Lecture 15

Ben Rosenberg

June 30, 2021

Note: All material covered today may be on the final exam, but only as extra credit.

Nondeterministic Pushdown automata

- Somewhat between finite automata and "everything else" (Turing machines)
- There is a theorem that converts between pushdown automata and context-free grammars
- Is somewhat like a stack?

PDAs are like a finite automaton but they have an extra tape that is called a **stack**.

Has an input $w_0 \in \Sigma^*$ on a finite input string. The *head* on the machine is read-only, right-only just like FSAs.

The machine looks at its current state and current symbol, and looks at the top symbol of the stack. The top symbol will be popped (removed) and a new string is going to be pushed onto the stack.

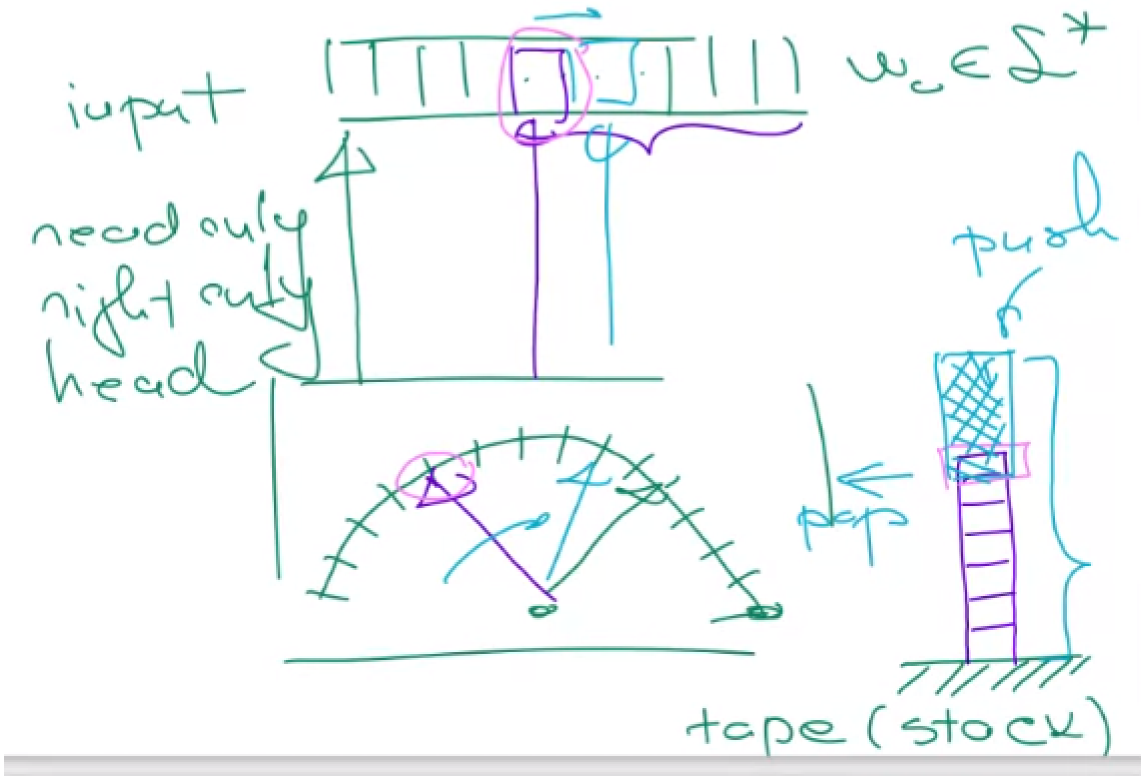


Figure 1: PDA illustration

Side note: NPDAs (nondeterministic pushdown automata) are NOT NECESSARILY EQUIVALENT to DPDAs (deterministic pushdown automata)

BUT: NPDAs *ARE* equivalent to CFGs (context-free grammars)

Definition 1: A **nondeterministic pushdown automaton** is a structure:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

such that:

- Q is the set of states
- Σ is the alphabet
- Γ is the tape alphabet (note: $\Sigma \cap \Gamma = \emptyset$; different from Turing machines!)
- δ is the transition function
- q_0 is the initial state
- F is the set of final (accept) states

Transition function

The transition function δ is a partial function again, defined as follows:

$$\delta : (Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})) \to \mathcal{P}(Q \times \Gamma^*)$$

The state is Q, the input symbol is in $\Sigma \cup \{\lambda\}$ (λ-moves are okay)

The top of stack symbol or λ is $\Gamma \cup \{\lambda\}$.

