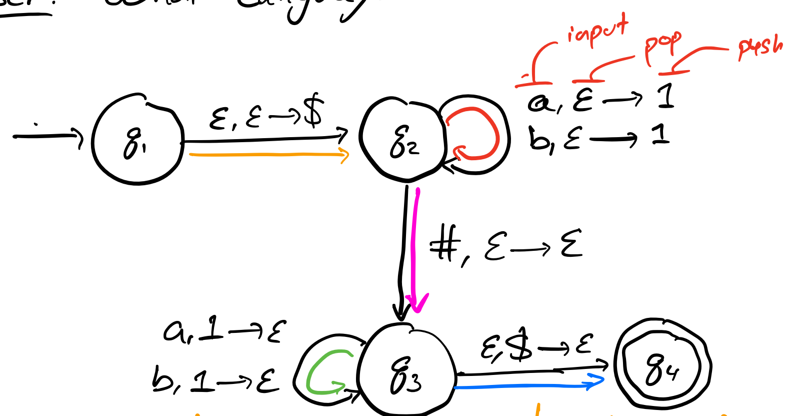
COMS W3261 — Lecture 7, Part 1

Equivalence of CFGs and PDA. Non-context free languages.

Teaser: What language does this PDA recognize?



1. Push \$ onto the stack. \$ will mark the bottom.
2. Read in a's and b's from the input, push a 1 onto the stack for every a or b that we read.
3. Read in a #.
4. Read in a's and b's and pop 1's off the stack.
5. Once our stack is exhausted, we pop the \$ and go to state q_4. If we're done reading input, we accept.

What does an accepting string look like?

$$\{(a \cup b)^k \# (a \cup b)^k \mid k \geq 0\}.$$

Announcements: HW #4 due Monday, 7/26/21 @ 11:59 PM EST.
If all homeworks average below ~85%, we may curve some up.

Readings: Sipser 2.2 (PDAs = CFGs)
Sipser 2.3 (Non-CFLs)

Today: 1. Review (PDA)

2. PDAs recognize the CFLs.

2.1) How to convert CFG $\to$ PDA

2.2) How to convert PDA $\to$ CFG.

3. Non-context free languages (and a new pumping lemma.)

1. Review: PDAs

Def. A Pushdown Automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$,

where Q is a finite set of states

Σ and Γ are finite input and stack alphabets,

q_0 is the start state,

$F \subseteq Q$ is the set of accept states,

and $\delta : \underbrace{Q}_{\text{a state}} \times \underbrace{\Sigma_\varepsilon}_{\text{an input symbol (or } \varepsilon)} \times \underbrace{\Gamma_\varepsilon}_{\text{a popped stack symbol (or } \varepsilon)} \longrightarrow \overbrace{\mathcal{P}}^{\text{this is the power set "all subsets of"}}(\underbrace{Q}_{\text{a new state}} \times \underbrace{\Gamma_\varepsilon}_{\text{a symbol to push}})$

A PDA accepts the input $w = w_1 w_2 \cdots w_m$, $w_i \in \Sigma_\varepsilon$ if there exists sequences of states and strings

$$r_0, r_1, \ldots, r_m \in Q$$
$$s_0, s_1, \ldots, s_m \in \Gamma^*,$$ such that

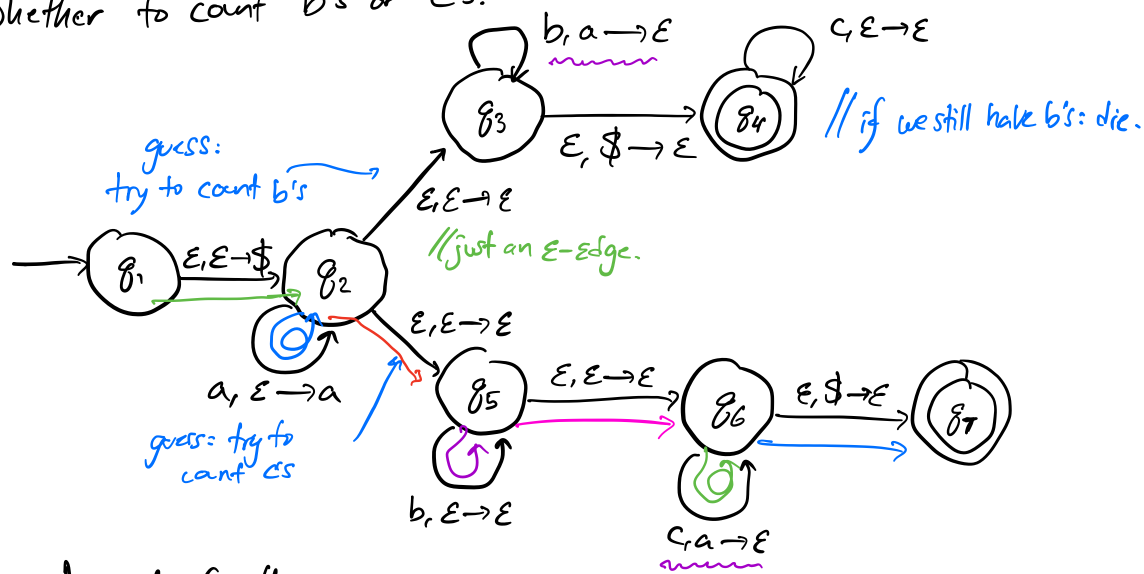
(1) $r_0 = q_0$, $r_m \in F$, $s_0 = \varepsilon$.

