le seul arbre restant dans la pile est l'arbre de dérivation.

Exercice 4.
Écrire une fonction `parse : token list -> arbre` qui prend une liste de tokens et renvoie l'arbre de dérivation correspondant.
