

# Sur la vérification de preuve et la preuve automatique d'appartenance d'un mot à une grammaire.

Ulysse Durand

## 1 Définitions

### Les grammaires formelles

Une grammaire formelle est un quadruplet  $G = (T, N_t, S, D)$  où :

- $T$  est l'alphabet des terminaux
- $N_t$  est l'alphabet des non terminaux
- $S \in N_t$  est l'axiome
- $D$ , un ensemble d'éléments de  $(N_t \cup T)^* \times (N_t \cup T)^*$ , est l'ensemble des règles de dérivation.

Soit  $\rightarrow$  la relation binaire définie sur  $(N_t \cup T)^*$  par

$$\forall m, m' \in (N_t \cup T)^*, m \rightarrow m' \iff \exists (a, b) \in D, u, v \in (N_t \cup T)^* / m = uav \text{ et } m' = ubv \quad (1)$$

On note  $\xrightarrow{*}$  la cloture transitive et réflexive de  $\rightarrow$ .

Pour  $x \in (N_t \cup T)^*$ , notons

$$\delta(x) := \{y \in (N_t \cup T)^* / x \xrightarrow{*} y\} \text{ et}$$

$$\eta(x) := \delta(x) \cap T^*.$$

Alors le langage de la grammaire formelle  $G$  est le suivant :

$$\mathcal{L}(G) := \eta(S)$$

Nous allons supposer que  $N_t$  est dénombrable et  $T$  est fini.

## Les grammaires non contextuelle

**Définition :** Une grammaire non contextuelle est une grammaire  $G = (T, N_t, S, D)$  telle que :

$$\forall (a, b) \in D, a \in N_t$$

**Exemple :** la grammaire non contextuelle des expressions arithmétiques suffixes.

$$\begin{aligned} D = & \{(\underline{S}, \underline{SS+})_1, (\underline{S}, \underline{SS*})_2, (\underline{S}, \underline{"N"})_3\} \\ & \cup \{(\underline{N}, \underline{C})_4 | (\underline{N}, \underline{NC})_5\} \\ & \cup \{(\underline{C}, \underline{0})_6 | (\underline{C}, \underline{1})_7\} \end{aligned}$$

(les indices ci-dessus correspondent à une énumération des règles de dérivation, par exemple,  $D_5 = (\underline{N}, \underline{NC})$ )

## Fôret de dérivation

Ici,  $G = (T, N_t, S, D)$  est une grammaire non contextuelle.

Une forêt est une liste d'arbres

Une preuve qu'un mot appartient à  $\eta(m)$  peut être donnée par une forêt de dérivation :

Si  $m$  est le mot vide, il s'agit de la forêt vide.

Sinon, soit  $n = |m|$ ,

Si  $n = 1$  et  $m_0 \in T$ , il s'agit d'un seul arbre où la racine est  $m$  et la sous forêt est la forêt de dérivation d'un  $b$  tel que  $(m, b) \in D$ .

Si  $n = 1$  et  $m_0 \in N_t$ , il s'agit de l'arbre de racine  $m$  et de sous forêt vide.

Si  $n > 1$ , il s'agit de la liste des arbres  $[f_1, \dots, f_n]$  où  $\forall i \in [1, n], f_i$  est un arbre de dérivation de la lettre  $m_i$ .

L'ensemble des forêts de dérivation de  $m$  est noté  $\mathcal{F}(m)$ .

Le parcours infixe des feuilles d'une forêt de dérivation de  $x$  donne un mot de  $\eta(x)$ . Un exemple sera donnée dans le cas d'une grammaire non contextuelle.

Pour  $a \in \mathcal{F}(x)$ , on note  $\mathcal{I}(a)$  le parcours infixe des feuilles de  $a$ .

## Exemple de preuve

Une preuve que  $\underline{"10""11"+"0"*} \in \mathcal{L}(G)$  :

Lemme 1 :  $\underline{"10"} \in \eta(S)$

En effet,  $\underline{S} \rightarrow_3 \underline{"N"} \rightarrow_5 \underline{NC} \rightarrow_6 \underline{N0} \rightarrow_4 \underline{C0} \rightarrow_7 \underline{"10"}$

L'arbre de dérivation fait aussi office de preuve, bien plus concise :

Lemme 2 :  $\underline{''11''} \in \eta(S)$  Preuve avec l'arbre suivant

On peut alors, en combinant les deux arbres précédents, prouver que  $\underline{''10''''11''} +$   
 $\in \eta(S)$  :

Avec l'arbre suivant, on prouve enfin que  $\underline{''10''''11'' + ''0''^*} \in \delta(S) \cap T^* = \mathcal{L}(G)$

:

On remarque bien qu'un parcours infixe des feuilles de l'arbre donne  $\underline{''10''''11'' + ''0''^*}$

## 2 Vérification de preuve - grammaire non contextuelle

La forêt de dérivation nous donne une bonne manière de vérifier une preuve, car une preuve est caractérisé par une forêt de dérivation.

