CS 3313
Foundations of Computing:

Introduction to Grammars

http://gw-cs3313-2021.github.io

A better formalism to define language ?
Some questions

- Set notation works but does not specify a way to generate the words/strings in the language
- Regular expressions work for regular languages but many interesting/useful languages are not regular
- Ex: how do you define a syntactically valid C program ?
- Ex: how do you define the construction of a sentence in the English language ?
  - Can we formally define what it means for a language to be ambiguous?

Grammars: Definition

- Precise mechanism to describe the strings in a language
- Definition: A grammar G (V,T,P,S) consists of:

  V: a finite set of *variable* or non-terminal symbols

  T: a finite set of *terminal* symbols (same as *alphabet* Σ)

  S: a variable called the *start* symbol

  P: a set of *productions* (also called production rules)
- Example 1:

  $V = \{ S \}$

  $T = \{ a, b \}$

  $P = \{ S \rightarrow aSb, S \rightarrow \lambda \}$
- Ex.2: <sentence> = <noun phrase><verb phrase>

Grammars - comments

- Basic idea is to use “variables” to stand for sets of strings (i.e., languages)
  - Variable for <nouns>, <verbs>
- These variables are defined recursively, in terms of one another
- Recursive rules (Productions) involve only concatenation
  - Alternate rules for a variable allow union
- Production rule is of the form $x \rightarrow y$ where x,y are $(V \cup T)^+$
  - Production rules are akin to the transition function of an automaton

Grammars: Derivation of Strings

- Beginning with the start symbol, strings are derived by repeatedly replacing string on left hand side symbols with the expression on the right-hand side of any applicable production
- Any applicable production can be used, in arbitrary order, until the string contains no variable symbols.
- Sample derivation using grammar in Example:

$V = \{ S \}$ $T = \{ a, b \}$ $P = \{ S \to aSb, S \to \lambda \}$

$S \Rightarrow aSb$ (applying first production)

$\Rightarrow aaSbb$ (applying first production)

$\Rightarrow aabb$ (applying second production)

Derivations – Formalism

- We say $\alpha A \beta$ => $\alpha\gamma\beta$ if A -> γ is a production.
- Example: S -> 01; S -> 0S1.
- S => 0S1 =>. 00S11 => 000111.

- we are replacing the occurrence of variable A in string $\alpha A \beta$ on LHS with the RHS of the production A -> γ

Iterated Derivation

- =>* means "zero or more derivation steps."
- Basis: α =>* α for any string α.
- Induction: if α =>* β and β => γ, then α =>* γ.
- Example:
  - Grammar: S -> 01; S -> 0S1.
  - S => 0S1 => 00S11 => 000111 i.e., S derives 000111 in 3 steps
  - Thus S =>* S; S =>* 0S1;
    S =>* 00S11; S =>* 000111.

The Language Generated by a Grammar

- Definition: For a given grammar G, the language generated by G, L(G), is the set of all terminal strings derived from the start symbol
- If G is a CFG, then L(G), the *language of G*, is

  L(G)= *{w | S =>* w and w is a string over set T}*.
- Example: G has productions S -> ϵ and S -> 0S1.
- L(G) = $\{0^n 1^n \mid n \geq 0\}$.

- To show a language L is generated by G:
  - Show every string in L can be generated by G
  - Show every string generated by L is in G

Grammars and Language Classes

- By placing constraints on what type of productions are allowed, we define different language classes.
- Regular Grammars = Regular Languages
- Context Free Grammars = Context Free languages

Regular Grammars

- In a right-linear grammar, *at most one variable symbol appears on the right side of any production*. If it occurs, it is the rightmost symbol.
- In a left-linear grammar, *at most one variable symbol appears on the right side of any production*. If it occurs, it is the leftmost symbol.
- A *regular grammar* is either right-linear or left-linear.
- Example: a regular (right-linear) grammar:

V = { S }, T = { a, b }, and productions $S \rightarrow abS \mid a$

Right-Linear Grammars Generate Regular Languages

Theorem: It is always possible to construct a NFA to accept the language generated by a regular grammar G:

- Label the start state with S and a final state V_f
- For every variable symbol V_i in G, create a NFA state and label it V_i
- For each production of the form $A \rightarrow aB$, label a transition from state A to B with symbol a
- For each production of the form $A \rightarrow a$, label a transition from state A to V_f with symbol *a* (may have to add intermediate states for productions with more than one terminal on RHS)

