CS 273, Lecture 14
PDA to CFG conversion, Chomsky Normal form, Grammar-based induction

4 March 2008

This lecture covers the construction for converting a PDA to an equivalent CFG (Section 2.2 of Sipser). We also cover the Chomsky Normal Form for context-free grammars and an example of grammar-based induction.

If it looks like this lecture is too long, we can push the grammar-based induction part into lecture 15.

1 NFA to CFG conversion

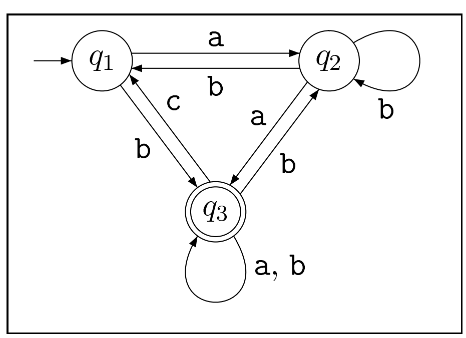
Before converting a PDA to a context-free grammar, let's first see how to convert an NFA to a context-free grammar.

The idea is quite simple: We introduce a symbol in our grammar for each state in the given NFA $N = (Q, \Sigma, \delta, q_0, F)$. We introduce a symbol for every state, and a rule for every transition. In particular, a state q_j would correspond to a symbol L_j (naturally, the language generated by L_j is the suffix language of the state q_j). A transition $q_j \in \delta(q_i, x)$, for $x \in \Sigma_\epsilon$, would be translated into the rule

$$L_i \to xL_j$$

We also add for any accepting state q_i the rule $L_i \to \epsilon$. (Note, that x can be ϵ. Then the ϵ-transition of moving from q_i to q_j, is translated into the rule $L_i \to L_j$.

As a concrete example, consider the NFA on the right. We introduce a CFG with symbols L_1, L_2, L_3, where L_1 is the initial symbol. We have the following rules:



$$\begin{aligned} L_1 &\to \mathtt{a}L_2 \mid \mathtt{b}L_3 \\ L_2 &\to \mathtt{b}L_1 \mid \mathtt{b}L_2 \mid \mathtt{a}L_3 \\ L_3 &\to \mathtt{b}L_2 \mid \mathtt{c}L_1 \mid \mathtt{a}L_3 \mid \mathtt{b}L_3 \mid \epsilon. \end{aligned}$$

Interestingly, the state q_3 is an accept state, and as such we add the transition $L_3 \to \epsilon$ to the rules.

2 PDA to CFG conversion

In this section, we will show how to convert a PDA into an equivalent CFG. We will first convert the PDA into a "normalized" form, and then we will generate a context free grammar (CFG) that explicitly writes down all the input strings that get the PDA from one state to another. Next, we will convert this normalized PDA into a CFG.

2.1 From PDA to a normalized PDA

Given a PDA N' we would like to convert into a PDA N that has the following three properties:

(A) There is a single accept state q_{accept}.

(B) It empties the stack before accepting.

(C) Each transition either pushes a symbol to the stack or pop it, but not both.

Transforming a given PDA into an equivalent PDA with these properties might seem like a tool order initially, but in fact can be easily done with some care.

(A) There is a single accept state q_{accept}.

This can be easily enforced by creating a new state q_{accept}, and creating ϵ-transitions from the old accept states to this new accept state.

(B) It empties the stack before accepting.

This can be easily enforced by pushing a special character \$ into the stack in the start state (introducing a new start state in the process). Next, we introduce a new temporary state q_{temp} which replaces q_{accept}, which has transitions popping all characters from the stack (excepting \$), and finally, we introduce the transition

$$q_{\mathrm{temp}} \xrightarrow{\$} q_{\mathrm{accept}}.$$

And of course, q_{accept} is the only accept state in this new automata.

(C) Each transition either pushes a symbol to the stack or pop it, but not both.

One bad case for us is a transition that both pushes and pops from the stack. For example, we have a transition

$$q_i \xrightarrow{\mathtt{x,b} \rightarrow \mathtt{c}} q_j.$$

(Read the character x from the input, pop $\mathtt{b}$ from the stack, and push $\mathtt{c}$ instead of it.) To remove such transitions, we will introduce a special state q'_{temp}, and introduce two transitions – one doing the pop, the other doing the push. Formally, for the above transition, we will introduce the transitions

$$q_i \xrightarrow{\mathtt{x,b} \rightarrow \epsilon} q'_{\mathrm{temp}} \quad \text{and} \quad q'_{\mathrm{temp}} \xrightarrow{\epsilon,\epsilon \rightarrow \mathtt{c}} q_j.$$

Similarly, if we have a transition that neither pushes nor pops anything, we replace it with a sequence of two transitions that push and then immediately pop some newly-created dummy stack symbol.