(en pratique, 'e sera `string` ).

Par induction on peut montrer  $m \in \eta(x) \iff$  il existe  $p$  une forêt de dérivation dont le parcours infixe des feuilles est  $m$  et les racines sont les lettres de  $x$ .

NB :  $A_{[p,q]}$  correspond à la forêt  $A$  restreinte à ses arbres d'indice  $i/p \leq i \leq q$  et  $A@B$  correspond à la concaténation des forêts  $A$  et  $B$ .

$$\begin{array}{|l}
 m \in \eta(x) \\
 \iff \exists n \in \mathbb{N} / \exists m_1, m_2, \dots, m_n \in (N_t \cup T)^* / x \rightarrow m_1 \rightarrow m_2 \rightarrow \dots \rightarrow m_n \text{ et } m_n \in T^* \\
 (2_n) \\
 \mathcal{P}(n) := ((2_n) \implies \exists A \in \mathcal{F}(x) / \mathcal{I}(x) = m) \\
 \\
 \left| \begin{array}{l}
 \mathcal{P}(0) : x \in \eta(x) \implies x \in T^* \text{ donc } [B(x_1); \dots; B(x_{|x|})] \text{ convient.} \\
 \forall n \in \mathbb{N}^*, \mathcal{P}(n-1) \implies (m \in \eta(m_1) \implies \exists A \in \mathcal{F}(m_1) / \mathcal{I}(A) = m) \\
 \\
 \text{et comme } x \rightarrow m_1, \exists (a, b) \in D, u, v \in (N_t \cup T)^* / x = uav \text{ et } m_1 = ubv \\
 \\
 \text{Soit } p := |u|, q := |v|, r := |b| \\
 \\
 \text{alors } A|_{[1,p]} \in \mathcal{F}(u), A|_{[p+1,p+r]} \in \mathcal{F}(b), A|_{[p+r+1,p+r+q]} \in \mathcal{F}(v) \\
 \\
 \text{Donc } A|_{[1,p]} @ A' @ A|_{[p+r+1,p+r+q]} \in \mathcal{F}(x) \text{ où } A' = N(b, A|_{[p+1,p+r]}) \\
 \\
 \text{Soit } \mathcal{P}(n).
 \end{array} \right.
 \end{array}$$

La réciproque se montre avec un algorithme qui à un arbre de dérivation associe une séquence  $m_1, m_2, \dots, m_n \in (N_t \cup T)^*$  telle que  $x \rightarrow m_1 \rightarrow m_2 \rightarrow \dots \rightarrow m_n$

En entrée, nous avons  $f_0$ , une forêt de dérivation.

$f$  est une copie de  $f_0$ .

Tant que  $f$  n'est pas constituée que de feuilles :

On prend  $a$  un arbre qui n'est pas une feuille dans  $f$

Remplacer  $a$ , dans  $f$ , par ses fils.

Lire les racines de la forêt de gauche à droite.

En remplaçant  $a$  dans  $f$  par ses fils, on dérive notre mot une fois.

Alors  $a \in \mathcal{F}(S)$  est une preuve que  $\mathcal{I}(a) \in \mathcal{L}(G)$ .

Alors, vérifier une preuve donnée par une forêt consiste juste à vérifier que la forêt est bien une forêt de dérivation.

### 3 Preuve automatique - grammaire non contextuelle

Toujours dans une grammaire non contextuelle

Avec une grammaire dont l'axiome est  $S$ , pour générer automatiquement une preuve, l'idée est qu'à chaque étape, on dispose d'une liste de mots prouvés, et à chaque étape, on fait toutes les dérivations possibles de notre liste de mots prouvés pour obtenir une liste de mots prouvés plus grande.

Deux approches coexistent alors, soit on fait toutes les dérivations possibles dans le sens direct, soit dans le sens indirect.

#### La méthode bottom-up

On part des  $\{(b_1, []); \dots; (b_n, [])\} \mid (a, b) \in D \text{ et } b \in T^*, n = |b|\}$  et on construit d'autres arbres de dérivation à partir des règles de dérivation.

Par la méthode bottom-up, on générerait dans l'ordre :  
à l'étape 0 : à l'étape 1 : à l'étape 2 : à l'étape 3 :

On prouve par induction simple qu'à l'étape  $k$ , on a généré tous les arbres de dérivations possibles de profondeur  $\leq k$ .

#### La méthode top-down

On part de l'axiome  $S$  de la grammaire non contextuelle pour construire des mots de  $\delta(S)$  en remplaçant un nonterminal  $x$  d'un mot par un  $b$  d'une règle de dérivation  $(x, b)$ .

