Lec 11. Pushdown Automata and CFG

Eunjung Kim

PDA FOR $L = \{w \cdot w^R : w \in \{0,1\}^*\}$

PDA FOR $L = \{w \in 0^n 1^m : n \geq m\}$

PDA FOR $L = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$

PDA FOR $L = \{w \in a^i b^j c^k : i = j$ OR $i = k\}$

PDA, seemingly more powerful

A slightly general form of PDA which pops a string in Γ^* and pushes Γ^* can be converted into a usual one.

PDA WITH SPECIFIC CONDITIONS

A given PDA can be transformed to satisfy any combination of the following conditions.

1. It has a single accept state q_{accept}.

2. It empties its stack before accepting.

3. Each transition move either pushes a symbol onto the stack (push move) or pops a symbol off the stack (pop move), but does not do both at the same time.

Equivalence of CFG and PDA

Theorem

A language is context-free if and only if some pushdown automaton recognizes it.

- ($\Rightarrow$): converting a CFG to an equivalent PDA.
- ($\Leftarrow$): converting a PDA to an equivalent CFG.

$\Rightarrow$: Converting CFG to PDA

- From a context-free grammar $G = (V, \Sigma, R, S)$, we aim to construct a PDA P such that $L(P) = L(G)$.

$\Rightarrow$: Converting CFG to PDA

- From a context-free grammar $G = (V, \Sigma, R, S)$, we aim to construct a PDA P such that $L(P) = L(G)$.

- Key idea: we design PDA P which simulates a leftmost derivation of w.

$\Rightarrow$: Converting CFG to PDA

- From a context-free grammar $G = (V, \Sigma, R, S)$, we aim to construct a PDA P such that $L(P) = L(G)$.

- Key idea: we design PDA P which simulates a leftmost derivation of w.
  1. "matching" the input symbol and the stack symbol if the stack symbol is an element of Σ.
  2. "replacing" the stack symbol A by z if A is a variable of G and there is a rule $A \rightarrow z$.
  3. while maintaining, in the stack, the suffix of a string $w \in (\Sigma \cup V)^*$ s.t. $S \Rightarrow^*_{lm} w$ starting with the leftmost variable in w.

$\Rightarrow$: CONVERTING CFG TO PDA

$L = \{0^n 1^n : n \geq 0\}$ is the language of the grammar $S \to 0S1 \quad | \quad \epsilon$.

$\Rightarrow$: Converting CFG to PDA

Construct a PDA P as follows.

1. There are three states q_{start}, q, q_{accept}.
2. The stack alphabet is $V \cup \Sigma \cup \{\$\}$.
3. Initially, P places the marker \$ onto the (empty) stack, then the start symbol S of CFG G.
4. It loops at the state q and executes the following unless the stack symbol is \$
   - If the stack symbol is $A \in V$, then P nondeterministically chooses a rule of the form $A \to \gamma$ and pushes γ onto stack so that the first symbol of γ is at the top.
   - If the stack symbol is $a \in \Sigma$, then P reads the symbol $a \in \Sigma$ in the input, pop a, and stays in the current state. If a cannot be read ("does not match"), no move is defined and the current computation branch dies out.
5. If the stack symbol is \$, then it goes to q_{accept}. The input string is accepted if the string has been read fully. If not, the current branch dies out.

$\Rightarrow$: Converting CFG to PDA

Construct a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ from $G = (V, \Sigma, R, S)$:

1. $Q = \{q_0, q, q_{accept}\}$. $\Gamma = V \cup \Sigma \cup \{\$\}$.
2. $\delta(q_0, \epsilon, \epsilon) = \{(q, S\$)\}$.
3. For each stack symbol in $V \cup \Sigma \cup \{\$\}$
   - for every $A \in V$: $\delta(q, \epsilon, A) = \{(q, \gamma) : \text{ for all rules } A \to \gamma \text{ in } G\}$
   - for every $a \in \Sigma$: $\delta(q, a, a) = \{(q, \epsilon)\}$
   - $\delta(q, \epsilon, \$) = \{(q_{accept}, \epsilon)\}$.

$\Rightarrow$: Converting CFG to PDA

Construct a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ from $G = (V, \Sigma, R, S)$:

1. $Q = \{q_0, q, q_{accept}\}$. $\Gamma = V \cup \Sigma \cup \{\$\}$.
2. $\delta(q_0, \epsilon, \epsilon) = \{(q, S\$)\}$.
3. For each stack symbol in $V \cup \Sigma \cup \{\$\}$
   - for every $A \in V$: $\delta(q, \epsilon, A) = \{(q, \gamma) : \text{ for all rules } A \to \gamma \text{ in } G\}$
   - for every $a \in \Sigma$: $\delta(q, a, a) = \{(q, \epsilon)\}$
   - $\delta(q, \epsilon, \$) = \{(q_{accept}, \epsilon)\}$.

