Goal: Define machine that accepts context-free languages (CFLs), useful in designing parsers from context-free grammars (CFGs).

Three main models of computation we study (plus variants):

1. **Finite automata** (DFAs, NFA's, ε-NFA's)

   - Transition function: δ(q₀, x)
   - Configuration: (qᵢ, xᵢ, ..., xₙ)
   - Initial state: q₀
   - Accepting state: q_f

2. **Push-down automaton** = (ε-nondeterministic) finite automaton with a stack for extra memory

   - Input: x
   - Finite state control
   - Accept (x ∈ L)
   - Reject (x ∉ L)
   - Stack: Z₀, top of stack Z
   - Transition: δ(q₀, x) = q₁
   - Push: P and push Zₓ

3. **Turing machine** = finite state controller with read/write access to the input tape (of unbounded length)

   - Input tape: x₁, x₂, ..., xₙ
Pushdown Automata (PDA)

* the biggest problem with finite automata was insufficient memory
  physically unrealistic but disagrees with our intuition for ideal computation
  (we should be able to decide \( L = \{ 0^n1^n \mid n \geq 0 \} \))

- let's add memory
  - **Stack**
    - can push things in, or **pop things out**
    - but only off the **top** (unlike, say, **RAM**)
    - **unbounded**
  - **Control** (**E-NFA**)

  **Transitions** Depend on
  \[
  \begin{aligned}
  \text{Effect} & \quad \text{· Input character} \\
  & \quad \text{· Current state} \\
  & \quad \text{· Stack top} \\
  & \quad \text{· New state} \\
  & \quad \text{· Replace stack top by some string}
  \end{aligned}
  \]

**Example** \( L = \{ 0^n1^n \mid n \geq 0 \} \)

**Idea**

* While Input = 0
  - Push A on stack
  * While Input = 1
  - Pop A from stack

**Accept** when input is over, no A's left on the stack

---

![Diagram of PDA transitions](attachment:diagram.png)

**Example** (state, remaining input, stack)

\[
\begin{align*}
(p, 0011, 20) & \rightarrow (p, 011, A20) \\
(p, 11, A20) & \rightarrow (q, 1, A20) \\
(q, \varepsilon, Z_0) & \rightarrow (q, \varepsilon, \varepsilon) \\
(r, \varepsilon, \varepsilon) & \rightarrow \text{ACCEPT} \\
\end{align*}
\]
Definition: A pushdown automaton (PDA) is a tuple
\[ P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F) \]
where:
- \( Q \) = finite set of states
- \( \Sigma \) = finite input alphabet
- \( \Gamma \) = finite stack alphabet
- \( q_0 \in Q \) start state
- \( Z_0 \in \Gamma \) start symbol
- \( F \subseteq Q \) accepting states

Transition function \( \delta \)
\[ \delta(q, a, X) = (p, \lambda) \]
where \( q, p \in Q, a \in \Sigma, X \in \Gamma, \lambda \in \Gamma^* \)

means In state \( q \), if input being read is \( a \) and stack top is \( X \),
then go to state \( p \), replace \( X \) by \( a \) on stack

Example: Our previous PDA, formally
\( \Sigma = \{ 0, 1 \}, \Gamma = \{ \epsilon, A, 0, 3 \}, Q = \{ p, q, r \}, F = \{ r \} \)

Transitions:
\[ \delta(p, 0, Z_0) = (p, AZ_0) \quad \text{on input } 0, \]
\[ \delta(p, 0, A) = (p, AA) \quad \text{add A to stack} \]
\[ \delta(p, 1, A) = (q, \epsilon) \quad \text{on input 1, move to q} \]
\[ \delta(q, 1, A) = (q, \epsilon) \quad \text{on input 1, consume A} \]
\[ \delta(q, \epsilon, Z_0) = (r, \epsilon) \quad \text{hit stack bottom} \]

Deterministic PDA: \( \delta: Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^* \)
-the above example can be made deterministic

A general PDA is nondeterministic: \( \delta: Q \times (\Sigma \cup \{ \epsilon \}) \times \Gamma \rightarrow Q \times \Gamma^* \)
-control is like an \( \epsilon \)-NFA

Powers

Regular languages = \text{POWER}(\epsilon\text{-NFA}) < \text{POWER}(\text{DPDA}) < \text{POWER}(\text{PDA}) = \text{CFL}

