Context-Free Grammars

* jump between higher-level language to assembly is bigger than assembly to machine so compiler is more complicated
* below is main components of a typical compiler:

Diagram

Description automatically generated

* + through Scanner, we get sequence of tokens
  + Parser is syntactic analysis stage
    - takes sequence of tokens and checks for syntax errors
  + Context Sensitive Analysis is semantic analysis stage
    - requirements that go beyond syntax like types in assignment statements matching, variable declaration before use, etc.
  + finally, compiler moves to Synthesis stage and generates intended output
* balanced parentheses problem: consider Σ = {(, )} and L = {w: w is a balanced string of parentheses}
  + this is not a regular expression because it can’t be built with a DFA
  + once we have an arbitrarily large number of open parentheses, we need to ensure we have the same number of closing parentheses to match
  + in DFA, each pair requires an extra 2 states to keep track of how deeply nested the expression is
  + need to use context-free language instead of regular language
* context-free languages are exactly regular languages with the addition of recursion
* grammar is language of languages
  + i.e. grammars help us describe what we’re allowed and not allowed to say
* context-free languages are typically expressed as a context-free grammars
* Context-Free Grammar (CFG) is a 4-tuple (N, T, P, S):
  + N is a finite, non-empty set of non-terminal symbols
  + T is an alphabet: a finite, non-empty set of terminal symbols
    - can use Σ rather than T
  + P is a finite set of productions/rules, each of the form A → β where A ∈ N and β ∈ (N ∪ T)
    - V = N ∪ T and call it vocabulary, which is set of all symbols in our language
  + S ∈ N is a start symbol
* e.g. CFG rep valid arithmetic expressions with arbitrary balanced parentheses:

Text

Description automatically generated

* conventions allow to state formal definitions concisely without having to repeatedly specify which set each variable belongs to:
  + lower-case letters from start of alphabet (i.e. a, b, c, …) are elements of T
    - i.e. terminals
  + lower-case letters from end of alphabet (i.e. …, w, y, z) are elements of T\*
    - i.e. words
  + upper-case letters start of alphabet (i.e. A, B, C, …) are elements of N
    - i.e. non-terminals
  + Greek letters (i.e., α, β, γ, …) are elements of V\*
    - i.e. sequences of terminals and non-terminals (N ∪ T)\*
  + S is always start symbol
* e.g. balanced parentheses string as a CFL:

Graphical user interface, text, application, letter

Description automatically generated

* over a CFG (N, T, P, S), we say that A directly derives γ, and write A ⇒ γ, if and only if there is a rule A → γ in P
  + “directly derives” relation rep a single application of a production rule
  + rewrite/expand a non-terminal by replacing LHS of production rule with RHS (which is a terminal)
  + given an arbitrary sequence αAβ, we use the definition of “directly derives” to say that αAβ ⇒ αγβ if and only if A → γ
* over a CFG (N, T, P, S), we say that α derives β, and write α ⇒\* β, if either α = β, or if there exists γ such that α ⇒ γ and γ ⇒\* β
  + derive sequence β from α through zero or more applications of production rules
* over a CFG (N, T, P, S), a derivation of a string of terminals x is a sequence α0α1 · · · αn such that α0 = S and αn = x and αi ⇒ α(i+1) for 0 ≤ i < n
* e.g. for the grammar of expressions with balanced parentheses, give a derivation for the word ID OP LPAREN ID OP ID RPAREN (i.e., show expr ⇒\* ID OP LPAREN ID OP ID RPAREN)

Text

Description automatically generated

* + each step of derivation chooses a non-terminal from the current αi and rewrites it by replacing it with the RHS of some rule for that non-terminal
* context-free languages are called “context-free” because regardless of any other symbol, the LHS of a rule can be replaced by its RHS
  + e.g.

Text

Description automatically generated with low confidence

* language of a CFG (N, T, P, S) is L(G) = {w ∈ T\*: S ⇒\* w}
  + a CFL are strings of terminals that have a derivation in G (i.e. words can be derived at the starting symbol of G)
  + a string that contains one or more non-terminals can’t be a word in a CFL
* a language L is context-free if and only if there exists a CFG G such that L = L(G)
  + every regular language is a CFL

Text, letter

Description automatically generated

* e.g.

