Chapter 4

Context Free Grammar & Parse Tree

Context Free Grammar & Parse Tree

- Context Free Grammars Concepts
  - Derivation
  - Recursive Grammars
  - Grammar Factoring
  - Syntax-Directed Translation
- Parse Tree
  - Deriving Strings
  - Ambiguity
  - Associativity of Operators
  - Operator Precedence

Context Free Grammars Concepts

- A **Context-free Grammar** is utilized to describe the syntactic structure of a language
- It is Characterized By:
  1. A Set of Tokens or Terminal Symbols
  2. A Set of Non-terminals
  3. A Set of Production Rules having the form:

     $$NT \rightarrow \{T, NT\}^*$$

  4. A Non-terminal Designated As the Start Symbol

Context Free Grammar Concepts

Context Free Grammars Example

$$list \rightarrow list + digit$$

$$list \rightarrow list - digit$$

$$list \rightarrow digit$$

$$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

OR

$$list \rightarrow list + digit \mid list - digit \mid digit$$

Context Free Grammars Concepts

Derivation

- A grammar derives strings by beginning with the start symbol and repeatedly replacing the non terminal by the right side of a production for that non terminal.
- Derivations are either Left Most Derivation or Right Most Derivations.
- Left (Right) most derivation is the process of accepting the given input string by rewriting the production rule with left (right) most non terminal only.

Context Free Grammars Concepts

Derivation Example:

Grammar : $E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid id$

Input: $id + id * id$

Derivation:

Left

$E \rightarrow \underline{E} + E$

$id + \underline{E}$

$id + \underline{E} * E$

$id + b * \underline{E}$

$id + id * id$

Right

$E \rightarrow E + \underline{E}$

$E + E * \underline{E}$

$E + \underline{E} * id$

$\underline{E} + id * id$

$id + id * id$

Context Free Grammars Concepts

Derivation Symbols

- Consider a non terminal A in the middle of a sequence of grammar symbols, as in αAβ, where α and β are arbitrary strings of grammar symbol.
- Suppose $A \rightarrow \gamma$ is a production. Then, we write:

  $\alpha A \beta \Rightarrow \alpha \gamma \beta$, if A derives γ in one step

  $\alpha A \beta \overset{*}{\Rightarrow} \alpha \gamma \beta$, if A derives γ in zero or more steps

  $\alpha A \beta \overset{+}{\Rightarrow} \alpha \gamma \beta$, if A derives γ in one or more steps
- If $S \overset{+}{\Rightarrow} \alpha$, where S is the start symbol of a grammar G, we say that α is a sentential form of G
  - If α contains only terminal symbols, then α is sentence.
  - If α contains one or more non-terminal symbols, then it is just a sentential form.

Context Free Grammars Concepts

Derivation Symbols Example:

Grammar :

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

1- $E \Rightarrow -E$

2- $E \overset{*}{\Rightarrow} -(E + E)$

3- $id + E * E$ is a sentential

4- $id + id * id$ is a sentence

Context Free Grammars Concepts

Recursive Grammars

A grammar is called a **recursive grammar** if it contains production rules that expanding a non-terminal according to these rules can lead to a string that includes the same non-terminal again. Otherwise it is called a non-**recursive grammar.**

A grammar is left/ right-recursive if and only if there exists a non-terminal symbol that can derive to a sentential form with itself as the leftmost/ rightmost symbol. Like ($A \rightarrow A\ \alpha$ / $A \rightarrow \alpha\ A$)