Cette méthode suit d'assez près la définition des mots de la grammaire non contextuelle

(on peut écrire la clôture transitive et réflexive  $\xrightarrow{*}$  comme :  $\xrightarrow{*} = \bigcup_{n \in \mathbb{N}} \rightarrow^n$ , avec le

produit usuel de deux relation binaires  $\forall \mathcal{R}, \mathcal{R}' \in X^2, \mathcal{R}\mathcal{R}' := \{(x, z) \in X^2 \mid \exists y \in X / (x, y) \in \mathcal{R} \text{ et } (y, z) \in \mathcal{R}'\}$ .

On génère en effet, avec cette méthode, à la  $n$ -ième étape,

$$A_n = \left\{ m \in (N \cup T)^* \mid S \left( \bigcup_{k \in [1, n]} \rightarrow^k \right) m \right\}.$$

Il est alors simple de vérifier que tout mot du langage de la grammaire sera généré en temps fini par cette méthode.

On génère alors dans l'ordre :

à l'étape 0 :  $\{S\}$ ,

à l'étape 1 :  $\{S, SS+, SS*, "N"\}$ ,

et à l'étape 2 :  $\{S, SS+, SS*, "N", SS * S+, SSS * +, SSS + *, SS + S*, "N" S+, S" N" +, "N" S*, S" N" *, "C", "NC"\}$ .

Cette méthode est bien plus proche de la définition et est directement applicable aux grammaire formelles.

## 4 Vérification de preuve - grammaire formelle

### Une nouvelle structure de données pour les preuves

Nous ne disposons plus des forêts de dérivation. Alors nous pouvons nous restreindre à la définition d'un mot qui dérive d'un autre. On va alors fournir, comme preuve, une liste de mots  $m_1, \dots, m_n$  tels que  $\forall i \in [1, n - 1], m_i \rightarrow m_{i+1}$ . Pour ceci nous allons renseigner de quelle manière le mot  $m_{i+1}$  dérive du mot  $m_i$  par la règle de dérivation  $(a, b)$ , par exemple en fournissant les indices délimitant  $a$  dans  $m_i$ , et  $b$  le remplaçant. Nous aboutissons alors à :

```
type 'e preuveformelle = (int*int*('e caractere list)) list
```

Nous prendrons comme exemple la grammaire formelle  $G = (T, N_t, S, D)$  où :

$T = \{\underline{a}, \underline{b}, \underline{c}\}$

$N_t = \{\underline{S}, \underline{B}\}$

$D = \{(\underline{S}, \underline{aBSc})_1, (\underline{S}, \underline{abc})_2, (\underline{Ba}, \underline{aB})_3, (\underline{Bb}, \underline{bb})_4\}$ ,

alors aabbcc est dans  $\mathcal{L}(G)$  car  $\underline{S} \rightarrow_1 \underline{aBSc} \rightarrow_2 \underline{aBabcc} \rightarrow_3 \underline{aaBbcc} \rightarrow_4 \underline{aabbcc}$ .

### Vérifier une preuve dans une telle structure de données

## 5 Preuve automatique - grammaire formelle

Chercher automatiquement une preuve d'appartenance d'un mot à une grammaire non contextuelle se fait déjà très bien avec les automates à pile. Nous allons ici nous intéresser plus spécifiquement aux grammaires formelles, et réutiliser notre méthode top-down. La méthode s'applique alors directement de la même manière.

Nous avons donc un algorithme qui permet théoriquement de prouver que n'importe quel mot de la grammaire appartient bien à cette dernière. Le problème étant la complexité de ce dernier, qui fonctionne comme une machine non déterministe où on ne coupe pas les instances arrivant à un état puit. La suite va donc consister à trouver, de manière heuristique ou non, des méthodes affirmant qu'à partir d'un mot  $m$ , on aura des difficultés à dériver en notre mot  $m'$  à prouver. Alors, nous allons arrêter les recherches de dérivations de  $S$  en  $m'$  qui passent par le mot  $m$ .

Pour ce faire, pour dériver un mot  $m$  en mot  $m'$ , on calcule  $S(x) := \{x \in (N \cup T)^* \mid m \rightarrow x\} = \{ubv \in (N_t \cup T)^* \mid \exists(a, b) \in D / \exists(u, v) \in (N_t \cup T)^* / x = uav\}$ , puis, pour chaque  $y \in S(x)$ , on cherche une dérivation de  $y$  en  $m'$ .

Pour calculer  $S(x)$ , il suffit de faire une recherche de facteur.

## Un parcours en largeur particulier

Nous allons utiliser un parcours qui nous permettra de trouver dans un graphe un chemin plus court entre un sommet donné et un sommet 'valide'. Ce parcours pourra aussi éviter les chemins passant par un sommet 'invalide'.

