Compiler Design

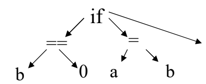
Syntax Analysis

Amey Karkare
Department of Computer Science and Engineering
IIT Kanpur
karkare@iitk.ac.in

Syntax Analysis

- Check syntax and construct abstract syntax tree

if	(	b	==	0	)	a	=	b	;



- Error reporting and recovery
- Model using context free grammars
- Recognize using Push down automata/Table Driven Parsers

2

Limitations of regular languages

- How to describe language syntax precisely and conveniently. Can regular expressions be used?
- Many languages are not regular, for example, string of balanced parentheses
  - ((((…))))
  - $\{ (^i)^i \mid i \geq 0 \}$
  - There is no regular expression for this language
- A finite automata may repeat states, however, it cannot remember the number of times it has been to a particular state
- A more powerful language is needed to describe a valid string of tokens

3

Syntax definition

- Context free grammars <T, N, P, S>
  - T: a set of tokens (terminal symbols)
  - N: a set of non terminal symbols
  - P: a set of productions of the form
    nonterminal →String of terminals & non terminals
  - S: a start symbol
- A grammar derives strings by beginning with a start symbol and repeatedly replacing a non terminal by the right hand side of a production for that non terminal.
- The strings that can be derived from the start symbol of a grammar G form the language L(G) defined by the grammar.

4

Examples

- String of balanced parentheses
  S → (S) S | ϵ

- Grammar
  list → list + digit
  | list – digit
  | digit
  digit → 0 | 1 | … | 9

  Consists of the language which is a list of digit separated by + or -.

5

Derivation

list ➔ list + digit
➔ list – digit + digit
➔ digit – digit + digit
➔ 9 – digit + digit
➔ 9 – 5 + digit
➔ 9 – 5 + 2

Therefore, the string 9-5+2 belongs to the language specified by the grammar

The name context free comes from the fact that use of a production X → … does not depend on the context of X

6

Examples …

- Simplified Grammar for C block
  block → '{' decls statements '}'
  statements → stmt-list | ϵ
  stmt–list → stmt-list stmt ';'
  | stmt ';'
  decls → decls declaration | ϵ
  declaration → …

7

Syntax analyzers

- Testing for membership whether w belongs to L(G) is just a "yes" or "no" answer
- However the syntax analyzer
  - Must generate the parse tree
  - Handle errors gracefully if string is not in the language
- Form of the grammar is important
  - Many grammars generate the same language
  - Tools are sensitive to the grammar

8

What syntax analysis cannot do!

- To check whether variables are of types on which operations are allowed
- To check whether a variable has been declared before use
- To check whether a variable has been initialized
- These issues will be handled in semantic analysis

9

Derivation

- If there is a production $A \rightarrow \alpha$ then we say that A derives α and is denoted by $A \Rightarrow \alpha$
- $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if $A \rightarrow \gamma$ is a production
- If $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$ then $\alpha_1 \Rightarrow^+ \alpha_n$
- Given a grammar G and a string w of terminals in $L(G)$, we can write $S \Rightarrow^+ w$
- If $S \Rightarrow^* \alpha$ where α is a string of terminals and non terminals of G then we say that α is a sentential form of G

10

Derivation …

- If in a sentential form only the leftmost non terminal is replaced then it becomes leftmost derivation
- Every leftmost step can be written as
  $wA\gamma \Rightarrow^{lm*} w\delta\gamma$
  where w is a string of terminals and $A \rightarrow \delta$ is a production
- Similarly, right most derivation can be defined
- An ambiguous grammar is one that produces more than one leftmost (rightmost) derivation of a sentence

11

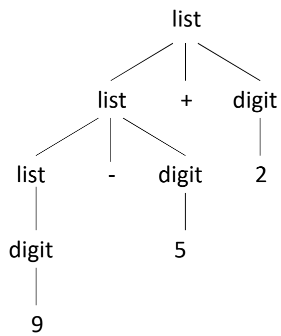
Parse tree

- shows how the start symbol of a grammar derives a string in the language
- root is labeled by the start symbol
- leaf nodes are labeled by tokens
- Each internal node is labeled by a non terminal
- if A is the label of anode and $x_1, x_2, \ldots x_n$ are labels of the children of that node then $A \rightarrow x_1\ x_2 \ldots x_n$ is a production in the grammar