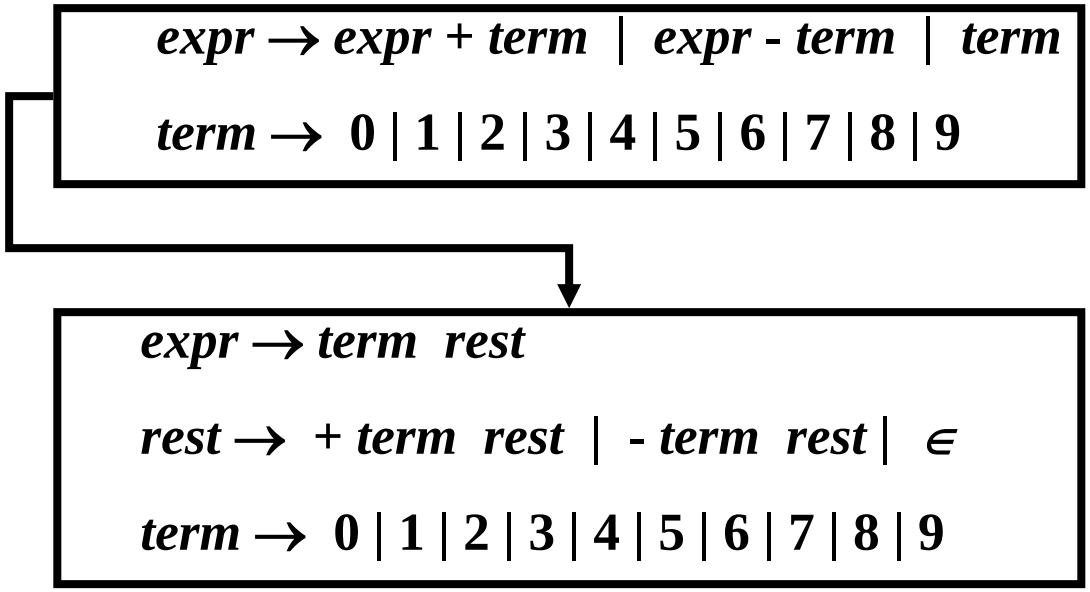
Direct recursion occurs when the definition can be satisfied with only one substitution. It has the form: ($A \rightarrow A\ \alpha \mid \alpha\ A$)

indirect recursion occurs when the definition can be satisfied with more than one substitution.
It has the form: ($A \rightarrow B\ \alpha$, $B \rightarrow A\ \gamma$) or
($A \rightarrow \alpha\ B$, $B \rightarrow \gamma\ A$)

Context Free Grammars Concepts

- In some situations, Left Recursion should be removed.
- The basic idea is to rewrite the recursive grammar $A \rightarrow A\alpha|\beta$ as $A \rightarrow \beta A'$, $A' \rightarrow \alpha A' \mid \in$

Example:



recursive grammar

non-recursive grammar

Context Free Grammars Concepts

Grammar Factoring

- When a production has more than one alternatives with common prefixes, then it is necessary to make right choice on production.

- This can be done through rewriting the production until enough of the input has been seen.

Example:

To perform left-factoring for the production, $A \rightarrow \alpha\ \beta \mid \alpha\ \gamma$,
we rewrite is as: $A \rightarrow\ \alpha\ A',\ A' \rightarrow\ \beta\ \gamma$

Parse Tree

- A parse tree is a tree that shows **how** the start symbol of a grammar derives a string in the language.

- More Formally, a Parse Tree for a CFG Has the Following Properties:
  - Root Is Labeled With the Start Symbol
  - Leaf Node Is a Token or $\in$
  - Interior Node Is a Non-Terminal
  - If $A \rightarrow x1x2\ldots xn$, Then A Is an Interior; $x1x2\ldots xn$ Are Children of A and May Be Non-Terminals or Tokens

Deriving Strings Using Grammars

Using the grammar defined on the earlier slide, we can derive the string: 9 - 5 + 2 as follows:

list → *list* + *digit*	**P1 :** *list* → *list* + *digit*
→ *list* - *digit* + *digit*	**P2 :** *list* → *list* - *digit*
→ *digit* - *digit* + *digit*	**P3 :** *list* → *digit*
→ 9 - *digit* + *digit*	**P4 :** *digit* → 9
→ 9 - 5 + *digit*	**P4 :** *digit* → 5
→ 9 - 5 + 2	**P4 :** *digit* → 2

Deriving Strings Using Grammars

This derivation could also be represented via a Parse Tree (parents on left, children on right)

