Context Free Languages VI

Life Beyond CFLs

Homework

- Homework #5
  - Due today
- Homework #6 (due 10/29)
  - From textbook
    - 7.1
    - 7.5b,c
    - 7.6 (first PDA only)
    - 7.13a,b (deterministic PDA)
    - 7.21a

Before We Start

- Any questions?

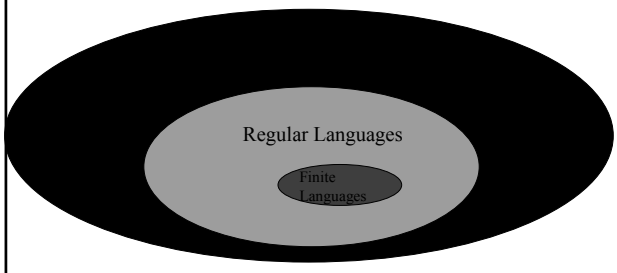
Languages

- Future Exam Question
  - What is a language?
  - What is a class of languages?

Context Free Languages

- Context Free Languages(CFL) is the next class of languages outside of Regular Languages:
  - Means for defining: Context Free Grammar
  - Machine for accepting: Pushdown Automata

Now our picture looks like



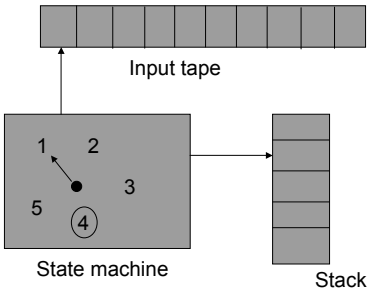
Context Free Grammars

- Let's formalize this a bit:
  - A context free grammar (CFG) is a 4-tuple: (V, Σ, S, P) where
    - V is a set of variables
    - Σ is a set of terminals
    - V and Σ are disjoint (I.e. $V \cap \Sigma = \varnothing$)
    - $S \in V$, is your start symbol

Context Free Grammars

- Let's formalize this a bit:
  - Production rules
    - Of the form $A \rightarrow \beta$ where
      - $A \in V$
      - $\beta \in (V \cup \Sigma)^*$ string with symbols from V and Σ
    - We say that γ can be derived from α in one step:
      - $A \rightarrow \beta$ is a rule
      - $\alpha = \alpha_1 A \alpha_2$
      - $\gamma = \alpha_1 \beta \alpha_2$
      - $\alpha \Rightarrow \gamma$

Pushdown Automata



Pushdown Automata

- Let's formalize this:
  - A pushdown automata (PDA) is a 7-tuple:
    - $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ where
      - Q = finite set of states
      - Σ = tape alphabet
      - Γ = stack alphabet (may have symbols in common w/ Σ)
      - $q_0 \in Q$ = start state
      - $Z_0 \in \Gamma$ = initial stack symbol
      - $A \subseteq Q$ = set of accepting states
      - δ = transition function

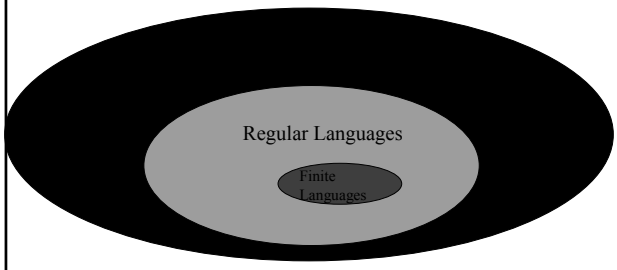
Pushdown Automata

- About this transition function δ:
  - During a move of a PDA:
    - At most one character is read from the input tape
      - Λ transitions are okay
    - The topmost character is popped from the stack
      - Unless it is Z_0
    - The machine will move to a new state based on:
      - The character read from the tape
      - The character popped off the stack
      - The current state of the machine
    - 0 or more symbols from the stack alphabet are pushed onto the stack.

