Context Free Languages

PDAs and CFLs

Before We Start

- Any questions?

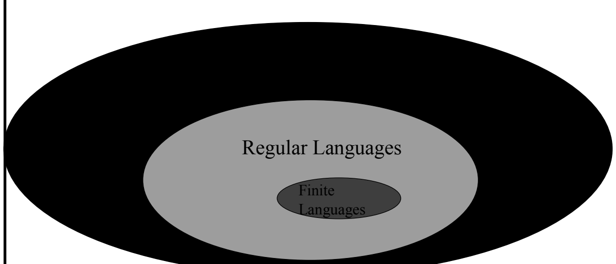
Languages

- Recall.
  - 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



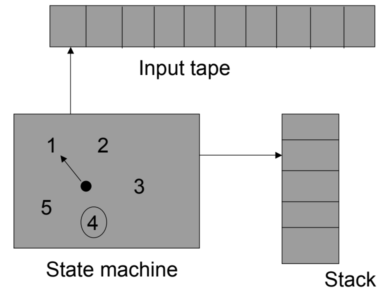
Context Free Grammars

- Let's formalize this a bit:
  - A context free grammar (CFG) is a 4-tuple: (V, T, S, P) where
    - V is a set of variables
    - T is a set of terminals
    - V and Σ are disjoint (I.e. $V \cap \Sigma = \emptyset$)
    - $S \in V$, is your start symbol

Context Free Grammars

- Let's formalize this a bit:
  - Production rules
    - Of the form $A \to \beta$ 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:
      - $A \to \beta$ 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
    - 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

- Show that PDAs and CFGs are equivalent.

- Questions?

Equivalence of CFG and PDA

1. Given a CFG, G, construct a PDA M, such that L(M) = L(G)
2. Given a PDA, M, define a CGF, G such that L(G) = L(M)

Step 1: CFG → PDA

- Given: A context free grammar G
- Construct: A pushdown automata M
- Such that:
  - Language generated by G is the same as
  - Language accepted by M.

Step 1: CFG → PDA

- Basic idea
  - Use the stack of the PDA to simulate the derivation of a string in the grammar.
    - Push S (start variable of G) on the stack
    - From this point on, there are two moves the PDA can make:
      1. If a variable A is on the top of the stack, pop it and push the right-hand side of a production $A \rightarrow \beta$ from G.
      2. If a terminal, a is on the top of the stack, pop it and match it with whatever symbol is being read from the tape.

Step 1: CFG → PDA

- Observations:
  - There can be many productions that have a given variable on the left hand side:
    - $S \rightarrow \varepsilon \mid 0S1 \mid 1S0$
  - In these cases, the PDA must "choose" which string to push onto the stack after pushing a variable.
    - I.e. the PDA being constructed in non-deterministic.

Step 1: CFG → PDA

- More observations:
  - A string will only be accepted if:
    - After a string is completely read
    - The stack is empty

Step 1: CFG → PDA

- Let's formalize this:
  - Let G = (V, T, S, P) be a context free grammar.
  - We define a pushdown automata
    - $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
  - Such that
    - $L(M) = L(G)$

Step 1: CFG → PDA

- Define M as follows:
  - $Q = \{ q_0, q_1, q_2 \}$
    - q_o will be the start state
    - q_1 will be where all the work gets done
    - q_2 will be the accepting state
  - $\Gamma = V \cup \Sigma \cup \{ Z_0 \}$ $\quad Z_0 \notin (V \cup \Sigma)$
  - $A = \{ q_2 \}$

Step 1: CFG → PDA

- Transition function δ is defined as follows:
  - $\delta(q_0, \varepsilon, Z_o) = \{ (q_1, SZ_o) \}$
    - To start things off, push S onto the stack and immediately go into state 1
  - $\delta(q_1, \varepsilon, A) = \{ (q_1, \alpha) \mid A \to \alpha$ is a production of G$\}$ for all variables A
    - Pop and replace variable.

