1 TIPE Sur la vérification de preuve et la preuve automatique

L'ensemble des propositions démontrables de mathématiques est le langage d'une grammaire contextuelle, et c'est sur ce principe que nous allons construire un vérificateur de preuve, puis un prouveur automatique.

2 Introduction

2.1 Les grammaires non contextuelles

Une grammaire non contextuelle 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 $(N_t \cup T)^\star \times (N_t \cup T)^\star$, est l'ensemble des règles de dérivation.

Soit $\to$ la relation binaire définie sur $(N_t \cup T)^\star$ par

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

On note $\overset{*}{\to}$ la cloture transitive et réflexive de $\to$.
Pour $x \in (N_t \cup T)^\star$, notons

$\delta(x) := \{y \in (N_t \cup T)^\star / x \overset{*}{\to} y\}$ et

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

Alors le langage de la grammaire G est le suivant :

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

2.2 Idée : les propriétés vraies comme dérivation d'une grammaire non contextuelle.

Comme grammaire non contextuelle, on peut compter celle d'une logique formelle mais ce qui nous intéresse vraiment, c'est la sémantique de notre logique. Alors on peut proposer une grammaire non contextuelle prenant comme axiome un symbole S, et des règles de dérivation (S, a) pour tout axiome a de notre système axiomatique. Nous ajouterons aussi les règles d'inférence aux règles de dérivation.

Alors n'importe quel mot du langage de la grammaire est une déduction d'axiomes par les règles d'inférence. Ce sont donc les propositions démontrables.

2.3 Définitions utiles

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 une lettre, il s'agit d'une forêt avec un seul arbre étant l'arbre où la racine est m et les sous arbres sont la forêt de dérivation d'un b tel que $(m, b) \in D$.

Si m est un mot, $n = |m|$, il s'agit de la concaténation des forêts $f_1, \ldots, f_n$ où $\forall i \in [1, n]$, f_i est une forêt de dérivation de 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.

2.4 Premier cas : limitons nous à une logique sans règle de remplacement (grammaire non contextuelle).

Définition : Une grammaire non contextuelle est une grammaire $G = (N_t \cup T)^\star$ telle que :

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

Exemple : la grammaire des expressions arithmétiques suffixes.

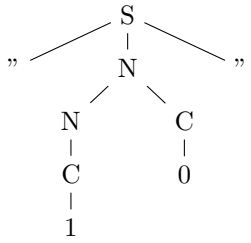
$$\begin{aligned} D =& \{(S, SS+)_1, (S, SS*)_2, (S, "N")_3\} \\ & \cup \{(N, C)_4 | (N, NC)_5\} \\ & \cup \{(C, 0)_6 | (C, 1)_7\} \end{aligned}$$

(les indices dessus correspondent à une énumération des règles de dérivation, par exemple, $D_5 = (N, NC)$)

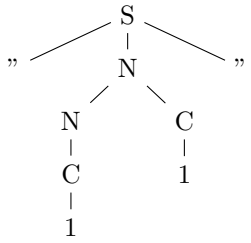
Une preuve que "10""11"+"0"* $\in \mathcal{L}(G)$:
Lemme 1 : "10" $\in \eta(S)$

En effet, $S \to_3$ "N" $\to_5$ "NC" $\to_6$ "$N0$" $\to_4$ "$C0$" $\to_7$ "10"

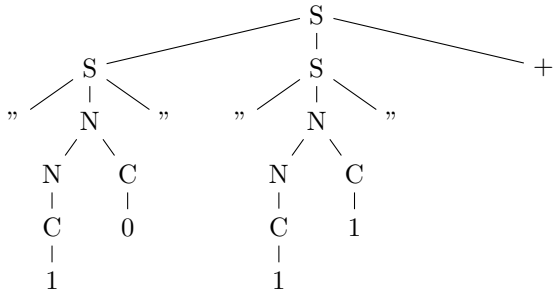
L'abre de dérivation fait aussi office de preuve, bien plus concise :



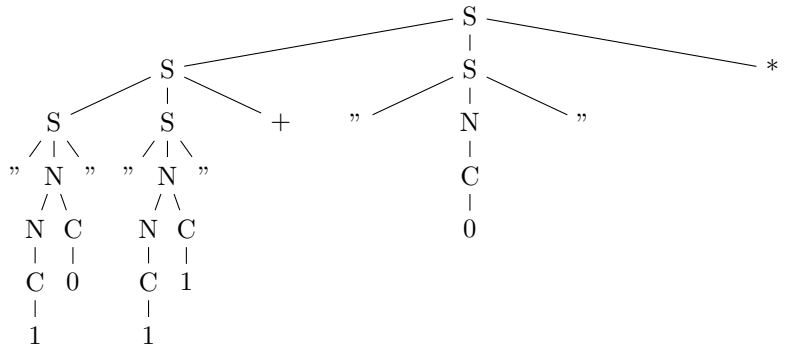
Lemme 2 : "11" $\in \eta(S)$ Preuve avec l'arbre suivant