list → *list* + *digit*

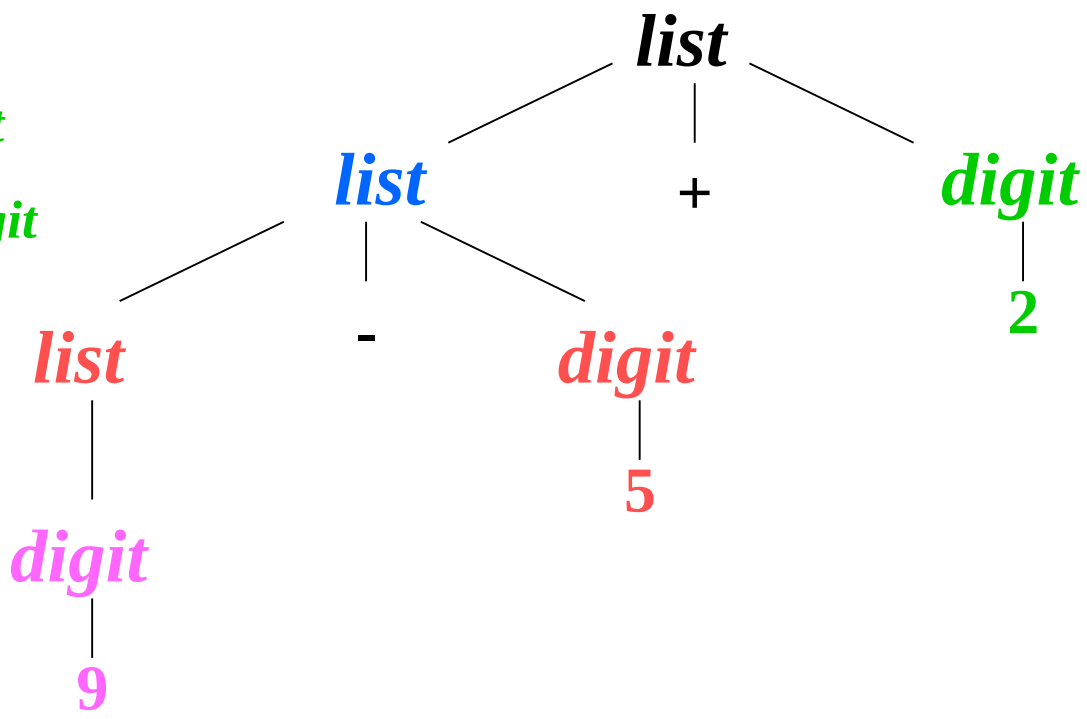
→ *list* - *digit* + *digit*

→ *digit* - *digit* + *digit*

→ 9 - *digit* + *digit*

→ 9 - 5 + *digit*

→ 9 - 5 + 2



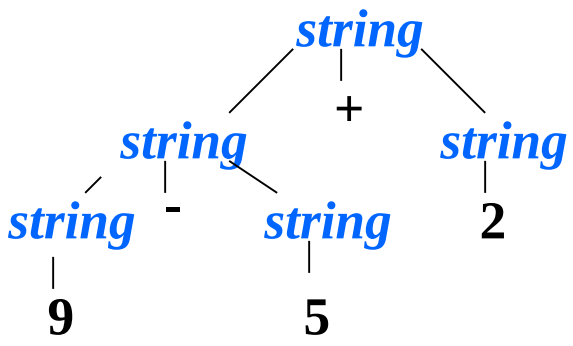
Ambiguity

A grammar that generates two parse trees for the same input is said to be ambiguous grammar.

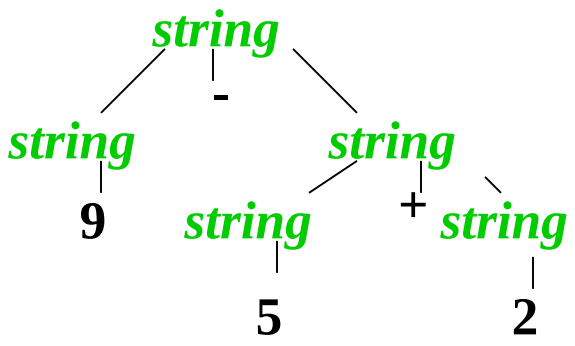
Input: 9 – 5 + 2

Grammar:

***string* → *string* + *string* | *string* – *string* | 0 | 1 | …| 9**



Parse tree 1



Parse tree 2

Associativity of Operators

- To solve the problem of ambiguity, extra restrictions such as **associativity** and **precedence** should be added to obtain only one parse tree for the input expression.

- Associativity of Operators

  Associativity means that when the same operator appears in the input stream, then which operator occurrence should be applied first.