Plan for today

- Just in time for Halloween!
  - The Return of the Pumping Lemma
- Closure Properties and Decision Properties for CFLs

Now our picture looks like



Is there anything out here?

Just when you thought it was safe

- Return of the Pumping Lemma
- Recall:
  - With Regular Languages
    - The Pumping Lemma showed that if a string in a regular language is "long enough", it would have to visit at least one state in the corresponding FA more than once.

Just when you thought it was safe

- Return of the Pumping Lemma
  - But before we start that!

Chomsky Normal Form

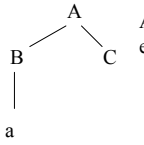
- Chomsky Normal Form
  - A context free grammar is in Chomsky Normal Form (CNF) if every production is of the form:
    - $A \rightarrow BC$
    - $A \rightarrow a$
    - Where A,B, and C are variables and a is a terminal.

Theory Hall of Fame

- Noam Chomsky
  - The Grammar Guy
  - 1928 –
  - b. Philadelphia, PA
  - PhD – UPenn (1955)
    - Linguistics
  - Prof at MIT (Linguistics) (1955 - present)
  - Probably more famous for his leftist political views.

Chomsky Normal Form

- If we can put a CFG into CNF, then we can calculate the "depth" of the longest branch of a parse tree for the derivation of a string.



At most 2 branches at every node

Chomsky Normal Form

- 4 Step process:
  1. Remove Λ Productions
  2. Remove Unit Productions
  3. Terminal Productions
  4. Variable Productions

Removing Λ-Productions

- A Λ-Productions is a production of the form
  - $A \rightarrow \Lambda$
  - Basic idea
    - Very similar to removing Λ transitions from a NDFA-Λ
    - Find the set of all variables A such that $A \Rightarrow^* \Lambda$ (set of nullable variables)
    - For all productions that contain a nullable variable on the right hand side, add a production that eliminates the nullable from the right hand side

Removing Λ-Productions

- We must be a bit careful here
  - If Λ is in a CFL, then the production $S \rightarrow \Lambda$ must be in the production set.
  - The algorithm to be described will generate L – {Λ}

Removing Λ-Productions

- Step 1: Find the set of nullable variables:
  - Recursive definition of a nullable variable
    - Let G = (V, Σ, S, P) be a CFG
    1. Any variable A such that there is a production in P: $A \rightarrow \Lambda$ is a nullable variable
    2. If P contains $A \rightarrow B_1B_2B_3\ldots B_N$ and $B_1, B_2, \ldots B_N$ are nullable variables, then A is nullable
    3. No other variables in V are nullable

Removing Λ-Productions

- Step 1: Find the set of nullable variables:
  - Algorithm:
    - Let N be your set of nullables
    - Add to N all variables A such that $A \rightarrow \Lambda$
    - While new variables are being added to N
      - If there is a production $B \rightarrow \alpha$ and all symbols of α are in N, add B to N

Removing Λ-Productions

- Step 1: Find the set of nullable variables:
  - Example:
    - $S \rightarrow AB$
    - $A \rightarrow aAA \mid \Lambda$
    - $B \rightarrow bBB \mid \Lambda$
    - All variables are nullable
      - A and B are nullable since $A \rightarrow \Lambda$ and $B \rightarrow \Lambda$
      - S is nullable since $S \rightarrow AB$ and A and B are nullable

Removing Λ-Productions

- Step 2: Remove nullable variables
  - For all productions $A \rightarrow \beta$ where β contains nullable variables, add a new production with each nullable removed from β

Removing Λ-Productions

Step 2: Remove nullable variables Example:

- $S \rightarrow AB$
- $A \rightarrow aAA \mid \Lambda$
- $B \rightarrow bBB \mid \Lambda$

