Unit VI: Language, grammar and Automata

LANGUAGE AND GRAMMAR

Terminologies

- Grammars
  - Used to generate sentences of a language and to determine if a given sentence is in a language
  - Formal languages, generated by grammars, provide models for programming languages (Java, C, etc) as well as natural language --- important for constructing compilers
- Finite-state machines (FSM)
  - FSM are characterized by a set of states, an input alphabet, and transitions that assigns a next state to a pair of state and an input. We'll study FSM with and without output. They are used in language recognition (equivalent to certain grammar)but also for other tasks such as controlling vending machines

Terminologies

- Formal Language Formal language is a language that is specified by a well-defined set of rules of syntax
- A formal grammar G is any compact, precise definition of a language L. A grammar implies an algorithm that would generate all legal sentences of the language.

Terminologies

- **Vocabulary/Alphabet (V/Σ)**: A vocabulary (or alphabet) V is a finite, nonempty set of elements called symbols.
  - Eg: V={1,0} or (a,b)
- **Word/String(w)**: A word (or sentence) over V is a string of finite length elements of V. Length of string is denoted by |w|
  - Eg: w=ababaa, w=1010101
- **Empty String λ** The empty string or null string, denoted by λ, is the string containing no symbols
- **Set of words**: The set of all words over V is denoted by $V*$.
- **Language**: collection of string over alphabet. A language over V is a subset of $V*$
  - **Σ**={a,b}, L_1={set of all string of length 2} L_1={ab,ba,aa,bb}
  - L_2={set of all string that begin with b} L_2={b,ba,bba,bbb,..}

Terminologies

- **Syntax:** rule
- **Semantics:** meaning
- **Formal language**: specified by well defined set of rules of syntax
- **Natural language**: spoken language
  - Example: frog writes neatly (formal language is OK. Natural language has no meaning)
- **Terminals (T)**: elements of vocabulary cannot be replaced by other symbols
- **Non terminals(N)**: which can be replaced by other symbols
- **Production rule**: rules that specify when we can replace string with another string. At least one non terminal is required in left hand side

Phase structure grammar

- A phrase-structure grammar or grammar G is defined by quad tuple
- G= (V, T, S, P) consists of a
  - vocabulary V , (capital letters)
  - terminal symbols T, (small letters)
  - a start symbol S from V ,
  - and a finite set of productions P.
- The set V – T is denoted by N. Elements of N are called nonterminal symbols. Every production in P must contain at least one nonterminal on its left side.
- **Example**
- Let G = (V , T , S, P), where V = {a, b, A, B, S}, T = {a, b}, S is the start symbol, and P = {S → ABa, A → BB, B → ab, AB → b}. G is an example of a phrase-structure grammar

Examples

- Let G be the grammar with vocabulary V = {S, A, a, b}, set of terminals T = {a, b}, starting symbol S, and productions P = {S → aA, S → b, A → aa}. What is L(G), the language of this grammar?
  - From the start state S we can derive aA using the production S → aA. We can also use the production S → b to derive b. From aA the production A → aa can be used to derive aaa. No additional words can be derived. Hence, L(G) = {b, aaa}.
- Let G be the grammar with vocabulary V = {S, 0, 1}, set of terminals T = {0, 1}, starting symbol S, and productions P = {S → 11S,S → 0}. What is L(G), the language of this grammar
  - S→0 gives 0
  - S→ 11S gives 110, 11110,1111110,… after each iteration, we can add two 1
  - So L(G)={0,110,11110,…}

Examples

- Give a phrase-structure grammar that generates the set $\{0^n 1^n \mid n = 0, 1, 2, \ldots\}$
    - Two productions can be used to generate all strings consisting of a string of 0s followed by a string of the same number of 1s, including the null string. The first builds up successively longer strings in the language by adding a 0 at the start of the string and a 1 at the end. The second production replaces S with the empty string. The solution is the grammar G = (V , T , S, P), where V = {0, 1, S}, T = {0, 1}, S is the starting symbol, and the productions are
    - $S \rightarrow 0S1$
    - $S \rightarrow \lambda$

