Syntax Analysis (Parsing)

Lexical analyser:

- Reads the source program
- Produces the sequence of tokens and delivers it to the parser (for syntax analysis)
- Tokens are represented by regular expressions

Syntax of a programming language:

- Expressed by a context free grammar
- Alphabet of the grammar - the set of tokens

Context free grammar:

- A finite way of describing the infinite number of strings
- Given by a start symbol and production rules
- String - a valid program of the source language

Derivation of a string s in grammar:

1. Let $A \to \gamma$ be a production rule
2. Let α, β be strings (with terminals and/or non-terminals)
3. Then $\alpha A \beta \Rightarrow \alpha\gamma\beta$ means "derives in one step"
4. $\gamma_1 \overset{*}{\Rightarrow} \gamma_2$ means "derives in zero or more steps"
5. $\gamma_1 \overset{+}{\Rightarrow} \gamma_2$ means "derives in one or more steps

1 Derivations

Grammar:

$$\begin{array}{lll} E \to E+T & T \to T\times F & F \to (E) \\ E \to T & T \to F & F \to x \\ & & F \to y \end{array}$$

Considering the string $(x+y)\times x$

$$\begin{aligned} E &\Rightarrow T \Rightarrow T\times F \Rightarrow F\times F \Rightarrow (E)\times F \Rightarrow (E+T)\times F \\ &\Rightarrow (T+T)\times F \Rightarrow (F+T)\times F \Rightarrow (x+T)\times F \\ &\Rightarrow (x+F)\times F \Rightarrow (x+y)\times F \Rightarrow (x+y)\times x \end{aligned}$$

Leftmost derivation - at each stage replace the leftmost non-terminal using a production rule

Rightmost derivation - at each stage replace the rightmost non-terminal using a production rule

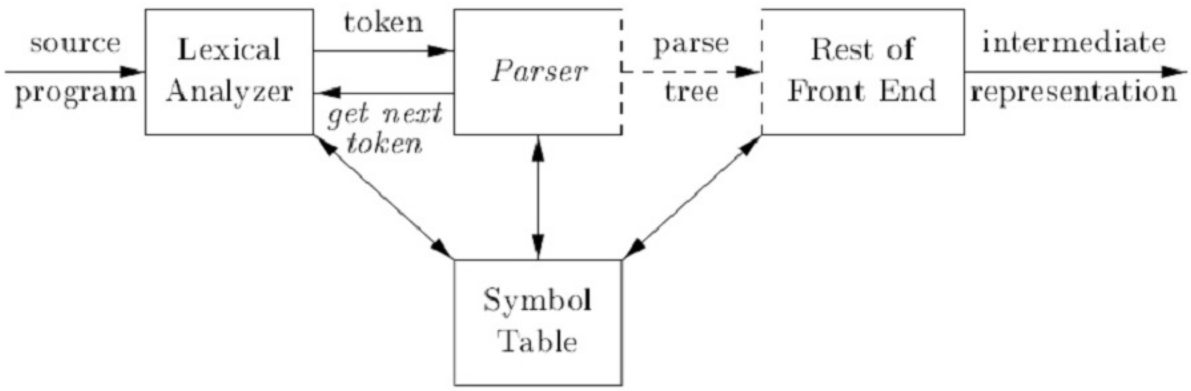
2 Parsing

Purpose of a parser:

- Given a string of tokens, to check whether it belongs to the language
- If yes, to find a derivation of this string in the grammar
- If not, to report useful syntax errors

For well formed strings of tokens (programs):

- The parser constructs a syntax tree (parse tree)
  - A graphical representation of the derivation of the string in the grammar
- The parse tree is passed to the next phase of the compiler (semantic analysis)



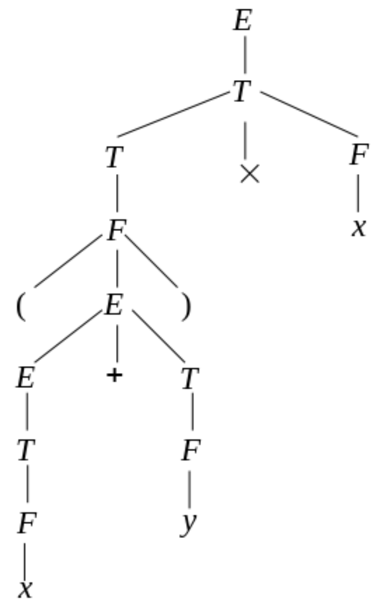
In the parse tree:

- Each internal node:
  - Is marked by a non-terminal
  - Represents the application of a production rule
- Each leaf is marked by a terminal
- All the leaves given the input string

Two main methods for constructing a parse tree:

- The top-down approach
  - Start from the root (labelled with the start symbol)
  - Continue down to the leaves
- The bottom-up approach
  - Start from the leaves
  - Continue up to the root

So creating a parse tree from the previous example looks like this



3 Ambiguity

Ambiguous grammar:

- If there is more than one parse tree for the same string
- Equivalently: there exists more than one leftmost (or rightmost) derivation of the same string

To prove a grammar is ambiguous just find a string of terminals that is produced by two parse trees

A string with two parse trees may have two meanings, therefore we need:

- To use additional rules to resolve ambiguities
- Or to design unambiguous grammars

Common occurrences here are things that are normally solved with BODMAS, e..g what is 9-5+2

3.1 Resolving ambiguity

Two solutions:

- Use diambiguating rules that "throw away" undesired parse trees
- Construct an equivalent unambiguous grammar

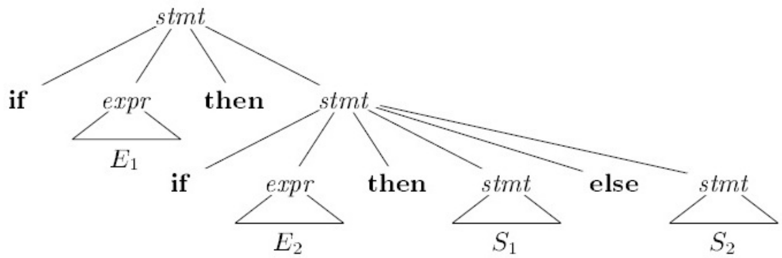
Example of disambiguating rules:

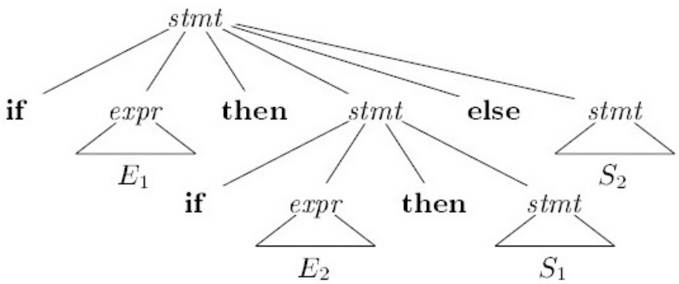
- Impose rules defining the relative precedence of operators when we have two different operators
- Operator * has higher precedence than +

The dangling else grammar

$$\begin{aligned} stmt \rightarrow\ & \mathbf{if}\ expr\ \mathbf{then}\ stmt \\ |\ & \mathbf{if}\ expr\ \mathbf{then}\ stmt\ \mathbf{else}\ stmt \\ |\ & \mathbf{other} \end{aligned}$$

Problem - when we read from left to right, which else matches with which else





Most languages prefer the first tree - match each else with the closest unmatched then

To rewrite this into an unambiguous grammar

- A statement appearing between a then and else must be ”matched”, i.e. it can’t end with an unmatched then
- A matched statement is
  - Either an if-then-else statement
  - Or any other unconditional statement

4 Abstract syntax tree

Recall

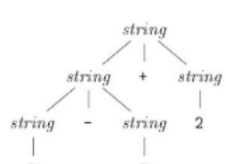
- The parse tree represents the steps of the derivation of a string
- Every internal node:
  - Is marked by a non-terminal
  - Represents the application of a production rule

Often parse trees are very complicated

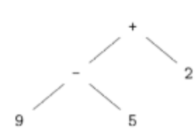
- Many non-terminals in the grammar:
  - Are auxillary non terminals
  - Do not represent operations.
- We need a simpler representation of string derivations

Abstract syntax tree:

- Much simpler than the parse tree
- Every internal node represents an operation and no a non-terminal



parse tree



abstract syntax tree (AST)

We can add annotations (or attributes) to the nodes of each tree

Attributed:

- Detailed information about semantics
- e.g. type information, location in memory ...