- All variables are nullable

Removing Λ-Productions

- Step 2: Remove nullable variables Example:
  - Consider: $S \rightarrow AB$
    - Add to P: $S \rightarrow A$ and $S \rightarrow B$
  - Consider: $A \rightarrow aAA$
    - Add to P: $A \rightarrow aA$ and $A \rightarrow a$
  - Consider: $B \rightarrow bBB$
    - Add to P: $B \rightarrow bB$ and $B \rightarrow b$

Removing Λ-Productions

- Step 2: Remove nullable variables
  - Our grammar now looks like:
    - $S \rightarrow AB \mid A \mid B$
    - $A \rightarrow aAA \mid aA \mid a \mid \Lambda$
    - $B \rightarrow bBB \mid bB \mid b \mid \Lambda$

Removing Λ-Productions

- Step 3: Remove your Λ-Productions
  - Example:
    - Remove $A \rightarrow \Lambda$ and $B \rightarrow \Lambda$
    - Our final grammar looks like:
      - $S \rightarrow AB \mid A \mid B$
      - $A \rightarrow aAA \mid aA \mid a$
      - $B \rightarrow bBB \mid bB \mid b$
  - Questions?

Removing Unit Productions

- A Unit Productions is a production of the form
  - $A \rightarrow B$ where A and B are variable
- Basic idea
  - Very similar to removing Λ productions
  - For each variable A, find the set of all variables B such that $A \Rightarrow^* B$ by just following unit productions (A-derivable)
  - For all variables B that are A derivable and for all productions $B \rightarrow \alpha$, add the production $A \rightarrow \alpha$

Removing Unit Productions

- Step 0: Remove Λ-Productions using the previous algorithm.

Removing Unit Productions

- Step 1: For all variables A find the set of A-derivable variables:
  - Recursive definition of A-derivable
    1. If $A \rightarrow B$ then B is A-derivable
    2. If C is A derivable and $C \rightarrow B$ (and $B \neq A$), then B is A derivable
    3. No other variables are A-derivable.

Removing Unit Productions

- Step 1: For all variables A find the set of A-derivable variables:
  - Example:
    - $S \rightarrow S + T \mid T$
    - $T \rightarrow T * F \mid F$
    - $F \rightarrow (S) \mid a$
    - Let's find the set of S-derivable variables:
      - T is S derivable since $S \rightarrow T$
      - F is S derivable since $T \rightarrow F$ and T is S derivable

Removing Unit Productions

- Step 1: For all variables A find the set of A-derivable variables:
  - Example:
    - $S \rightarrow S + T \mid T$
    - $T \rightarrow T * F \mid F$
    - $F \rightarrow (S) \mid a$
    - S-derivable = {T, F}
    - T-derivable = {F}
    - F-derivable = ∅

Removing Unit Productions

- Step 2: For each variable A, if B is A-derivable, for each non-unit production $B \rightarrow \beta$, add the production $A \rightarrow \beta$

Removing Unit Productions

- Step 2:
  - Example:
    - $S \rightarrow S + T \mid T$
    - $T \rightarrow T * F \mid F$
    - $F \rightarrow (S) \mid a$
    - S-derivable = {T, F}
    - T-derivable = {F}
    - Add to P: $S \rightarrow T * F$, $S \rightarrow (S) \mid a$
    - : $T \rightarrow (S) \mid a$

Removing Unit Productions

- Step 2:
  - Our new grammar now looks like:
    - $S \rightarrow S + T \mid T * F \mid (S) \mid a \mid T$
    - $T \rightarrow T * F \mid (S) \mid a \mid F$
    - $F \rightarrow (S) \mid a$

Removing Unit Productions

- Step 3: Remove Unit Productions
  - Our final grammar looks like:
  - Our new grammar now looks like:
    - $S \rightarrow S + T \mid T * F \mid (S) \mid a$
    - $T \rightarrow T * F \mid (S) \mid a$
    - Remove $S \rightarrow T$, $T \rightarrow F$
  - Questions

Recall our goal

- Chomsky Normal Form
  - A context free grammar is in Chomsky Normal Form (CNF) if every production is of the form:
    - $A \rightarrow BC$
    - $A \rightarrow a$
    - Where A,B, and C are variables and a is a terminal.