Examples

- Find a phrase-structure grammar to generate the set $\{0^m 1^n \mid m \text{ and } n$ are nonnegative integers$\}$.
  - First method:
    - The grammar G1 has alphabet $V = \{S, 0, 1\}$; terminals $T = \{0, 1\}$;
    - Production rules: $S \rightarrow 0S$, $S \rightarrow S1$, $S \rightarrow \lambda$
  - Second method:
    - The grammar G2 has alphabet $V = \{S, A, 0, 1\}$; terminals $T = \{0, 1\}$; and
    - Productions rules: $S \rightarrow 0S$, $S \rightarrow 1A$, $S \rightarrow 1$, $A \rightarrow 1A$, $A \rightarrow 1$, and $S \rightarrow \lambda$.

Type of Grammar/Chomsky classification

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton

Type- 0 grammar(Unrestricted)

- Type-0 grammars generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.
- They generate the languages that are recognized by a Turing machine.
- The productions can be in the form of $\alpha \rightarrow \beta$ where
- α is $(N + T)^* N (N + T)^*$
- β is $(N + T)^*$
- This meaning left hand side must contain at least one non terminal

$S \rightarrow ACaB$
$Bc \rightarrow acB$
$CB \rightarrow DB$
$aD \rightarrow Db$

Type 1 grammar(Context sensitive grammar)

- Type-1 grammars(CSG/Length increasing grammar, Non-contracting grammar)
- The productions must be in the form
- $\alpha A \beta \rightarrow \alpha \gamma \beta$
- where $A \in N$ (Non-terminal)
- and $\alpha, \beta, \gamma \in (T \cup N)^*$ (Strings of terminals and non-terminals)
- The strings α and β may be empty, but γ must be non-empty.
- The rule $S \rightarrow \varepsilon / \lambda$ is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.

- That is the count of symbol in LHS is less than or equal to RHS

$AB \rightarrow AbBc$

$A \rightarrow bcA$

$B \rightarrow b$

Type-2 grammar (Context Free)

- Type-2 grammars generate context-free languages and recognized by non deterministic push down automata
- The productions must be in the form $A \rightarrow \gamma$
- where $A \in N$ (Non terminal)
- and $\gamma \in (T \cup N)^*$ (String of terminals and non-terminals).

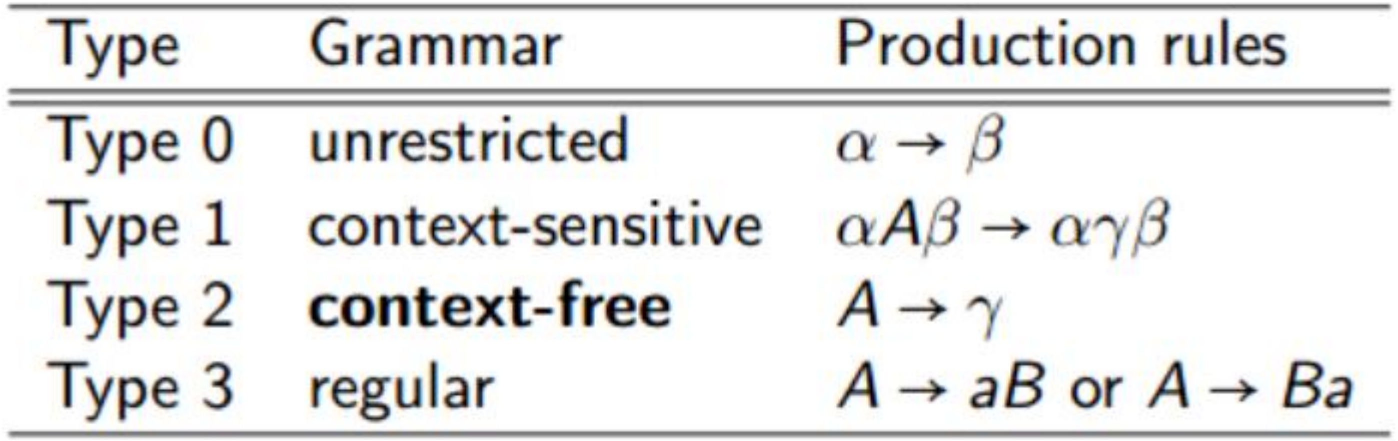
$S \rightarrow X\, a$
$X \rightarrow a$
$X \rightarrow aX$
$X \rightarrow abc$
$X \rightarrow \varepsilon$

