Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Grammaires algébriques
Récursivité à gauche

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Messages d'erreur avec numéro de ligne

```
> a.out <bench-source.c
syntax error near line 754
>
```

```
%{
#include <stdio.h>
int yylex();
void yyerror(char *);
extern int lineno;
%}
%%
(...)
%%
void yyerror(char *s){
   (...)
}
```

Dans le programme Flex

Déclarer et initialiser un compteur de lignes

Compter les lignes

Dans le programme Bison

Déclarer le compteur de lignes en **`extern`**

Redéfinir **`yyerror()`**

Respecter le type du paramètre : **`yyerror(char *s)`**

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Actions dans Bison

```
%token <ident> IDENT
%token <num> NUM
%%
e   : e '+' t      {printf("+");}
    | t
    ;
t   : t '*' f      {printf("*");}
    | f
    ;
f   : '(' e ')'
    | IDENT        {printf("%s", $1);}
    | NUM          {printf("%d", $1);}
    ;
```

On peut attacher des actions aux règles

L'analyseur syntaxique réalise l'action quand il a identifié tous les symboles du membre droit

Exemples d'actions

- Afficher des traces à l'écran
- Insérer des traces dans une copie de la donnée
- Traduire
- Signaler des erreurs sémantiques

Ordre de réalisation des actions

Exploration de l'arbre de dérivation en profondeur (*depth-first*)

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Grammaires équivalentes

E --> *E* + *E* | *E* - *E* | *E* * *E* | *E* / *E* | (*E*) | **N**

E --> *E* + *T* | *E* - *T* | *T*
T --> *T* * *F* | *T* / *F* | *F*
F --> (*E*) | **N**

Deux grammaires sont équivalentes si elles engendrent le même langage

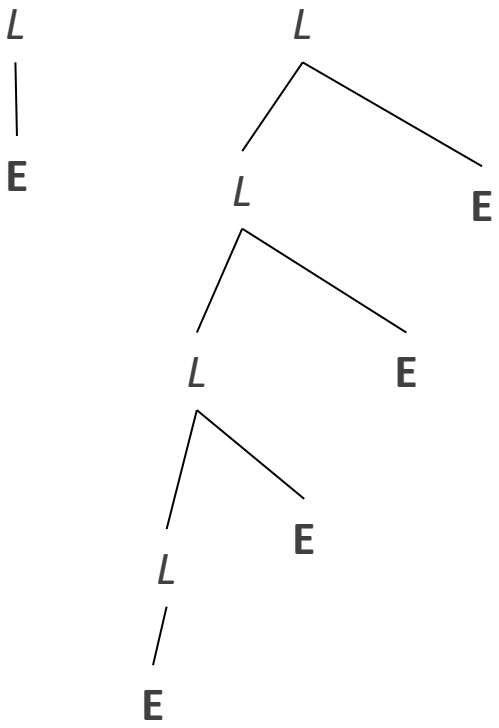
Pourquoi changer de grammaire si on garde le même langage ?

- pour éliminer les ambiguïtés
- pour la compatibilité avec l'analyseur
- pour la lisibilité

Les trois raisons ne vont pas toujours dans le même sens

Une grammaire pour les listes

L --> *L* **E** | **E**



Arbres en arête de poisson avec les feuilles à droite

Traduction en analyseur descendant

```
void L(void)          /* WRONG */
{
     if(second_next_token=='E') {
          L();
          match(next_token);
     }
     else match(next_token);
}
```

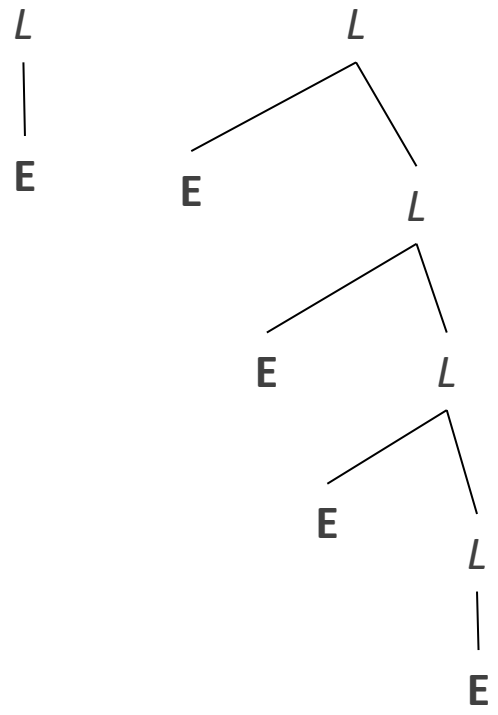
S'il y a plusieurs **E**, la fonction L() s'appelle récursivement sans que rien n'ait changé

Boucle infinie

Récursivité à gauche

Une autre grammaire pour les listes

L --> **E** L | **E**



Arbres en arête de poisson avec les feuilles à gauche

Traduction en analyseur descendant

```
void list(void)
{
      if(second_next_token=='E') {
            match(next_token);
            list();
      }
      else match(next_token);
}
```

