INF431

Grammaires
Sujet proposé par Frédéric Magniez
Version: 1131:2574M

Introduction

L'objectif de ce sujet est de vous faire travailler sur la notion de grammaire qui est à la base des langages informatiques, et sur l'analyse syntaxique.

1 Pour commencer…

On considère la grammaire sur l'alphabet de symboles terminaux $\{a, b\}$, ayant comme unique symbole non-terminal le symbole de départ S, et composée des deux productions suivantes :

$$S \longrightarrow \epsilon \qquad S \longrightarrow aSb$$

Question 1 Montrer que S peut se récrire en $aaabbb$ en donnant l'arbre de production. ◇

On considère maintenant les langages d'alphabet $\{a, b\}$ qui vérifient les deux propriétés suivantes :

1. Le mot vide ϵ est dans le langage.
2. Si W est dans le langage alors aWb y est aussi.

Question 2 Donner trois langages distincts, dont le plus petit langage, qui satisfont ces deux propriétés. Montrer que la grammaire de la question 1 engendre exactement le plus petit langage. ◇

2 JavaScript Object Notation

JSON (JavaScript Object Notation) est un format de données textuel et générique. Il permet de représenter de l'information structurée. Un objet JSON est de trois types possibles (voir figure 1) :

— Une valeur terminale : un booléen, un nombre, une chaîne de caractères ou la constante `null` ;

— Un tableau d'objets, éventuellement vide, de la forme $[objet_1, \ldots, objet_k]$;

— Une liste associative : une liste de paires (nom, objet), éventuellement vide, de la forme $\{nom_1 : objet_1, \ldots, nom_k : objet_k\}$;

Question 3 Écrire la grammaire de ce langage, en supposant que l'on dispose des symboles terminaux `bool` (pour les booléens), `num` (pour les nombres), `string` (pour les chaînes de caractères), `null`, `lbrace` (« { »), `rbrace` (« } »), `lbrack` (« [»), `rbrack` («] »), `comma` (« , ») et `colon` (« : »).

3 Ambiguïté

Une grammaire est ambiguë si elle permet d'engendrer un même mot (sur l'alphabet des symboles non-terminaux) de deux façons différentes. L'ambiguïté est à éviter si possible car elle rend les analyseurs non déterministes et donc non efficaces.

```
{   "menu":
    {
        "id": "file",
        "value": "File",
        "popup":
        {
            "menuitem":
                [
                    { "value": "New", "onclick": "CreateNewDoc()" },
                    { "value": "Open", "onclick": "OpenDoc()" },
                    { "value": "Close", "onclick": "CloseDoc()" }
                ]
        }
    } }
```

FIGURE 1 – Exemple d'objet JSON

Comme nous allons le voir sur un exemple, il est parfois possible de transformer une grammaire ambiguë en une grammaire non-ambiguë équivalente. Considerer la grammaire suivante, qui est un fragment de la quasi-totalité des langages de programmation :

Instr	$\rightarrow$	`if` Expr `then` Instr
Instr	$\rightarrow$	`if` Expr `then` Instr `else` Instr
Instr	$\rightarrow$	`do_a` | `do_b`
Expr	$\rightarrow$	`x=0` | `y=0`

Question 4 Donner deux arbres de production pour une même expression. ◇

Le problème essentiel vient de la précédence, c'est à dire ici des règles de priorité. Dans le cas des langages de programmation, le `else` se rapporte en général au dernier `if` rencontré.

Question 5 Proposer une grammaire non-ambiguë pour ce langage. Vérifier qu'elle engendre le même langage. Argumenter, sans le prouver, que cette nouvelle grammaire est non-ambiguë. ◇

4 Algorithme d'analyse syntaxique CYK

En 1967, l'algorithme Cocke-Younger-Kasami (CYK) à base de programmation dynamique a été découvert. Un de ses avantages est qu'il fonctionne pour toute grammaire. Sa complexité est cubique en la taille de la séquence à analyser, et linéaire en le nombre de productions de la grammaire (si celle-ci est en forme normale de Chomsky).

La première étape de l'algorithme CYK consiste à normaliser les grammaires, c'est-à-dire à les transformer en des grammaires équivalentes mais sous une forme contrainte, dite normale de Chomsky. Elle consiste à n'avoir que des productions de l'une des trois formes suivantes :

— $S \rightarrow \epsilon$, où S est le symbole non-terminal initial de la grammaire.

— $X \rightarrow YZ$, où Y et Z sont des symboles non-terminaux.

— $X \rightarrow a$, où a est un symbole terminal.

De plus, si la production $S \rightarrow \epsilon$ est présente, alors S ne peut apparaître dans le membre droit d'aucune production.

Question 6 Mettre la grammaire suivante sous forme normale de Chomsky :

$$S \rightarrow TbT \qquad T \rightarrow TaT \mid ca$$

Nous supposons maintenant donnée une grammaire $\mathcal{G}$ sous forme normale de Chomsky constituée de g productions.

Question 7 Montrer que $\mathcal{G}$ reconnait le mot vide ϵ si et seulement si $S \to \epsilon$ est une production de $\mathcal{G}$. Dans ce cas, donner une grammaire sous forme normale de Chomsky qui engendre le même langage que $\mathcal{G}$ privé du mot vide ϵ avec $(g-1)$ productions. $\diamond$

On suppose dorénavant que $S \to \epsilon$ n'est pas une production. Fixons une séquence $w := w_1 w_2 \dots w_n$ de n symboles terminaux, avec $n \geq 1$. L'analyse syntaxique consiste à décider si $S \to^* w$, c'est-à-dire si la séquence w peut se dériver de S par une série de productions de $\mathcal{G}$.

Vous avez vu en cours qu'il était possible de ramener ce problème de décision à la résolution d'un système d'équations booléennes positives. Lorsque la grammaire est sous forme normale de Chomsky, la résolution du système peut s'effectuer par programmation dynamique en considérant uniquement les variables booléennes $H_{i,k,X}$, pour chaque intervalle $[i,k[$ et non-terminaux X, avec $1 \leq i < k \leq n+1$, de sorte que $H_{i,k,X}$ est vraie si et seulement si $X \to^* w_i w_{i+1} \dots w_{k-1}$.

Question 8 Donner un algorithme pour décider en temps $\mathrm{O}(g \times n^3)$ si $S \to^* w$ dans $\mathcal{G}$. $\diamond$