Chomsky Normal Form

- Given a CFG G, there is an equivalent CFG, G' in Chomsky Normal form such that
  - $L(G') = L(G) - \{\Lambda\}$

Chomsky Normal Form

- Step 1:
  - Remove Λ-Productions
- Step 2:
  - Remove Unit Productions

Chomsky Normal Form

- After steps 1 & 2:
  - All productions are of the form:
    - $A \rightarrow a$ where A is a variable and a is a terminal
    - $A \rightarrow \beta$ where $|\beta| \geq 2$ and β contains variables and/or terminals.
  - Step 3: Derive terminals from new variables:
    - For all productions of the 2nd type: $A \rightarrow \beta$, for all terminals a in β, create a new variable X_a
    - Add a new production $X_a \rightarrow a$
    - Replace a in β with X_a

Chomsky Normal Form

- Step 3:
  - Let's go back to our first example:
    - S → AB | A | B
    - A → aAA | aA | a
    - B → bBB | bB | b
  - Removing unit transitions:
    - S → AB | aAA | aA | a | bBB | bB | b
    - A → aAA | aA | a
    - B → bBB | bB | b

Chomsky Normal Form

- Step 3:
  - Define new productions: $X_a \rightarrow a$ and $X_b \rightarrow b$ and replace instance of a with X_a, similarly for b
    - S → AB | aAA | aA | a | bBB | bB | b
    - A → aAA | aA | a
    - B → bBB | bB | b
  - New:
    - S → AB | X_aAA | X_aA | a | X_bBB | X_bB | b
    - A → X_aAA | X_aA | a
    - B → X_bBB | X_bB | b
    - $X_a \rightarrow a$
    - $X_b \rightarrow b$

Chomsky Normal Form

- After steps 1 & 2 & 3:
  - All productions are of the form:
    - A → a where A is a variable and a is a terminal
    - A → β where $|\beta| \geq 2$ and β contains only variables.
  - Step 4:
    - For all productions of type 2 where $|\beta| > 2$, replace the production with a series of new productions each having exactly 2 variables on the right
    - Best illustrated with an example

Chomsky Normal Form

- Step 4:
  - The production:
    - A → BCDBCE
  - Would be replaced with
    - $A \rightarrow BY_1$
    - $Y_1 \rightarrow CY_2$
    - $Y_2 \rightarrow DY_3$
    - $Y_3 \rightarrow BY_4$
    - $Y_4 \rightarrow CE$

Chomsky Normal Form

- Step 4:
  - Back to our example
    - S → AB | $\underline{X_a AA}$ | X_aA | a | $\underline{X_b BB}$ | X_bB | b
    - A → $\underline{X_a AA}$ | X_aA | a
    - B → $\underline{X_b BB}$ | X_bB | b
    - $X_a \rightarrow a$
    - $X_b \rightarrow b$
  - Add productions
    - $Y_1 \rightarrow AA$
    - $Y_2 \rightarrow BB$

Chomsky Normal Form

- Step 4:
  - Our final grammar
    - S → AB | $\underline{X_a Y_1}$ | X_aA | a | $X_b Y_2$ | X_bB | b
    - A → $X_a Y_1$ | X_aA | a
    - B → $X_b Y_2$ | X_bB | b
    - $Y_1 \rightarrow AA$
    - $Y_2 \rightarrow BB$
    - $X_a \rightarrow a$
    - $X_b \rightarrow b$
  - Questions

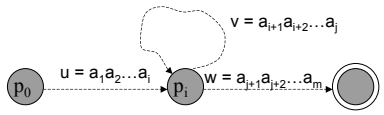
CNF

- Any grammar can be placed into CNF
- Questions?
- Breaktime

The Pumping Lemma for RL

- Statement of the pumping lemma for RL
  - Let L be a regular language.
  - Then there exists a constant n (which varies for different languages), such that for every string $x \in L$ with $|x| \geq n$, x can be expressed as x = uvw such that:
    1. $|v| > 0$
    2. $|uv| \leq n$
    3. For all $k \geq 0$, the string $uv^k w$ is also in L.