(2) For all $i \in 0, 1, \ldots m-1$, $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, and $s_i = at$, $s_{i+1} = bt$ for some $a, b \in \Gamma$ and $t \in \Gamma^*$.

Example. Build a PDA that recognizes the language

$$L = \{a^i b^j c^k \mid i, j, k \geq 0, \text{ and } i = j \text{ OR } i = k\}$$

Idea: push a's onto the stack, then nondeterministically guess whether to count b's or c's.



Accept if #a's = #b's or #a's = #c's.

Test string:
aabcc
(only one accepting branch)

state	stack
q_1	ε
q_2	$\$$
q_2	$a\$$
q_2	$aa\$$
q_5	$aa\$$
q_5	$aa\$$
q_6	$aa\$$

state	stack
q_6	$a\$$
q_6	$\$$
q_7	ε

✓

---

2. Pushdown Automata recognize the context-free languages.

Recall: A language is context-free if some context-free grammar describes it.

We'll show:

Theorem: A language is context-free if and only if some Pushdown Automaton recognizes it.

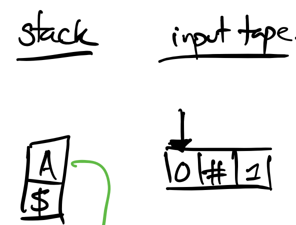
Follows immediately from two lemmata (lemmas).

Lemma 1. (CFG → PDA). If a language is context-free, some PDA recognizes it.

Lemma 2. (PDA → CFG). If a PDA recognizes some language, that language is context-free.

Idea: To prove Lemma 1, convert generic CFG → PDA. CFGs derive every string from a series of substitution rules. We'll show a PDA that nondeterministically guesses which rule to use, guess all leftmost derivations and checks if any matches the input string.

How our PDA will work:

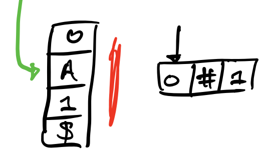


(1) Push $ and the start symbol.

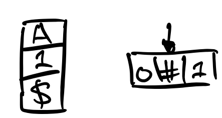
Consider PDA operation $L = \{0^n \# 1^n \mid n \geq 0\}$, and the string $0\#1$.

Given the grammar $G: A \rightarrow 0A1 \mid \#$ for L.

(2) If the top of the stack is a variable, nondeterministically choose a substitution rule and implement it. (Example: $A \rightarrow 0A1$.)



(3) If the top of the stack is a terminal, read an input character. If it matches the stack, pop the stack. If not, the branch dies.

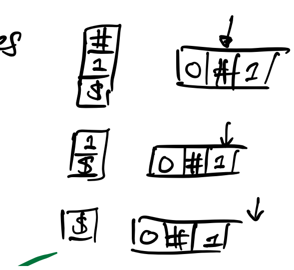


(4) Repeat steps 2 and 3 until the branch dies or we see $.

(choose $A \rightarrow \#$)

(pop #, read #)

(pop 1, read 1)



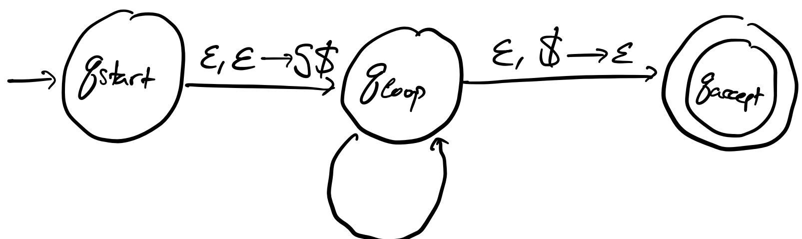
(5) Accept if all input has been read. ✓

Lemma 1. If a language is context-free, some PDA recognizes it.