Step 1: CFG → PDA

- Transition function δ is defined as follows:
  - $\delta(q_1, a, a) = \{ (q_1, \varepsilon) \}$ for all terminals a
    - Pop and match terminal.
  - $\delta(q_1, \varepsilon, Z_0) = \{ (q_2, Z_0) \}$
    - After all reading is done, accept only if stack is empty.
  - No other transitions exist for M

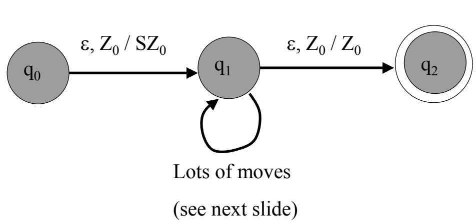
Step 1: CFG → PDA

- Let's look at an example:
  - Remember the CFG for odd length palindromes:
    - $S \to a \mid b$
    - $S \to a\,S\,a \mid b\,S\,b$
  - Let's convert this to a PDA.

Step 1: CFG → PDA

- Example:
  - $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
  - $Q = \{ q_0, q_1, q_2 \}$
  - $\Sigma = \{ a, b \}$
  - $\Gamma = \{ a, b, S, Z_0 \}$
  - $F = \{ q_2 \}$

Step 1: CFG → PDA



Step 1: CFG → PDA

State	Tape input	Stack	Move(s)
q_1	ε	S	(q_1, a) (q_1, b) (q_1, aSa) (q_1, bSb)
q_1	a	a	(q_1, ε)
q_1	b	b	(q_1, ε)

Step 1: CFG → PDA

- Let's run M on abbba
  - $(q_0, abbba, Z) \mapsto (q_1, abbba, SZ)$
  - $\mapsto (q_1, abbba, aSaZ)$ // push
  - $\mapsto (q_1, bbba, SaZ)$ // match
  - $\mapsto (q_1, bbba, bSbaZ)$ // push
  - $\mapsto (q_1, bba, SbaZ)$ // match
  - $\mapsto (q_1, bba, bbaZ)$ // push
  - $\mapsto (q_1, ba, baZ)$ // match
  - $\mapsto (q_1, a, aZ)$ // match
  - $\mapsto (q_1, \varepsilon, Z)$ // match
  - $\mapsto (q_2, \varepsilon, Z)$ // accept

Step 1: CFG → PDA

- Questions?

Step 2: PDA → CFG

- Given: A pushdown automata M
- Define: A context free grammar G
- Such that:
  - Language accepted by M is the same as
  - Language generated by G

Pushdown Automata

- Strings accepted by a PDA by <u>Final State</u>
  - Start at (q_0, x, Z_0)
    - Start state q_0
    - X on the input tape
    - Empty stack
  - End with (q, ε, β)
    - End in an accepting state $(q \in F)$
    - All characters of x have been read
    - Some string on the stack (doesn't matter what).

Pushdown Automata

- Strings accepted by a PDA by <u>Empty Stack</u>
  - Start at (q_0, x, Z_0)
    - Start state q_0
    - X on the input tape
    - Empty stack
  - End with $(q, \varepsilon, \varepsilon)$
    - End in <u>any</u> state
    - All characters of x have been read
    - Stack is empty

Final State vs. Empty Stack

- The two means by which a PDA can accept are equivalent wrt the class of languages accepted
  - Given a PDA M such that L = L(M), there exists a PDA M' such that L = N(M')
  - Given a PDA M such that L = N(M), there exists a PDA M' such that L = L(M')

Final State $\rightarrow$ Empty Stack

- Final State $\rightarrow$ Empty Stack
  - Given a PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$ and $L = L(P_F)$ then there exists a PDA P_N such that $L = N(P_N)$
  - We will build such a PDA

Accept by Empty Stack

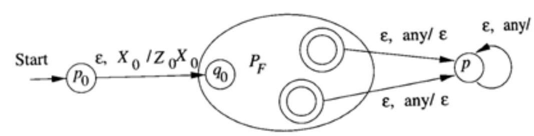
- Final State $\rightarrow$ Empty Stack
  - Basic idea
    - Transitions of P_N will mimic those of P_F
    - Create a new state in P_N that will empty the stack.
    - The machine can move into this new state whenever the machine is in an accepting state of P_F

