CS 3313
Foundations of Computing:

Exam2 Review

https://gw-cs3313.github.io/

Pushdown Automata (PDA)

- Intuition: think of PDA as a NFA with stack storage
  - Stack storage implies first-in last-out
    - If you use stack as counter variable, then effectively one counter
- Nondeterministic: the PDA can have a choice of next moves

  For each choice, the PDA can:
  1. Change state, and also
  2. Replace the top symbol on the stack by a sequence of zero or more symbols.
     - Zero symbols = “pop.”
     - Many symbols = sequence of “pushes”.
- Stack alphabet = symbols that can be stored on stack
  - To store the type of input read, we can associate a stack symbol with each input symbol, ex: X for a, Y for b, etc.
  - Z is starting stack symbol

Example 1: L = { w | w has equal number of a's and b's}

- Logic: (start state is q_0 and start stack is Z)
  - Stack symbols X for input a, Y for input b
  - For every a in the input, there is a b that cancels it out
  - Push X if you read an a, with Z or X at TOS
  - Push Y if you read a b, with Z or Y at TOS
  - If you read a and Y is TOS, then "pop" (cancel a with a previous b)
  - If you read b and X is TOS, then "pop" (cancel b with a previous a)

  - If you have an equal number then at the end of input you will have
    - Empty string to read on input
    - Start stack on TOS
    - (q, λ, Z) is the ID (Instantaneous description)

This is the type of reasoning (w/ states) you want to provide; instead of just rewrite everything in δ form.

Example 2: L = { $a^m b^n$ | n=m or n=2m }

- Strings of the form aaabbb or aaabbbbbb
- Use non-determinism to design the PDA:
  - PDA1 accepts $n=m$ or
  - PDA2 accepts $n=2m$
  - Start PDA in q_0 and jump non-deterministically on empty string input to PDA1 or PDA2
  - Important: in both PDA1 and PDA2, to check for pattern of *b*'s after *a*'s, we need to change state after reading *b*.
- PDA1 starts in q_1
  - For every *a* in input, push X. For every *b*, pop one X
- PDA2 starts in q_2
  - For every a in input, push 2 X's. For every b, pop one X
- If either PDA1 or PDA2 accept then PDA M accepts.

Example 3: $L = \{ w w^R \mid w \text{ in } \{a,b\}^* \}$

- Recognizing wcw^R was "easy"
  - Keep pushing input until you read *c*
  - After reading *c*, compare input with TOS
  - The location of *c* gives you the 'midpoint' (the middle of the string)
- In ww^R there is no *c* in input to denote midpoint
- So "guess" non-deterministically
  - If it is the midpoint then input you read = last input you read (i.e. TOS)
  - After midpoint, "match" input with TOS (*read a, X is TOS; read b with Y as TOS*)…until you hit end of string and *Z* as TOS
- Ex: abbbba
  - a ↑ bbbba can this be the midpoint?
  - ab↑bbba can this be the midpoint?
  - abb ↑ bba can this be midpoint ?
  - abbb↑ ba can this be midpoint ?

Questions regarding PDAs from HW

Note how to pop multiple chars.

Context Free Grammars

- Designing grammar for a given language
  - Ex: { ww^R }: generating from the "outside" to the "inside" –key characteristic of CFG
  - Ex: {a^nb^{2n} }:
    - generate from outside to inside
    - "count" two b's for every a, generate 2 b's for every a
  - Ex: {a^nb^n} ∪ {a^nb^{2n}} : "parallel" generation – generating two parallel paths
    - One for {a^nb^n} and other for {a^nb^{2n}}
  - Ex: : $\{a^m b^n c^k \mid m = n + k,\ m, n, k \geq 0\}$
    - Each generated a is matched with either a b or a c
    - "progressively" generating: one path leads to another

Cleanup and Simplification of CFGs

- Useless productions & useless symbols
  - Useless if variable A does not derive any terminal string
  - Useless if variable A is not reachable from S
- Removing λ productions
  - Nullable variables: variable that can derive empty string
  - Replace nullable variables in production
    - Can lead to unit productions
- Removing unit productions

Grammar Cleanup Example

$S \to aA \mid AC \mid aBB$ $\quad A \to aaA \mid \lambda$

$B \to bB \mid bbC$ $\quad C \to B$

1. Remove λ-productions

Order matters!

