CSE 2001: Introduction to Theory of Computation
Summer 2013

Week 7: CFL and Pushdown Automata

Yves Lespérance

Course page: http://www.cse.yorku.ca/course/2001

Slides are mostly taken from Suprakash Datta’s for Winter 2013

Next

• Chapter 2:
  • Pushdown Automata
More examples of CFLs

- \( L(G) = \{0^n1^{2n} \mid n = 1,2,\ldots \} \)
- \( L(G) = \{xx^R \mid x \text{ is a string over } \{a,b\} \} \)
- \( L(G) = \{x \mid x \text{ is a string over } \{1,0\} \text{ with an equal number of } 1 \text{'s and } 0 \text{'s} \} \)

Next: Pushdown automata (PDA)

Add a stack to a Finite Automaton

- Can serve as type of memory or counter
- More powerful than Finite Automata
- Accepts Context-Free Languages (CFLs)
- Unlike FAs, nondeterminism makes a difference for PDAs. We will only study non-deterministic PDAs and omit Sec 2.4 (3rd Ed) on DPDAs.
Pushdown Automata

*Pushdown automata* are for context-free languages what finite automata are for regular languages.

PDAs are *recognizing automata* that have a single stack (= memory):

**Last-In First-Out pushing and popping**

Non-deterministic PDAs can make non-deterministic choices (like NFA) to find accepting paths of computation.

---

**Informal Description PDA (1)**

input $w = 00100100111100101$

The PDA $M$ reads $w$ and stack element. Depending on

- input $w_i \in \Sigma^*$,
- stack $s_j \in \Gamma^*$, and
- state $q_k \in Q$

the PDA $M$:

- jumps to a new state,
- pushes an element $\Gamma^*$

(nondeterministically)
Informal Description PDA (2)

input $w = 00100100111100101$

After the PDA has read complete input, M will be in state $\in Q$

If possible to end in accepting state $\in F \subseteq Q$, then M accepts w

Formal Description of a PDA

A Pushdown Automata M is defined by a six tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, with

- $Q$ finite set of states
- $\Sigma$ finite input alphabet
- $\Gamma$ finite stack alphabet
- $q_0$ start state $\in Q$
- $F$ set of accepting states $\subseteq Q$
- $\delta$ transition function

$$\delta: Q \times \Sigma \times \varepsilon \rightarrow \mathcal{P}(Q \times \Gamma)$$
Example 2.9:
The PDA first pushes "$ 0^n $" on stack. Then, while reading the $1^n$ string, the zeros are popped again.
If, in the end, $ $ is left on stack, then “accept”

Machine Diagram for $0^n1^n$

On $w = 000111$ (state; stack) evolution:

- $q_1$: $\epsilon \rightarrow (q_2; \epsilon) \rightarrow (q_2; 0) \rightarrow (q_2; 00) \rightarrow (q_3; 00) \rightarrow (q_3; \epsilon)$
- $q_4$: $\epsilon \rightarrow (q_2; \epsilon) \rightarrow (q_2; 0) \rightarrow (q_2; 00) \rightarrow (q_3; 00) \rightarrow (q_3; \epsilon)$

This final $q_4$ is an accepting state
On $w = 0101$ (state; stack) evolution:
$(q_1; \varepsilon) \rightarrow (q_2; \$) \rightarrow (q_2; 0\$) \rightarrow (q_3; \$) \rightarrow (q_4; \varepsilon)$ …
But we still have part of input “01”.
There is no accepting path.

An important example

- $L = \{a^ib^ia^k | i=j \text{ or } i=k \}$
  (Example 2.16, p 115. 3rd ed)

- Try $L = \{ww^R | w \text{ is any binary string} \}$
PDAs and CFL

Theorem 2.20 (2.12 in 2nd Ed):
A language L is context-free if and only if there is a pushdown automata M that recognizes L.

Two step proof:
1) Given a CFG G, construct a PDA $M_G$
2) Given a PDA M, make a CFG $G_M$

Converting a CFL to a PDA

- Lemma 2.21 in 3rd Ed
- The PDA should simulate the derivation of a word in the CFG and accept if there is a derivation.
- Need to store intermediate strings of terminals and variables. How?
Idea

• Store only a suffix of the string of terminals and variables derived at the moment starting with the first variable.
• The prefix of terminals up to but not including the first variable is checked against the input.
• A 3 state PDA is enough.

Converting a PDA to a CFG

• Lemma 2.27 in 3rd Ed
• Design a grammar equivalent to a PDA
• Idea: For each pair of states \(p, q\) we have a variable \(A_{pq}\) that generates all strings that take the automaton from \(p\) to \(q\) (empty stack to empty stack).
Some details

Assume

- Single accept state
- Stack emptied before accepting
- Each transition either pops or pushes a symbol

• Can create rules for all the possible cases (p 122 in 3rd Ed)