Compilers
Context Free Grammars

LEIC

FEUP-FCUP

2022

Compiler steps

Input program

↓

Lexical analysis

↓

list of *tokens*

↓

Parsing

↓

AST

↓

Semantic Analysis

↓

AST & Symbol Table

Code generation

↓

intermediate code

Instruction selection

↓

symbolic *assembly*

↓

Register allocation

↓

real *assembly*

↓

Assembler & linker

↓

executable code

This lecture

Syntactic Analysis

Context free grammars

Examples

Syntactic analysis

- Check that a program syntax is correct with respect to a given grammar e.g.:
  - open and closed brackets `{ }`, `( )` match
  - operators $+$, $*$, etc. respect their arity;
  - instructions end correctly `;`)
- Note that a sentence may have a correct syntax and still does not make sense
  Example (Chomsky, 1957): "*Colorless green ideas sleep furiously*"
- The (*parser*) builds an abstract syntax tree (AST) from a list of *tokens* (or outputs a syntax error)
- Main framewok: context free grammars

This lecture

Syntactic Analysis

Context free grammars

Examples

Context free grammars

A context free grammar $G = (\Sigma, N, S, P)$ is defined by:

Σ set of *terminal* symbols;

N set of *non-terminal* symbols;

$S \in N$ initial symbol;

P set of de *production rules* $X \to \alpha$ where:

- X is non-terminal;
- α is a sequence (maybe empty) of terminal or non-terminal symbols

Example

Terminal symbols:

$$\Sigma = \{a, b\}$$

Non-terminal symbols:

$$N = \{S, B\}$$

Initial symbol:

$$S$$

Production rules:

$$
\begin{aligned}
S &\rightarrow aSB \\
S &\rightarrow \varepsilon \\
S &\rightarrow B \\
B &\rightarrow Bb \\
B &\rightarrow b
\end{aligned}
$$

Derivations

A derivation relation $\Rightarrow$ replaces a non-terminal symbol by the right-hand side of its corresponding rule.

Example:

$$
\begin{align}
S &\rightarrow aSB \tag{1}\\
S &\rightarrow \varepsilon \tag{2}\\
S &\rightarrow B \tag{3}\\
B &\rightarrow Bb \tag{4}\\
B &\rightarrow b \tag{5}
\end{align}
$$

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{2}{\Rightarrow} aaBB \overset{4}{\Rightarrow} aaBbB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb$$

Language defined by a grammar

- Beginning with the initial symbol...
- expand the non-terminals using the corresponding production rules...
- when there only terminal symbols: we reach a *word* described by the grammar.

For the previous grammar G:

$$S \Rightarrow aSB \Rightarrow aaSBB \Rightarrow aaBB \Rightarrow aaBbB \Rightarrow aabbB \Rightarrow aabbb$$

Thus: $aabb \in L(G)$.

Language defined by a grammar (cont.)

Thus, if $G = (\Sigma, N, S, P)$ then

$$L(G) = \{w \in \Sigma^* : S \Rightarrow^* w\}$$

($\Rightarrow^*$ is the *transitive closure* of the derivation.)

Exercise

$$
\begin{aligned}
G: \quad S &\to aSB \\
S &\to \varepsilon \\
S &\to B \\
B &\to Bb \\
B &\to b
\end{aligned}
$$

Describe the language produced by G.

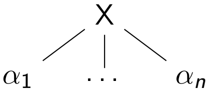
- Where may we have occurrences of a and b?
- What is the relation between the *number* of as and bs?

Syntax trees

Each production rule

$$X \to \alpha_1 \ldots \alpha_n$$

corresponds to a node with n sub-trees:

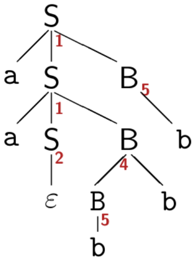


Syntax trees (cont.)