Associativity of Operators

For example, given the expression:

operand1 operator operand2 operator operand3

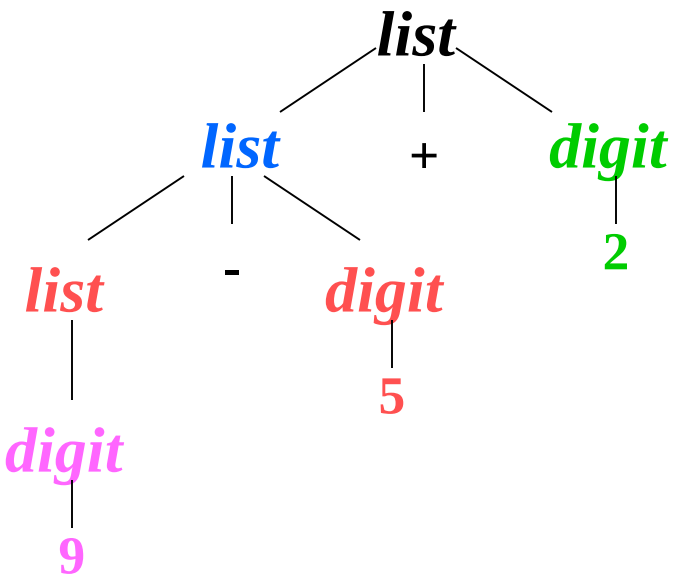
- the "operator" is left associative if it is applied first to "operand1" and "operand2 " and then to "operand3". Like (+, - , * , /)

- the "operator" is right associative, if it is applied applied first to "operand2" and "operand3 " and then to "operand1". Like (=, exponent)

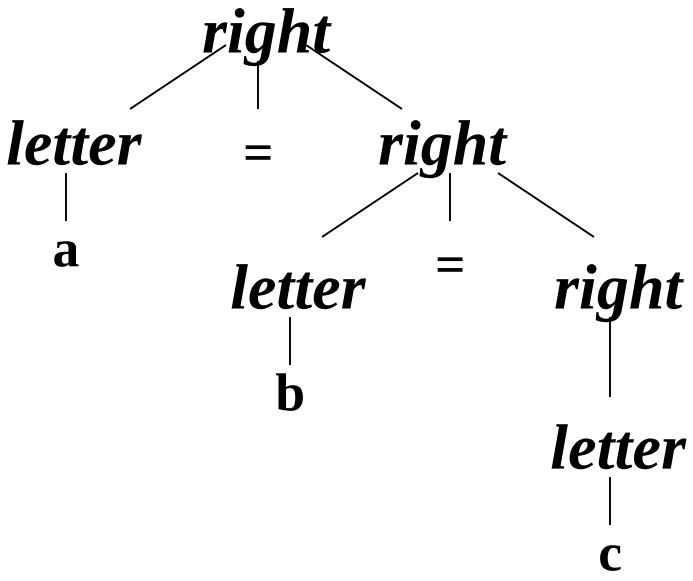
Associativity of Operators

Left vs. Right

9 – 5 + 2



a = b + c



list → list + digit |

| list - digit | digit

digit → 0 | 1 | 2 | …| 9

right → letter = right | letter

letter → a | b | c | …| z

Associativity of Operators

- The language of arithmetic expressions with + -
  - (ambiguous) grammar that does not enforce associativity

    string → ***string*** **+** ***string*** **|** ***string*** **–** ***string*** **| 0 | 1 | …| 9**

  - non-ambiguous grammar enforcing left associativity (parse tree will grow to the left)

    string → ***string*** **+** ***digit*** **|** ***string*** **-** ***digit*** **|** ***digit***

    digit → **0 | 1 | 2 | …| 9**

  - non-ambiguous grammar enforcing right associativity (parse tree will grow to the right)

    string → ***digit*** **+** ***string*** **|** ***digit*** **-** ***string*** **|** ***digit***

    digit → **0 | 1 | 2 | …| 9**

Operator Precedence

- Most programming languages have operator precedence rules that state the order in which operators are applied.

- Operators precedence rules can be incorporated directly into a Context Free Grammar to obtain only one parse tree for the input expression.

- Ambiguity is avoided.

Operator Precedence

What about

9 – 5 + 2

9 + 5 * 2

(9 + 5) * 2 ?

Typically { **( )** ; *** /** ; **+ -** } **is precedence order**

$$expr \rightarrow expr + term \mid expr - term \mid term$$

$$term \rightarrow term * factor \mid term / factor \mid factor$$

$$factor \rightarrow digit \mid (\ expr \ )$$

$$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid \ldots \mid 9$$

Precedence Achieved by:

***expr* & *term* for each precedence level**

Rules for each are associate to the left

The End