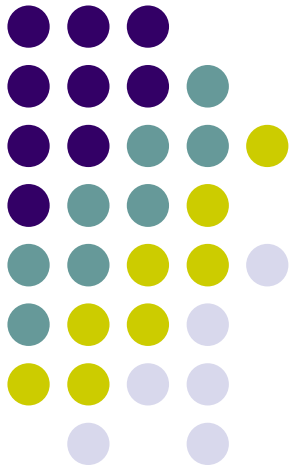
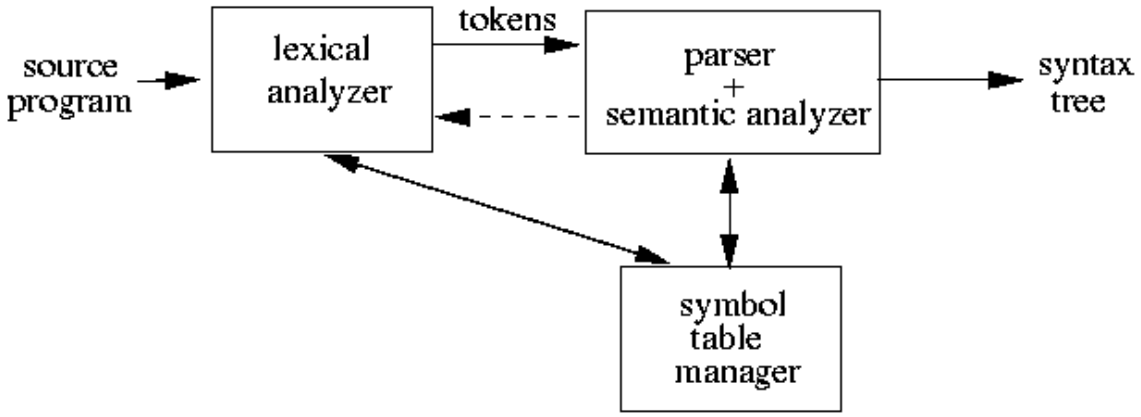
Syntax Analysis (Parsing)

Lecture - 5



Overview

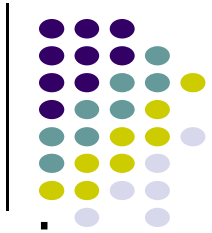


Main Task: Take a token sequence from the scanner and verify that it is a syntactically correct program.

Secondary Tasks:

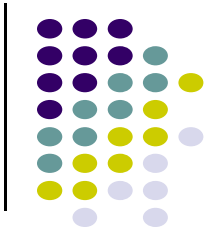
- Process declarations and set up symbol table information accordingly, in preparation for semantic analysis.
- Construct a syntax tree in preparation for intermediate code generation.

Grammars



- Every programming language has precise **grammar rules** that describe the **syntactic structure** of well-formed programs
  - In C, the rules state how functions are made out of parameter lists, declarations, and statements; how statements are made of expressions, etc
- Grammars are easy to understand, and **parsers** for programming languages can be constructed automatically from certain classes of grammars
- **Parsers** or **syntax analyzers** are generated for a particular grammar
- **Context-free grammars** are usually used for syntax specification of programming languages

What is Parsing or Syntax Analysis?



- A parser for a grammar of a programming language
  - verifies that the string of tokens for a program in that language can indeed be generated from that grammar
  - reports any syntax errors in the program
  - constructs a parse tree representation of the program (not necessarily explicit)
  - usually calls the lexical analyzer to supply a token to it when necessary
  - could be hand-written or automatically generated is based on context-free grammars.
- Grammars are generative mechanisms like regular expressions
- Pushdown automata are machines recognizing **context-free languages** (like FSA for RL)

Context-free Grammars

- A *context-free grammar* for a language specifies the syntactic structure of programs in that language.
- Components of a grammar:
  - a finite set of tokens (obtained from the scanner);
  - a set of variables representing “related” sets of strings, e.g., *declarations*, *statements*, expressions.
  - a set of rules that show the structure of these strings.
  - an indication of the “top-level” set of strings we care about.

Context-free Grammars: Definition

Formally, a context-free grammar *G* is a 4-tuple

G = (N, T, P, S), where:

- N: Finite set of non-terminals
- T: Finite set of terminals
- S ∈ N: The start symbol
- P: Finite set of productions, each of the form $A \longrightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$

- Usually, only P is specified and the first production corresponds to that of the start symbol
- Ex:

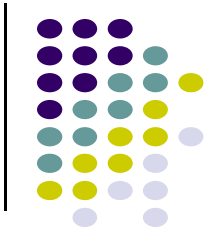
(1)
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow id$

(2)
$S \rightarrow 0S0$
$S \rightarrow 1S1$
$S \rightarrow 0$
$S \rightarrow 1$
$S \rightarrow \epsilon$

(3)
$S \rightarrow aSb$
$S \rightarrow \epsilon$

(4)
$S \rightarrow aB \mid bA$
$A \rightarrow a \mid aS \mid bAA$
$B \rightarrow b \mid bS \mid aBB$

Ways of writng CFG

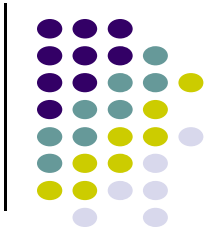


$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow (E)$$
$$E \rightarrow id$$

E → E + E | E *E | (E) | id

E → E + E
| E * E
| (E)
| id

Context-free Grammars: Terminology



- The *language* of a grammar $G = (N,T,P,S)$ is

  $L(G) = \{ w \mid w \in T^* \text{ and } S \Rightarrow^* w \}$.

  The language of a grammar contains only strings of terminal symbols.