Récursivité à droite

Récursivité à gauche

L --> *L* **E** | **E**

L --> **E** *L* | **E**

L'analyse descendante n'est pas applicable à la première grammaire en raison de la **récursivité à gauche**

La deuxième grammaire est récursive à droite : pas de problème

Définition

Une grammaire est récursive à gauche ssi

$\exists\, x \in V \;\; \exists\, w \in (A \mid V)^+ \;\; x \xrightarrow{*} xw$

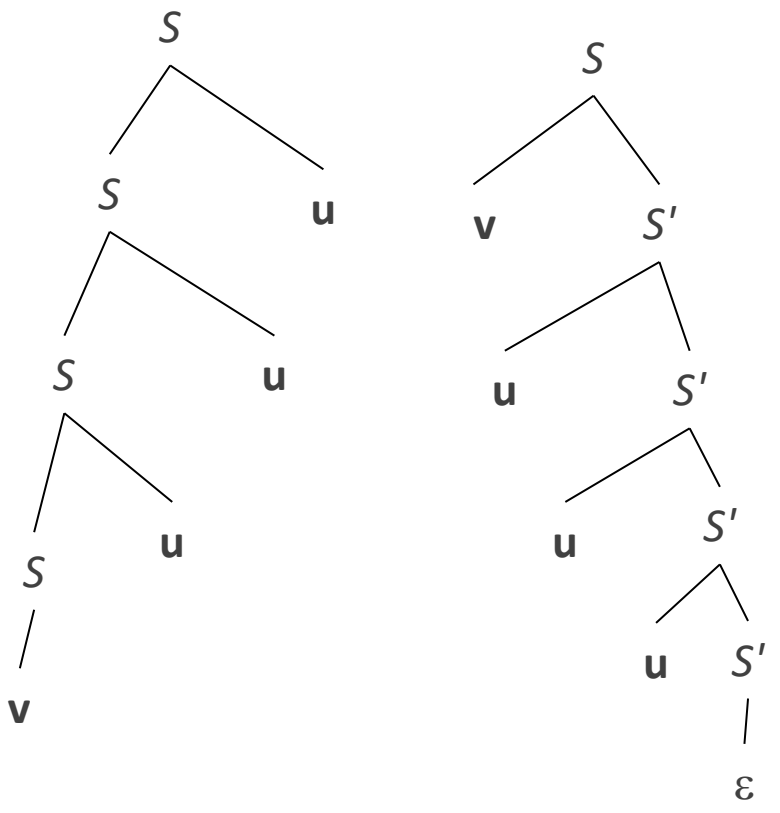
Forme normale de Greibach

Source: University of Waterloo, Canada

Pour toute grammaire algébrique telle que ε n'appartient pas au langage engendré, il existe une grammaire équivalente dont toutes les règles sont de la forme T --> **a** U_1 U_2 ... U_n, où **a** est un terminal, n ≥ 0 et les U_i sont des non-terminaux

Cette grammaire n'est pas récursive à gauche

Suppression de la récursivité à gauche



$S \rightarrow S\,\mathbf{u} \mid \mathbf{v}$

$$\begin{cases} S \rightarrow \mathbf{v}\,S' \\ S' \rightarrow \mathbf{u}\,S' \mid \varepsilon \end{cases}$$

Pour pouvoir faire l'analyse descendante

Remplacer une grammaire récursive à gauche par une grammaire équivalente qui ne l'est pas

Les deux grammaires ci-contre engendrent le même langage *vu**

Lemme d'Arden

$S \dashrightarrow S\,u \mid v$ $\qquad u \in A^+, v \in A^*$

Langage engendré : vu^*

$$\begin{cases} S \dashrightarrow v\,S' \\ S' \dashrightarrow u\,S' \mid \varepsilon \end{cases}$$

Langage engendré : vu^*

La nouvelle grammaire est équivalente et n'est pas récursive à gauche

Le raisonnement marche aussi dans des cas plus compliqués

Lemme d'Arden

Cas où la grammaire a aussi des règles dont le membre gauche n'est pas S

On regroupe d'abord toutes les règles dont le membre gauche est S

$S \dashrightarrow S\,u \mid v \qquad u \subseteq (A \mid V)^+ \quad v \subseteq (A \mid V)^*$

$S \overset{*}{\dashrightarrow} vu^*$

$S \dashrightarrow S\,u_1 \mid S\,u_2 \mid \ldots \mid S\,u_m \mid v_1 \mid v_2 \mid \ldots \mid v_n \qquad u_i \in (A \mid V)^+,\ v_i \in (A \mid V)^*$

$S \overset{*}{\dashrightarrow} (v_1 \mid v_2 \mid \ldots \mid v_n)(u_1 \mid u_2 \mid \ldots \mid u_m)^*$

$$\begin{cases} S \dashrightarrow v\,S' \\ S' \dashrightarrow u\,S' \mid \varepsilon \end{cases}$$

$S \overset{*}{\dashrightarrow} vu^*$

$$\begin{cases} S \dashrightarrow v_1\,S' \mid v_2\,S' \mid \ldots \mid v_n\,S' \\ S' \dashrightarrow u_1\,S' \mid u_2\,S' \mid \ldots \mid u_m\,S' \mid \varepsilon \end{cases}$$

$S \overset{*}{\dashrightarrow} (v_1 \mid v_2 \mid \ldots \mid v_n)(u_1 \mid u_2 \mid \ldots \mid u_m)^*$

Quelles sont les conditions pour que S ne soit pas récursif à gauche dans la nouvelle grammaire ?