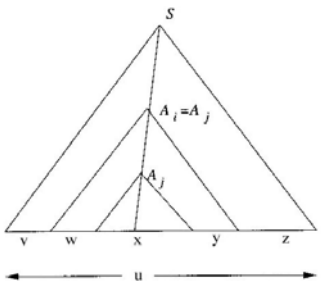
The Pumping Lemma for RL



The Pumping Lemma for CFLs

- With CFLs
  - strings are distinguished by their derivation (or parse trees) based on the productions of a CFG.
  - The idea behind the Pumping Lemma for CFLs:
    - If a string is long enough, then at least one variable in it's derivation will have to be repeated.
    - We can repeatedly reapply productions for the repeated variable ("pump you up") and the resultant string will also be in the language

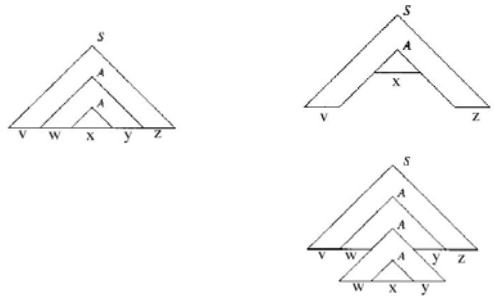
The Pumping Lemma for CFLs



The Pumping Lemma for CFLs

- $S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vwxyz$
  - So $A \Rightarrow^* x$ but also $A \Rightarrow^* wAy$
  - We can then write instead:
    - $S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vw^2Ay^2z \Rightarrow^* vw^3Ay^3z$
    - And so on…

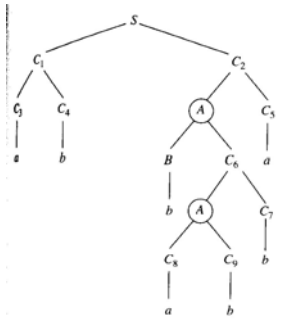
The Pumping Lemma for CFLs



The Pumping Lemma for CFLs

- How long is long enough?
  - Recall Chomsky Normal Form
    - A context free grammar is in Chomsky Normal Form (CNF) if every production is of the form:
      - $A \rightarrow BC$
      - $A \rightarrow a$
    - Where A,B, and C are variables and a is a terminal.

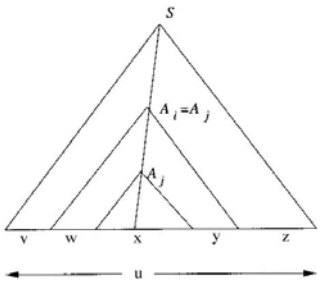
The Pumping Lemma for CFLs



The Pumping Lemma for CFLs

- The parse tree for a grammar in CNF will be a binary tree
  - A binary tree having more than 2^{k-1} leaf nodes must have a height (longest path) > k.
  - If we let k be the number of number of variables in our grammar, then
    - For any string x, where $|x| > 2^k$
    - At least one variable in the longest path will be repeated.

The Pumping Lemma for CFLs



The Pumping Lemma for CFLs

- Let L be a CFL. Then there is an integer n so that for all strings u, where $|u| \geq n$, u can be expressed as u = vwxyz where
  - $|wy| > 0$
  - $|wxy| \leq n$
  - For any $m \geq 0$, $vw^mxy^mz \in L$
- $n = 2^{p+1}$ where p = number of variables

The Pumping Lemma for CFLs

- The real strength of the pumping lemma is proving that languages are not context free
  - Proof by contradiction
    - Assume that the language to be tested is a CFL
    - Use the pumping lemma to come to a contradiction
    - Original assumption about the language being a CFL is false
- You cannot prove a language to be a CFL using the Pumping Lemma!!!!

