Lecture 11: PA $\equiv$ CFG

Theorem 1 *If a language is context-free, then it is accepted by a PA.*

The proof is a constructive proof. We first sketch the idea. We construct a PA from the given CFG as follows:

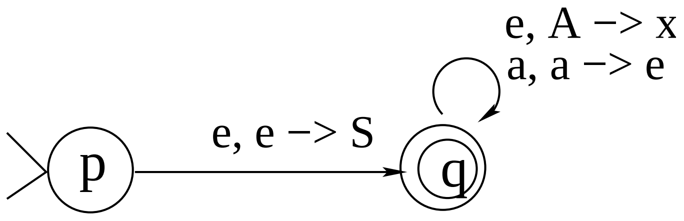
1. Push the start nonterminal of the CFG onto the stack
2. Repeat the following steps:
   (a) If the top of the stack is a nonterminal, say A, nondeterministically select one of the rules for A, and replace A in the stack by the string on the right hand side of the rule (recall: leftmost symbol in the string ends up topmost in the stack).
   (b) If the top of the stack is a terminal symbol, say a, read the next symbol from the input and compare it with a. If they match, repeat step (2). If not, reject this branch of non-determinism.

Recall: a string is accepted by a PA iff *there exists* a computation sequence such that when the entire input string is read, the PA is in a final state and the stack is empty.

PA $\equiv$ CFG

Proof:

- Given a CFG $G = (V, \Sigma, R, S)$.
- Construct the following PA M that accepts $L(G)$.
  $M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$ where Δ contains the following transitions:
  1. $((p, e, e), (q, S))$
  2. $((q, e, A), (q, x))$ for each rule $A \rightarrow x$ in R
  3. $((q, a, a), (q, e))$ for each $a \in \Sigma$



The transitions of M are designed such that what is stored in the stack in fact mimics a leftmost derivation of the input string.

That is, M has two main types of transitions:

– remove from the top of stack any terminal symbols and matches them against symbols in the input string (this exposes the leftmost nonterminal in the stack)

– remove from top of the stack a nonterminal, say A, and pushdown the right-hand side x of some rule $A \rightarrow x$.

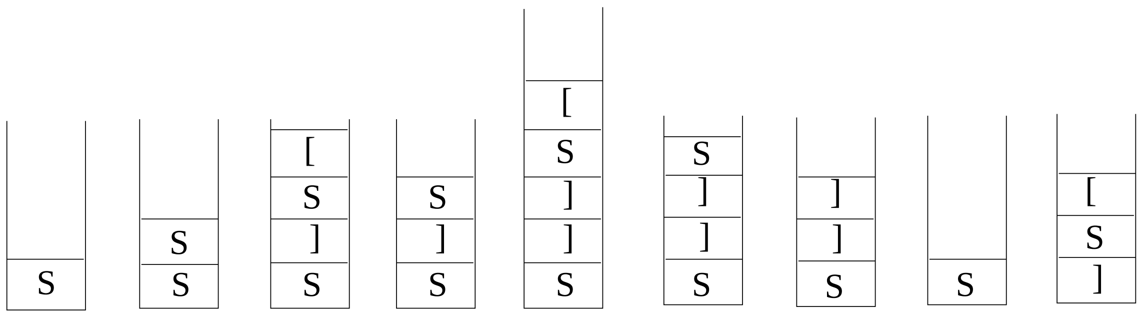
Example:

Given a CFG $G(V, \Sigma, R, S)$ where

$$V = \{S, [,]\}, \Sigma = \{[,]\}$$
$$R = \{S \to SS, S \to [S], S \to e\}$$

Leftmost derivation of the string [[ ]][ ]:

$S \Rightarrow SS \Rightarrow [S]S \Rightarrow [[S]]S \Rightarrow [[]]S \Rightarrow [[]][S] \Rightarrow [[]][]$



We can construct a PA that accepts $L(G)$:

$M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$ where

$$\begin{aligned}\Delta = \{ \quad & ((p, e, e), (q, S)), \\ & ((q, e, S), (q, SS)), \\ & ((q, e, S), (q, [S])), \\ & ((q, e, S), (q, e)), \\ & ((q, [, [), (q, e)), \\ & ((q,],]), (q, e))\}\end{aligned}$$

