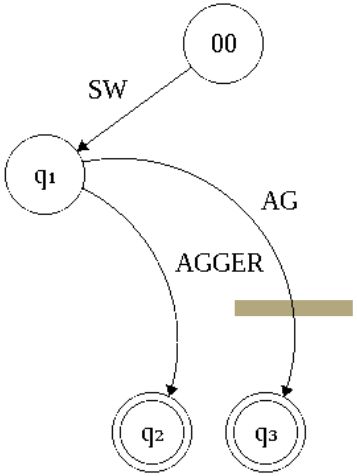


CS Theory

Midterm Review

Things to Think About:

Properties of regular languages, and the meaning of irregularity

Rules for manipulating regular languages (+, *, .)

Properties of CFGs

Relationship between CFLs and RLs, PDAs and FSM

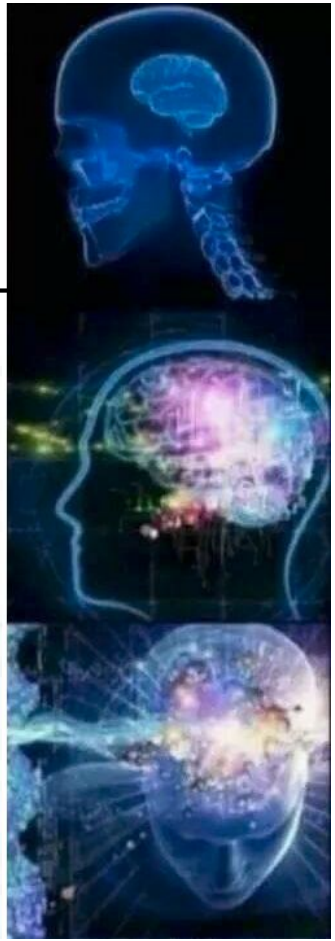
Design (NFA/DFA, PDA)

Compare and Contrast (think about the thing have learned in relation to each other)

Regular Languages

Context Free Languages

Recursively enumerable languages



In this Review

Regular Languages, proving regularity, closure properties

Context Free Languages, proving CF, closure properties

WHAT IS NON-DETERMINISM

Compare RLs and CLs

Algebraic Rules for RLs

Regular Languages

The first class of languages we saw was the class of regular languages. A language is regular if and only if it can be recognized by one of the following:

- An NFA
- A DFA
- A Regular Expression.

The above three representations are equivalent and we have seen methods of going from one to the other. (eg. subset construction for NFA to DFA, GNFA method with state ripping to go from an NFA to a REGEX, and a recursive transformation of any REGEX to a NFA).

Proving Regularity

We have also looked at proving how a language is regular. Here are some ways
to show it:

- Design a DFA recognizing the language
- Design an NFA recognizing the language
- Design a regular expression generating the language
- Use closure properties: (details on next slide)

Reminder of RL Closure Properties

- Regular Languages are closed under:
  - Complementation
  - intersection
  - Union
  - Concatenation
  - Kleene star (*) «that thing
- Ex: to Prove L is regular: show that L can be expressed as a combination of two regular languages using these operations (e.eg show L = L1 U L2 or L1 = ~L2)

- LM = ML ?

  Substitute 0 for L , 1 for M $\rightarrow$ Is 01 =10 ? No!

  This gives a counterexample: L={0}, M={1}
- L(M+N)=LM+LN?

  0(1+2) = 01+02 ? Yes (by definition of concatenation)
- (R+S)* = (R*S*)*

  (0+1)* = (0*1*)*: both are = set of all strings over {0,1}
- R*R* = R*

  0*0* = 0* : both sides are = set of all strings of 0's

UNDERSTAND WHY BOTH DIRECTIONS

- Two directions to prove that L(A) = desired language L
- $L(A) \subseteq L$: for every input string x, if x is accepted by A then x is in L
- $L \subseteq L(A)$: for every input string x, $x \in L \Rightarrow x \in L(A)$

Can often do both directions at same time (if and only if)

Proving Not Regular

- Pumping Lemma
- Closure properties,
- Ex using closure:
- To show L not regular, assume it is regular, and show that combining L with other languages known to be regular using closed properties gives a language we know not to be regular (contradiction!)
- L_1 intersection $L_2 = L_3$ where L_2 is known to be regular, and L_3 is known to be not regular, means we can conclude that L_1 was not regular.

Review Non-Determinism: It's ok to make mistakes!

- As long as SOME PATH EXISTS the non-deterministic machines accepts
- If your machine has multiple options on a given symbol, then it is NON-DETERMINISTIC
- Many paths in a non-deterministic string will fail, but as long as one works the string is accepted
- Similarly, a string that should be rejected must have NO VALID computation path.

CFL (Context Free Languages)

- A language L is context free if and only if it satisfies one of the following:
  - L can be generated by a CFG
  - L can be recognized by a PDA
- We saw conversion from CFG to equivalent PDA
- We saw how to construct PDAs

Proving Context Free

- Give CFG for the language
- Give PDA for the language
- Use closure properties

Constructing CFGs

- Recall recursive structure of CFLs
- Productions have variables (A), start variable (S), terminals (1):
  - $S \rightarrow A$
  - $A \rightarrow 0A1 \mid 1$
- Visualize using parse trees

Question: How do you construct a CFG for the language L = palindromes consisting of 0's and 1's?

Constructing PDAs

- Push and pop operations
  - Push: (a, Ɛ -> b)
  - Pop: (a, b -> Ɛ)
- Start by pushing $ to represent bottom of stack
- As before, Ɛ is a spontaneous transition
- Empty stack vs. end of input

Question: How do you construct a PDA for the following language?

$S \rightarrow AB$ $A \rightarrow aAb \mid Ɛ$ $B \rightarrow cB \mid Ɛ$

Limitations of CFLs

- Stack can differentiate between symbols, depending on design of PDA

- However, stack can only help us measure the relationship
  - No “size of stack” property
- For example, the language of all possible number of 1’s ($L = 1^n$) is not a CFL (why?)
- Is $L = \{a^n b^n c^n\}$ a CFL?

CFL Closure Properties

- Union
- Concatenation
- Kleene

NOT closed under

- Intersection
- Complement

IGNORE THE RECURSIVE PARTS (they come later)