The Pumping Lemma for CFLs

- The Pumping Lemma game
  - To show that a language L is not a CFL
    - Assume L is context free
    - Choose an "appropriate" string x in L
    - Express x = uvwxy following rules of pumping lemma
    - Show that uv^kwx^kz is not in L, for some k
    - The above contradicts the Pumping Lemma
    - Our assumption that L is context free is wrong
    - L must not be context free

The Pumping Lemma for CFLs

- Example:
  - $L = \{ a^ib^ic^i \mid i \geq 1 \}$
  - Strings of the form abc where number of a's, b's and c's are equal
  - Let's play!
  - Assume that L is context free. Then by the pumping lemma all strings u with $|u| \geq n$ can be expressed as u = vwxyz and
    - $|wy| > 0$
    - $|wxy| \leq n$
    - For any $m \geq 0$, $vw^mxy^mz \in L$

The Pumping Lemma for CFLs

- Example
  - $L = \{ a^ib^ic^i \mid i \geq 1 \}$
  - Choose an appropriate $u = a^nb^nc^n = vwxyz$
  - Since $|wxy| \leq n$ then wxy must consists of
    - All a's or all b's or all c's
    - Some a's and some b's
    - Some b's and some c's

The Pumping Lemma for CFLs

- In all three cases
  - vw^2xy^2z will not have an equal number of a's b's and c's.
  - Pumping Lemma says $vw^2xy^2z \in L$
  - Can't contradict the pumping lemma!
  - Our original assumption must be wrong.
  - L is not context-free.

The Pumping Lemma for CFLs

- By the same argument (same choice of u), we can show that:
  - $L = \{ x \in \{ a,b,c\}^* \mid n_a(x) = n_b(x) = n_c(x) \}$
- Is not context free

The Pumping Lemma for CFLs

- Another Example:
  - L = { $a^i b^j c^k$ | i < j and i < k }
  - Number of a's is less than the number of b's and the number of c's
  - Let's play!
  - Assume that L is context free. Then by the pumping lemma all strings u with |u| ≥ n can be expressed as u = vwxyz and
    - |wy| > 0
    - |wxy| ≤ n
    - For any m ≥ 0, $vw^m xy^m z \in L$

The Pumping Lemma for CFLs

- Example
  - L = { $a^i b^j c^k$ | i < j and i < k }
  - Choose an appropriate $u = a^n b^{n+1} c^{n+1} = vwxyz$
  - Since |wxy| ≤ n then wxy must consists of
    - Case 1: All a's or all b's or all c's
    - Case 2: Some a's and some b's
    - Case 3: Some b's and some c's

The Pumping Lemma for CFLs

- Let's consider each case individually:
  - Case 1: All a's or all b's or all c's
    - If wxy consists of all a's then there will be k such that when we pump w and y k times, the number of a's will be greater than n+1
    - If wxy consists of all b's then vw^0xy^0z will contain the same number or less b's than a's
    - If wxy consists of all c's then vw^0xy^0z will contain the same number or less c's than a's

The Pumping Lemma for CFLs

- Let's consider each case individually:
  - Case 2: Some a's and some b's
    - If wxy consists of only a's and b's then there will be k such that when we pump w and y k times, the number of a's will be greater than n+1 (# of c's)
    - Relationship between a's and b's might be maintained, but not the relationship between a's and c's

The Pumping Lemma for CFLs

- Let's consider each case individually:
  - Case 2: Some b's and some c's
    - If wxy consists of only b's and c's then vw^0xy^0z will contain the same number or less c's or b's than a's

The Pumping Lemma for CFLs

- In all cases
  - We found a "pumped" (or unpumped) string that the pumping lemma said should be in the langauge but did not maintain the relationship of a's to b's and c's as specified in the language.
  - Can't contradict the pumping lemma!
  - Our original assumption must be wrong.
  - L is not context-free.