- A is nullable, replace $S \to aA \mid AC$ and $A \to aaA$ with $S \to aA \mid AC \mid a \mid C$ and $A \to aaA \mid aa$ to get grammar:
  - $S \to aA \mid AC \mid a \mid C \mid aBB$ and $A \to aaA \mid aa$
  - $B \to bB \mid bbC$ $\quad C \to B$

2. Remove unit productions: $S \to C$, $C \to B$, and $S \Rightarrow^* B$ to get:
  - $S \to aA \mid AC \mid a \mid bB \mid bbC \mid aBB$ $\quad A \to aaA \mid aa$
  - $B \to bB \mid bbC$ $\quad C \to bB \mid bbC$

3. Remove useless Prod.

- (i) B, C do not derive terminal string: eliminate all productions containing B, C
- (Note: Not just erasing all the B's and C's.)
- Final grammar: $S \to aA \mid a$ $\quad A \to aaA \mid aa$

CNF grammar

- : A CFG $G = (V, T, P, S)$ is in Chomsky Normal Form (CNF) if all productions are of the form
  - $A \rightarrow BC$, or
  - $A \rightarrow a$,
- Ex 1: G_1 with production rules:
  - $S \rightarrow AS \mid a$
  - $A \rightarrow SA \mid b$
    - Is G_1 in CNF?
- Ex 2: G_2 with production rules:
  - $S \rightarrow AS \mid AAS$
  - $A \rightarrow SA \mid aa$
    - Is G_2 in CNF?...Convert:
    - $S \rightarrow AS \mid AD_1$ $\qquad D_1 \rightarrow AS$
    - $A \rightarrow SA \mid B_a B_a$ $\qquad B_a \rightarrow a$

CYK Algorithm

- w_{ij} : substring (of input w) that starts at position i and has length j
  - $w = w_{1n}$
- V_{ij} : set of variables that derive w_{ij}
  - *If $S \in V_{1n}$ then w is generated by the grammar*
- $V_{ij} = \bigcup_{1 \leq k \leq j-1} \{ A \mid A \rightarrow BC \text{ and } B \in V_{ik} , C \in V_{i+k\, j-k} \}$

Example: Application of CYK Algorithm

- S →AB | BC   A→ BA | a   B → CC | b   C→ AB | a
- w = baaba (length 5), so i,j iterate from 1 to 5.

i →

j ↓

b	a	a	b	a
B	A, C	A, C	B	A, C
S, A	B	S, C	A, S	
{}	B	B		
{}	B			
S, A, C				

Pattern of computation

S is in V_{15} therefore w is in L(G)

Questions regarding CFGs from HW

Properties of CFLs

- Closure Properties…CFLs are closed under:
  - Union, Concatenation, Star Closure
  - Homomorphism
  - Intersection with regular languages
- Decision properties: algorithms for
  - Emptiness, finiteness, membership (does w belong to the language)
- To prove a language is CFL, provide a CFG or PDA
- To prove a language is not CFL, prove using pumping lemma
  - Assume it is CFL and derive a contradiction

Statement of the CFL Pumping Lemma

For every context-free language L

There is an integer n, such that

For every string z in L of length $\geq$ n

There exists z = uvwxy such that:

1. $|vwx| \leq n$.
2. $|vx| > 0$.
3. For all $i \geq 0$, uv^iwx^iy is in L.

- You cannot fix the value of n
- vwx can fall anywhere in the string as long as it satisfies $|vwx| \leq n$
  => have to consider all cases for vwx

Examples L_1: { a^i | i is a prime number}

- *L_1: { a^i | i is a prime number} …NOT CFL*
  - Intuition: We need to run some kind of algorithm that has to remember which primes have been checked with i.
  - Application of pumping lemma similar to proof that this language is not regular – and we only have one case for splitting the string into uvwxy
- *L_2: { w | w {a,b,c}*, and $n_a(w) = n_b(w) * n_c(w)$ }*
  - Intuition: we need to keep track of number of b's and c's, and then multiply the two…multiplication using repeated addition implies we need to store two variables ($n_b(w)$ and $n_c(w)$): likely not context free
  - This language does not place restrictions on the pattern
    - We can have a's after b's etc.

L_2: { w | w {a,b,c}*, and $n_a(w) = n_b(w) * n_c(w)$ }

- Assume context free, let n be the constant of the lemma
- We need to pick values for $n_a(w)$, $n_b(w)$, $n_c(w)$ which will make it easy to prove the $n_a(w)$ in pumped string cannot be the product of $n_b(w)$ and $n_c(w)$
- Additionally, pick a pattern that makes it easier to determine the different cases of *vwx*

