Pushdown Automata

Homework

- Homework #3
  - Will return today
- Homework #4
  - due today
- Homework #5
  - Assigned today

Homework #5

- Homework 5 (due 10/18)
  - From textbook
    - Exercise 6.2.1 b,c / Exercise 6.2.2 b (use JFLAP)
    - Exercise 6.2.5
    - Exercise 6.3.2 (by empty stack or final state)
    - Exercise 6.3.4
    - Exercise 6.4.1a,b,c

Final Exam

- Finals schedule has been posted
  - Tuesday, November 15th
  - 2:45pm – 4:45pm
  - 70-1620

Plan

- Today
  - Pushdown Automata
- Thursday
  - CFG <-> PDA Equivalence

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

Plan for today

- Introduction to Pushdown Automata

Pushdown Automata

- A pushdown automata (PDA) is essentially:
  - An NFA-ε with a stack
  - A “move” of a PDA will depend upon
    - Current state of the machine
    - Current symbol being read in
    - Current symbol popped off the top of the stack
  - With each “move”, the machine can
    - Move into a new state
    - Push symbols on to the stack

Pushdown Automata

- The stack
  - The stack has its own alphabet
  - Included in this alphabet is a special symbol used to indicate an empty stack. (Z₀)
    - This special symbol should not be removed from the stack.
  - Note that the basic PDA is non-deterministic!

Pushdown Automata

- Let’s formalize this:
  - A pushdown automata (PDA) is a 7-tuple:
    - \( M = (Q, \Sigma, \Gamma, \delta, q₀, Z₀, F) \) where
      - \( Q \) = finite set of states
      - \( \Sigma \) = tape alphabet
      - \( \Gamma \) = stack alphabet (may have symbols in common w/ \( \Sigma \))
      - \( q₀ \in Q \) = start state
      - \( Z₀ \in \Gamma \) = initial stack symbol
      - \( F \subseteq Q \) = set of accepting states
      - \( \delta \) = transition function
Pushdown Automata

- About this transition function $\delta$:
  - During a move of a PDA:
    - At most one character is read from the input tape
    - $\varepsilon$ 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.

- Formally:
  - $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow (\text{finite subsets of } Q \times \Gamma^*)$
  - Domain:
    - $Q = \text{state}$
    - $(\Sigma \cup \{\varepsilon\}) = \text{symbol read off tape}$
    - $\Gamma = \text{symbol popped off stack}$
  - Range:
    - $Q = \text{new state}$
    - $\Gamma^* = \text{symbols pushed onto the stack}$

- Example:
  - $\delta(q, a, a) = (p, aa)$
  - Meaning:
    - When in state $q$,
    - Reading in an $a$ from the tape
    - With an $a$ popped off the stack
  - The machine will
    - Go into state $p$
    - Push the string "aa" onto the stack

- Configuration of a PDA
  - Gives the current “configuration” of the machine
  - $(p, x, \alpha)$ where
    - $p$ is the current state
    - $x$ is a string indicating what remains to be read on the tape
    - $\alpha$ is the current contents of the stack.

- Move of a PDA:
  - We can describe a single move of a PDA:
    - $(q, x, \alpha) \rightarrow (p, y, \beta)$
    - If:
      - $x = ay, \alpha = \gamma X, \beta = YX$
      - $\delta(q, x, y)$ includes $(p, Y)$ or
        - $\delta(q, \varepsilon, y)$ includes $(p, Y)$ and $x = y$.

- Moves of a PDA
  - We can write:
    - $(q, x, \alpha) \rightarrow^* (p, y, \beta)$
    - If
      - You can get from one configuration to the other by applying 0 or more moves.
Pushdown Automata

• Strings accepted by a PDA by Final State
  – Start at \((q_0, x, Z_0)\)
  • Start state \(q_0\)
  • \(X\) on the input tape
  • Empty stack
  – End with \((q, \varepsilon, \beta)\)
    • 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

• The language accepted by a PDA
  – Let \(M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)\) be a PDA
  – The language accepted by \(M\) by final state,
    • Denoted \(L(M)\) is
      • The set of all strings \(x\) that are accepted by \(M\) by final state

Pushdown Automata

• Let’s look at an example:
  – \(L = \{ xcx^r \mid x \in \{a,b\}^* \}\)
  – The PDA will have 4 states
    • State 0 (initial) : reading before the ‘c’
    • State 1: read the ‘c’
    • State 2: read after ‘c’, comparing chars
    • State 3: (accepting): move only after all chars read and stack empty
PDA Example

- Transition for abcba
  - \( (q_0, \text{abcb}, Z) \rightarrow (q_0, \text{bcba}, a) \) // push a
  - \( \rightarrow (q_0, \text{cba}, ba) \) // push b
  - \( \rightarrow (q_1, \text{ba}, ba) \) // goto 1
  - \( \rightarrow (q_2, \text{ba}, ba) \) // \( \varepsilon \) trans
  - \( \rightarrow (q_2, \text{a}, \text{a}) \) // pop b
  - \( \rightarrow (q_2, \varepsilon, Z) \) // pop a
  - \( \rightarrow (q_3, \varepsilon, Z) \) // Accept!