Accept by Empty Stack

- Final State $\rightarrow$ Empty Stack
  - We must be careful though
    - P_F may crash when the stack is empty.
    - In those cases we need to assure that P_N does not accept
    - To solve this:
      - Create a new empty stack symbol X_0 which is placed on the stack before P_F s empty stack marker (Z_0)
      - Z_0 will only be popped by the new “stack emptying state
    - The first move of P_N will be to place Z_0X_0 on P_N stack.

Accept by Empty Stack

- Final State $\rightarrow$ Empty Stack

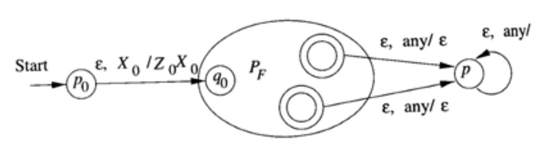


Empty Stack $\rightarrow$ Final State

- Must show:
  - A string x is accepted by P_N (by empty stack) iff it is accepted by P_F (by final state)
  - If x is accepted by P_N (empty stack) then it is accepted by P_F (final state)
  - If x accepted by P_F (final state) then it is accepted by P_N (empty stack)

Accept by Empty Stack

- Final State $\rightarrow$ Empty Stack



Final State vs. Empty Stack

- We showed: Final State $\rightarrow$ Empty Stack.
  - Given a PDA that accepts by final state, we can build a PDA that accepts by empty stack
- the inverse can be shown: Empty Stack $\rightarrow$ Final State
  - Given a PDA that accepts by empty stack, we can build a PDA that accepts by final state.
- Showing that PDAs that accept by empty stack and PDAs that accept by final state are equivalent.

Questions?

Step 2: PDA $\rightarrow$ CFG

- Given: A pushdown automata M that accepts by <u>empty stack</u>
- Define: A context free grammar G
- Such that:
  - Language accepted by M is the same as
  - Language generated by G

Step 2: PDA $\rightarrow$ CFG

- Basic idea
  - We define variables in G to be triplets:
    - [p, A, q] will represent a variable, that can generate all strings x that:
      - Upon reading x on the PDA tape will
      - Take you from state p to state q in the PDA and
      - Have a "net result" of popping A off the stack
    - In essence, A is "eventually" replaced by x
    - Note that it may take many moves to get there.

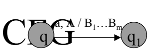
Step 2: PDA $\rightarrow$ CFG

- Productions of G
  1. For all states p in M, add the production
     - $S \rightarrow [q_0 Z_0 q]$
       - Following these productions will generate all strings that start at q_o, and result in an empty stack. Final state is not important.
       - In other words, all strings accepted by M.

Step 2: PDA $\rightarrow$ CFG

- More Productions of G
  2. For every $q, q_1 \in Q$, $a \in \Sigma \cup \{\varepsilon\}$ and $A \in \Gamma$
     - If $\delta(q, a, A)$ contains (q_1, ε) then add
       - $[qAq_1] \rightarrow a$
     - Meaning you can get from q to q_1 while popping A from the stack by reading an a.

Step 2: PDA $\rightarrow$ CFG



- Even More Productions of G
  3. For every $q, q_1 \in Q$, $a \in \Sigma \cup \{\varepsilon\}$ and $A \in \Gamma$
     - If $\delta(q, a, A)$ contains $(q_1, B_1 B_2 \ldots B_m)$ then
     - For every possible sequence of states $q_2, \ldots q_{m+1}$
     - Add
       - $[qAq_{m+1}] \rightarrow a[q_1 B_1 q_2]\,[q_2 B_2 q_3] \ldots [q_m B_m q_{m+1}]$
     - Meaning:
       - one way to pop A off the stack and to get from q to q_{m+1} is to
         - » read an a
         - » use some input to pop B_1 off the stack (bring you from q_1 to q_2 in the process),
         - » While in q_2, use some input to pop B_2 off the stack (bringing you to q_3 in the process)
         - » And so on…