Text

Description automatically generated

* + often use shorthand S → ε|aSb to rep multiple productions for the same non-terminal
  + often just write productions rules since other parts of CFG can be inferred from the rules
  + when not explicitly stated, non-terminal on LHS of first rule is start symbol S
  + derivation for the string aaabbb: S ⇒ aSb ⇒ aaSbb ⇒ aaaSbbb ⇒ aaabbb

Derivations and Parse Trees

* a parse tree is a visual representation of the structure of an input word
  + root is start symbol
  + each non-leaf node is a non-terminal and its immediate descendants are the RHS of the rule that was used for the non-terminal in the derivation
  + leaf nodes rep terminals or ε
    - common to exclude ε leaf node for ε productions (only case in which a non-terminal can also be a leaf node)
* a derivation uniquely defines a parse tree
* an input string can have more than one parse tree
  + e.g. the below derivations correspond to diff parse trees

Text, letter

Description automatically generated

* + - first one:

A picture containing text, watch

Description automatically generated

* + - second one:

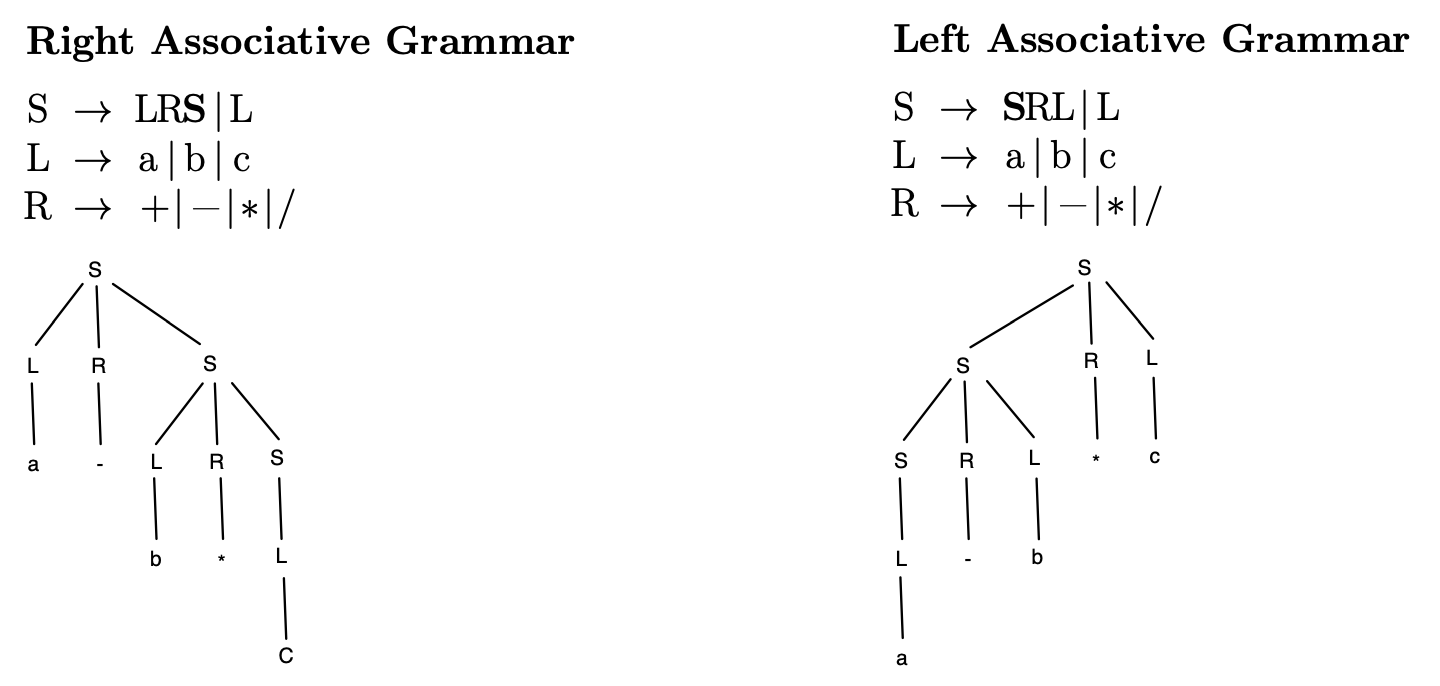
Chart

Description automatically generated

* in a Leftmost Derivation (i.e. Left Canonical Derivation), we always expand the leftmost non-terminal first
  + each step has the form: xAγ ⇒ xαγ
* in a Rightmost Derivation (i.e. Right Canonical Derivation), we always expand the rightmost non-terminal first
  + each step has the form: βAx ⇒ βαx
* x rep string of terminals and α, β and γ are elements of V\* (words over the vocabulary)

Ambiguous Grammars

* a grammar is ambiguous if there’s a word in the language which has more than one distinct leftmost/rightmost derivation
  + if we only cared about recognition (i.e. if w ∈ L(G)), this wouldn’t matter since any derivation would prove the word is in the language
* can force a grammar to be left/right associative by insisting on how the recursion works



* due to depth-first traversal, deeper parts of the tree get evaluated first and therefore have higher precedence

Parsing Algorithms

* context-free languages can be recognized by model of computation call Pushdown Automata (PDA)
  + a PDA is a Finite Automaton with addition of a stack