Midterm II review

Date: 4/17/2017

Time: 10:10-11am

Location: In class

Closed book, closed notes

Main Topics

- Background on Regular Languages
  - Reg. lang. closure properties + DFA minimization
- CFGs
- PDAs
- CFLs & pumping lemma
- CFG simplification & normal forms

Regular Languages (Background)

- Building DFA, NFA, ε-NFA
- Building regular expressions
- Closure property results of regular languages
- Which languages cannot be regular and why?
  - Property
  - Pumping lemma

You need to know all material covered prior to Midterm I

CFGs

- G=(V,T,P,S)
- Derivation, recursive inference, parse trees
  - Their equivalence
- Leftmost & rightmost derivation
  - Their equivalence
  - Generate from parse tree
- Regular languages vs. CFLs
  - Right-linear & left-linear grammars

CFGs

- Designing CFGs (tips & techniques):
  - Making your own start symbol for combining grammars
    - Eg., $S_{new} \Rightarrow S_1 \mid S_2$ (or) $S_{new} \Rightarrow S_1 S_2$
  - Matching symbols & nested structures: (e.g., S => a **S** b | …)
  - Replicating nested structures side by side: (e.g., S => a S b **S**)
  - Use variables for specific purposes (similar to states)
  - To go to an “acceptance” from a variable
    - ==> end the recursive substitution by making it generate terminals directly
    - A => w
  - Conversely, to *not* go to acceptance from a variable, have recursion (loop back to same variable either directly or indirectly)

Proving CFGs are correct

- You will use induction either on
  - Input string length
  - Derivation length

To show: "IF a string is of a particular form (e.g., balanced paranthesis), THEN it will be generated by G
  - Use induction on string length

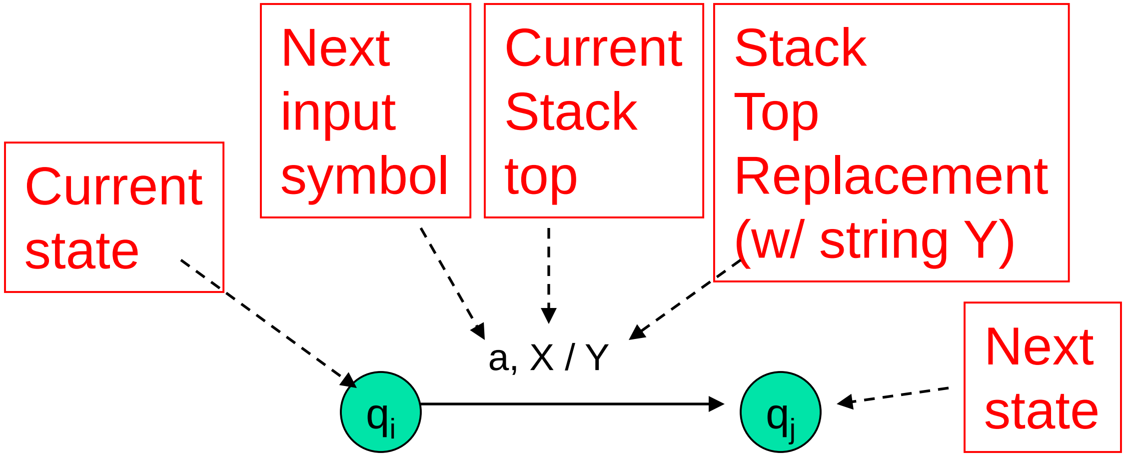
To show: "IF a string is generated by L(G), THEN it is of a particular form (e.g., balanced paranthesis)"
  - Use induction on derivation length

CFGs & ambiguity

- Ambiguity of CFGs
  - To show that a CFG is ambiguous, given one input string in the language which has more than one parse tree
    - (or equivalenty, >1 leftmost/rightmost derivation)
  - Finding one example is sufficient
- A CFL is *inherently ambiguous* if all grammars for that language are going to be ambiguous
- Converting ambiguous CFGs to non-ambiguous CFGs
  - Not possible for inherently ambiguous CFLs
  - For unambiguous CFLs, use ambiguity resolving techniques (e.g., precedence)

