Université de Nice-Sophia Antipolis — Licence d'Informatique — 3e année
Analyses — 2004–2005

TD N° 2

Analyse lexicale et Automates

Diagramme d'états et grammaire des aspects lexicaux

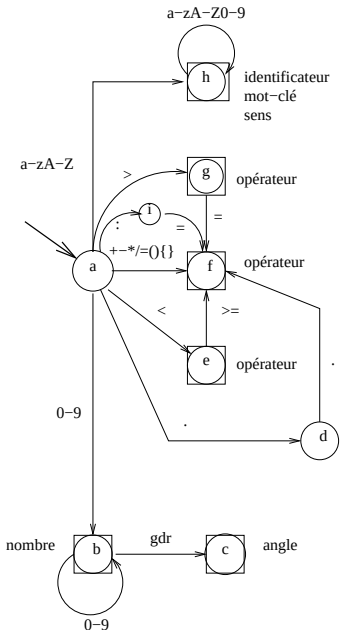
Dans les définitions ou descriptions des langages de programmation, les aspects lexicaux ne sont en général pas définis à l'aide d'expressions régulières ou de grammaires régulières. Quand ils ne sont pas donnés simplement sous la forme d'énumérations en langue naturelle, ils sont décrits sous la forme de grammaires algébriques, d'où il faut tirer ceux des non-terminaux qui sont des lexèmes. Les exemples du TD 01 le montraient bien.

Nous allons voir dans ce TD comment passer d'une telle description sous forme de grammaire algébrique à un automate d'états finis décrit sous la forme d'un diagramme. En fait, comme l'opération inverse est plus facile, c'est par celle-là que nous allons commencer.

Rappels sur les notations :

X ?	0 ou 1 fois X
X+	1 ou plusieurs fois X
X*	0 ou plusieurs fois X
X | Y	X ou Y
[abc]	ensemble des terminaux a, b et c

1. Voici le diagramme d'états qui vous a servi pour le TP 01. Pour ce langage inspiré de *Logo*, les mots-clés et sens sont reconnus par le même état de reconnaissance que les identificateurs.



Déduisez-en la grammaire algébrique qui vous permet de décrire les aspects lexicaux de ce langage.

Grammaire des lexèmes du langage à la Logo

Axiome = identificateur, mot-clé, sens, opérateur, angle et nombre

N = { identificateur, mot-clé, sens, opérateur, angle, nombre, chiffre, lettre, caractère-spécial }

T = { les chiffres, les lettres majuscules et minuscules, le plus, le moins, l'étoile, le slash, l'égal, l'inférieur, le supérieur, la parenthèse ouvrante et fermante, les accolades ouvrante et fermante, <>, <=, >=, le point-point, :=, tantque, si, sinon, pour, poser, lever, avancer, tourner }

P = {

```
identificateur →
lettre (lettre | chiffre)*
mot–clé → tantque | si | sinon | pour | poser | lever | avancer | tourner
sens → gauche | droite
opérateur →
caractère–spécial | <> | <= | >= | .. | :=
angle → nombre [dgr]
nombre → chiffre+
chiffre → 0 | 1 | ... | 9
lettre → a | ... | z | A | ... | Z
caractère–spécial →
+ | – | * | / | = | < | > | ( | ) | { | }
```

}

. . . . . .

2. La grammaire ci-dessous présente, de manière très simplifiée, les aspects lexicaux du langage *Ada*.

Grammaire des nombres *Ada*

Axiome = littéral_numérique

N = { littéral_numérique, littéral_décimal, entier, exposant, littéral_basé, base, entier_basé, chiffre_généralisé }

