CFGs and PDAs are Equivalent

We provide algorithms to convert
a CFG to a PDA and vice versa.

CFGs and PDAs are Equivalent

We now prove that a language is generated by some CFG if and only if it is accepted by some PDA

The proof consists of two conversions.

Conversion from CFG to PDA

For the conversion from CFG to PDA, the idea is that the PDA guesses the leftmost derivation and checks that it is correct. (***proof by simulation***)

Each step of the derivation is a string. The machine guesses the next step. The problem is to do this with only a stack.

Terminals to the left of leftmost variable are not stored, but are matched with input string. Only current string from leftmost variable onwards is stored on stack, with *leftmost* symbol on top.

Operating the PDA

The PDA starts with start variable on the stack. At each step:

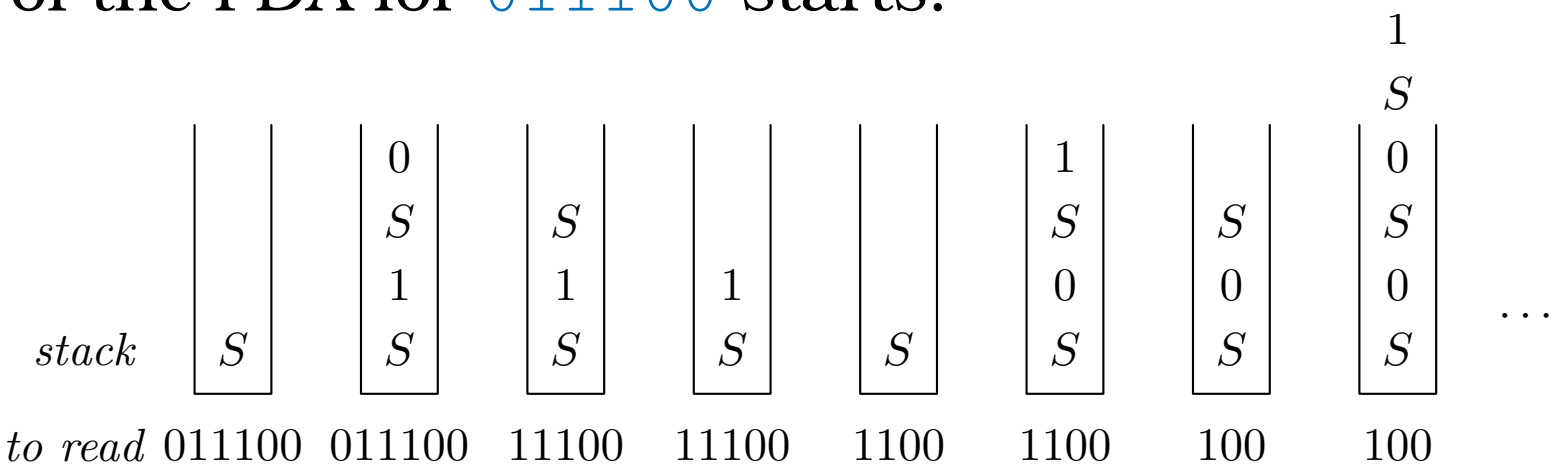
- If stack-top is a variable, then PDA guesses a production and replaces variable by RHS of that production.
- If stack-top is a terminal, then PDA pops it and reads one symbol from input and checks that the two match.

Accept if and only if reach empty stack at end of string. (Note that PDA not guaranteed to halt.)

Example

$S \to \mathtt{0}S\mathtt{1}S \mid \mathtt{1}S\mathtt{0}S \mid \varepsilon$

The sequence of stacks for the accepting branch of the PDA for `011100` starts:



