

For every CFG there is an equivalent PDA

It has been said many times that PDA's are an abstract machine *just* powerful enough to recognize the languages generated by CFG's. In order to prove this one must show that for any CFG G there exists a PDA M s.t. $L(M)=L(G)$. Furthermore one must also show that for any PDA M there exists a CFG G s.t. $L(G) = L(M)$.

It is interesting to point out that PDA's were formally introduced by Oettinger in 1961 and CFG's by Chomsky in 1956. The equivalence between these two were first perceived by Chomsky in 1962 and Evey in 1963.

In both directions you will see a constructive proof. (i.e. A construction followed by a proof that the construction does what it says it does.)

Theorem 1

If $G = (V, \Sigma, S, P)$ is a CFG, there exists a PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, A)$ such that $L_f(m) = L(G)$.

Proof: We define our PDA M as follows:

$Q = \{q_0, q_1, q_2\}$ - It is interesting to note that ANY CFG can be recognized (via final state) by a PDA with only three states.

$\Gamma = V \cup \Sigma \cup \{Z_0\}$, $Z_0 \notin V \cup \Sigma$ - Our stack symbols are the grammar variables, a stack start symbol and the input alphabet.

$A = \{q_2\}$ - Only need one accepting state for ANY CFG.

We must now define our PDA's transition function δ

1. $\delta(q_0, \epsilon, Z_0) = \{(q_1, SZ_0)\}$
2. For $A \in V$, $\delta(q_1, \epsilon, A) = \{(q_1, \alpha) | (A \rightarrow \alpha) \in P\}$
3. For each $a \in \Sigma$, $\delta(q_1, a, a) = \{(q_1, \epsilon)\}$
4. $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$

The proof can be completed by showing $L(G) \subseteq L_f(M)$ and $L_f(M) \subseteq L(G)$ (both via induction) which is left as an exercise for the reader. \diamond

The basic idea of the construction is to use the start state and a required ϵ -transition first move to transit to state q_1 while placing the grammar's start symbol on the stack. Whenever a grammar variable is on the top of the stack we can pop it off replacing it with its righthand side string.

Whenever the input symbol matches the stack's top (grammar terminals) we eat the input and pop the symbol off the stack. Finally when input is exhausted and Z_0 is all that is left on the stack (nothing was ever placed under it), the PDA transits to the accepting state.

Hopefully one sees that the constructed PDA will correctly guess the steps necessary for a leftmost string derivation for $x \in L(G)$. Simply put if at some point in the derivation the current string looks like $x\alpha$, where $x \in \Sigma^*$ (a string of terminals), then at some point in the PDA's simulation of the leftmost derivation, the input string read so far is x and that stack contains α .

It should be clear that this PDA is NOT a DPDA. The nondeterminism is introduced by there being multiple productions for any given variable A . Any nondeterminism inherent in the grammar is replicated in the PDA.

For every PDA there is an equivalent CFG

A word (or two) is in order before looking at how to construct a CFG out of any PDA M . Assume w.l.o.g. that the given PDA M accepts language L via empty stack $L = L_e(M)$. (Given a PDA M' that accepts L via final state one can convert M' into a PDA M such that $L = L_e(M) = L_f(M')$.)

We already know what the terminal symbols are, but what to use for the grammar's variables? It should be obvious that one cannot use the PDA's states; there are probably not enough of them for our purposes. Furthermore one cannot use either the set Γ of stack symbols (not enough information encoded) or all the possible stack configurations (infinitely many). Instead our variables will be things of the form $[p, A, q]$ where $p, q \in Q$; $A \in \Gamma$. This will certainly provide more variables than needed. (No need to worry though: you should know the algorithm for eliminating useless variables).

Whenever the PDA can move from state p to state q , read $a \in \Sigma \cup \{\epsilon\}$ from the input stream and pop A , $A \in \Gamma$, off the stack (a true pop operation, not a replace) we introduce into the grammar the production $[p, A, q] \rightarrow a$. Hence our grammar's variables incorporate both state information and stack top symbol information.

It will be good if the PDA and grammar to operate as they did above: simulate leftmost derivations. Another way of stating this is that the string read so far by the PDA will be the initial portion of the current string in a leftmost derivation, and what is on the stack will correspond to the remaining portion of the current string.

Theorem 2

If $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, A)$ is a PDA there exists a CFG $G = (V, \Sigma, S, P)$ such that $L_e(M) = L(G)$

Proof: The CFG G is defined as follows:

$V = \{S\} \cup \{[p, A, q] \mid A \in \Gamma; p, q \in Q\}$ – more than is needed but it is harmless to include useless variables.

The productions of the grammar are:

- For every $q \in Q$; $(S \rightarrow [q_0, Z_0, q]) \in P$
- If $q_1, q_2 \in Q$; $a \in \Sigma \cup \{\epsilon\}$; $A \in \Gamma$; and $\delta(q_1, a, A)$ contains (q_2, ϵ) then $([q_1, A, q_2] \rightarrow a) \in P$
- If $m \geq 1$; $q_0, q_1 \in Q$; $a \in \Sigma \cup \{\epsilon\}$; $A \in \Gamma$; $\delta(q_0, a, A)$ contains $(q_1, B_1 B_2 \dots B_m)$ for some string $B_1, B_2, \dots, B_m \in \Gamma$ then for every possible choice (i.e. assignment) of $q_a, \dots, q_m \in Q$, the production

$$[q_0, A, q_m] \rightarrow a[q_1, B_1, q_a][q_a, B_2, q_b] \dots [q_l, B_m, q_m]$$

is included in P

The proof is completed by showing $L(G) \subseteq L_e(M)$ and $L_e(M) \subseteq L(G)$ (both via induction) which are left as exercises for the reader. \diamond

It is my opinion that the transformation from a CFG to a PDA makes “sense” (whatever that means). On the other hand the transformation from a PDA to a CFG appears like Voodoo magic; how can this possibly work. One must first resign oneself to the observation that this algorithm produces many many useless productions and variables. It is also true that while one can certainly produce strings from the grammar that are in $L_e(M)$, it is also true that one cannot produce any strings using this grammar that are not in $L_e(M)$.

It might help to know that the variables and productions of G have been defined in such a way that a leftmost derivation in G of a string x is a simulation of the PDA M when fed x as input. The variables that appear in any step of a leftmost derivation in G correspond to the symbols on the stack of M at a time when M has seen as much of the input as the grammar has already generated. Put another way, the intention is that $[p, A, q]$ derive x if and only if x causes M to erase an A from its stack by some sequence of moves beginning in state q and ending in state p .