CS 4510: Automata and Complexity **Fall 2019**

Lecture 14: Context Free Grammars

October 16, 2019

Lecturer: Santosh Vempala *Scribe: Aditi Laddha*

14.1 Review

Context-free grammar: Context-free grammar(CFG), we define the following:

V: set of variables,
$S \in V$: start symbol,
Σ: set of terminals, and
R: production rules, $V \to (V \cup \Sigma)^*$

14.2 Pushdown Automaton

Pushdown Automaton: A pushdown automaton(PDA) is defined as

Q: the set of states,
$q_0 \in Q$: the start state,
$F \subseteq Q$: the set of accept states,
Σ: Input alphabet,
Γ: Stack alphabet, and
δ: the transition function for PDA

where $\delta : Q \times (\Sigma \cup \epsilon) \times (\Gamma \cup \epsilon) \to 2^{Q \times (\Gamma \cup \epsilon)}$. Note that the transition function is non-deterministic and defines the following stack operations:

- Push: $(q, a, \epsilon) \to (q', b')$
  In state q on reading input letter a, we go to state q' and push b' on top of the stack.
- Pop: $(q, a, b) \to (q', \epsilon)$
  In state q on reading input letter a and top symbol of the stack b, we go to state q' and pop b from top of the stack.

Example 1: $L = \{0^i 1^i : i \in \mathbb{N}\}^*$

- CFG G_1:

$$S \to \epsilon$$
$$S \to SS_1$$
$$S_1 \to \epsilon$$
$$S_1 \to 0S_1 1$$

- PDA:

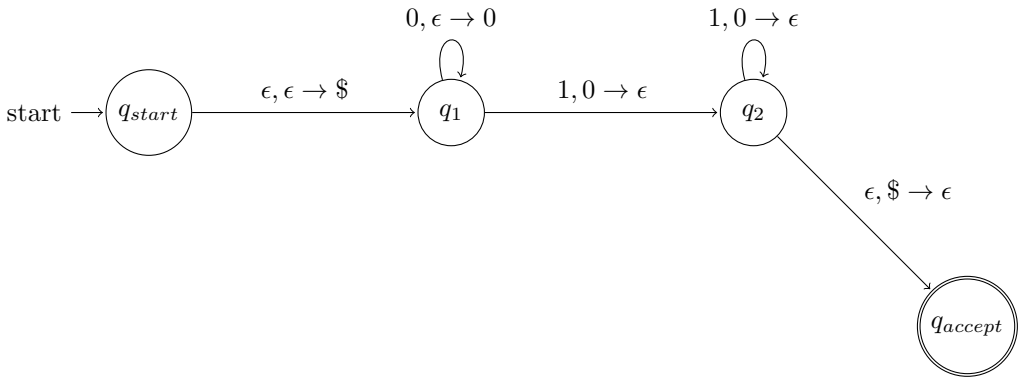

Figure 14.1: PDA for L

Example 2: The set of balanced parenthesis

- CFG G_2:

$$S \to \epsilon$$
$$S \to SS$$
$$S \to (S)$$

- PDA:

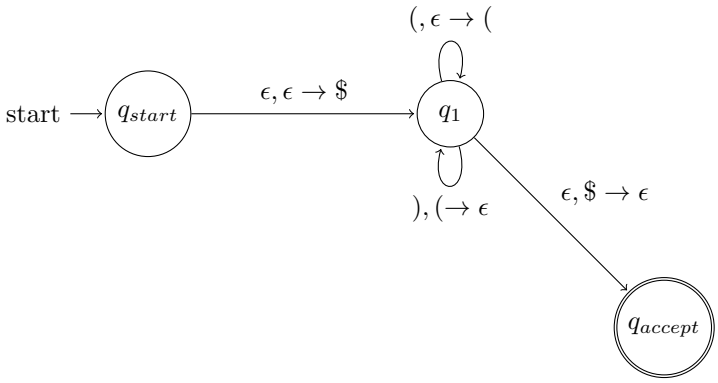

Figure 14.2: PDA for balanced parenthesis

How can we generate (()(())) using these production rules? There can be multiple ways to generate the same terminal string from S. One way might be to replace the leftmost symbol in an expression at every

step. This gives the following derivation

S	
(S)	(rule 3)
(SS)	(rule 2)
$((S)S)$	(rule 3)
$(()S)$	(rule 1)
$(()(S))$	(rule 3)
$(()((S)))$	(rule 3)
$(()(()))$	(rule 1)