In the end of this normalization process, we end up with an equivalent PDA N that complies with our requirements.

2.2 From a normalized PDA to CFG

2.2.1 intuition

Consider a run of a normalized PDA $N = (Q, \Sigma, \delta, q_{\text{init}}, F)$ that accepts a word w. It starts at a state q_{init} (with an empty stack), and ends up at q_{accept}, again, with an empty stack. As such, it is natural to define for any two states $p, q \in Q$, the language $L_{p,q}$ of all the strings that starts at p with an empty stack, and end up in q with an empty stack.

For each pair of states p and q, we will have a symbol $S_{p,q}$ in our CFG for the language $L_{p,q}$. $S_{p,q}$ will generate all the strings in $L_{p,q}$. The language of N is then $L_{q_{\text{init}},q_{\text{accept}}}$.

So, consider a word $w \in L_{p,q}$ and how the PDA N works for this word. In particular, consider the stack as the PDA starting at p (with an empty stack) handles w.

Stack is empty in the middle. If during this execution, the stack ever becomes empty at some intermediate state r, then a word of $L_{p,q}$ can be formed by concatenating a word of $L_{p,r}$ (that got N from state p into state r with an empty stack), and a word of $L_{r,q}$ (that got N from r to q).

Stack never empty in the middle. The other possibility is that the stack is never empty in the middle of the execution as N transits from p to q, for the input $w \in L_{p,q}$. But then, it must be that the first transition (from p into say p_1) must have been a push, and the last transition into q (from say q_1) was a pop. Furthermore, the this pop transition, popped exactly the character pushed into the stack by the first transition (from p to p_1). Thus, if the PDA read the character x (from the input) as it moved from p to p_1 and read the letter y (from the input) as it moved from q_1 to q, then

$$w = xw'y,$$

where w' is an input that causes the PDA N to start from p_1 with an empty stack, and end up in q_1 with an empty stack. Namely, $w' \in L_{p_1 q_1}$.

Formally, if there is a push transition (pushing z into the stack) from p to p_1 (reading x) and pop transition from q_1 to q (popping the same z from the stack and reading y), then a word in $L_{p,q}$ can be constructed from the expression

$$xL_{p_1,q_1}y.$$

Notice that x and/or y could be ϵ, if one of the two transitions didn't read anything from the input.

2.2.2 The construction

We now explicitly state the construction. First, for every state p, we introduce the rule

$$S_{p,p} \to \epsilon.$$

The case that the stack is empty in the middle of transitioning from p to q is captured by introducing, for any states p, q, r of N, we define the following rule in our CFG:

$$S_{p,q} \to S_{p,r} S_{r,q}.$$

As for the other case, that the stack is never empty, we specify for any given states p, p_1, q_1, r of N, such that there is a push transition from p to p_1 and a pop transition from q_1 to r (that push and pop the same letter), we introduce an appropriate rule. Formally, for any p, p_1, q_1, r, if there are transitions in N of the form

$$\underbrace{p \xrightarrow{\text{x},\epsilon\to\text{z}} p_1}_{\text{push } z} \quad \text{and} \quad \underbrace{q_1 \xrightarrow{\text{y},\text{z}\to\epsilon} q}_{\text{pop } z}.$$

The introduce the rule

$$S_{p,q} \to x S_{p_1,q_1} y$$

into the CFG.

We create such rules for all possible choices of states of N. Let C be the resulting grammar. This completes the description of how we constructed the CFG equivalent to the given PDA N.

We claim that $S_{q_{\text{init}},q_{\text{accept}}}$ in the grammar C generates all the words that the PDA N accepts.

Remark 2.1 At the start of our construction, we got rid of all the transitions that don't touch the stack at all. Another option would have been to handle them with a variation of our second type of context-free rule. That is, we have a transition from a state p to p_1 that does not touch the stack (and reads the character x from the input). A small extension of the above construction would give us the transition:

$$S_{p,q} \to x S_{p_1,q}.$$

2.3 Proof of correctness

Here, prove that the language generated by $S_{q_{\text{init}},q_{\text{accept}}}$ is the language recognized by the PDA N.

Claim 2.2 *If the string w can be generated by $S_{p,q}$ then there is an execution of N stating at p (with an empty stack) and ending at q (with an empty stack).*

Proof: The proof is by induction on the number n of steps used in the derivation generating w from $S_{p,q}$

For the base of the induction, consider $n = 1$. The only rules in C that have no symbols in them are of the form

$$S_{p,p} \rightarrow \epsilon.$$

Which implies the claim trivially.

