Some notes on the PDA to CFG Construction: **Theorem 3.37**

G. Pecelli


This discussion starts at the bottom of p. 134, and takes us to the middle of p. 135.

The problem is how to use the info in the NPDA to generate the rules of the corresponding CFG. We must have a way to generate the variables of the grammar; the terminals are already given. The transitions in the NPDA are all of the form $\delta(p, a, A) \rightarrow (q, B)$ or $(q, BA)$ or $(q, \epsilon)$ (the latter are actually sets, but I take the three possible forms of the elements of the range, for simplicity). $a$ can be a character of the input alphabet or $\epsilon$. $A$ and $B$ can be any elements of $\Gamma \cup \{\$\}$. That’s what we have. The first part of the proof of Thm. 3.37, p. 134, is just a setup to justify being able to claim that the transitions are limited to the forms 1,2,3,4 on slides 24 - 25 (or p. 134). 3 includes both popping the top and pushing a new symbol or just popping the top. 4 takes care of pushing without previous popping. For those who want to try it, the discussion in http://www1.cs.columbia.edu/~zeph/3261/L12/L12.ppt solves the same problem in a slightly different manner, but the idea (and the need) is the same (thanks to Mr. I. White for the URL).

Hopefully, we are OK to here.

How can we construct grammar rules from the new NPDA?

One observation would be that, if we consider a string $x$ being “consumed” by the NPDA starting in state $q$ and with some symbol, say $A$, as tos (imagine the rest of the stack as “walled off”: you will ignore anything below $A$ in the discussion that follows), the NPDA would go through a number of transitions and, if successful, would end in some state $p$, having consumed the string $x$ and having emptied the stack (except for the walled-off part). The customary notation for this (0 or more steps) sequence of transitions is

$$(q, x, A) \vdash^* (p, \epsilon, \epsilon),$$

where we are leaving the dependence on the stack below the original $A$ out of the computation. If it makes it easier think of $A$ as the only symbol on the stack - we are dealing with “internal strings” rather than strings in the language.

From the point of view of derivations we could introduce the corresponding notation

$$(q, A, p) \Rightarrow^* x,$$

(or, using notation closer to that of the Columbia handout mentioned above: $A_{qp} \Rightarrow^* x$) since the automaton “consuming” the string is equivalent to the grammar “generating” it. The expression
⟨q, A, p⟩ is just a symbol (at the level of grammar), while it also has a meaning associated with the NPDA from which it is obtained: it denotes the strings that can be consumed in a path from q to p starting with tos A and ending when the contributions of A are finished (we would start working with symbols below the original A). As the totality of the strings accepted by the automaton are strings for which

\[(s, x, $) \vdash^* (f, \epsilon, \epsilon),\]

where the $ in third position actually denotes an empty stack, the same totality of strings will be denoted by ⟨s, $, f⟩ (the start symbol for the grammar, or $sf, in the notation of the Columbia handout). Note that we should start with (s₁, x, $) and thus with (s₁, $, f) but we leave this out, being associated with the one instruction that cannot perform a pop: recall that, on the very first move, all we do is push a $.

This suggests that, for every possible input-output pair of the transition relation for the NPDA (except for this first one), we should construct one or more rules for the CFG.

We can construct one such symbol ⟨q, A, p⟩ (A \neq \epsilon), for every pair of states and stack symbol in the definition of the NPDA. We should note that it is not necessary that every possible ⟨q, A, p⟩ correspond to an actual string being generated. You would expect that many such symbols will not generate anything, because there is no path in the NPDA from q to p that starts with A as tos and stops in p when the contribution of A is completed.

Before we continue, we rewrite (as an example of what we should be ready for: pop A, consume x₁ and push B)

\[(q, x, A) = (q, x_1 x_2 \ldots x_n, A) \vdash (q_1, x_2 \ldots x_n, B) \vdash^* (p, \epsilon, \epsilon)\]

The corresponding derivations can be written (somewhat incorrectly but suggestively: see case 2 below for the precise notation and interpretation) as

\[⟨q, A, p⟩ \Rightarrow ⟨q, A, q₁⟩(q₁, B, p) \Rightarrow x₁(q₁, B, p) \Rightarrow^* x₁x₂ \ldots x_n,\]

where B is the stack symbol at the top of the stack after consuming x₁, and q₁ is the state reached. Using the other notation, we would have Aqp \Rightarrow Aqq₁B_{q₁,p} \Rightarrow x₁B_{q₁,p} \Rightarrow^* x₁ \ldots x_n.

What we still need to do is examine all the types of transitions that arise in the NPDA and write down the corresponding CFG rules.

**Note:** to repeat, the only pure push operation occurs at the beginning, when we introduce the $ as tos. After that, any push operation is preceded by a pop or by a pop-push pair where the tos is replaced by itself. Note further that these are represented by single transitions in the NPDA and each will give rise to a rule.