How to implement a transition such as $\{(q, \gamma) \in \delta(q, \epsilon, A)$ when γ is a string, not necessarily a symbol in Γ_ϵ?

$\Leftarrow$: Converting PDA to CFG

Step A. Streamlining the PDA.

1. It has a single accept state q_{accept}.
2. It empties its stack before accepting.
3. Each transition move either pushes a symbol onto the stack (push move) or pops a symbol off the stack (pop move), but does not do both at the same time.

$\Leftarrow$: Converting PDA to CFG

Step B. Variables A_{pq} for all $p, q \in Q$.

1. Meaning of A_{pq}: we intend to design CFG G so that

$$L(A_{pq}) := \{w : A \Rightarrow_G^* w\}$$

   coincides with

$$\{w : (p, w, \epsilon) \vdash_P^* (q, \epsilon, \epsilon)\}$$

2. Take A_{st} as the start variable of CFG G, where $s = q_0$ and $t = q_{accept}$.
3. Then $L(G)(= L(A_{st}))$ coincides with

$$\{w : (q_0, w, \epsilon) \vdash_P^* (q_{accept}, \epsilon, \epsilon)\},$$

   which is precisely $L(P)$.

$\Leftarrow$: Converting PDA to CFG

Step C. Designing a production rule for the variable A_{pq}.

1. For a string w in

$$\{w : (p, w, \epsilon) \vdash_P^* (q, \epsilon, \epsilon)\},$$

two situation can occur when P runs on w.

   A. the stack gets empty while running
   B. the symbol pushed at the beginning is never popped till the last moment.

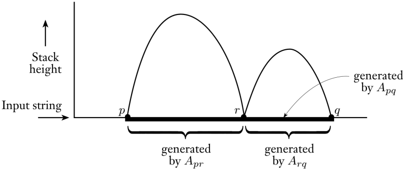

Figure 2.28, Sipser 2012

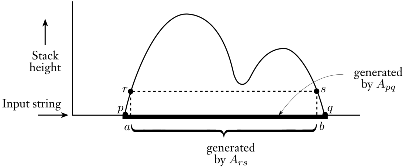

Figure 2.29, Sipser 2012

$\Leftarrow$: Converting PDA to CFG

Step C. Designing a production rule for the variable A_{pq}.

1. For a string w in
$$\{w : (p, w, \epsilon) \vdash_P^* (q, \epsilon, \epsilon)\},$$
two situation can occur when P runs on w.
   - A. the stack gets empty while running: i.e. $w \in L(A_{pr}) \cdot L(A_{rq})$.
   - B. the symbol pushed at the beginning is never popped till the last moment. i.e. $w \in aL(rs)b$ for all $a, b \in \Sigma$ whenever $\delta(p, a, \epsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ϵ) for some $u \in \Gamma$.
2. The trivial case $(p, \epsilon, \epsilon) \vdash_P^* (p, \epsilon, \epsilon)$.

Each case is simulated by the next rules.
- case A: $A_{pq} \to A_{pr}A_{rq}$ for all $p, q, r \in Q$
- case B: $A_{pq} \to aA_{rs}b$ for all $p, q, r, s \in Q$ and $a, b \in \Sigma_\epsilon$, and $u \in \Gamma$ such that $\delta(p, a, \epsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ϵ).
- case C: $A_{pp} \to \epsilon$.

The new CFG G contains all the above rules.

$\Leftarrow$: Converting PDA to CFG

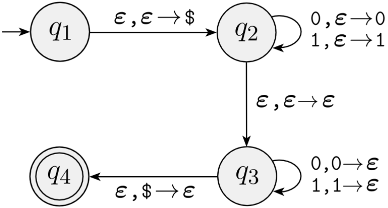

Figure 2.19, Sipser 2012

Why the conversions produce equivalent PDA / CFG?

(A quick words, which you can turn into a correctness proof.)

$\Rightarrow$: from CFG to PDA

- As an invariant, at each step of PDA's run on w,

  (the prefix of w that P already read) $\circ$ (the string in the stack, save \$) ($\star$)

  forms a leftmost derivation from S, implying $L(P) \subseteq L(G)$.

- For any $w \in (\Sigma \cup V)^*$ with $S \Rightarrow^*_{lm} w$, there is a run of P ending in a configuration with ($\star$). Especially, there is a run which ends up with an empty stack (save \$) after having read all symbols in the input, implying $L(G) \subseteq L(P)$.

- Use induction to argue both.

Why the conversions produce equivalent PDA / CFG?

(A quick words, which you can turn into a correctness proof.)

$\Leftarrow$: from PDA to CFG

- For both $L(P) \subseteq L(G)$ and $L(G) \subseteq L(P)$, Tedious induction...