The Pumping Lemma for CFLs

- By the same argument (same choice of u), we can show that:
  - $L = \{ x \in \{a,b,c\}^* \mid n_a(x) < n_b(x) \text{ and } n_a(x) < n_c(x) \}$
- Is not context free

The Pumping Lemma for CFLs

- Questions?

Closure Properties

- We already seen that CFLs are closed under:
  - Union
  - Concatenation
  - Kleene Star
- Regular Languages are also closed under
  - Intersection
  - Complementation
  - Difference
- What about Context Free Languages?

Closure Properties

- Sorry, Charlie
  - CFLs are <u>not</u> closed under intersection
  - Meaning:
    - If L_1 and L_2 are CFLs then $L_1 \cap L_2$ is not necessarily a CFL.

Closure Properties

- CFLs are <u>not</u> closed under intersection
  - Example:
    - $L_1 = \{a^i b^j c^k \mid i < j \}$
    - $L_2 = \{a^i b^j c^k \mid i < k \}$
    - Are both CFLs

Closure Properties

- CFLs are <u>not</u> closed under intersection

$L_1 = \{a^i b^j c^k \mid i < j \}$	$L_2 = \{a^i b^j c^k \mid i < k \}$
$S \rightarrow ABC$	$S \rightarrow AC$
$A \rightarrow aAb \mid \Lambda$	$A \rightarrow aAc \mid B$
$B \rightarrow bB \mid b$	$B \rightarrow bB \mid \Lambda$
$C \rightarrow cC \mid \Lambda$	$C \rightarrow cC \mid c$

Closure Properties

- CFLs are not closed under intersection
  - $L_1 \cap L_2 = \{a^i b^j c^k \mid i < j \text{ and } i < k\}$
  - Which we just showed to be non-context free.

Closure Properties

- Sorry, Charlie
  - CFLs are not closed under complement
  - Why?
    - $L_1 \cap L_2 = (L_1' \cup L_2')'$

Closure Properties

- Sorry, Charlie
  - CFLs are not closed under difference
  - Why?
    - $L' = \Sigma^* - L$
    - We know Σ^* is regular, and as such is also a CFL.
    - If CFLs were closed under difference, then $\Sigma^* - L = L'$ would always be a CFL
    - But we showed that CFLs are not closed under complement

Closure Properties

- What went wrong?
  - Can't we apply the same construction as we did for the complement of RLs?
    - Reverse the accepting / non-accepting states
    - PDAs can "crash".
      - I.e Fail by having no place to go.
      - PDAs can "crash" in accepting or non-accepting state
      - Making non-accepting states accepting will not handle crashes.

Closure Properties

- What went wrong?
  - Can't we apply the same construction as we did for the intersection of RLs?
    - The states of M are an ordered pair (p, q) where $p \in Q_1$ and $q \in Q_2$
    - Informally, the states of M will represent the current states of M_1 and M_2 at each simultaneous move of the machines.

Closure Properties

- What went wrong?
  - Can't we apply the same construction as we did for the intersection of RLs?
    - The problem is the stack.
    - Although we could try the same thing for PDAs and have a combined machine keep track of where both PDAs are at any one time.
    - We can't keep track of what's on both stacks at any given tine.

Closure Properties

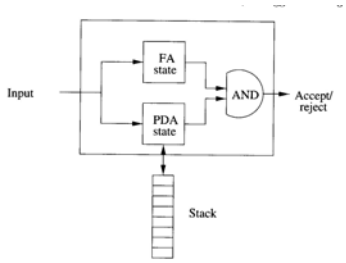
- However, if one of the CFLs does not use the stack (I.e. it is an FA), then we can build a PDA that accepts $L_1 \cap L_2$.
- In other words:
  - If L_1 is a context free language and L_2 is a regular language, then $L_1 \cap L_2$ is context free.