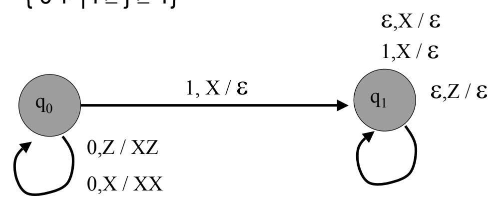
Step 2: PDA $\rightarrow$ CFG

- One can show by induction (though we won't) that
  - $[qAp] \Rightarrow^* x$ iff $(q, x, A) \mapsto^* (p, \varepsilon, \varepsilon)$
  - More specifically $[q_0Z_0p] \Rightarrow^* x$ and since we added the productions $S \rightarrow [q_0Z_0p]$ for all p, then $x \in L(G)$
  - On the flip side $S \rightarrow [q_0Z_0p]$ will always be the first production of any derivation of G
    - $(q_0, x, Z_0) \mapsto^* (p, \varepsilon, \varepsilon)$
    - So x is accepted by empty stack
    - $x \in L(M)$

Step 2: PDA $\rightarrow$ CFG

- Example

$L = \{ 0^i1^j \mid i \geq j \geq 1\}$



Step 2: PDA $\rightarrow$ CFG

- Example
  - $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
  - $Q = \{ q_0, q_1 \}$
  - $\Sigma = \{ 0, 1 \}$
  - $\Gamma = \{ X, Z \}$
  - $Z_0 = Z$
  - $F = \varnothing$

Step 2: PDA $\rightarrow$ CFG

- Corresponding CFG
  - Type 1 productions
  - $S \rightarrow [q_0Zq_1]$
  - $S \rightarrow [q_0Zq_0]$

Step 2: PDA $\rightarrow$ CFG

- Corresponding CFG
  - Type 2 productions
  - $[q_0Xq_1] \rightarrow 1$
  - $[q_1Xq_1] \rightarrow 1$
  - $[q_1Xq_1] \rightarrow \varepsilon$
  - $[q_1Zq_1] \rightarrow \varepsilon$

Step 2: PDA $\rightarrow$ CFG

- Corresponding CFG
  - Type 3 productions
  - Transitions to consider:
    - $\delta(q_0, 0, Z) = (q_0, XZ)$
    - $\delta(q_0, 0, X) = (q_0, XX)$

Step 2: PDA $\rightarrow$ CFG

- Corresponding CFG
  - Type 3 productions
    - $\delta(q_0, 0, X) = (q_0, XX)$
      - Look for all sequences of states $q_0 q_b q_c$
      - q_b and q_c can be either q_0 or q_1

q_b	q_c
q_0	q_0
q_1	q_0
q_0	q_1
q_1	q_1

Step 2: PDA $\rightarrow$ CFG

- Corresponding CFG
  - Type 3 productions
    - $\delta(q_0, 0, X) = (q_0, XX)$
  - Add productions
    - $[q_0Xq_0] \rightarrow 0[q_0Xq_0][q_0Xq_0]$
    - $[q_0Xq_0] \rightarrow 0[q_0Xq_1][q_1Xq_0]$
    - $[q_0Xq_1] \rightarrow 0[q_0Xq_0][q_0Xq_1]$
    - $[q_0Xq_1] \rightarrow 0[q_0Xq_1][q_1Xq_1]$