Lemme d'Arden

$S \dashrightarrow S\,u \mid v$ $\quad u \subseteq (A \mid V)^+$ $\quad v \subseteq (A \mid V)^*$

$S \overset{*}{\dashrightarrow} vu^*$

$S \dashrightarrow v\,S'$

$S' \dashrightarrow u\,S' \mid \varepsilon$

$S \overset{*}{\dashrightarrow} vu^*$

Grammaire récursive à gauche

expr --> *expr* **+** *term*
expr --> *expr* **-** *term*
expr --> *term*
term --> **0**
term --> **1**
...
term --> **9**

Grammaire équivalente non récursive à gauche

expr --> *term expr'*
expr' --> **+** *term expr'*
expr' --> **-** *term expr'*
expr' --> *ε*
term --> **0**
term --> **1**
...
term --> **9**

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Lemme d'Arden

$S \dashrightarrow S\,u \mid v$ $\quad u \subseteq (A \mid V)^+$ $\quad v \subseteq (A \mid V)^*$

$S \overset{*}{\dashrightarrow} vu^*$

$$\begin{cases} S \dashrightarrow v\,S' \\ S' \dashrightarrow u\,S' \mid \varepsilon \end{cases}$$

$S \overset{*}{\dashrightarrow} vu^*$

$$\begin{cases} E \dashrightarrow E + T \mid E - T \mid T \\ T \dashrightarrow T * F \mid T / F \mid F \\ F \dashrightarrow (\,E\,) \mid \mathbf{N} \end{cases}$$

$$\begin{cases} E \dashrightarrow T\,E' \\ E' \dashrightarrow + T\,E' \mid - T\,E' \mid \varepsilon \\ T \dashrightarrow F\,T' \\ T' \dashrightarrow * F\,T' \mid / F\,T' \mid \varepsilon \\ F \dashrightarrow (\,E\,) \mid \mathbf{N} \end{cases}$$

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Lemme d'Arden

Forme matricielle du lemme d'Arden

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

ANTLR

```
NAME:           ('a'..'z')+ ;
ENDSYMBOL:      '.' ;
start_rule:     expression ENDSYMBOL ;
expression:     term ('+' term)* ;
term:           factor ('*' factor)* ;
factor:         NAME
              | LPAREN expression RPAREN ;
```

Générateur open-source de compilateurs en Java

Spécifier l'analyseur lexical et l'analyseur syntaxique dans le même fichier (extension .g)

Relax NG

```
<addressBook>
 <card email="js@example.com">
  <name>John Smith</name>
 </card>
 <card email="fb@example.net" >
  <name>Fred Bloggs</name>
 </card>
</addressBook>
```

Format de schémas XML

Qu'est-ce qu'un schéma XML ?

Une grammaire qui indique quelles balises sont valides et quelle syntaxe elles doivent respecter

Éléments

Attributs

Relax NG, syntaxe compacte

```
<addressBook>
 <card email="js@example.com">
  <name>John Smith</name>
 </card>
 <card email="fb@example.net">
  <name>Fred Bloggs</name>
 </card>
</addressBook>
```

```
element addressBook {
 element card {
  attribute email { text },
  element name { text },
  element note { text }?
 }*
}
```

Expression régulière approximativement équivalente :

```
<addressBook>
 (<card (email=".*")>
  <name>.*</name>
  (<note>.*</note>)?
 </card>)*
</addressBook>
```

Relax NG, syntaxe compacte

```
<addressBook>
  <card email="js@example.com">
    <name>John Smith</name>
    <note>check phone</note>
  </card>
  <card email="fb@example.net">
    <name>Fred Bloggs</name>
  </card>
</addressBook>
```

```
start = AddressBook
AddressBook = element addressBook { Card* }
Card = element card { Email, Name, Note? }
Email = attribute email { text }
Name = element name { text }
Note = element note { inline }
inline = (text
  | element bold { inline }
  | element italic { inline }
  | element span {
    attribute style { text }?,
    inline
    })*
```

Relax NG, syntaxe compacte

Grammaire algébrique approximativement équivalente

```
AddressBook --> <addressBook> Card*
    </addressBook>
Card --> <card Email> Name Note?
    </card>
Email --> email=".*"
Name --> <name>.*</name>
Note --> <note> inline </note>
inline --> text
    | <bold> inline </bold>
    | <italic> inline </italic>
    | <span (style=".*")?> inline </span>
```

```
start = AddressBook
AddressBook = element addressBook { Card* }
Card = element card { Email, Name, Note? }
Email = attribute email { text }
Name = element name { text }
Note = element note { inline }
inline = (text
  | element bold { inline }
  | element italic { inline }
  | element span {
      attribute style { text }?,
      inline
      })*
```