Why? since can ignore the stack,
\text{POWER}(DFA) \subseteq \text{POWER}(\text{DPDA})
since \( \{ 0^n 1^m | n \geq 0 \} \) is not regular,
\text{POWER}(DFA) \neq \text{POWER}(\text{DPDA})

\text{Unlike for finite-state automata}

Example: \( L = \{ x \epsilon \{ a, b \}^* | x \epsilon \{ 0, 1 \}^* \} \)
is accepted by a PDA but by no DPDA; it can't guess where \( x \) ends and \( x \epsilon \) begins.
\( \{ x2x^2 | x \epsilon \{ 0, 1 \}^* \} \) has DPDA.
Example \[ L = \{ w w^R \mid w \in \{a, b\}^* \} \]

Goal: Construct PDA \( M \) for \( L \)

Idea: In state \( q_0 \), push \( x \) onto the stack, one by one

- Guess midpoint and \( \varepsilon \)-move to \( q_1 \)
- State \( q_1 \), match \( w^R \) with stack

Observe: Stack will pop \( w \) in reverse order, allowing matching \( w \) with \( w^R \)

- Guess midpoint
  - \( \varepsilon, a/A \)
  - \( \varepsilon, b/B \)
  - \( \varepsilon, \varepsilon/\varepsilon \)

- On input \( a \)
  - Add \( A \) to stack

Example: On input \( aabbaa \),

\[ (q_0, aabbaa, Z_0) \]

\[ \Rightarrow (q_0, aabbaa, A Z_0) \]

\[ \Rightarrow (q_1, aabbaa, A Z_0) \]

\[ \Rightarrow (q_0, baa, A A Z_0) \]

\[ \Rightarrow (q_1, baa, BAAZ_0) \]

\[ \Rightarrow (q_1, a, AA Z_0) \]

\[ \Rightarrow (q_1, a, A Z_0) \]

\[ \Rightarrow (q_1, \varepsilon, Z_0) \]

\[ \Rightarrow (q_2, \varepsilon, \varepsilon) \]

Accept

Of course other execution traces are possible since \( M \) is nondeterministic.
Example: \( L = \{ x \in \{0,1\}^* : x \text{ has an equal number of 0s and 1s} \} \)

Idea: Make sure the stack is always either

\( A^n Z^o \) or \( B^n Z^o \)

where \( n \) is the number of extra 0s that have been seen

(ie in the first case, or extra 1s in the second case)

Exercise: \( L = \{ x \in \{0,1\}^* : x \text{ has twice as many 0s as 1s} \} \)

CFG: \( S \rightarrow E | SOS | SOS | SISOS | SISOSOS \)

PDA?

Main Theorem for PDAs:

For any language \( L \subseteq \Sigma^* \),

there exists a context-free grammar that generates \( L \)

if and only if

there exists a push-down automaton that accepts \( L \) (ie., recognizes \( L \))

Thus \( \text{POWER}(\text{PDAs}) = \text{POWER}(\text{CFGs}) = \text{context-free languages} \)

(analogous to \( \text{POWER}(\text{DFAs}) = \text{POWER}(\text{REGs}) = \text{regular languages} \))

Two directions for the proof:

1. Turn a CFG \( G \) into a PDA \( M \)
2. Turn a PDA into a CFG

Two main ingredients in the proof:

- Chomsky normal form for CFGs
- Equivalence between empty stack acceptance and final state acceptance
Modes of PDA Acceptance

By Final State: when input is over, in final state
final state language \( L(M) \)
\[
L(M) = \{ x \in \Sigma^* \mid (q_0, x, z_0) \xrightarrow{*} (p, \varepsilon, \varepsilon) \text{ where } p \in F, \varepsilon \in \Gamma^* \}
\]

By Empty Stack: when input is over, stack is empty
empty-stack language \( N(M) \)
\[
N(M) = \{ x \in \Sigma^* \mid (q_0, x, z_0) \xrightarrow{*} (p, \varepsilon, \varepsilon) \text{ for some } p \in Q \}
\]

Either mode gives the same power:

**Theorem**: \( L = L(M_1) \) for some PDA \( M_1 \)
if and only if
\( L = N(M_2) \) for some PDA \( M_2 \).