Topic 6 - Grammar & Regular Languages

Week 11

Grammar

grammar — set of rules to connect strings

context-free grammar (CFG) — represent languages that cannot be represented by regular languages or FAs

- recursively describes structure of strings
- compact
- 4-tuple (V,Σ,R,S), where:
    - V = Variables — finites set of symbols; usually upper case letters
    - Σ = Terminals — finite set of letters; disjointed from V; usually lower case letters (ϵ is NOT included)
    - R = Rules — finite set of mappings with each mapping takes a variable and returns a string of variables and terminals
    - Start variable = S — member of V; usually on left hand side of top rule
- example process:

Generating strings

1. Start from the starting symbol, read its rule
2. Find a variable in the rule of the starting symbol and replace it with a rule of that variable
3. Repeat step 2 until there are no variables left.

- A derivation is a sequence of substitutions in generating a string
- There may be more than one rule for a variable. Then we can use "|" symbol to indicate or.
- For example: $S \to bSa|\ ba$

For example:

- $S \to bSa$
- $S \to ba$

$S \Rightarrow bSa \Rightarrow bbaa$

$S \Rightarrow bSa \Rightarrow bbSaa \Rightarrow bbbaaa$

$S \Rightarrow bSa \Rightarrow bbSaa \Rightarrow bbbSaaa \Rightarrow bbbbaaaa$

We say u derives v, or $u \Rightarrow^* v$ if there is a derivation from u to v.

  - S is V; a,b is Σ; bSa or ba is R; S is also S

derivation — sequence of substitutions in generating a string

Context-Free grammar

language of context-free grammar — set of all strings that can be derived from a grammar

- form. def. if G = (V,Σ,R,S) then L(G) = {w ∈ Σ*|S→*w}
- All strings can be derived from the starting symbol using rules of grammar
- example: what is the language of the following grammar: G_2:

S → aS | T ⟶ "|" is like "or" or "∪"

T→b | ϵ

a few strings in L(G_2): a, ab, b, ϵ, aa, aab

a few strings NOT in L(G_2): ba, abb,aabb

Therefore the language of G_2 is the union of all a and all a plus one b

L(G_2) = a*∪a*b = {a^i b^j | 0 ≤ i, 0 ≤ j ≤ 1}

designing CFG for a given context-free language:

1. analyise the language; find and dissect the strings
2. find recursive relation in structure of language
3. using recursive relations, building the grammar

- checklist for creating CFG:
    1. CONSISTENCY: is all strings generated by grammar fit the description
    2. COMPLETENESS: all strings in the description can be generated by grammar
        a. vice versa of CONSISTENCY
    3. TERMINATING RECURSIONS: all recursion used in grammar terminate
- example:

- Example 4: Design a CFG for the following $\{a^m b^n | n \geq m\}$
- $a^m b^n = a^m \ b^{n-m} \ b^m$
- If $i = n - m = 0$
  - Strings of the form $a^m b^m$: $S \rightarrow aSb|\epsilon$
- If $i > 0$:
  - Strings of the form b^i: $U \rightarrow bU|b$
    - $S \rightarrow aSb|U|\epsilon$
    - $\mathrm{U} \rightarrow bU|b$

  - note that in i>0 there is no a because as you can see the number of a is 0 whereas the number b is i
  - alternate way:

    S → aSb | U

    U → bU | ϵ

Week 12

- context-free languages are made of

  context-free grammar (CFG)
- regular languages are made of

  regular expressions (RE)
- Can all REs be converted to CFG?

  yes
- can all CFGs be converted to REs?

  no

---

practice problems for converting REs to CFGs:

- ▼ converted ab* to CFG

  b* = U → bU|ϵ

  ab* = S → aU

  CFG:

  S → aU

  U → bU|ϵ

- ▼ convert ab* ∪ b* to CFG

  b* = U → bU|ϵ

  ab* = S → aU

  CFG:

  S → aU|U

  U → bU|ϵ

- ▼ convert ab^+ ∪ b^+b to CFG

  b^+ = U → bU|b ⟶ not "or ϵ" because it's a + rather than a *

  ab^+ = S → aU

  b^+b = S → bU

  CFG:

  S → aU|bU

  U → bU|b

- ▼ convert Σ*aΣ* to CFG

  Σ* = (a ∪ b)*

  U → aX

  U → bX

  U → ϵ ⟶ since * (if + then instead of ϵ, add a|b)

  Σ* = U → aU|bU|ϵ

  CFG:

S → UaU

U → aU|bU|ϵ

- for further studies (will not be part of test daw) review Chomsky normal form