On peut alors, en combinant les deux arbres précédants, prouver que "10""11"+ $\in \eta(S)$:



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



On remarque bien qu'un parcours infixe des feuilles de l'arbre donne "10""11"+"0"*

3 Vérification de preuve

On peut faire un programme qui vérifie la validité de notre arbre de dérivation.
Dans la suite nous allons supposer que N_t est dénombrable et T fini.

```
type 'e caractere = T of 'e | Nt of int

type 'a regle = ('a ) * ('a list);;

type 'e preuve =
    B of ('e caractere)  |
    P of ('e caractere) * ('e preuve list) * ('e caractere) regle

type 'e grammairemoche = {
    terminaux : 'e array ;
    nonterminaux : 'e array ;
    axiome : 'e caractere ;
    regles : ('e caractere) regle array
};;
```

(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.

$$
\begin{aligned}
& m \in \eta(x) \\
& \qquad \iff \exists n \in \mathbb{N}, m_1, m_2, \ldots, m_n \in (N_t \cup T)^\star / x \to m_1 \to m_2 \to \cdots \to m_n \text{ et } m_n \in T^\star \\
& \mathcal{P}(n) := \implies \exists A \in \mathcal{F}(x) / \mathcal{I}(x) = m
\end{aligned}
$$

$\mathcal{P}(0) : x \in \eta(x) \implies x \in T^\star$ donc $[B(x_1); \ldots; B(x_{|x|})]$ convient.
$\forall n \in \mathbb{N}^*, \mathcal{P}(n-1) \implies (m \in \eta(m_1) \implies \exists A \in \mathcal{F}(m_1)/\mathcal{I}(m_1) = m)$

et comme $x \to m_1, \exists (a,b) \in D, u, v \in (N_t \cup T)^\star / x = uav$ et $m_1 = ubv$

Soit $p := |u|, q := |v|, r := |b|$

alors $A|_{[1,p]} \in \mathcal{F}(u), A|_{[p+1,p+r]} \in \mathcal{F}(b), A|_{[p+r+1,p+r+q]} \in \mathcal{F}(v)$

Donc $A|_{[1,p]} @ A' @ A|_{[p+r+1,p+r+q]} \in \mathcal{F}(x)$ où $A' = N(b, A|_{[p+1,p+r]})$

Soit $\mathcal{P}(n)$.

TODO :

Faire la reciproque

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

Pour vérifier qu'une preuve $P(x, l, r)$ est valide, il n'y a que quelques étapes :

- Vérifier que chaque arbre de la forêt l est une preuve valide
- Vérifier que le nonterminal du membre de gauche de la règle de dérivation est bien x
- Vérifier que la dérivation appartient bien à la grammaire

D'où la fonction suivante :

```
let rec verif_preuve g p =
    match p with
    |B c -> true
    |P (x,l,r) ->
            (
                List.for_all
                (verif_preuve g)
                l
            )
            &&
            (fst r = x)
            &&
            (Array.mem r g.regles)
```

4 Preuve automatique

4.1 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. On peut visualiser ce processus par un graphe qui grandit à chaque étape.

Deux approches coexistent alors,

La première est dite top-down où l'on part de l'axiome S de la grammaire 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).

L'autre est dite bottom-up où l'on part des $\{[B(b_1); \ldots; B(b_n)] | (a, b) \in D$ et $b \in T^\star, n = |b|\}$ et on construit d'autres arbres de dérivation à partir des règles de dérivation.

Alors par la méthode top-down, on générerait dans l'ordre :
à l'étape 0 : $\{S\}$,
à l'étape 1 : $\{S, SS+, SS*, "N"\}$,
et à l'étape 2 : $\{S, SS+, SS*, "N", SS*S+, SSS*+, SSS+*, SS+S*,$
$"N"S+, S"N"+, "N"S*, S"N"*, "C", "NC"\}$.

Par la méthode bottom-up, on générerait dans l'ordre :
à l'étape 0 : $\{0, 1\}$
à l'étape 1 : $\{0, 1\}$
à l'étape 2 : $\{0, 1\}$
à l'étape 3 : $\{"0", "1", 10, 01\}$

TODO :

dessiner les arbres expliquant mieux la methode bottom-up
faire la preuve que tout arbre de derivation de S est generable en temps fini avec la methode bottom-up
faire la preuve que tout arbre de derivation de S est generable en temps fini avec la methode top-down