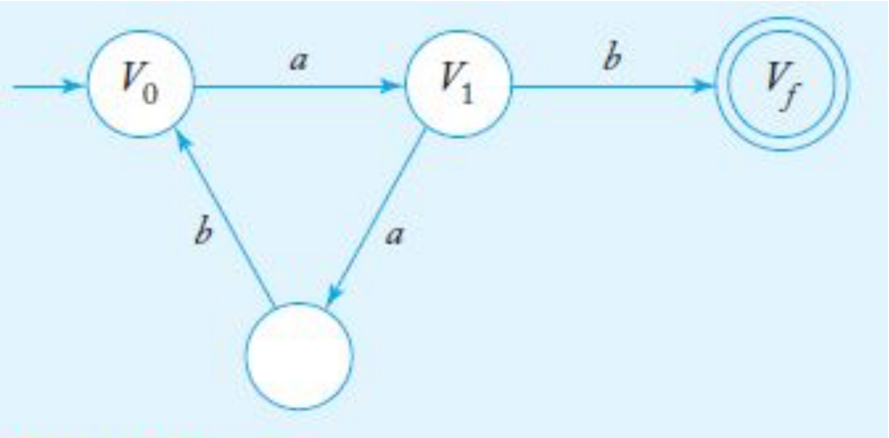
Example: Construction of a nfa to accept a language L(G)

Given the regular grammar G with productions

$V_0 \to aV_1$

$V_1 \to abV_0 \mid b$

a nondeterministic fa to accept L(G) can be constructed systematically



Assignment: Read Chapter 3.2 in textbook to complete coverage of regular grammars

Context Free Grammars

- A context free grammar is a grammar G=(V,T,P,S) where all production rules are of the form: $V \rightarrow (V \cup T)^*$
  - Production rules have exactly one variable on the left and a string consisting of variables and terminals on the right.

$V = \{ S \}$

$T = \{ a, b \}$

$P = \{ S \rightarrow aSb, S \rightarrow \lambda \}$

Grammars for Programming Languages

- The syntax of constructs in a programming language is commonly described with grammars
  - Commonly referred to as **Backus-Naur Form (BNF)**
- Assume that in a hypothetical programming language,
  - Identifiers consist of digits and the letters a, b, or c
  - Identifiers must begin with a letter
- Productions for a sample grammar:

  <id> $\rightarrow$ <letter> <rest>

  <rest> $\rightarrow$ <letter> <rest> | <digit> <rest> | λ

  <letter> $\rightarrow$ a | b | c

  <digit> $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- Example: Check the formal grammar that defines the syntax of Python at https://docs.python.org/3/reference/grammar.html

CFGs Example: *{ $a^i b^i$ | i >= 0 }*

- $S \rightarrow aSb \mid \epsilon$
- Generating recursively: each time an a is generated, a matching b is generated at the same time; can generate the matching pair indefinitely; with ϵ being the base case for the purposes of 1) generating $w = \epsilon$, and 2) terminating the recursion.
- S = aSb =>$aaSbb$ => $aaaSbbb$ =>$aaabbb$
- Can prove by induction, on length of derivation, that

  L(G) = { {a^i b^i }

CFG Example 2: { wcw^R | w in $\{a,b\}^*$ }

- Generating from the "outside" to the "inside": key characteristic of CFGs
- $S \rightarrow aSa \mid bSb \mid c$
- Intuition: at derivation step 1, S => aSa or S => bSb
- and then after another k steps, assume S $=>^*$ xcy
  - x,y are in {a,b}*
- Therefore: S => aSa $=>^*$ a xcy a
  - The two a's are equidistant from c but in "opposite directions"
    - Leftward or rightward
- S => aSa => abSba => abaSaba => abacaba
- Observation: Replace $S \rightarrow$ c by $S \rightarrow \epsilon$ (empty string) in the productions and we get grammar for ww^R !

CFGs: Example 2
A grammar for (arithmetic) expressions

- G = (V, T, P, <expr>) *variables enclosed in < >*
- V = { <expr>, <term>, <factor>} T = { a,+,X, (,) }
- P: <expr> → <expr> + <term> | <term>

  <term> → <term> X <factor> | <factor>

  <factor> → (<expr>) | a

(a) Is (a + a) X a generated by the grammar ?

(b) Is aaXa generated by the grammar ?

(a): <expr> => <term> => <term> X <factor> => <term> X a

<term>Xa => <factor>Xa => (<expr>) X a =>

=> (<expr> + <term>) Xa => (<expr> +<factor>)Xa

=> (a + a) X a

CFG Exercise

- G= ({S,A,B}, { (,) }, P, S)   alphabet/terminals are (,)
- $S \rightarrow SS \mid ASB \mid AB$
- A → (B →)
- What language does this grammar generate ?
- Check if grammar generates ((())), ())(, (()())
- What language does this grammar generate ?