Type-3 grammars (Regular)

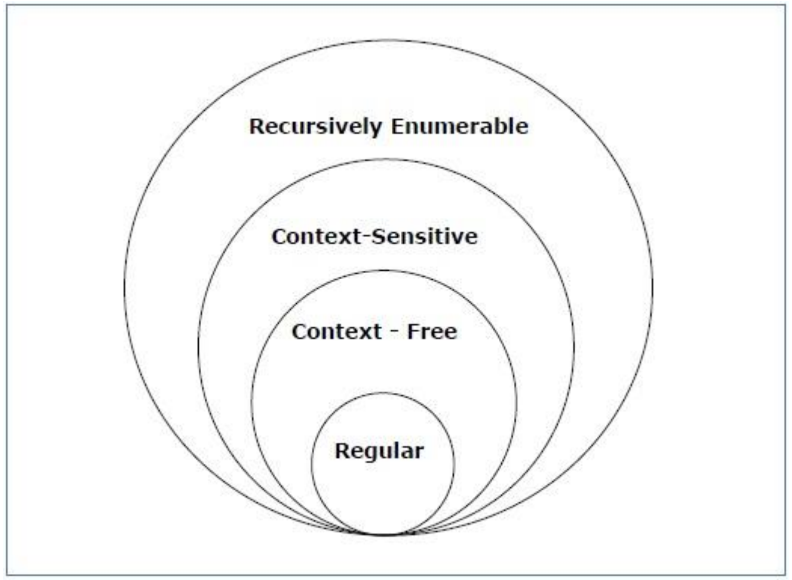
- Type-3 grammars generates context free languages which is accepted by Push Down Automata
- Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.
- The productions must be in the form X → a or X → aY(right linear) or X → Ya (left linear)
- where X, Y ∈ N (Non terminal)
- and a ∈ T (Terminal)
- The rule S → ε/ λ is allowed if S does not appear on the right side of any rule.
- Example

X → ε
X → a | aY
Y → b

Type



Type	Grammar	Production rules
Type 0	unrestricted	$\alpha \rightarrow \beta$
Type 1	context-sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type 2	**context-free**	$A \rightarrow \gamma$
Type 3	regular	$A \rightarrow aB$ or $A \rightarrow Ba$



- Type 0 grammars are completely unrestricted,
- whereas Type 1 grammars require all production rules to contain at least one non-terminal on the left-hand side (= LHS) of the rule which then expands to whatever sequence of non-terminal and terminal symbols on the right-hand side (= RHS), i.e. the LHS can not contain only terminal symbols.
- Type 2 allows only one non-terminal on the LHS;
- Type 3 additionally prohibits sequences other than one non-terminal and one terminal, or just one terminal on the RHS.