Alors on fournira à ce parcours une fonction `elimine` et une fonction `termine` toutes deux de type `sommet -> bool`.

Ce parcours est donc un parcours en largeur depuis un sommet  $x_0$  qui garde en mémoire le chemin parcouru et qui s'arrête dès qu'un sommet  $s$ , tel que `termine s`, est parcouru. Il renvoie alors un chemin de  $x_0$  à  $s$ . Lorsque le parcours passe par un sommet  $s$  tel que `elimine s`, il ne parcourt pas ensuite ses voisins.

En voici une implémentation OCaml :

## Enfin, un prouveur automatique d'appartenance d'un mot à une grammaire formelle

Il ne nous reste plus qu'à appliquer notre parcours sur un graphe où les sommets sont les mots de  $(N_T \cup T)^*$ , et les arêtes sont  $\{(a, b) \in (N_T \cup T)^* \mid a \rightarrow b\} = \{(a, b) \in (N_T \cup T)^* \mid b \in S(a)\}$ .

Par la suite, nous allons apporter des améliorations, en fournissant des fonctions `elimine`.

## 6 Amélioration pour les grammaires croissantes

Une grammaire croissante est une grammaire telle que :

$$\forall(a, b) \in D, |a| \leq |b| \quad (2)$$

On a alors une première propriété très simple,  $\forall m, m' \in \delta(S), m \xrightarrow{*} m' \implies |m| \leq |m'|$ .

Alors, dans la recherche de dérivations de  $S$  vers  $m$ , on peut supprimer les "branches de recherche" qui partent d'un mot de longueur  $> |m|$ . Il ne reste qu'à faire le même parcours que précédemment avec la fonction suivante comme fonction `elimine`.

## 7 Amélioration dans le cas général : déduction sur le nombre d'occurrence de chaque lettre

Rappelons, le problème : étant donné un mot  $m$  d'une grammaire, comment le dériver en un mot  $m'$  ? Pour tout  $l \in N \cup N_t$ , on va chercher un ensemble  $q(l)$  qui majore  $|\delta(m)|_l$ .

Ainsi,  $m \xrightarrow{*} m' \implies |m'|_l \in q(l)$ , la contraposée nous sera utile :

$$|m'|_l \notin q(l) \implies \neg(m \xrightarrow{*} m')$$

Ce qui pourra nous permettre de réduire notre champ de recherche. En effet,

si pour un mot  $x \in \delta(m)$ ,  $\forall l \in (T \cup N_t)$ ,  $|\delta(x)|_l \cap q(l) = \emptyset$ , alors  $m' \notin \delta(x)$ .

Pour calculer  $|\delta(x)|_l \cap q(l)$ , nous allons utiliser un graphe semblable à un automate.

Les sommets sont des états de  $Q \subset (\mathcal{P}(\mathbb{N}))^{T \cup N_t}$  tels que :

$$\begin{aligned} &\forall q, q' \in Q, \exists l \in (T \cup N_t) / q(l) \cap q'(l) = \emptyset \\ &\text{et } \forall m \in \delta(S), \exists q \in Q / \forall l \in (T \cup N_t), |m|_l \in q(l) \end{aligned}$$

Ainsi à tout mot  $x \in \delta(S)$ , on peut associer un unique état  $q$  tel que  $\forall l \in (N_t \cup T)$ ,  $|x|_l \in q(l)$

Les arêtes du graphe sont les dérivation possibles d'une famille majorante  $q$  à une autre  $q'$ .

$$\forall (q, q') \in Q^2, (q, q') \in A \iff \exists x, x' \in (N_t \cup T)^* / x \rightarrow x' \text{ et } \forall l \in N_t \cup T, |x|_l \in q(l) \text{ et } |x'|_l \in q'(l)$$

Ainsi, si  $m$  correspond à un état  $q$ , et  $m'$  à un état  $q'$ , alors  $m \xrightarrow{*} m' \implies q'$  est accessible depuis  $q$ .

**Les états  $q$  sous la forme  $q \in Q = \{\{0\}, \mathbb{N}^*\}^{(N_t \cup T)}$**

Exemple avec la grammaire suivante :  $D = \{(S, \underline{abc})_0, (\underline{abc}, \underline{ab})_1, (\underline{b}, \underline{k})_2, (\underline{c}, \underline{ak})_3, (\underline{kak}, \underline{aa})_4, (\underline{a}, \underline{aaa})_5\}$

Le graphe est le suivant :

Et voici ce à quoi correspondent les différents états (sommets) du graphe :

Le graphe réduit suivant sera utile :

Ainsi, nous pouvons tout de suite affirmer qu'il est impossible de dériver aaabakab en akkckckaaakck (en effet,  $\psi$  n'est pas accessible depuis  $\delta$ )

Intéressons nous au calcul des arrêtes du graphe, qui ne requiert pas une complexité phénoménale.