CS410 TD 1 & 2
Analyse lexicale et grammaires

Antoine Bouthors Christophe Deleuze

18 octobre 2006

1 Introduction

"Quelle phase détecte cette erreur ?"

Donner des exemples de programmes en C contenant des erreurs détectées par les différentes phases d'analyse (lexicale, syntaxique et sémantique), et des erreurs à l'exécution.

2 Analyse lexicale

2.1 Unités lexicales

Identifer les lexèmes qui dénotent les unités lexicales dans les programmes suivants. Donner des valeurs d'attributs raisonnables pour ces unités lexicales.

– Pascal :

```
function max( i, j : integer) : integer;

     { retourne le maximum des entiers i et j }
     begin
        if i > j then max := i
        else max := j
     end;
```

– C :

```
int max(i, j) int i, j;

     /* retourne le maximum des entiers i et j */
     {
        return i>j?i :j;
     }
```

– Fortran 77 :

```
FUNCTION MAX (I, J)

     C   RETOURNE LE MAXIMUM DES ENTIERS I ET J
            IF (I .GT. J) THEN
              MAX = I
            ELSE
              MAX = J
            END IF
         RETURN
```

2.2 Expressions régulières

– Construire une définition régulière pour les nombres non signés en Pascal. Ils sont de la forme `5280`, `39.67`, `6.336E64`, `1.89E-4`, `1.0`. `1.` n'est pas un nombre valide. Écrivez deux versions, sans et avec les opérateurs `+` et `?`.

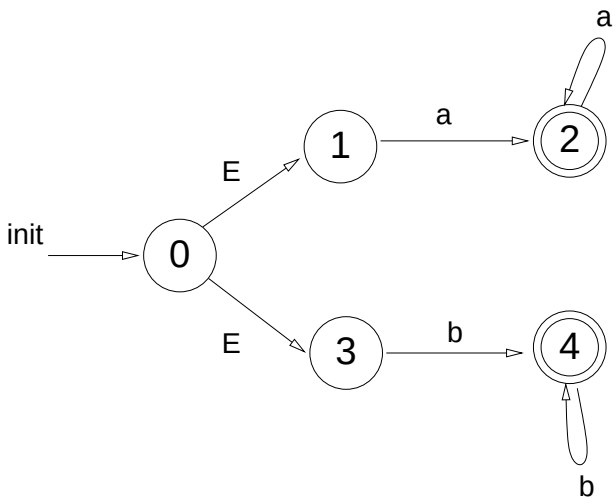

FIG. 1 – Automate fini non déterministe

– Donner une définition régulière de **id** correspondant à la définition informelle : *suite de lettres, chiffres et soulignements qui commence par une lettre, sans qu'il y ait deux soulignements consécutifs ni un soulignement final.*

2.3 Construction d'un AFN

– Ecrire la table de transition de l'automate fini non déterministe (AFN) de la figure 1.
– Construire l'AFN pour l'expression régulière `(a|b)*abb` avec la méthode de Thompson.

2.4 Simulation d'un AFN

– Décrire (peut être informellement) comment on peut simuler un AFN.
– Imaginer une méthode permettant de construire un AFD à partir d'un AFN.

2.5 Présentation de l'outil lex/flex

3 Grammaires et dérivation

3.1 Priorité des opérateurs

Soit la grammaire suivante :

E → *E A E* | (*E*) | id
A → + | *

– Montrer qu'elle est ambigüe par un exemple.
– Proposer une grammaire résolvant l'ambigüité.

3.2 Ambigüité du "sinon en suspens"

Soit la grammaire suivante :

instr → si *expr* alors *instr*
| si *expr* alors *instr* sinon *instr*
| autre

– Montrer qu'elle est ambigüe par un exemple.
– Trouver une grammaire supprimant cette abmigüité.

3.3 Dérivation d'une liste

Proposer une grammaire dérivant une suite d'instructions séparées par des points-virgules.

3.4 Ecriture d'une grammaire

Ecrire une grammaire pour le langage "paramètres d'une fonction C"