Proof. Let $G = (V, \Sigma, R, S)$ be a CFG. We show how to build an equivalent PDA $P = (Q, \Sigma, \Gamma, \delta, q_{start}, F)$.

Σ $\quad V \cup \Sigma$ $\quad$ ✓ $\quad \{q_{accept}\}$

PDA skeleton:



$\varepsilon, A \to w$ for every rule $A \to w \in R$.

push a string?

$a, a \to \varepsilon$ for every terminal $a \in \Sigma$.

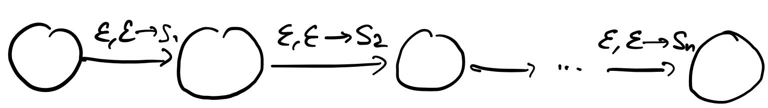
Suppose we reach q_{accept} with no more input to read. Then we must have generated and popped precisely the input string. (If we accept, the string was derivable from S.)

Suppose we get as input a string s derivable from S. Then there exists some derivation for s, and some branch reaches q_{accept}.

[very informal] argument that P recognizes the same language as G.

Detail 1: how do we push strings?

push $s = s_1 s_2 \cdots s_n$, $s_i \in \Gamma$ as follows:



Detail 2: What does the transition function look like formally?

$\delta(q_{start}, \varepsilon, \varepsilon) = \{(q_{loop}, S\$)\}$ // shorthand for pushing \$, then S as above

$\delta(q_{loop}, \varepsilon, A) = \{(q_{loop}, w) \mid A \to w \text{ a rule in } R\}$

$\delta(q_{loop}, a, a) = \{(q_{loop}, \varepsilon)\}$

$\delta(q_{loop}, \varepsilon, \$) = \{(q_{accept}, \varepsilon)\}$

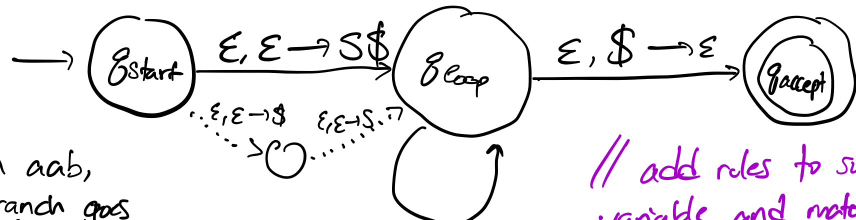
(δ maps to the null state $\emptyset$ for all other inputs.) ■

Sketch — Section 2.2 in Sipser for full details.
Need to know: just how to convert CFG to PDA.

Example: CFG → PDA.

Consider G: $S \to aTb \mid b$

$T \to Ta \mid \varepsilon$



// add rules to substitute each variable and match each terminal

on aab, one branch goes as follows:

state	stack
q_{start}	ε
q_{loop}	$S\$$
q_{loop}	$aTb\$$
q_{loop}	$Tb\$$
q_{loop}	$Tab\$$
q_{loop}	$ab\$$
q_{loop}	$b\$$ ...
q_{loop}	$\$$
q_{accept}	ε

$\varepsilon, S \to aTb$

$\varepsilon, S \to b$

$\varepsilon, T \to Ta$

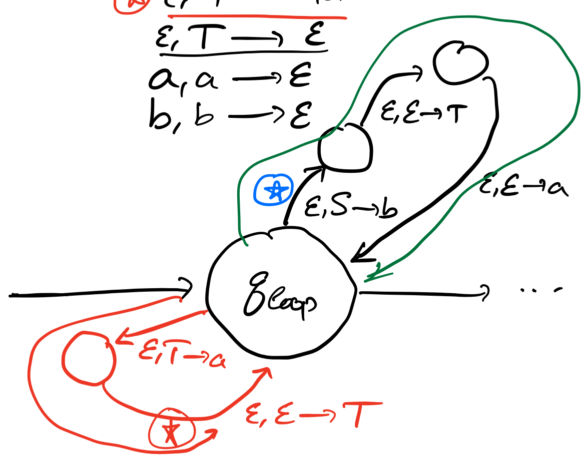
$\varepsilon, T \to \varepsilon$

$a, a \to \varepsilon$

$b, b \to \varepsilon$

pushing strings

substitute the starred edges for the gadgets below.



Example: aab. $S \Rightarrow aTb \Rightarrow aTab \Rightarrow aab.$

Next: Sketch PDA $\to$ CFG.

Non-context free languages.