Class 6: Pushdown Automata

Spring 2010
University of Virginia
David Evans

Revisiting the Pumping Lemma

If input string is longer than $|Q|$, some state must repeat.

If $A$ is a regular language, then there is some number $p$ (the pumping length) where for any string $s \in A$ and $|s| \geq p$, $s$ may be divided into three pieces, $s = xyz$, such that $|y| > 0$, $|xy| \leq p$, and for any $i \geq 0$, $xy^iz \in A$.

Pumping Game for Language $A$

1. Player 1: picks $p$
2. Player 2: picks $s \notin A$ and $|s| \geq p$
3. Player 1: picks $x, y, z$, $|y| > 0$, $|xy| \leq p$, $s = xyz$
4. Player 2: picks $i \geq 0$

Player 1 wins if $xy^iz \notin A$, Player 2 wins if $xy^iz \in A$

If Player 1 can always win: $A$ is regular
If Player 2 can always win: $A$ is not regular

Pump-Priming Game for Language $A$

1. Player 1: picks $p$
2. Player 2: picks $s \not\in A$ and $|s| \geq p$
3. Player 1: picks $x, y, z$, $|y| > 0$, $|xy| \leq p$, $s = xyz$
4. Player 2: picks $i \geq 0$

Player 1 wins if $xy^iz \not\in A$, Player 2 wins if $xy^iz \in A$

If Player 1 can always win: $A$ is regular
If Player 2 can always win: $A$ is not regular

$A = \{ w \mid w$ has more 0s than 1s $\}$

Menu

- Revisiting and Reversing the Pumping Lemma
- Recognizing Non-Regular Languages
- Pushdown Automata
A Complementary View

• Regular Languages are closed under complement: if $A$ is regular, then $\overline{A}$ is regular

Proof sketch:
DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizes $A$.
DFA $\overline{M} = (Q, \Sigma, \delta, q_0, Q - F)$ recognizes $\overline{A}$.

• Thus, Player 2 wins by showing either $A$ or $\overline{A}$ is non-regular.

DFA + Counter?

Accept: count = 0

DFA + Counter?

Accept if count == 0.

DFA + Stack = Deterministic Pushdown Automaton

Accept if stack is empty.

DFA + Stack = Deterministic Pushdown Automaton

Accept if stack is empty.
Formalizing DPDA

\[ (Q, \Sigma, \Gamma, \delta, q_0, F) \]

- \( Q \) — finite set of states
- \( \Sigma \) — input alphabet
- \( \Gamma \) — stack alphabet
- \( \delta : Q \times \Sigma \times \Gamma \rightarrow (Q \times \Gamma) \cup \{\emptyset\} \) — transition function
- \( q_0 \in Q \) — start state
- \( F \subseteq Q \) — accepting states

Note: Sipser only defines Nondeterministic PDA and calls it PDA.

DPDA Transitions

\[ \delta : Q \times \Sigma \times \epsilon \rightarrow Q \times \Gamma \]

\[ \Sigma_\epsilon = \Sigma \cup \{\epsilon\} \quad \Gamma_\epsilon = \Gamma \cup \{\epsilon\} \]

Inputs: state, alphabet symbol or \( \epsilon \), popped stack symbol or \( \epsilon \)
Outputs: state, stack symbol to push or \( \epsilon \)

It is a deterministic machine: must always follow possible \( \epsilon \)-input transition.

Deterministic Pushdown Automaton

\[ \Sigma = \{a, b\} \quad \Gamma = \{\$, +\} \quad A = a^n b^n \]

This looks like nondeterminism, but it is not: must always take possible \( \epsilon \)-transitions a state with an \((\epsilon, \delta)\) transition cannot have any other \((a, \delta)\) transitions; a state with an \((a, \epsilon)\) transition cannot have any other \((a, \delta)\) transitions.

Computing Model for DPDA

First, we need to model the stack!