Summary

- Type 2 grammars are called context free because nonterminal symbol that is left side of production can be replaced in string whenever it occurs, no matter what is in string
- Type 1 grammar is context sensitive because non terminal symbols in left side is surrounded by some terminal characters
- $\{0^n1^n \mid n = 0, 1, 2,...\}$ is a context-free language
- $\{0^m1^n \mid m, n = 0, 1, 2,...\}$ is a regular language

Example

- Consider following grammar
  - $S \rightarrow ACaB$
  - $Bc \rightarrow acB$
  - $CB \rightarrow DB$
  - $aD \rightarrow Db$
- Determine whether given grammar is context sensitive, context free or regular grammar or none of these

- A. Checking for regular(type-3)
  - Production rule type
    - $A \rightarrow a$
    - $A \rightarrow Ba$
    - Production rules are not in above form
- B. Checking for context free(type-2)
    - $A \rightarrow \gamma$
    - They are not in above form
- C. Checking for context sensitive (type-1)
    - $\alpha A \beta \rightarrow \alpha \gamma \beta$
    - Production rules are in this form

- Write context free grammar that generates palindrome string over Σ(a,b).
  - Solution
  - $S \rightarrow aSa \mid bSb$
  - $S \rightarrow \lambda \mid a \mid b$

Backus–Naur Form

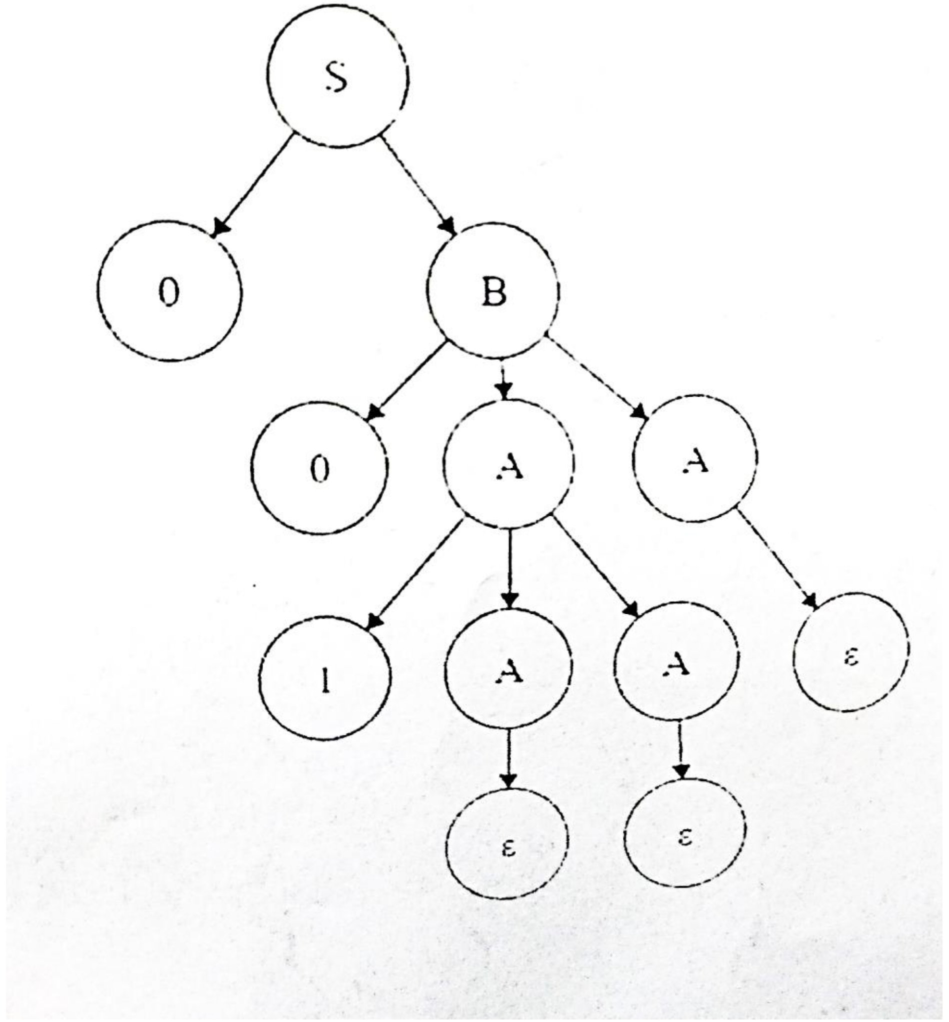
- There is another notation that is sometimes used to specify a type 2 grammar, called the Backus–Naur form (BNF), after John Backus, who invented it, and Peter Naur, who refined it for use in the specification of the programming language ALGOL
- The Backus–Naur form is used to specify the syntactic rules of many computer languages, including Java
- Instead of listing all the productions separately, we can combine all those with the same nonterminal symbol on the left-hand side into one statement. Instead of using the symbol → in a production, we use the symbol ::=.
- We enclose all nonterminal symbols in brackets, , and we list all the right-hand sides of productions in the same statement, separating them by bars.
- For instance, the productions A → Aa, A → a, and A → AB can be combined into A ::= $\langle A \rangle$ a | a | $\langle A \rangle \langle B \rangle$

