Pushdown Automata and CFG Equivalence

Stack memory.
Context-free recognition.

PDA design • Stack traces • CFG equivalence

Learning outcomes

Model

Describe a pushdown automaton (PDA) and its stack operations.

Design and trace

Construct PDAs for matched blocks and balanced parentheses; trace configurations.

Prove equivalence

Convert a CFG into an NPDA and explain the reverse construction.

Why a stack?

Finite automata

A DFA has finite memory and cannot recognize $\{a^n b^n : n \geq 0\}$.

Pushdown automata

A PDA adds an unbounded last-in, first-out stack. Push for each a; pop for each b.

Expressive power

Nondeterministic PDAs recognize exactly the context-free languages. Deterministic PDAs recognize a proper subclass.

Formal definition of an NPDA

Seven-tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. Q: states; Σ: input alphabet; Γ: stack alphabet.

Initial and final components

q_0 is the initial state; Z_0 is the initial stack symbol; F is the set of accepting states.

Transition function

$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to$ finite subsets of $Q \times \Gamma^*$. A move replaces the top symbol with a stack string.

Transitions and stack operations

Notation

a, X → γ means consume a (or ε), remove top X and replace it by γ; leftmost symbol of γ is the new top.

Push and pop

a, X → AX pushes A above X. b, A → ε pops A.

No stack change

ε, X → X changes state without reading input or modifying the stack.

Configurations and acceptance

Instantaneous description

(q, w, α): current state q, unread input w, stack α (top at left).

Final-state acceptance

$(q_0,w,Z_0) \vdash^* (f,\varepsilon,\alpha)$, where $f \in F$. The stack need not be empty.

Empty-stack acceptance

$(q_0,w,Z_0) \vdash^* (q,\varepsilon,\varepsilon)$. Both modes require full input consumption and are equivalent for NPDAs.

Example 1: matching a and b

Language

L = {$a^n b^n$: n ≥ 0}. Use states p (push), r (pop), f (final), with initial stack Z.

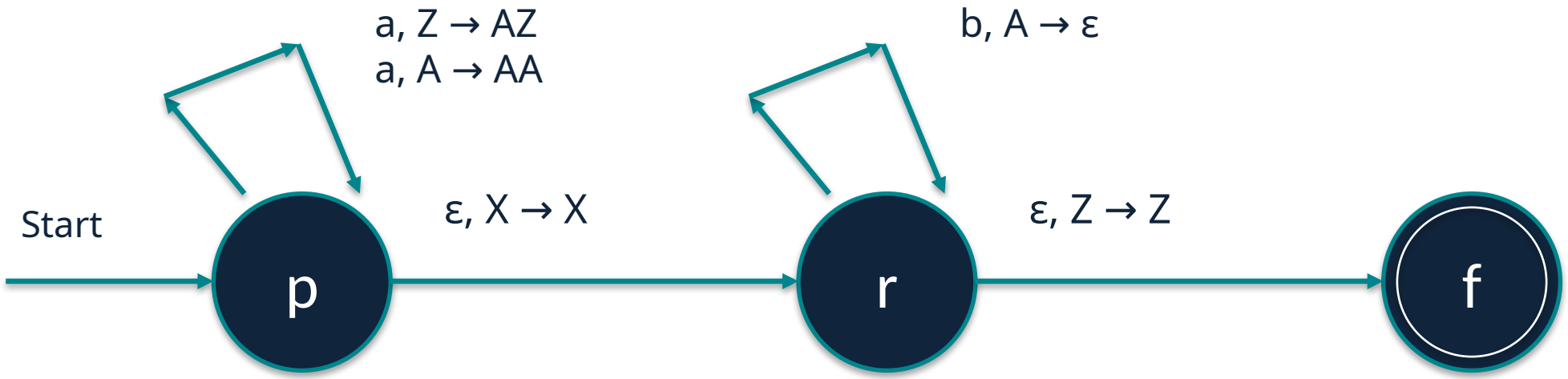
Push phase

In p: a,Z → AZ; a,A → AA. Switch p → r on ε,X → X for X ∈ {A,Z}.

Pop and accept

In r: b,A → ε. Move r → f on ε,Z → Z. No other moves. Only f is accepting.