$(p, [[]][], e) \vdash (q, [[]][], S) \vdash (q, [[]][], SS) \vdash (q, [[]][], [S]S) \vdash (q, []][], S]S) \vdash (q, []][], [S]]S) \vdash (q,]][], S]]S) \vdash (q,]][],]]S) \vdash (q,][],]S) \vdash (q, [], S) \vdash (q, [], [S]) \vdash (q,], S]) \vdash (q,],]) \vdash (q, e, e)$

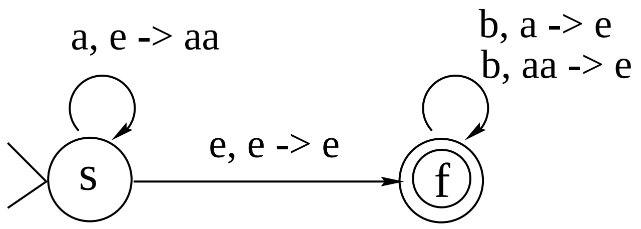
- *M mimics the leftmost derivation of the input string.*

Theorem 2 *If a language is accepted by a PA, then it is a context-free language.*

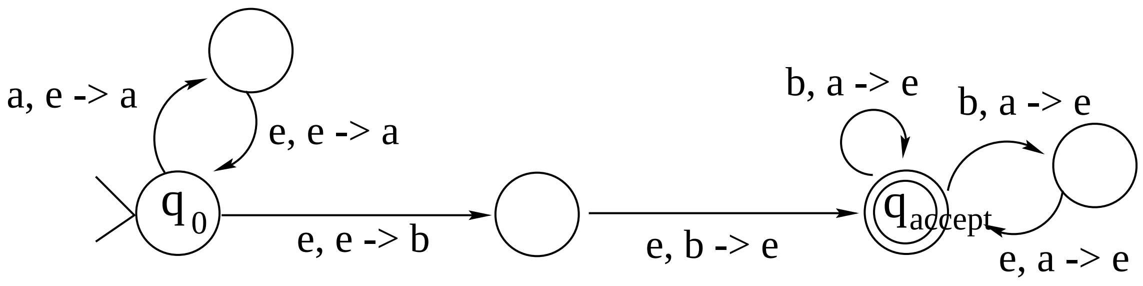
Proof: (We shall skip the proof. It is outlined here for completeness, details are in Michael Sipser's book.)

- Given a PA $P = (K, \Sigma, \Gamma, \Delta, q_0, F)$
- Convert P to a simple equivalent PA, M. Then design a CFG G such that $L(G) = L(M)$.
- M has the following features:
  1. It has a single accepting state, q_{accept}
  2. Each transition must be in one of the following 2 formats, where $p, q \in K, a \in \Sigma \cup \{e\}, t \in \Gamma$:
     - $((q, a, e), (p, t))$
     - $((q, a, t), (p, e))$

Example:
$\{a^m b^n : m \leq n \leq 2m\}$



Convert to an equivalent PA:



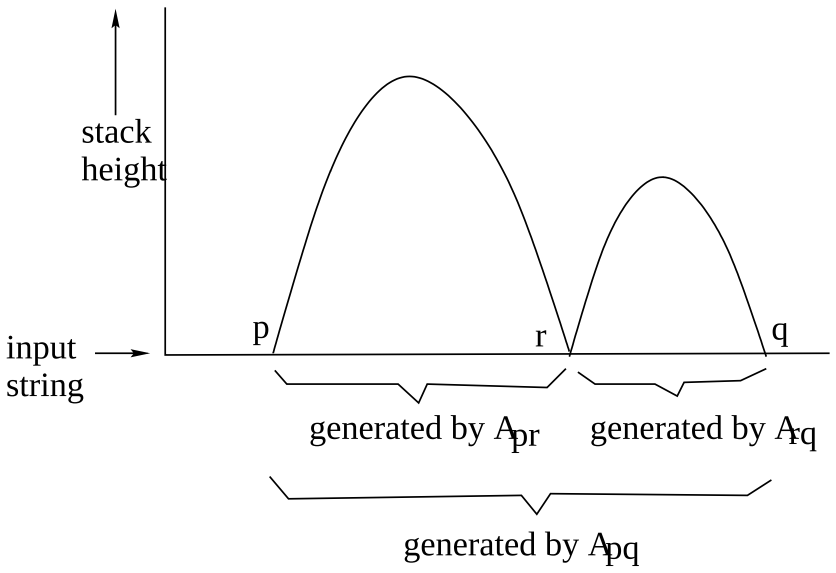
PA $\equiv$ CFG

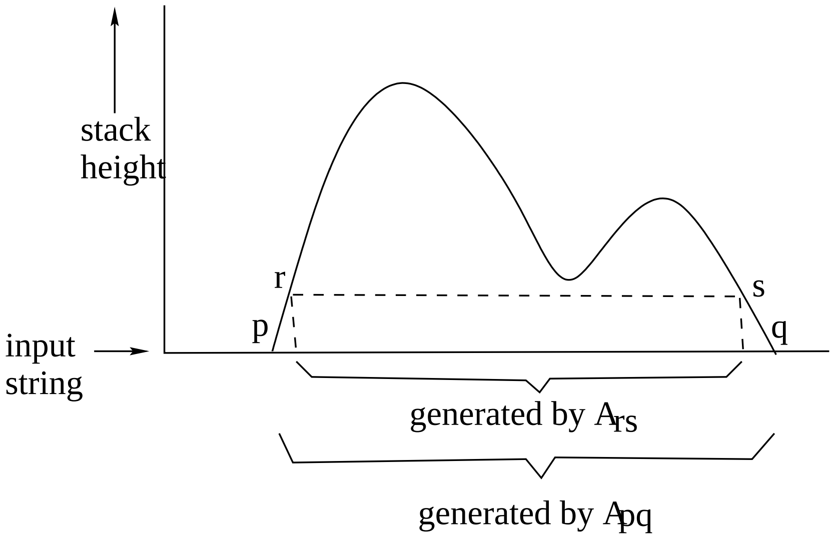
Design G from the simple PA M so that the nonterminal A_{pq} generates all strings that take M from state p to q, starting and ending with empty stack.
Note: $L(G)$ is the set of strings generated from the nonterminal $A_{q_0,q_{accept}}$.

Observed behaviors of PA M:

- For any string w that takes M from p to q, starting and ending with empty stack, M's first move must be a push (because M can't pop an empty stack and all transitions either pop or push one symbol), and M's last move on w must be a pop (so that the stack is empty)
- M's computation on w has two possibilities:
  1. The symbol popped at the last computation step is the symbol pushed in the first step, and that symbol is not popped during any intermediate computation step.
     Then, we simulate the computation sequence with the rule $A_{pq} \rightarrow aA_{rs}b$, where a is the symbol read in the first move, b is the symbol read in the last move, $a, b \in \Sigma \cup \{e\}$, r is the state following p, s is the state preceding q.

2. The initially pushed symbol was popped in an intermediate step. Then simulate this computation with the rule $A_{pq} \rightarrow A_{pr}A_{rq}$, where r is the intermediate state when the initially pushed symbol was popped.


stack height
input string
p
r
q
generated by A_pr
generated by A_rq
generated by A_pq


stack height
input string
r
p
s
q
generated by A_rs
generated by A_pq

PA $\equiv$ CFG

Given a *simple* PA $M = (K, \Sigma, \Gamma, \Delta, q_0, \{q_{accept}\})$.

We can construct a CFG $G = (V, \Sigma, R, S)$ where
$S = A_{q_0, q_{accept}}$
$V = \{A_{pq} : p, q \in K\} \cup \Sigma$
The rules in R are constructed as follows:

1. for each $p, q, r, s \in K$, $t \in \Gamma$, $a, b \in \Sigma \cup \{e\}$, if $((p, a, e), (r, t)) \in \Delta$ and $((s, b, t), (q, e)) \in \Delta$,
   then put the rule $A_{pq} \rightarrow a A_{rs} b$ in R.
2. for each $p, q, r \in K$, put the rule $A_{pq} \rightarrow A_{pr} A_{rq}$ in R.
3. for each $p \in K$, put the rule $A_{pp} \rightarrow e$ in R.

To be complete and rigorous, we should show that A_{pq} generates w if and only if w can take M from p with empty stack to q with empty stack (see details in Sipser's book).