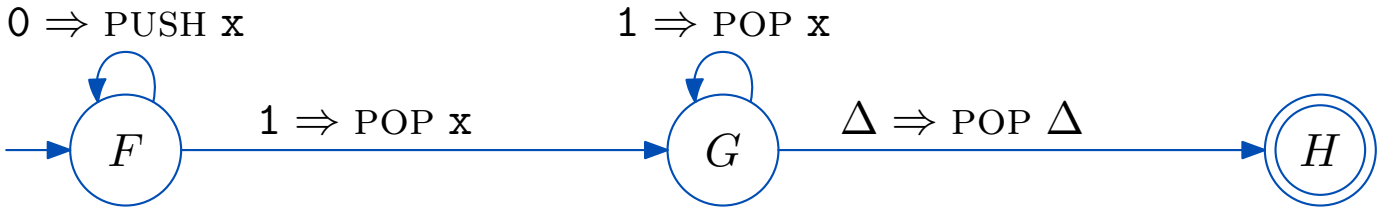
An Alternative Representation of PDA

We show next how to convert from PDA to CFG.

This procedure is slightly easier if PDA is in another form: where only the reading states are retained and stack operations are part of the transitions.

Example

Here is earlier PDA for $\mathtt{0}^n\mathtt{1}^n$ redrawn:



Formal Definition

In this light, a ***formal definition*** of a PDA is a 7-tuple $(Q, \Sigma, \Gamma, q_0, h_a, \Delta, \delta)$ where:

- Q is finite set of states;
- Σ is finite input alphabet;
- Γ is finite stack alphabet;
- q_0 is start state;
- h_a is accept state;
- Δ is empty-stack and end-of-string symbol; and
- δ is transition function.

The transition function needs more explanation. . .

The Transition Function

The transition function is a function from $Q \times (\Gamma \cup \{\Delta\}) \times (\Sigma \cup \{\varepsilon, \Delta\})$ to finite subsets of $Q \times (\Gamma \cup \{\Delta\})^*$.

That is, it looks at current state, current stack-top symbol, and (possibly) input symbol; then it updates the state and replaces the current stack-top symbol by a string of symbols:

pop = replace stack-top symbol by nothing;
push = replace stack-top symbol by itself and a new symbol.

Example

For example, $\delta(q, \texttt{A}, \texttt{a}) = \{(p, \texttt{AB})\}$ means that in state q with A on stack-top, on reading a the PDA pushes symbol B and changes to state p.

Conversion from PDA to CFG

The idea for the conversion from PDA to CFG is to make each step in a derivation correspond to a move by the PDA. This is far from trivial!

We use the earlier alternative representation. Further, we assume that: (a) stack of the PDA M is empty if and only M is in the accept state; (b) every move is either a push of a single symbol or a pop of a single symbol (Exercise).

The Variables are Triples

The variables of the CFG will be *all* the ordered triples $\boxed{q\,\mathtt{A}\,p}$ where q, p are states of the PDA, and $\mathtt{A}$ is a symbol from the stack alphabet.

Invariant for Triples

We construct the CFG such that for all strings w:

$$\boxed{q\,\mathtt{A}\,p} \stackrel{*}{\Longrightarrow} w \quad \Leftrightarrow$$

M can go from state q to p while reading w with the net effect of popping symbol $\mathtt{A}$.

$[q_0, \Delta, h_a]$ will be start variable for the CFG: it will generate string w if and only if one can go from q_0 to h_a while reading w and popping the empty-stack symbol; that is, w is accepted.

The Productions: Pops

The two types of moves of the PDA become productions.

Consider a ***pop*** move; say $(p, \varepsilon) \in \delta(q, \mathtt{A}, \mathtt{a})$. That is, in state q the PDA can pop symbol $\mathtt{A}$ while reading $\mathtt{a}$ and changing to state p ($\mathtt{a}$ can be ε).

Then we add one production:

$$\boxed{q\,\mathtt{A}\,p} \rightarrow \mathtt{a}$$

The Productions: Pushes

Consider a ***push*** move; say $(p, \mathtt{AB}) \in \delta(q, \mathtt{A}, \mathtt{a})$. That is, in state q with $\mathtt{A}$ on stack-top, one can push symbol $\mathtt{B}$ while reading $\mathtt{a}$ and changing to state p.