The above examples suggest a relation between CFGs and PDAs and motivate the following question.
Question: For any context free grammar G, does there exist a PDA that accepts exactly the set of strings generated by G? For any pushdown automaton P, does there exist a context free grammar which generates exactly the set of strings accepted by P?
The answer to both questions is yes and we will see one side of the proof in the following section.

14.3 Equivalence between CFG and PDA

Theorem 14.1 *A language is context free if and only if some pushdown automaton recognizes it.*

Given a context-free grammar G, following is an algorithm to create a pushdown automaton P that accepts the language generated by G.

Algorithm 1: Converting CFG to PDA

1. Push \$ symbol on stack
2. Push start variable, S on stack
3. If the top of the stack is a terminal, then match it with input letter
4. If the top of the is a variable, A then non-deterministically select a rule for A, say $A \to \alpha_1\alpha_2 \dots \alpha_k$ where $\alpha_i \in V \cup \Sigma$, and replace A with the right hand side of the rule
5. If the top of the stack is \$, enter accept state

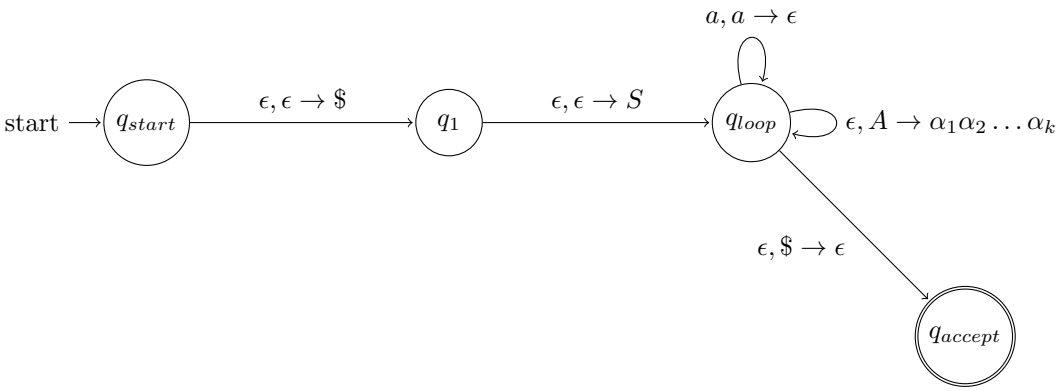

Figure 14.3: State diagram for P

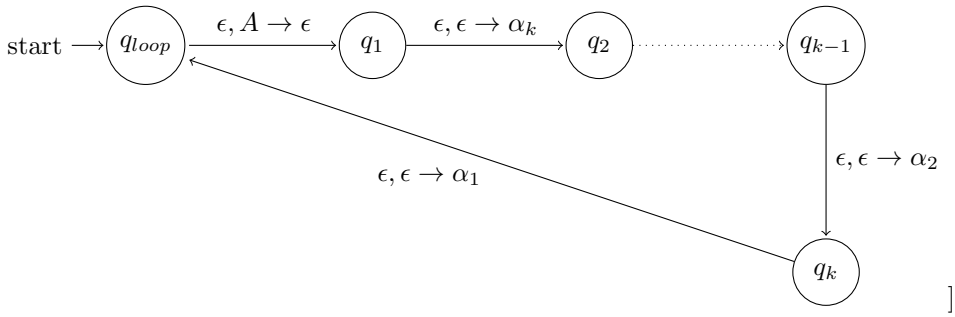

Figure 14.4: Implementation of $A \to \alpha_1\alpha_2 \ldots \alpha_k$

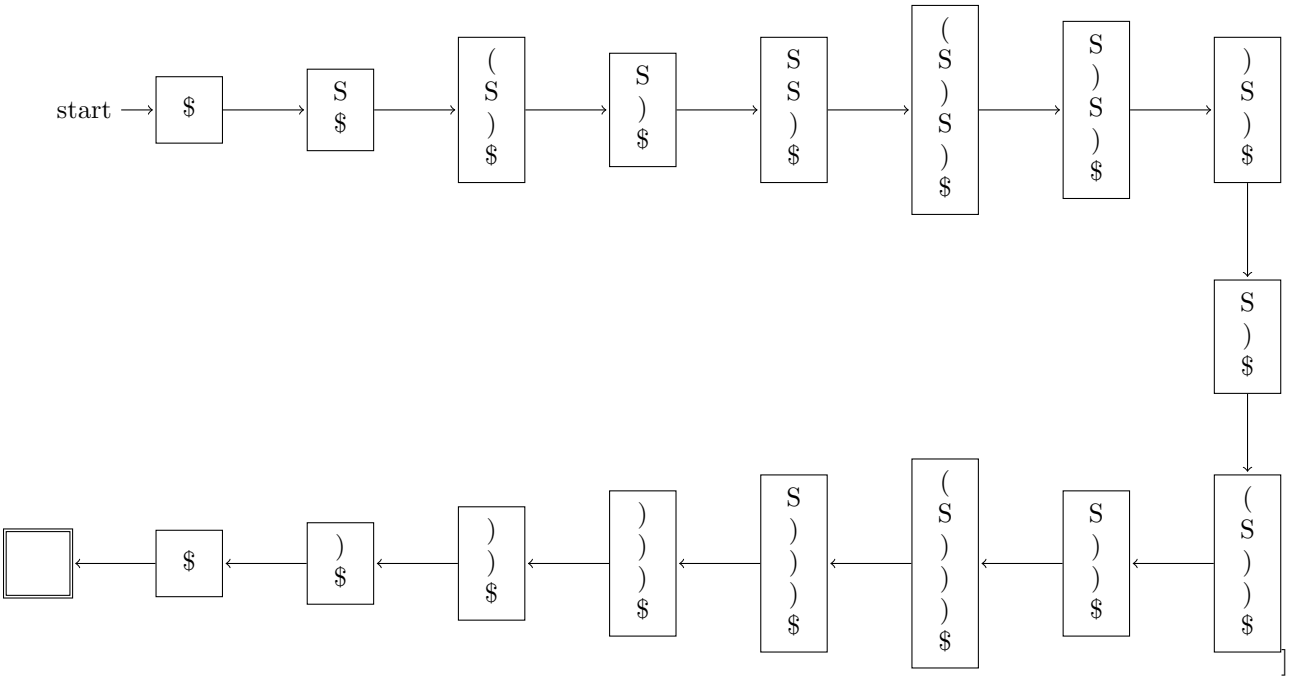

Figure 14.5: Stack trace of one branch of P corresponding to G_2 on input (()(()))

For the other side of the proof, see the reference.

14.4 Another example

Show that the language $\{x \in \{0,1\}^* : x \text{ is a palindrome}\}$ is context-free. For this problem, the CFG is easy:

$$S \to \epsilon$$
$$S \to 0$$
$$S \to 1$$
$$S \to 0S0$$
$$S \to 1S1$$

What about the PDA? If we knew the "middle" of the input, we could simply push the first "half", then start matching the input with the top of the stack and keep popping till we see the $ symbol. If the input agrees with the stack, and we reach the end of the input and the bottom of the stack at the same time, the PDA accepts. But how to know the middle? This is where we can use the inherent non-determinism of a PDA. At every letter of the input, the PDA can either continue to push it to the stack or it can transition to a new state where from now on it only compares the input letter to the top of the stack and pops the top if they match and repeat. So, if the input is a palindrome then there is at least one sequence of transitions where the PDA reaches accept state at the end of the input. Recall that the PDA accepts as long as there is *some* sequence of valid transitions that lead it to accept.

14.5 Reference

- Ch 2.2 Pushdown Automata, "Introduction to the Theory of Computation"