The target here is $\mathcal{P}(Q \times \Gamma^*)$.

Q is finite, Γ is finite, but $|\Gamma^*| = \aleph_0$ and $|Q \times \Gamma^*| = \aleph_0$ and $|\mathcal{P}(Q \times \Gamma^*)| > \aleph_0$

So, we have the restriction that the outputs are $x \in \mathcal{P}(Q \times \Gamma^*)$ and $|x| \leq \aleph_0$.

Our transition function is given in 5-tuples as follows:

$$[q, a, A, p, \alpha]$$

where:

- $q \in Q$ is the current state
- $a \in \Sigma$ is the current input symbol
- $A \in \Gamma$ is the current top of stack (gets "popped")
- $p \in Q$ is the next state
- $\alpha \in \Gamma^*$ is the next top of the stack (gets "pushed")

Configuration

The machine has in its configuration the following:

- state, q
- unprocessed input, w
- the *entire* stack content, σ

We write this as (q, w, σ).

Initial configuration:

$$(q_0, w_0, \lambda)$$

Here, we have q_0 as the initial state, w_0 as the entire input string, and λ as **an empty stack**.

Upon moving, we get the new configuration:

$$(p, x, \sigma_1)$$

by performing the calculation:

if $[q, a, T, p, \Pi] \in \delta$: $w = ax$, where $a \in \Sigma \cup \{\lambda\}$ and $\Pi \in \Gamma^*$.

Convention: Write stack as thing that sits on the ground, and add or subtrack by pushing and popping.

By convention, we put the **bottom on the left** and the **top on the right**.

So, σ had a bottom, and a top, $\sigma = \beta T$. After that, it became $\sigma_1 = \beta\Pi$.

$\sigma_1\sigma_1 \in \Gamma^*$.

M accepts w_0 if there exists a computation that takes M from (q_0, w_0, λ) to (f, λ, λ) where $f \in F$, f is a final state, and the input and stack are both *empty*. We start with an empty stack and end with an empty stack.

It must clear the input, clear the stack, and be in a final state.

Examples

Example: Something intrinsically context-free

1. $L_2 = \{a^n b^n | n \geq 0\}$

We can do this with PDAs because it has *memory* and can remember how many a's and b's it has pushed.

On an a, we push it to the stack. Then, on the b's, we check the b's against the a's and return when we have as many b's as a's.

$M = (Q, \Sigma, \Gamma, \delta, q, F)$

$Q = \{q\}$

$\Sigma = \{a, b\}$

$\Gamma = \{A\}$

$F = \{p\}$

$\delta :$

- $[q, a, \lambda, q, A]$ (push a)
- $[q, b, A, p, \lambda]$ (pop a)
- $[p, b, A, p, \lambda]$ (pop a)

This is **incorrect** because it misses λ which is a good string.

We can change this δ to be correct:

$\delta :$

- $[q, a, \lambda, q, A]$ (push)
- $[p, b, A, p, \lambda]$ (pop)
- $[q, \lambda, \lambda, p, \lambda]$ (jump at *any time*)

This last transition will not break the machine's functionality as any jump from `a` to `b` will result in either a nonempty stack at the end of the input string or a nonempty input string at the end of the stack, preventing it from accepting.

Algorithms

Algorithm $\textcircled{1}$:

Input: Finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ that accepts a language L

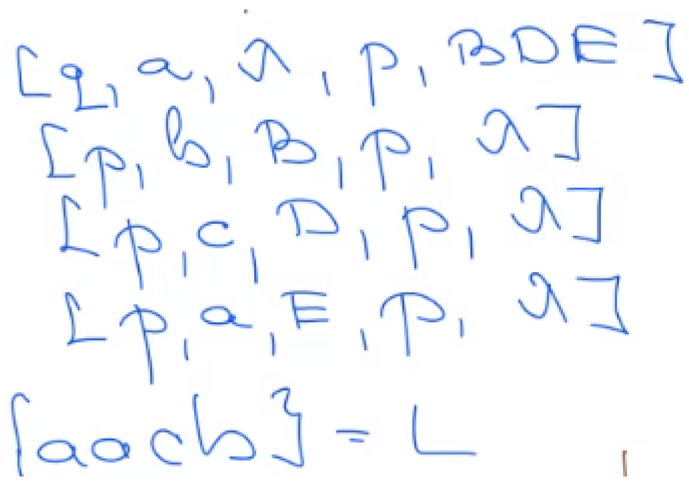
Output: Pushdown automaton $M_1 = (Q, \Sigma, \Gamma_1, \delta_1, q_1, F_1)$ that accepts L

Construction:

- ignore the stack
- $Q_1 = Q, \Gamma_1 = \emptyset, q_1 = q_0, F_1 = F$, and:
  - whenever: $[q, a, p] \in \delta$, with $q, p \in Q$ and $a \in \Sigma$
  - then: $[q, a, \lambda, p, \lambda] \in d_1$.