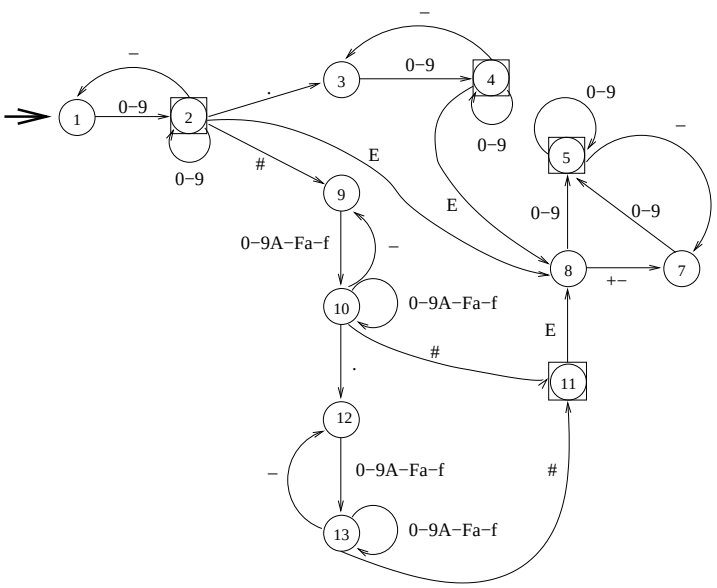
T = { les chiffres, les lettres, le souligné, le dièse, le plus, le moins, le point }

P = {

littéral_numérique →
littéral_décimal | littéral_basé
littéral_décimal →
entier (. entier)? exposant?
entier → chiffre (trait_bas? chiffre)*
exposant → E [+−]? entier
littéral_basé →
base # entier_basé (. entier_basé)? # exposant?
base → entier
entier_basé →
chiffre_généralisé (trait_bas? chiffre_généralisé)*
chiffre_généralisé →
chiffre | A–F | a–f

}

Construisez le diagramme d'états correspondant.



3. Même exercice pour le langage *Modula-2*.

Grammaire des nombres *Modula*

Axiome = nombre

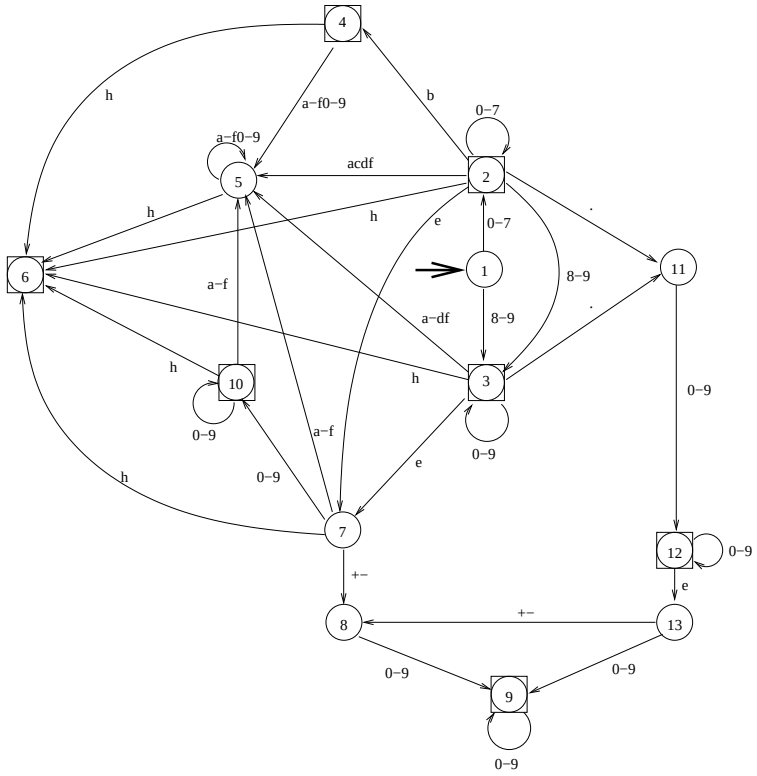
N = { nombre, entier, octal, décimal, hexadécimal, réel, fraction, exposant, zéroà7, hexa, chiffre }

T = { les chiffres, A-F, a-f, H, h, le plus, le moins, le point }

P = {

```
nombre → entier | réel
entier → octal | décimal | hexadécimal
octal → zéroà7+ [Bb]
décimal → chiffre+
hexadécimal → chiffre hexa* [Hh]
réel → décimal ((fraction? exposant) | (fraction exposant?))
fraction → . décimal
exposant → [Ee] [+−]? décimal
zéroà7 → 0 | 1 | ... | 7
hexa → chiffre | a | ... | f | A | ... | F
chiffre → 0 | 1 | ... | 9
```

}



. . . . . .