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

2 Définitions

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 :

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

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

En OCaml :

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

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

type 'e csg = {
    terminaux : 'e array ;
    nonterminaux : 'e array ;
    axiome : 'e caractere ;
    reglesc : ('e caractere) reglec array
};;
```

2.2 Les grammaires contextuelles

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

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

```
type 'a reglenc = ('a list) * ('a list);;

type 'e cfg = {
    terminaux : 'e array ;
    nonterminaux : 'e array ;
```

```
    axiome : 'e caractere ;
    reglesnc : ('e caractere) reglenc array
};;
```

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, $D_5 = (N, NC)$)

2.3 Fôret de dérivation

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

Une forêt est une liste d'arbres

```
type 'a arbre = 'a * 'a foret and
type 'a foret = 'a arbre list
```

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 $n = 1$, il s'agit d'un seul arbre où la racine est m et chaque sous arbres est la forêt de dérivation d'un b tel que $(m, b) \in D$.
> >
> > Si $n > 1$, il s'agit de la liste des arbres $[f_1, \ldots, f_n]$ où $\forall i \in [1, n], f_i$ est un arbre de dérivation de la lettre m_i.

```
let test_foret_deriv f m gram = match m with
    |[] -> f = []
    |[x] ->
        List.exists
        (fun (a,b) -> x = a && test_foret_deriv f b gram)
        gram.reglesnc
    |t1::q1 ->
        match f with
            |[] -> false
            |t2::q2 -> (test_foret_deriv [t2] [t1]) && (test_foret_deriv q2 q1)
```

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

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

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.

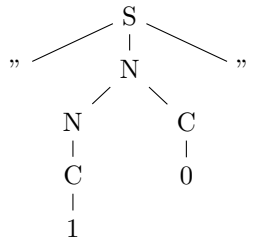
Pour $a \in \mathcal{F}(x)$, on note $\mathcal{I}(a)$ le parcours infixe des feuilles de a.

2.4 Exemple de preuve

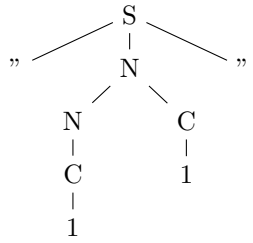
Une preuve que "10""11"+"0"* $\in \mathcal{L}(G)$:
Lemme 1 : "10" $\in \eta(S)$

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

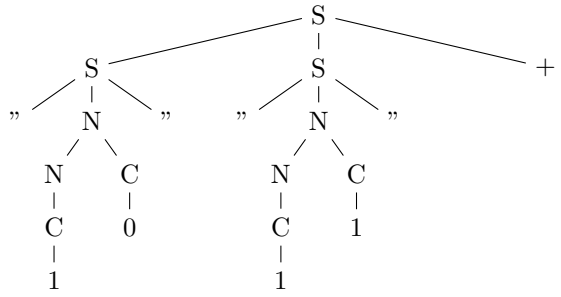
L'abre de dérivation fait aussi office de preuve, bien plus concise :



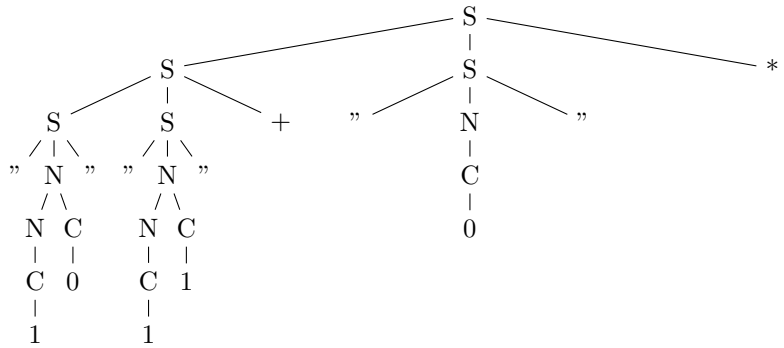
Lemme 2 : $"11" \in \eta(S)$ Preuve avec l'arbre suivant



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



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



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

3 Vérification de preuve pour une 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.

> $m \in \eta(x)$
> $\iff \exists n \in \mathbb{N} / \exists m_1, m_2, \ldots, m_n \in (N_t \cup T)^\star / x \to m_1 \to m_2 \to \cdots \to m_n$ et $m_n \in T^\star$ (2_n)
> $\mathcal{P}(n) := ((2_n) \implies \exists A \in \mathcal{F}(x) / \mathcal{I}(x) = m)$
>
> > $\mathcal{P}(0) : x \in \eta(x) \implies x \in T^\star$ donc $[B(x_1); \ldots; B(x_{|x|})]$ convient.
> > $\forall n \in \mathbb{N}^*, \mathcal{P}(n-1) \implies (m \in \eta(m_1) \implies \exists A \in \mathcal{F}(m_1) / \mathcal{I}(A) = m)$
> >
> > 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 $\mathcal{P}(n)$.

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

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_0
> >
> > 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.

4 Preuve automatique en 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.

4.1 La méthode bottom-up

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.

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

0 1

à l'étape 1 :

C C 0 1
| |
0 1

à l'étape 2 :

N N C C 0 1
| | | |
C C 0 1
| |
0 1

à l'étape 3 :

N N N N S S N N C C 0 1
/ \ / \ / \ / \ / | \ / | \ | | | |
N C N C N C N C " N " " N " C C 0 1
| | | | | | | | | | | |
C 0 C 1 C 0 C 1 C C 0 1
| | | | | | |
0 0 1 1 0 1

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

4.2 La méthode top-down

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).

Cette méthode suit d'assez près la définition des mots de la grammaire
(on peut écrire la clôture transitive et réflexive $\overset{*}{\to}$ comme : $\overset{*}{\to} = \bigcup_{n\in\mathbb{N}} \to^n$ où $\to^k$ est $\to$ composée k fois).

On génère en effet, avec cette méthode, à la n-ième étape, $A_n = \left\{ m \in (N \cup T)^\star \mid S \left(\bigcup_{k\in[1,n]} \to^k \right) m \right\}$.
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 contextuelles.

4.3 En grammaire contextuelle

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 contextuelles, 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.