Modeling the Stack

\[ T = \{\ldots, \} \]

\[ \text{Stack alphabet:} \]

\[ \text{push}(\gamma, h) \rightarrow h_1 \gamma \]

\[ \text{pop}(h_1, h) \rightarrow \gamma \]
Modeling the Stack

**push:** $\Gamma^* \times \Gamma_e \rightarrow \Gamma^*$

$\text{push}(s, \epsilon) = s$

$\text{push}(s, h_p \in \Gamma) = h_p s$

**pop:** $\Gamma^* \rightarrow \Gamma^*$

$\text{pop}(h_s s) = h_t \times s$

$\text{pop}(\epsilon) = \text{undefined (error)}$

**pop:** $\Gamma^* \times \Gamma_e \rightarrow \Gamma^*$

$\text{pop}(s, \epsilon) = s$

$\text{pop}(h_s, h_t) = s$

$\text{pop}(h_s s, h_t \neq h_t) = \text{undefined (error)}$

This would be a normal way to define a stack, but not what we use!

DPDA Computation Model

$\delta : Q \times \Sigma \times \Gamma_e \rightarrow Q \times \Gamma_e$

$\delta^* : Q \times \Sigma^* \times \Gamma_e \rightarrow Q \times \Gamma_e$

$\forall q \in Q, \forall a \in \Sigma, x \in \Sigma^*, \gamma \in \Gamma^* :$

$\delta^*(q, \epsilon, \gamma) = (q, \gamma)$

Dealing with $\epsilon$-transitions

Remember the NFA $\subseteq$ DFA proof?

**Proof by Construction.** Given $N = (Q, \Sigma, \delta, q_0, F)$, an NFA recognizing some language $L$, we construct a DFA $N' = (Q', \Sigma, \delta', q'_0, F')$ that recognizes the same language:

- $Q' = \mathcal{P}(Q)$
- $\delta' : Q' \times \Sigma \rightarrow Q'$ is defined to capture all possible states resulting from $\delta$ transitioning from the input state:
  - $\delta'(r, a) = \bigcup_{r \in R} E(r, a)$
  - $q'_0 = E(q_0)$
- $F' = \{ q' \in Q' \land q \notin F' \}$

where $E : Q' \rightarrow Q$: is the epsilon-transition function defined by:

$E(q) = q \cup \bigcup_{r \in \delta(q, \epsilon)} E(r)$

DPDA Computing Model

$\delta^*(q, \epsilon, \gamma) = E(q, \gamma)$

$\delta(q, a, h_t) \rightarrow (q_t, h_p) \Rightarrow \delta^*(q, ax, h_t \gamma) = \delta^*(q_t, x, h_p \gamma)$

where $(q_r, \gamma_r) = E(q_t, h_p \gamma)$

Where $E$ is the forced-follow $\epsilon$-transitions function defined by:

$E : Q \times \Gamma^* \rightarrow Q \times \Gamma^*$

$\delta(q, \epsilon, \gamma) = \emptyset : E(q, \gamma) = (q, \gamma)$

$\delta(q, \epsilon, h_t \gamma) = (q_t, h_p \gamma) :$

$E(q, h_t \gamma) = E(q_t, h_p \gamma)$

Acceptance: PDA accepts $w$ when:

**Accepting State Model**

$\delta^*(q_0, w, \epsilon) \rightarrow (q_f, s) \land q_f \in F$

**Empty Stack Model**

$\delta^*(q_0, w, \epsilon) \rightarrow (q, \epsilon)$

Is the set of languages accepted by DPDA with each model the same?

A good, original proof is worth a challenge bonus.

(Finding a published proof is not.)
Power of DPDAs

\[ L(DPDA) \supseteq L(DFA) \]?

1. Prove there is some DPDA that recognizes every regular language.
   Construct DPDA from DFA by adding empty stack transitions:
   \[ \delta_{DPDA}(q, a, \epsilon) = (\delta_{DFA}(q, a), \epsilon) \]

2. Prove there is some language that can be recognized by a DPDA that cannot be recognized by any PDA.
   \[ A = \{ a^n b^n | n \geq 0 \} \]

Charge

- **Thursday:**
  - Non-equivalence of NPDA and DPDA
  - Context-Free Grammars

- **Next Tuesday:**
  - PS2 Due
  - Languages that cannot be recognized by NPDA