Step 2: PDA $\rightarrow$ CFG

- Corresponding CFG
  - Type 3 productions
    - $\delta(q_0, 0, Z) = (q_0, XZ)$
      - Look for all sequences of states $q_0 q_b q_c$
      - q_b , q_c can be either q_0 or q_1

q_b	q_c
q_0	q_0
q_1	q_0
q_0	q_1
q_1	q_1

Step 2: PDA $\rightarrow$ CFG

- Corresponding CFG
  - Type 3 productions
    - $\delta(q_0, 0, Z) = (q_0, XZ)$
  - Add productions
    - $[q_0Zq_0] \rightarrow 0[q_0Xq_0][q_0Zq_0]$
    - $[q_0Zq_0] \rightarrow 0[q_0Xq_1][q_1Zq_0]$
    - $[q_0Zq_1] \rightarrow 0[q_0Xq_0][q_0Zq_1]$
    - $[q_0Zq_1] \rightarrow 0[q_0Xq_1][q_1Zq_1]$

Step 2: PDA $\rightarrow$ CFG

- Complete grammar $G = (V, \Sigma, S, P)$
- $V = \{$
  - S, $[q_0Xq_0]$, $[q_0Zq_0]$,
  - $[q_0Xq_1]$, $[q_0Zq_1]$,
  - $[q_1Xq_0]$, $[q_1Zq_0]$,
  - $[q_1Xq_1]$, $[q_1Zq_1]$,
  - $[q_1Xq_1]$
  - $\}$

Step 2: PDA $\rightarrow$ CFG

- P =
  - $S \rightarrow [q_0Zq_1]$ (1) $[q_0Xq_0] \rightarrow 0[q_0Xq_0][q_0Xq_0]$ (7)
  - $S \rightarrow [q_0Zq_0]$ (2) $[q_0Xq_0] \rightarrow 0[q_0Xq_1][q_1Xq_0]$ (8)
  - $[q_0Xq_1] \rightarrow 1$ (3) $[q_0Xq_1] \rightarrow 0[q_0Xq_0][q_0Xq_1]$ (9)
  - $[q_1Xq_1] \rightarrow 1$ (4) $[q_0Xq_1] \rightarrow 0[q_0Xq_1][q_1Xq_1]$ (10)
  - $[q_1Xq_1] \rightarrow \varepsilon$ (5) $[q_0Zq_0] \rightarrow 0[q_0Xq_0][q_0Zq_0]$ (11)
  - $[q_1Zq_1] \rightarrow \varepsilon$ (6) $[q_0Zq_0] \rightarrow 0[q_0Xq_1][q_1Zq_0]$ (12)
  - $[q_0Zq_1] \rightarrow 0[q_0Xq_0][q_0Zq_1]$ (13)
  - $[q_0Zq_1] \rightarrow 0[q_0Xq_1][q_1Zq_1]$ (14)

Step 2: PDA → CFG

- Let's try a derivation for 00011
  - $S \rightarrow \underline{[q_0Zq_1]}$ // P1

    $\rightarrow 0\ \underline{[q_0Xq_1]}\ [q_1Zq_1]$ // P14

    $\rightarrow 00\ \underline{[q_0Xq_1]}\ [q_1Xq_1]\ [q_1Zq_1]$ // P10

    $\rightarrow 000\ \underline{[q_0Xq_1]}\ [q_1Xq_1]\ [q_1Xq_1]\ [q_1Zq_1]$ // P10

    $\rightarrow 0001\ \underline{[q_1Xq_1]}\ [q_1Xq_1]\ [q_1Zq_1]$ // P3

    $\rightarrow 00011\ \underline{[q_1X\,q_1]}\ [q_1Zq_1]$ // P4

    $\rightarrow 00011\ \varepsilon\ \underline{[q_1Zq_1]}$ // P5

    $\rightarrow 00011\ \varepsilon\ \varepsilon$ // P6

Summary

- What have we learned?
  - (We really don't need to see the CFG corresponding to a PDA, do we?) ☺

Summary

- What we have really learned?
  - Given a CFG, we can build a PDA that accepts the same language generated by the CFG
  - Given a PDA, we can define a CFG that can generate the language accepted by the PDA.
- Looking for a machine to accept CGLs?
  - The pushdown automata fits the bill!

Next time

- Closure Properties for CFLs
- Decision Algorithms for CFLs
- Just when you thought it was safe…
  - The Pumping Lemma for CFLs