Derivation Tree

- A derivation in the language generated by a context-free grammar can be represented graphically using an ordered rooted tree, called a derivation, or parse tree.
  - The root of this tree represents the starting symbol.
  - The internal vertices of the tree represent the nonterminal symbols that arise in the derivation.
  - The leaves of the tree represent the terminal symbols that arise

Derivation tree

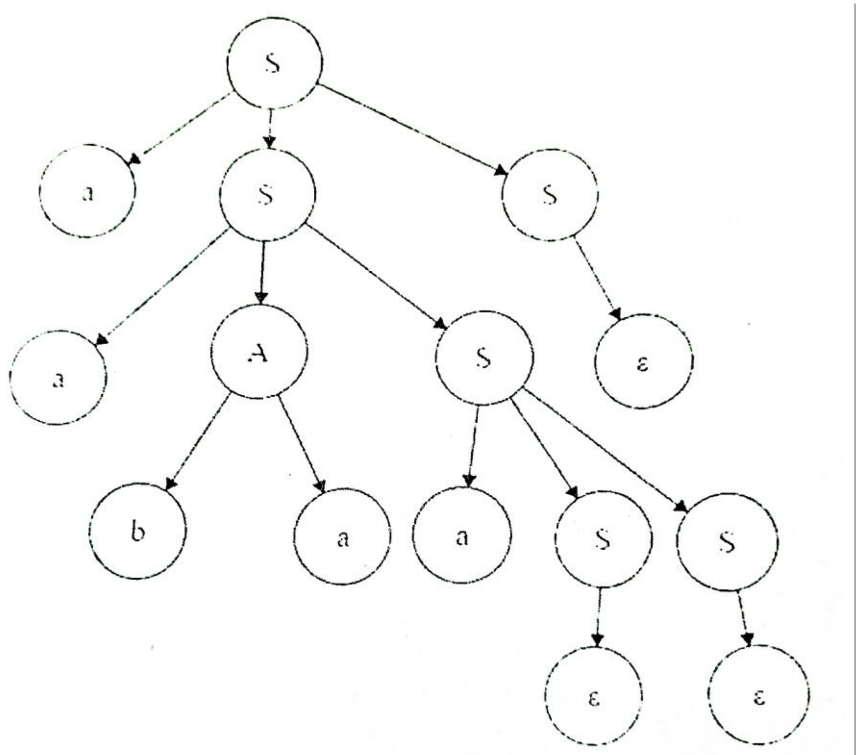
- Consider following grammar
  - G={N,T,P,S} where production rule is given by
    - S→0B
    - A→1AA/ λ
    - B→0AA
  - Construct derivation tree for string “001”
- Solution
  - S→0B
  - S→00AA
  - S→001AAA
  - S→001



Example 2

- Consider following grammar
  - G={N,T,P,S} where production rule is given by
    - S→aAS/aSS/e
    - A→SbA/ba
  - Construct derivation tree for string “aabaa”
- Solution
  - S→aAS
  - S→aaSSAS
  - S→aaAS
  - S→aabaS
  - S→aabaaSS
  - S→aabaa