PDAs

- PDA ==> ε-NFA + “a stack”
- P = (Q,∑,Γ, δ,q_0,Z_0,F)
- δ(q,a,X) = {(p,Y), …}
- ID : (q, aw, XB) |--- (p,w,AB)
- State diagram way to show the design of PDAs



PDA - common mistakes

- Transition notation
  - <u>Goal:</u> ***push*** symbol 0 on top of the current stack top symbol 1

$$q_i \xrightarrow{a,\ 1\ /\ 0} q_j$$ X Why?

$$q_i \xrightarrow{a\ /\ 1\ 0} q_j$$ X Why?

$$q_i \xrightarrow{a,\ 1\ /\ 10} q_j$$ X Why?

$$q_i \xrightarrow{a,\ 1\ /\ 01} q_j$$ ✓

PDA - common mistakes…

- Transition notation
  - Goal: ***pop*** stack top if stack top is 0

$q_i \xrightarrow{a,\ 0\ /\ 0} q_j$ X Why?

$q_i \xrightarrow{a\ /\ 0} q_j$ X Why?

$q_i \xrightarrow{a,\ 00\ /\ 0} q_j$ X Why?

$q_i \xrightarrow{a,\ 0\ /\ \varepsilon} q_j$ ✓

> Remember:
> you can *push* multiple symbols in one step, but can *pop* only one symbol at a time

Design tips for PDAs

- Take advantage of the two types of PDAs
  - Acceptance by empty stack
    - If no more input <u>and</u> stack becomes empty
  - Acceptance by final state
    - If no more input <u>and</u> end in final state
- Convert one form to another
- Assign state for specific purposes
- Push to “remember” and Pop to “tally”
- Introducing your own stack symbols may help
- Take advantage of non-determinism

PDA design restrictions

- Feel free to design an empty stack PDA or final state PDA unless otherwise explicitly specified
  - This is meant for design convenience
- But if I ask you design a specific type of PDA in the question, then show a direct construction
  - i.e., do not convert one to another
- Same applies for PDA vs. CFG
  - i.e., If I ask you to design a PDA, then give a direct construction (do not convert from CFG)
  - Same for CFG

Conversion procedures

- Be familiar with:
  - CFG => PDA conversion
  - PDA empty stack => PDA final state
  - PDA final state => PDA empty stack

Follow the algorithms described in class.
if you come up with an ad hoc way that works for that example but not necessarily for others, then that could lead to reduction of points

CFG Simplification

1. Eliminate ε-productions: A => ε
   - ==> substitute for A (with & without)
   - Find nullable symbols first and substitute next
2. Eliminate unit productions: A=> B
   - ==> substitute for B directly in A
   - Find unit pairs and then go production by production
3. Eliminate useless symbols
   - Retain only reachable and generating symbols
   - Order: first generating test, and then reachability test

- Order is important : steps (1) => (2) => (3)

Chomsky Normal Form

- All productions of the form:
  - A => BC   or   A=> a
- Grammar does not contain:
  - Useless symbols, unit and ε-productions
- Converting CFG (without S=>*ε) into CNF
  - Introduce new variables that collectively represent a sequence of other variables & terminals
  - New variables for each terminal
- CNF ==> Parse tree size
  - If the length of the longest path in the parse tree is n, then $|w| \leq 2^{n-1}$.

Pumping Lemma for CFLs

- Then there exists a constant n, s.t.,
  - if z is any string in L s.t. $|z| \geq n$, then we can write $z=uvwxy$, subject to the following <u>conditions:</u>
    1. $|vwx| \leq n$
    2. $vx \neq \varepsilon$
    3. For all $k \geq 0$, uv^kwx^ky is in L

Using Pumping Lemmas for CFLs

- Steps:
  1. Let n be the P/L constant
  2. Pick a word z in the language s.t. $|z| \geq n$
     - (choice critical - any arbitrary choice may not work)
  3. z=uvwxy
  4. First, argue that because of conditions (1) & (2), the portions covered by vwx on the main string z will have to satisfy some properties
  5. Next, argue that by pumping up or down you will get a new string from z that is not in L

Refer to the exercises done in class as examples

Closure Properties for CFL

- CFLs are closed under:
  - Union
  - Concatenation
  - Kleene closure operator
  - Substitution
  - Homomorphism, inverse homomorphism
- CFLs are *not* closed under:
  - Intersection
  - Difference
  - Complementation

Good luck !!