Example:

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{2}{\Rightarrow} aaBB \overset{4}{\Rightarrow} aaBbB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb$$

corresponds to the tree:



$$\begin{array}{llll} G: & S & \to aSB & (1) \\ & S & \to \varepsilon & (2) \\ & S & \to B & (3) \\ & B & \to Bb & (4) \\ & B & \to b & (5) \end{array}$$

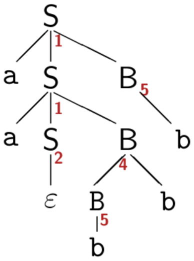
Syntax trees (cont.)

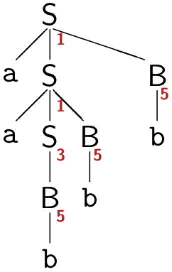
$S \Rightarrow^* aabbb$ may have two different derivations:

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{2}{\Rightarrow} aaBB \overset{4}{\Rightarrow} aaBbB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb \tag{6}$$

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{3}{\Rightarrow} aaBBB \overset{5}{\Rightarrow} aabBB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb \tag{7}$$

(6) corresponds to the tree on the left-hand side; (7) corresponds to the tree on the right-hand side.





$$G: \begin{array}{lll} S & \rightarrow aSB & (1) \\ S & \rightarrow \varepsilon & (2) \\ S & \rightarrow B & (3) \\ B & \rightarrow Bb & (4) \\ B & \rightarrow b & (5) \end{array}$$

Ambiguous grammars

A grammar is ambiguous if it produces words with different syntax trees.

G is ambiguous

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{2}{\Rightarrow} aaBB \overset{4}{\Rightarrow} aaBbB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb$$

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{3}{\Rightarrow} aaBBB \overset{5}{\Rightarrow} aabBB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb$$

because the two previous derivations correspond to two different syntax trees.

Ambiguous grammars

A grammar is ambiguous if it produces words with different syntax trees.

G is ambiguous

$$S \stackrel{1}{\Rightarrow} aSB \stackrel{1}{\Rightarrow} aaSBB \stackrel{2}{\Rightarrow} aaBB \stackrel{4}{\Rightarrow} aaBbB \stackrel{5}{\Rightarrow} aabbB \stackrel{5}{\Rightarrow} aabbb$$

$$S \stackrel{1}{\Rightarrow} aSB \stackrel{1}{\Rightarrow} aaSBB \stackrel{3}{\Rightarrow} aaBBB \stackrel{5}{\Rightarrow} aabBB \stackrel{5}{\Rightarrow} aabbB \stackrel{5}{\Rightarrow} aabbb$$

because the two previous derivations correspond to two different syntax trees.

Note:

- different derivations may correspond to the same syntax tree
- an ambiguous grammar must produce *different syntax tree* and not only *different derivations*

This lecture

Syntactic Analysis

Context free grammars

Examples

Arithmetic expressions

Non-terminals: E

Terminals (tokens): num $\quad + \quad * \quad (\quad)$

Production rules:

$$
\begin{aligned}
E &\rightarrow E{+}E \\
E &\rightarrow E{*}E \\
E &\rightarrow \text{num} \\
E &\rightarrow (E)
\end{aligned}
$$

Or...:

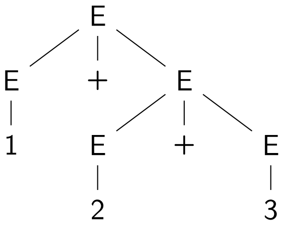
$$E \rightarrow E{+}E \;\mid\; E{*}E \;\mid\; \text{num} \;\mid\; (E)$$

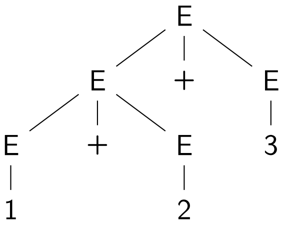
Arithmetic expressions (cont.)

- This grammar is *ambiguous*

Arithmetic expressions (cont.)