- Transition for abcb
  - \( (q_0, \text{abcb}, Z) \rightarrow (q_0, \text{bcb}, a) \) // push a
  - \( \rightarrow (q_0, \text{cb}, ba) \) // push b
  - \( \rightarrow (q_1, \text{b}, ba) \) // goto 1
  - \( \rightarrow (q_2, \text{b}, ba) \) // \( \varepsilon \) trans
  - \( \rightarrow (q_2, \varepsilon, \text{a}) \) // pop b
  - Nowhere to go // Reject!

Pushdown Automata

- I bet you’re wondering if JFLAP can handle PDAs!
  - Yes, it can…
  - Let’s take a look.

- Let’s look at another example:
  - \( L = \{ xx' | x \in \{ \text{a,b} \}^* \} \)
  - Basic idea for building a PDA
    - Much like last example, except
      - This time we don’t know when to start popping and comparing
      - Since PDAs are non-deterministic, this is not a problem

Pushdown Automata

- Let’s look at another example:
  - \( L = \{ xx' | x \in \{ \text{a,b} \}^* \} \)
  - The PDA will have 3 states
    - State 0 (initial): reading before the center of string
    - State 1: read after center of string, comparing chars
    - State 2 (accepting): after all chars read, stack should be empty
    - The machine can choose to go from state 0 to state 1 at any time:
      - Will result in many “wrong” set of moves
      - All you need is one “right” set of moves for a string to be accepted.
PDA Example

• Let’s see a bad transition set for abba
  – \((q_0, \text{abba}, Z) \mapsto (q_0, bba, a)\) \// push a
  – \(\mapsto (q_0, ba, ba)\) \// push b
  – \(\mapsto (q_0, a, bba)\) \// push b
  – \(\mapsto (q_1, a, bba)\) \// \(\varepsilon\) trans
  – Nowhere to go \// Reject!

PDA Example

• Let’s see a good transition set for abba
  – \((q_0, \text{abba}, Z) \mapsto (q_0, bba, a)\) \// push a
  – \(\mapsto (q_0, ba, a)\) \// push b
  – \(\mapsto (q_1, ba, ba)\) \// \(\varepsilon\) trans
  – \(\mapsto (q_1, a, a)\) \// pop b
  – \(\mapsto (q_1, \varepsilon, Z)\) \// pop a
  – \(\mapsto (q_2, \varepsilon, Z)\) \// Accept!

Pushdown Automata

• “Let’s go to the video tape”
  – Actually JFLAP…

Deterministic PDAs

• As mentioned before
  – Our basic PDA in non-deterministic
  – We can define a Deterministic PDA (DPDA) as follows:
    • Let \(M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)\) be a PDA
    • \(M\) is deterministic if:
      – \(\delta(q, a, X)\) has at most one element
      – \(\text{If } \delta(q, c, X) \neq \emptyset \text{ then } \delta(q, a, X) = \emptyset \text{ for all } a \in \Sigma\)

Deterministic PDAs

• In other words:
  – There is no configuration where the machine has a “choice” of moves
    • Each transition has at most 1 element.
    • If you can make a \(\varepsilon\)-transition from a state with a given symbol on the stack,
      – You cannot make that same transition on any tape input symbol.
Deterministic PDAs

• A language \( L \) is a deterministic context-free language (DCFL) if there is a DPA that accepts \( L \).

PDA Example

• Example:
  \[ L = \{ x \in \{ a, b \}^* \mid n_a(x) > n_b(x) \} \]

  – First using a PDA:
    • Let the stack store the “excess” of one symbol over another
      – If more \( a \)’s have been read than \( b \)’s, \( a \)’s will be on the stack, and
        via versa
      – If \( a \) is on the stack and you read a \( b \), simple match the \( a \) with the
        \( b \).
      – If \( a \) is on the stack and you read an \( a \), we have one more extra \( a \) –
        Push it on the stack.
      – An empty stack means the number of \( a \)’s and \( b \)’s are equal.

PDA Example

• Example:
  \[ L = \{ x \in \{ a, b \}^* \mid n_a(x) > n_b(x) \} \]

  – The PDA will have 2 states:
    • State 0 (start) : where all the work gets done
    • State 1 (accepting) : one you’re in here, the machine
      stops.
  – The machine can “choose” to go into state 1 on
    \( a \in \) transition whenever an \( a \) is on the stack.