This automaton being simulated is *deterministic* (not that it matters because we can just convert it from NFA to DFA).

"How to fail"



$[q, a, \lambda, p, BDE]$
$[p, b, B, p, \lambda]$
$[p, c, D, p, \lambda]$
$[p, a, E, p, \lambda]$
$\{aacb\} = L$

Figure 2: Example PDA

Theorem (aside): A language is **context-free** if and only if it is accepted by some pushdown automaton.

Another example: Palindromes

$M = (Q, \Sigma, \Gamma, \delta, q, F)$

$Q = \{q\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \{\}$

$F = \{\}$

idea: put the first part onto the stack, and pair it off with the second part and pop it. The question is: when do we reach the middle?

What we will do is have the automaton *guess* the middle.

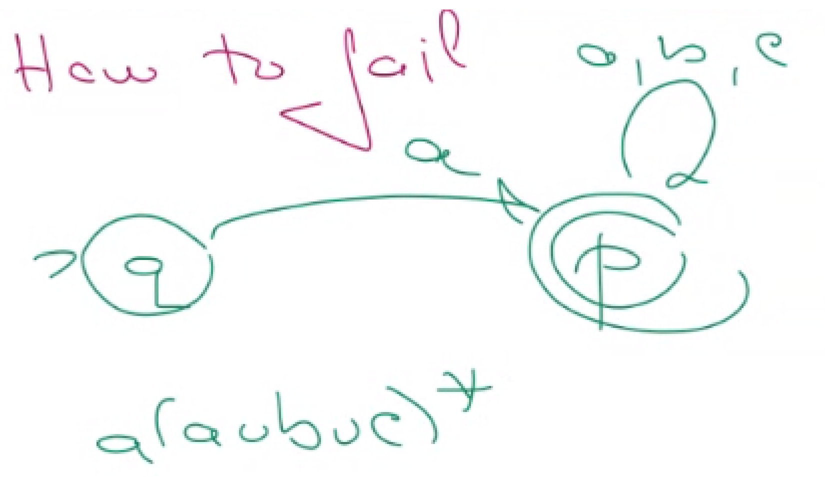


Figure 3: Incorrect FSA representation of the above PDA

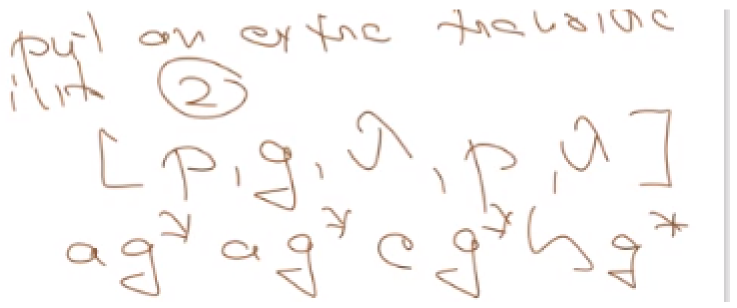


Figure 4: What would happen if we put another transition into our example PDA?

THeorem 4 (Aside): The intersection of a context-free language with a regular language is context-free.

δ :

- $[q, a, \lambda, q, A]$ (push)
- $[q, b, \lambda, q, B]$
- $[q, c, \lambda, q, K]$
- $[p, a, A, p, \lambda]$ (pop)
- $[p, b, B, p, \lambda]$
- $[p, c, K, p, \lambda]$
- $[q, \lambda, \lambda, p, \lambda]$ (λ-transition from q to p for even strings)
- $[q, (a \cup b \cup c), \lambda, p, \lambda]$ (guess the middle for an *odd* case [this is really 3 transitions, not 1])

Last example: Set of exactly those strings where the number of a's is equal to the number of b's.

Whichever you see first, you push it onto the stack. Then, an a comes along – push it. If it is a b, then you pair it off. Whichever you have a surplus off, you keep it on the stack. When the other comes in, you pair it off. Repeat this whenever the stack is empty.

Issue: don't know when the stack is empty. So, we put a sentinel at the beginning that lets us detect when the stack is empty.