Closure Properties

- Basic idea:
  - Like with the FA construction, let the states of the new machine keep track of the states of the PDA accepting L_1 (M_1) and the FA accepting L_2 (M_2).
  - Our single stack of the new machine will operate the same as the stack of the PDA accepting L_1
  - Accepting states will be all states that contain both an accepting state from M_1 and M_2.

Closure Properties

- Basic idea



Closure Properties

- Formally, let
  - $M_1 = (Q_1, \Sigma, \Gamma, q_1, Z_0, A_1, \delta_1)$ be a PDA that accepts L_1
  - $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$ be an FA that accepts L_2
  - Build $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ as follows:
    - $Q = Q_1 \times Q_2$
    - $q_0 = (q_1, q_2)$
    - $A = A_1 \times A_2$

Closure Properties

- Transition function
  - For $p \in Q_1$, $q \in Q_2$, $Z \in \Gamma$, $a \in \Sigma$
    - $\delta((p,q), a, Z) = \{ ((p', q'), \alpha) \mid (p', \alpha) \in \delta_1(p, a, Z)$ and $\delta_2(q, a) = q' \}$

$\delta((p,q), \Lambda, Z) = \{ ((p', q), \alpha) \mid (p', \alpha) \in \delta_1(p, \Lambda, Z) \}$

Closure Properties

- Summary
  - CFLs are closed under
    - Union, Concatenation, Kleene Star
  - CFLs are NOT closed under
    - Intersection, Difference, Complement
  - But
    - The intersection of a CFL with a RL is a CFL

Decision Properties

- Questions we can ask about context free languages and how we answer such questions.

Decision Properties

- Given regular languages, specified in any one of the four means, can we develop algorithms that answer the following questions:
  1. Is a given string in the language?
  2. Is the language empty?
  3. Is the language finite?

Decision Properties

- Membership
  - Unlike FAs, we can't just run the string through the machine and see where it goes since PDAs are non-deterministic.
    - Must consider all possible paths

Decision Properties

- Membership
  - Instead, start with your grammar in CNF.
    - The proof of the pumping lemma states that the longest derivation path of a string of size n will be 2n – 1.
    - Systematically generate all derivations with one step, then two steps, …, then 2n – 1 steps where the length of the string tested = n. If one of the derivations derive x, return true, else return false.

Decision Properties

- Emptiness
  - By the proof of the pumping lemma, if a grammar in CNF has p states, the longest string, not subject to the pumping lemma will have length $n = 2^{p+1}$.
    - Systematically generate all strings with lengths less than n.
    - Test each one using membership algorithm
    - If all fail and $\Lambda \notin L$, then L is empty
    - Else L is not empty.

Decision Properties

- Finiteness
  - Just as with RLs, a language is infinite if there is a string x with length between n and 2n
    - With RLs n = number of states in an FA
    - With CFLs $n = 2^{p+1}$ where p is the number of variables in the CFG
    - Systematically generate all strings with lengths between n and 2n
    - Run through membership algorithm
    - If one passes, L is infinite, if all fail, L is finite

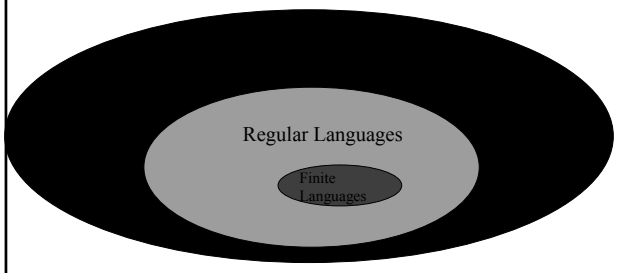
Decision Properties

- Questions?

Summary

- Pumping Lemma for CFLs
- Closure Properties
- Decision Properties

Now our picture looks like



Is there anything out here? YES

Next Time

- Next classes of languages
- However,
  - We start with the machine rather than the language
  - Move beyond simple language acceptance into the realm of computation.
- Enter…The Turing Machine!!!