Example 1: state diagram



X ∈ {A, Z}. Accept at f only after consuming the entire input.

Example 1: trace aabb

Push twice

$(p,aabb,Z) \vdash (p,abb,AZ) \vdash (p,bb,AAZ)$.

Switch and pop twice

$\vdash (r,bb,AAZ) \vdash (r,b,AZ) \vdash (r,\varepsilon,Z)$.

Accept

$\vdash (f,\varepsilon,Z)$. By contrast, abb leaves an unmatched b; aab leaves an unmatched A.

Example 2: balanced parentheses

Design

One working state p and final state f; stack alphabet {Z,A}. Push A for each opening parenthesis.

Transitions

In p: (,Z → AZ; (,A → AA;),A → ε. Move p → f on ε,Z → Z.

Trace (())

Stack sequence: Z → AZ → AAZ → AZ → Z. The string)(fails immediately because no A is available.

Determinism versus nondeterminism

DPDA

At most one move per state, input symbol and stack top; an ε-move excludes input-consuming moves for the same state and top.

NPDA

Can branch among alternatives. Acceptance requires at least one accepting computation.

Example

For $\{ww^R : w \in \{0,1\}^*\}$, an NPDA guesses the midpoint, then matches the second half against the stack.

CFG to NPDA: general construction

Initialize

Use one working state p, stack alphabet $V \cup \Sigma$, initial stack S. Accept by empty stack.

Expand variables

For each production $A \to \beta$, add $\delta(p,\varepsilon,A)$ containing (p,β). This simulates a leftmost derivation.

Match terminals

For each $a \in \Sigma$, add $\delta(p,a,a)$ containing (p,ε). Accept when input and stack are both empty.

CFG to NPDA: worked example

Grammar

S → aSb | ε. Initial stack S. Expansion moves: ε,S → aSb and ε,S → ε.

Terminal matching

Add a,a → ε and b,b → ε. The stack top is always written at the left.

Trace aabb

(p,aabb,S) ⊢ (p,aabb,aSb) ⊢ (p,abb,Sb) ⊢ (p,abb,aSbb) ⊢ (p,bb,Sbb).

CFG to NPDA: finish the trace

Choose ε-production

(p,bb,Sbb) ⊢ (p,bb,bb). The variable S disappears.

Match remaining terminals

⊢ (p,b,b) ⊢ (p,ε,ε). Accept by empty stack.

Why it works

Stack expansions simulate leftmost derivations; matching terminals verifies that the generated word equals the input.

NPDA to CFG: construction idea

Prepare

Use an NPDA accepting by empty stack; transitions replace a top symbol with zero or more symbols.

Variables

[pXq] generates strings taking the PDA from p with X on top to q after removing X, without disturbing the stack below.

Start rule

For each $q \in Q$ add $S \to [q_0 Z_0 q]$. These cover every possible state at empty-stack acceptance.

NPDA to CFG: production rules

Pop move

If $(r,\varepsilon) \in \delta(p,a,X)$, add $[pXr] \to a$ (a may be ε).

Push two symbols

If $(r,YZ) \in \delta(p,a,X)$, add $[pXq] \to a[rYt][tZq]$ for every $t,q \in Q$.

General replacement

For k pushed symbols, concatenate k variables with all intermediate states; for one symbol use $[pXq] \to a[rYq]$.

Reverse conversion: small example

PDA accepting by empty stack

One state p; initial stack Z. Moves: a,Z → AZ; b,A → ε; ε,Z → ε.

Generated productions

Let U=[pZp], V=[pAp]. Then S → U; U → aVU | ε; V → b.

Simplified grammar

S → abS | ε, generating (ab)*. Each a pushes A; the only way to remove A is to read b.

Exercises and recap

Exercise 1

Trace aaabbb in Example 1. Answer: push three A symbols, switch, pop three, then accept with Z.

Exercise 2

Convert S → 0S1 | ε to an NPDA. Answer: ε,S → 0S1 or ε; 0,0 → ε; 1,1 → ε; empty stack acceptance.

Takeaway

CFG ⇄ NPDA. Both characterize CFLs; the stack provides memory for nested structures. Next: CFL closure and pumping lemma.