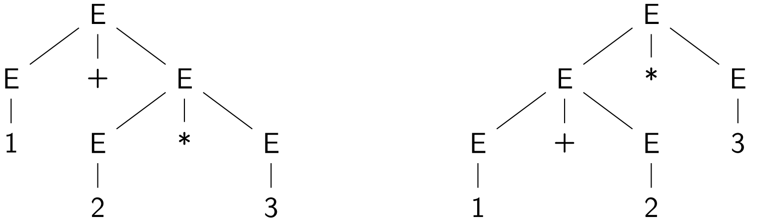
1+2+3:





Arithmetic expressions (cont.)

1+2*3:



How to eliminate ambiguity

For the previous example we must define:

- associativity properties

  left: 1+2+3 means $(1 + 2) + 3$

  right: 1+2+3 means $1 + (2 + 3)$

- a priority between + and $*$

  e.g. 1+2*3 means $1 + (2 \times 3)$ or $(1 + 2) \times 3$

How to eliminate ambiguity (cont.)

$$E \rightarrow E + T \qquad T \rightarrow T * F \qquad F \rightarrow \mathsf{num}$$
$$E \rightarrow T \qquad T \rightarrow F \qquad F \rightarrow (\ E\ )$$

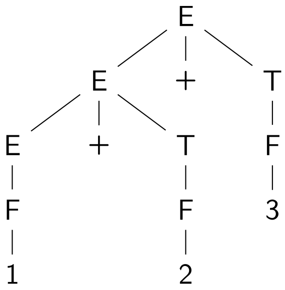
- In this grammar:

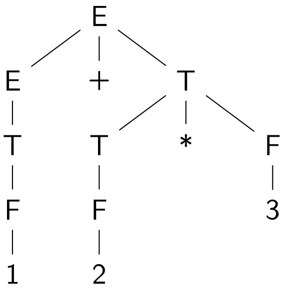
  expressions E are sums of *terms*;
  terms T are products of *factors*;
  factors F are constants or expressions between brackets.

- Productions of E and T with left recursion mean left associativity of $+$ and $*$

How to eliminate ambiguity (cont.)

Now 1+2+3 and 1+2*3 have unique syntaxt trees:





Example: sequences of statements

Non-terminals: S (*statements*) E (*expressions*)

Terminals (tokens): ident num = ( ) + , ; ++

Production rules:

$$
\begin{array}{ll}
S \to S \; ; \; S & E \to \text{ident} \\
S \to \text{ident} = E & E \to \text{num} \\
S \to \text{ident} \; {+}{+} & E \to E + E
\end{array}
$$

Example:

```
a = 17; b = 2
a = 0; (a++; b=a+5)
```

Example: sequences of statements (cont.)

Exercises:

1. Show that the previous grammar is ambiguous.
2. Rewrite the grammar to eliminate ambiguity.
   (Note: the problem is not only with expressions!)

"Dangling else"

if/then with optional *else*:

$$S \rightarrow \text{if } E \text{ then } S \text{ else } S$$
$$S \rightarrow \text{if } E \text{ then } S$$
$$S \rightarrow \textit{etc.}$$

Then

$$\text{if } e_1 \text{ then if } e_2 \text{ then } s_1 \text{ else } s_2$$

may have the two meanings:

$$\text{if } e_1 \text{ then } \{\text{if } e_2 \text{ then } s_1 \text{ else } s_2\} \tag{8}$$
$$\text{if } e_1 \text{ then } \{\text{if } e_2 \text{ then } s_1\} \text{ else } s_2 \tag{9}$$

Usually programming languages use (8): associate the *else* to the nearest *if*.

“Dangling else” (cont.)

Two new non-terminals: M (*matched statements*) and U (*unmatched statements*).

$$
\begin{aligned}
S &\rightarrow M \\
S &\rightarrow U \\[4pt]
M &\rightarrow \text{if } E \text{ then } M \text{ else } M \\
M &\rightarrow \textit{etc.} \\[4pt]
U &\rightarrow \text{if } E \text{ then } S \\
U &\rightarrow \text{if } E \text{ then } M \text{ else } U
\end{aligned}
$$

In practice: it may be better to solve these ambiguities in the parser implementation...