Thus, consider the case where $n > 1$, and assume that we proved that any word generated by at most n derivation steps (in the CFG grammar C) can be realized by an execution of the PDA N. We would like to prove the inductive step for $n + 1$. So, assume that w is generated from $S_{p,q}$ using $n + 1$ derivation steps. There are two possibilities for what is the first derivation rule used. The first possibility is that we used the rule

$$\underbrace{S_{p,q}}_{w} \underbrace{\rightarrow}_{=} \underbrace{S_{p,r}}_{w_1} \underbrace{S_{r,q}}_{w_2}.$$

As such, w_1 is generated from $S_{p,r}$ in at most $(n + 1) - 1 = n$ steps, and w_2 is generated from $S_{r,q}$ in at most $(n + 1) - 1 = n$ steps. As such, by induction, there is an execution of N starting at p and ending in r (with empty stack in the beginning and the end) and, similarly, there is an execution of N starting at r and ending q (with empty stack in the beginning and the end). By performing these two execution one of after the other, we end up with an execution starting at p and ending at q, with an empty stack on both ends, such that the PDA N reads the input w during this execution. Thus, this establishes the claim in this case.

The other possibility, is that w was derived by first applying a rule of the form

$$\underbrace{S_{p,q}}_{w} \underbrace{\rightarrow}_{=} \underbrace{x}_{x} \underbrace{S_{p_1,q_1}}_{w'} \underbrace{y}_{y}.$$

But then, by construction, the PDA N must have two transitions

$$p \xrightarrow{\mathtt{x},\epsilon\rightarrow\mathtt{z}} p_1 \quad \text{and} \quad q_1 \xrightarrow{\mathtt{y},\mathtt{z}\rightarrow\epsilon} q \tag{1}$$

that generated this rule. Furthermore, by induction, the word w' was generated from S_{p_1,q_1} using n derivation steps. As such, there exists a compliant execution X from p_1 to q_1 generating w'. Thus, if we start at p, apply the first transition of Eq. (1), then the execution X and then the second transition of Eq. (1), then we end up with a complaint execution of N that starts at p, ends at q (with empty stack on both ends), and reads the string w, which establishes the claim in this case, since we showed an execution that reads N. ■

Claim 2.3 *If there is an execution of N (with empty stack in both ends) starting at a state p and ending at a state q, that reads the string w, then w can be generated by $S_{p,q}$.*

Proof: The proof is somewhat similar to the previous proof. Consider the execution for w, and assume that it takes n steps. We will prove the claim by induction on n.

For $n = 0$, the execution is empty, and starts at p and ends at $q = p$. But then, w is ϵ, and it can be derived by $S_{p,p}$ since the CFG C has the rule $S_{p,p} \rightarrow \epsilon$.

Otherwise, for $n > 0$, assume by induction that we had proved the claim for all executions of length n, and we now consider an execution of length $n + 1$.

If the first transition in the execution is a push to the stack of a character z, and z is being popped by the last transition in the execution, then the first and last transitions are of the form

$$\underbrace{p \xrightarrow{\texttt{x},\epsilon\rightarrow\texttt{z}} p_1} \quad \text{and} \quad \underbrace{q_1 \xrightarrow{\texttt{y},\texttt{z}\rightarrow\epsilon} q},$$

respectively, and furthermore $w = xw'y$. As such, we have an execution of length $(n+1)-1 \leq n$ that reads w', and by induction, w' can be generated by the symbol S_{p_1,q_1}. But then, by construction, the rule

$$S_{p,q} \rightarrow xS_{p_1,q_1}y$$

is in the CFG C, and as such w can be generated by $S_{p,q}$, as claimed.

The other possibility is that z is being popped out at some earlier stages, as the PDA N enters a state r (after reading a prefix of w, denoted by w_1). But then, arguing as above, we can break w into two strings w_1 and w_2, such that $w = w_1w_2$, and by induction, w_1 can be generated by the rule $S_{p,r}$ and w_2 can be generated by the rule $S_{r,q}$. But then, the CFG C contains the rule

$$S_{p,q} \rightarrow S_{p,r}S_{r,q}.$$

Which implies that $S_{p,q}$ can generate the string w, as claimed. ■

As such, we conclude the following:

Lemma 2.4 *If the language L is accepted by a PDA N then L is a context-free language.*

Together with the results from earlier lectures, we can conclude the following.

Theorem 2.5 *A language L is context-free if and only if there is a PDA that recognizes it.*