S→aSS
S→aaAS
S→aabaS
S→aabaaSS
S→aabaa



Backus-Naur example

- The Backus–Naur form for a grammar that produces signed integers is
- ⟨signed integer⟩ ::= ⟨sign⟩ ⟨unsigned integer⟩
- ⟨sign⟩ ::= + | −
- ⟨unsigned integer⟩ ::= ⟨digit⟩ | ⟨digit⟩ ⟨*unsigned* integer⟩
- ⟨digit⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Derive -102 using grammar and construct derivation tree

- ⟨signed integer⟩ ::= ⟨sign⟩ ⟨unsigned integer⟩

  ::= -$\langle digit \rangle$ ⟨unsignedinteger⟩

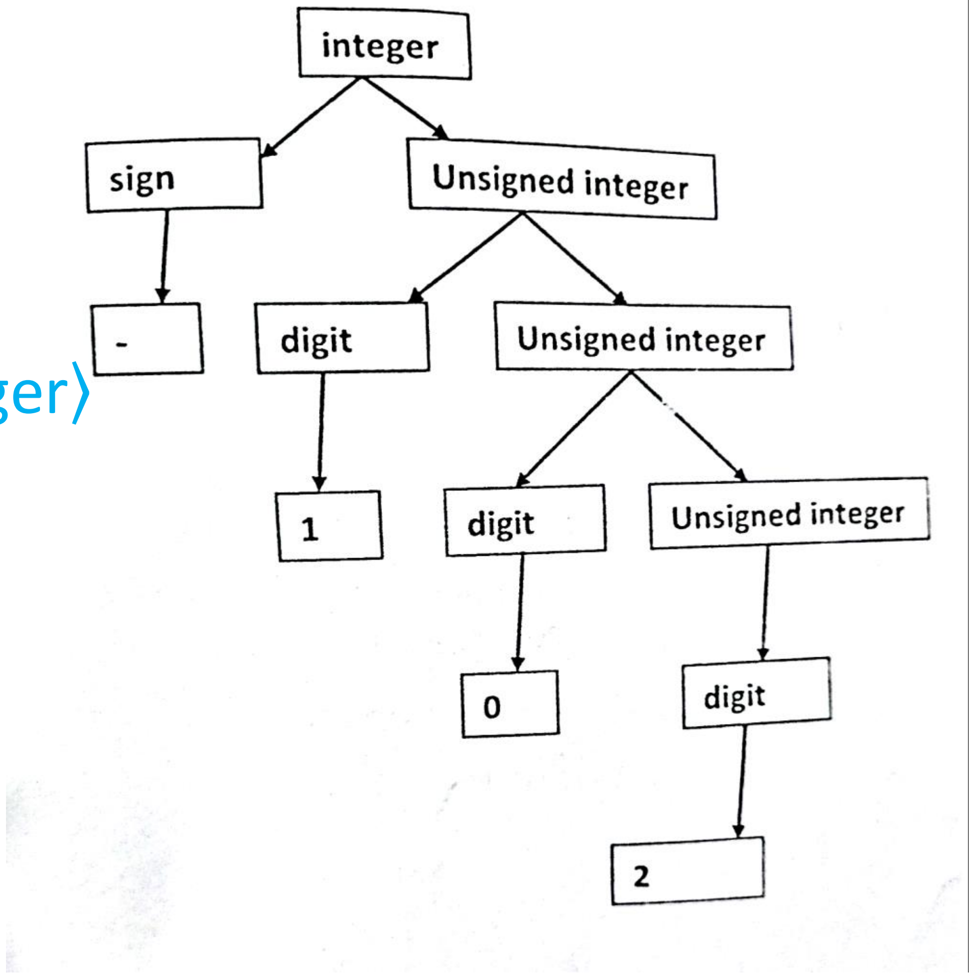
  ::= -$\langle digit \rangle \langle digit \rangle$⟨unsigned integer⟩

  ::= -1$\langle digit \rangle$ ⟨unsigned integer⟩

  ::= -10⟨unsigned integer⟩

  ::= -10$\langle digit \rangle$

  ::= -102



-