1. A pure pop operation (this could mean that this is the last operation or that the next operation is also a pop, using the portion of the stack below A): the image of δ′(q, a, A) contains (p, $), where a \in \Sigma \cup \epsilon. If there is no immediately subsequent push, we have either consumed an a \in \Sigma (with the tos usually matching it) or consumed nothing (a = \epsilon), and
popped tos. In either case, we have no more stack to use (anything below $A$ is “walled off”) and we have the rule

$$\langle q, A, p \rangle \rightarrow a$$

where we show the effects of a transition from state $q$ to the immediately following state $p$, with $A$ contributing only to this transition.

2. A pop followed by a push: the image of $\delta'(q, a, A)$ contains $(p, B)$, where $|B| = 1$, and $a \in \Sigma \cup \epsilon$ (we may be consuming an input character or not). At this point - after this instruction - the stack is not empty, and we have more actions to take. Our notation must show this: we are in state $q$, with $A$ as tos. As we pop $A$, the stack empties, but the push immediately following puts $B$ on the stack, so the stack has not emptied out, and we must indicate this. We don’t know what will happen in the next instruction (the $B$ will be popped, but it could be replaced by a $C$ or pushed again and “covered” by a $D$, or just popped taking us to case 1 above). So we do not know where (= in what state) the effect of being in state $q$ with $A$ as tos will terminate (= in what state we’ll be when the next move would take us to the portion of the stack below the original $A$). Since we are going to generate a string (= the string that the NPDA would consume starting from state $q$ and tos $A$, and ending in the state after which the next instruction will take us below the position of the original $A$), the beginning of the non-terminal symbol on the left will look like $\langle q, A, \_ \rangle$ (we don’t know the end state of this chain of instructions) and, on the right of the rule, the sentential form generated will look like $a\langle p, B, \_ \rangle$ (we execute one instruction and we don’t yet know where the chain will end, but it had better be the same place), where, if $a = \epsilon$, we have consumed no characters in the NPDA, nor generated any via the CFG rule (we have introduced a “unit rule”). Since we know we start in state $q$, but have no idea of which state we’ll ultimately end in, we need to create enough non-terminal symbols to take care of all possibilities: replace $\_$ by $r$, where $r$ ranges over all symbols in $Q \cup \{f\}$. The rules associated with this type of instruction are

$$\langle q, A, r \rangle \rightarrow a\langle p, B, r \rangle, \forall r \in Q \cup \{f\}.$$  

At this point one may raise an objection: why all of the possible $\langle q, A, r \rangle$ for all $r \in Q \cup \{f\}$? That’s not right...??? Surely some of the states $r$ are not reachable from $q$ with tos $A$ in the NPDA... Have we introduced rules that will generate a lot more strings? Well, some of those states may be reachable starting from $q$ with tos $A$ but not from $s$ with tos $\$ - which is the initial state of the NPDA...It would seem that all the strings accepted by the NPDA will be generated by this scheme, and, possibly, more... It certainly seems to allow for more “internal strings” to be generated (remember: we have to start from the start state to decide whether a string is accepted or not, starting in any other state does not count). The other alternative would be to start from $q$ with stack $A$ and use only those states $r$ that are reached via strings actually consumed by the NPDA between $q$ and $r$, with initial stack $A$. Since a proper CFL has infinitely many strings, this is not a feasible approach... That’s one reason why we generate so many rules.

3. A pop followed by two push operations: one to restore the symbol just popped, the second to push another symbol: the image of $\delta'(q, a, A)$ contains $(p, BA)$, where $|B| = 1$ and $a \in \Sigma \cup \epsilon$. The left hand side of the rule will look like $\langle q, A, \_ \rangle$, while the right-hand side must take care of the substrings generated by the instruction and both stack symbols: $a\langle p, B, -1 \rangle\langle -1, A, -2 \rangle$
(using indexed underscores to indicate still unknown states, where the index requires a match). The underscore in \( \langle q, A, \_ \rangle \) must match \( \_2 \) in \( \langle \_1, A, \_2 \rangle \), while \( \_1 \) also ranges over all states. The rule then:

\[
\langle q, A, r \rangle \rightarrow a(p, B, r') \langle r', A, r \rangle, \ \forall \ r, r' \in Q \cup \{f\}.
\]

At this point, all we need to prove is that

\[
\langle s, \$, f \rangle \Rightarrow^* x \Leftrightarrow (s, x, \$) \vdash^* (f, \epsilon, \epsilon).
\]

We will do something slightly more general:

\[
\langle q, A, p \rangle \Rightarrow^* x \Leftrightarrow (q, x, A) \vdash^* (p, \epsilon, \epsilon)
\]

for any states \( p \) and \( q \) and any stack symbol \( A \). In particular, this will prove that the almost indiscriminate generation of rules in 2) and 3) above does not give us more accepted strings, and not even more “internal strings”.

1. \( \langle q, A, p \rangle \Rightarrow^* x \Rightarrow (q, x, A) \vdash^* (p, \epsilon, \epsilon) \).
2. \( (q, x, A) \vdash^* (p, \epsilon, \epsilon) \Rightarrow \langle q, A, p \rangle \Rightarrow^* x \).

The proof is on pp. 135 and 136.