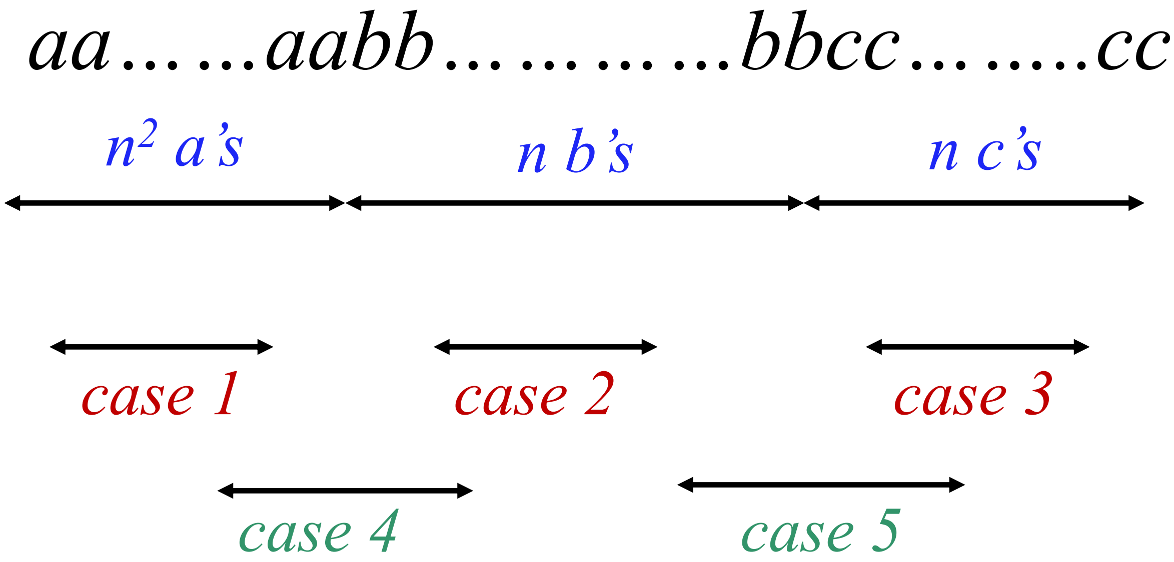
L_2: { w | w {a,b,c}*, and $n_a(w) = n_b(w) * n_c(w)$

- Assume context free, let n be the constant of the lemma
  - Additionally, pick a pattern and values that makes it easier to determine the different cases of *vwx*
- Let n be the constant and pick $z = a^m b^n c^n$ where $m = n^2$
  - *why pick this as z ?*
  - We want to construct an instance of $n_b(w) * n_c(w)$ which will make it easier to contradict: if we pick perfect squares then we know that the next perfect square after n^2 *is* $(n+1)^2$ which is $(2n+1)$ more than n^2
  - Lemma states, $|vwx| \leq n$ *and* $|vx| \geq 1$
- Next: look at the possible cases for where vwx could be
  - We need to find a contradiction for each of these cases

aa……aabb…………bbcc……..cc

L_2: { w | w {a,b,c}*, and $n_a(w)= n_b(w)*n_c(w)$

- look at the possible cases for where vwx could be
  - We need to find a contradiction for each of these cases



Observation:
vx in cases 1,2,3 consist of one type of symbol/terminal
vx in cases 4,5 consists of two types of symbols

Cases 4

$L_{3=}$ { w | w {a,b,c}*, and $n_a(w) = n_b(w) * n_c(w)$

- Case 4: *vx* consists of *j* a's and *k* b's – we don't care about the exact pattern
- **From conditions of the lemma**, $|vwx| \leq n$ *and* $|vx| \geq 1$ **implies**

  $(j+k) > 0$ *and* $(j+k) \leq n$

- Therefore, z' = uv^2wx^2y will have
  - $n_a(z') = (n^2 + j)$ *(number of a's)*
  - $n_b(z') = (n + k)$
  - $n_c(z') = n$
- Question: is $(n^2 + j) = n(n+k)$?
  - If $n^2 + j = n^2 + nk$ then $j = nk$
    - If $k=0$ then $j=0$ **contradiction since** $(j+k)>0$
    - If $k>0$ then $j = nk \geq n$ and thus $(j+k) > n$ **contradiction** since $(j+k) \leq n$

Example:

$L_3 = \{ x\, w\, w^R\, y \mid x=y,\ x,y \in \{0,1\}^*,\ w \in \{a,b\}^* \}$

- Intuition: While recognizing ww^R can be done using a stack, recognizing $x=y$ implies a stack storage is not sufficient
  - This property is like the language *ww* – see book for proof that it is not context free.
- Application of pumping lemma:
  - Approach 1: carefully choose the string so we can simplify the proof – choose $w=\lambda$ and $z= 0^n1^n0^n1^n$
  - Approach 2: use closure properties…
    - CFLs are closed under intersection with regular language
    - Therefore if L_3 is CFL, then $L = L_3 \cap \{0,1\}^*$ is CFL
    - But $L = \{ww \mid x \text{ in } \{0,1\}^*\}$
    - Observation: Applying closure properties can sometimes simplify the proof

Note the difference between a string and a language.

Questions regarding CFLs from HW