12

Example

Parse tree for 9-5+2



Ambiguity

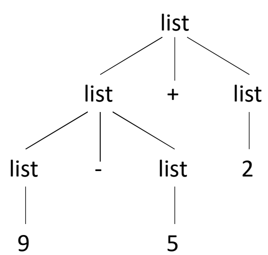
- A Grammar can have more than one parse tree for a string
- Consider grammar

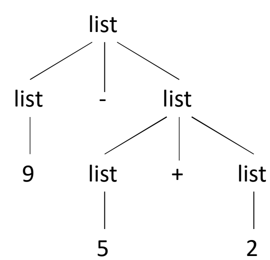
  list → list+ list

  | list – list

  | 0 | 1 | … | 9

- String 9-5+2 has two parse trees





Ambiguity …

- Ambiguity is problematic because meaning of the programs can be incorrect
- Ambiguity can be handled in several ways
  - Enforce associativity and precedence
  - Rewrite the grammar (cleanest way)
- There is no algorithm to convert automatically any ambiguous grammar to an unambiguous grammar accepting the same language
- Worse; there are inherently ambiguous languages!

Ambiguity in Programming Lang.

- Dangling else problem

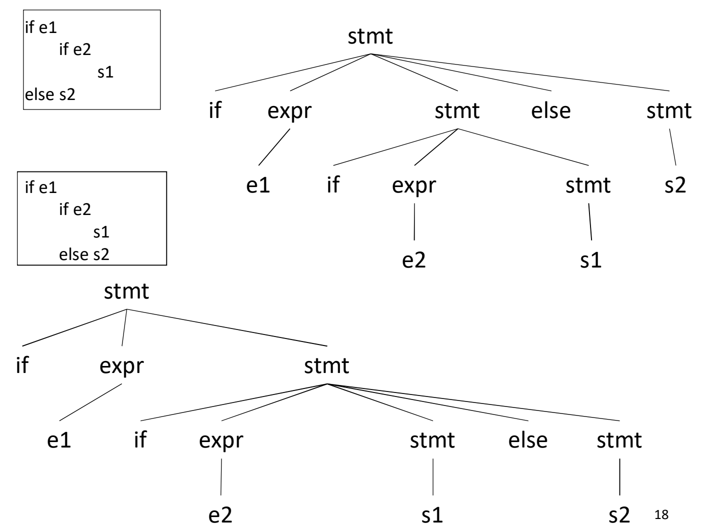
  stmt → if expr stmt

  | if expr stmt else stmt

- For this grammar, the string

  if e1 if e2 then s1 else s2

  has two parse trees



Resolving dangling else problem

- General rule: match each **else** with the closest previous **unmatched if**. The grammar can be rewritten as

  stmt → matched-stmt

  | unmatched-stmt

  matched-stmt → if expr matched-stmt

  else matched-stmt

  | others

  unmatched-stmt → if expr stmt

  | if expr matched-stmt

  else unmatched-stmt

Associativity

- If an operand has operator on both the sides, the side on which operator takes this operand is the associativity of that operator
- In a+b+c b is taken by left +
- +, -, *, / are left associative
- ^, = are right associative
- Grammar to generate strings with right associative operators

  right → letter = right | letter

  letter → a| b |…| z

Precedence

- String a+5*2 has two possible interpretations because of two different parse trees corresponding to (a+5)*2 and a+(5*2)
- Precedence determines the correct interpretation.
- Next, an example of how precedence rules are encoded in a grammar

Precedence/Associativity in the Grammar for Arithmetic Expressions

Ambiguous

E → E + E
  | E * E
  | (E)
  | num | id

3 + 2 + 5
3 + 2 * 5

- Unambiguous, with precedence and associativity rules honored

E → E + T | T
T → T * F | F
F → (E) | num
  | id

Parsing

- Process of determination whether a string can be generated by a grammar
- Parsing falls in two categories:
  - Top-down parsing:
    Construction of the parse tree starts at the root (from the start symbol) and proceeds towards leaves (token or terminals)
  - Bottom-up parsing:
    Construction of the parse tree starts from the leaf nodes (tokens or terminals of the grammar) and proceeds towards root (start symbol)