PDA Example

• Example:
  \[ L = \{ x \in \{ a, b \}^* \mid n_a(x) > n_b(x) \} \]

  – Removing the non-determinism :
    • Let the stack store 1 minus the “excess” of one
        symbol over another
    • The state will determine whether you have excess
        \( a \)’s or excess \( b \)’s

PDA Example

• Example:
  \[ L = \{ x \in \{ a, b \}^* \mid n_a(x) > n_b(x) \} \]

  – The PDA will have 2 states:
    • State 0 (start) : when \( n_a(x) \leq n_b(x) \)
      – Equality or surplus of \( b \)’s
    • State 1 (accepting) : when \( n_a(x) > n_b(x) \)
      – Surplus of \( a \)’s
PDA Example

- Example:
  \[ L = \{ x \in \{ a, b \}^* \mid n_a(x) > n_b(x) \} \]

Now you might be wondering…

We know that all DCFLs are CFLs

It can be shown…

- That the language pal:
  \[ \text{pal} = \{ x \in \{ a, b \}^* \mid x = x^r \} \]

- Cannot be accepted by any DPDA.

It can also be shown

- That all regular languages can be accepted by a DPDA.
  - Since an DFA is essentially a DPDA that doesn’t make use of the stack.

Now our picture looks like

Why DPDAs are important

- A compiler may wish to implement a PDA in software to parse a program given by a given grammar
- DPDAs and ambiguity
  - If L can be accepted by a DPDA, then L can be expressed by an unambiguous CFG
  - Not visa versa
Determinism vs. Non-Determinism

• Comparing FAs and PDAs
  – DPDAs allow for $\varepsilon$-transitions
  – DPDAs allow for no moves
  – FAs and NFAs are equivalent
  – PDAs and DPDAs are not equivalent

Questions

Pushdown Automata

• Strings accepted by a PDA by Final State
  – Start at $(q_0, x, Z_0)$
  • Start state $q_0$
  • $X$ on the input tape
  • Empty stack
  – End with $(q, \varepsilon, \beta)$
    • End in an accepting state $q \in F$
    • All characters of $x$ have been read
    • Some string on the stack (doesn’t matter what).

• Strings accepted by a PDA by Final State
  – Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA
  – $x$ is accepted by $M$ if
    $(q_0, x, Z_0) \rightarrow^* (q, \varepsilon, \beta)$
    • Where
      – $q \in Q$
      – $\beta \in \Gamma^*$

• Strings accepted by a PDA by Empty Stack
  – 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 any state
    • All characters of $x$ have been read
    • Stack is empty

Pushdown Automata

• The language accepted by a PDA
  – Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ be a PDA
  – The language accepted by $M$ by final state,
    • Denoted $L(M)$ is
    • The set of all strings $x$ that are accepted by $M$ by final state
  – The language accepted by $M$ by empty stack,
    • Denoted $N(M)$ is
    • The set of all strings $x$ that are accepted by $M$ by empty stack
  • We will show that all languages accepted by a PDA by final state will be accepted by an equivalent PDA by empty stack and visa versa
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

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.

Final State \( \rightarrow \) Empty Stack

- \( P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0, F) \)

Accept by Empty Stack

- Final State \( \rightarrow \) Empty Stack
  - For example:
    - Start \( P_f \)
    - \( P_f \) \( \rightarrow \) \( P_f \)
    - \( P_f \) \( \rightarrow \) \( P_f \)
    - \( P_f \) \( \rightarrow \) \( P_f \)
    - \( P_f \) \( \rightarrow \) \( P_f \)
Empty Stack $\rightarrow$ Final State

- Empty Stack $\rightarrow$ Final State
  - Given a PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$ and $L = \mathbb{N}(P_N)$ then there exists a PDA $P_F$ such that $L = L(P_F)$

  - We will build such a PDA

---

Empty Stack $\rightarrow$ Final State

- Empty Stack $\rightarrow$ Final State
  - Basic idea
    - Transitions of $P_F$ will mimic those of $P_N$
    - Create a new state in $P_F$ that will serve as the final state.
    - The machine can move into this new state whenever $P_N$ empties its stack.

---

Empty Stack $\rightarrow$ Final State

- Empty Stack $\rightarrow$ Final State
  - $P_F = (\cdot Q \cup \{p_o, p_f\}, \cdot \Sigma, \cdot \Gamma \cup \{X_0\}, \cdot \delta_F, \cdot p_0, \cdot X_0, \cdot p_f)$

---

Accept by Empty Stack

- Empty Stack $\rightarrow$ Final State

---

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.

- We showed: 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?

---

Next time

- PDAs…the perfect machine for CFLs…