Then we add the collection of productions:

$$\boxed{q\,\mathtt{A}\,r} \rightarrow \mathtt{a}\ \boxed{p\,\mathtt{B}\,s}\ \boxed{s\,\mathtt{A}\,r} \text{ for } \textit{every} \text{ state } r \text{ and } s.$$

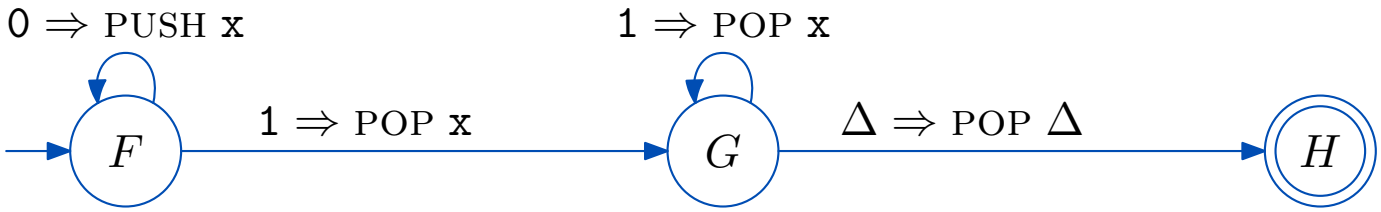
Essentially, there are two ways that effectively erase the stack-top symbol: (a) pop it, or (b) push something, and then later erase both symbols.

Proof Details Omitted

The rest of the proof is to show that every derivation in this grammar corresponds to a legitimate computation of the PDA.

Example Conversion

We use the PDA for $\{0^n 1^n\}$ from before.



Here goes. . .

Example CFG

Start symbol: $\boxed{F \,\Delta\, H}$

Popping occurs in three places:

$$\boxed{F \,\mathtt{x}\, G} \;\to \mathtt{1}$$
$$\boxed{G \,\mathtt{x}\, G} \;\to \mathtt{1}$$
$$\boxed{G \,\Delta\, H} \;\to \varepsilon$$

Pushing yields 18 productions of the form

$$\boxed{F\, A\, r} \;\to \mathtt{0}\; \boxed{F \,\mathtt{x}\, s}\; \boxed{s\, A\, r} \quad \text{for each } A \in \{\mathtt{x}, \Delta\}$$
and $r, s \in \{F, G, H\}$.

We could stop, but one can simplify:

Simplification

Omit productions that produce triples that are never LHS; this leaves

1) $\boxed{F \times F} \rightarrow 0\ \boxed{F \times F}\ \boxed{F \times F}$
2) $\boxed{F \times G} \rightarrow 0\ \boxed{F \times F}\ \boxed{F \times G}$
3) $\boxed{F \times H} \rightarrow 0\ \boxed{F \times F}\ \boxed{F \times H}$
4) $\boxed{F \triangle F} \rightarrow 0\ \boxed{F \times F}\ \boxed{F \triangle F}$
5) $\boxed{F \triangle G} \rightarrow 0\ \boxed{F \times F}\ \boxed{F \triangle G}$
6) $\boxed{F \triangle H} \rightarrow 0\ \boxed{F \times F}\ \boxed{F \triangle H}$
7) $\boxed{F \times G} \rightarrow 0\ \boxed{F \times G}\ \boxed{G \times G}$
8) $\boxed{F \triangle H} \rightarrow 0\ \boxed{F \times G}\ \boxed{G \triangle H}$

$\boxed{F \times F}$ is also unusable (why?); omit its rules.

Final CFG

This leaves us with the CFG with start variable $\boxed{F \triangle H}$:

$$\boxed{F \times G} \rightarrow 1$$
$$\boxed{G \times G} \rightarrow 1$$
$$\boxed{G \triangle H} \rightarrow \varepsilon$$
$$\boxed{F \times G} \rightarrow 0 \boxed{F \times G} \boxed{G \times G}$$
$$\boxed{F \triangle H} \rightarrow 0 \boxed{F \times G} \boxed{G \triangle H}$$

Whew!

Summary

There is an algorithm to convert a CFG to an equivalent PDA: the PDA guesses the leftmost derivation. The algorithm to convert from PDA to CFG is more complex.