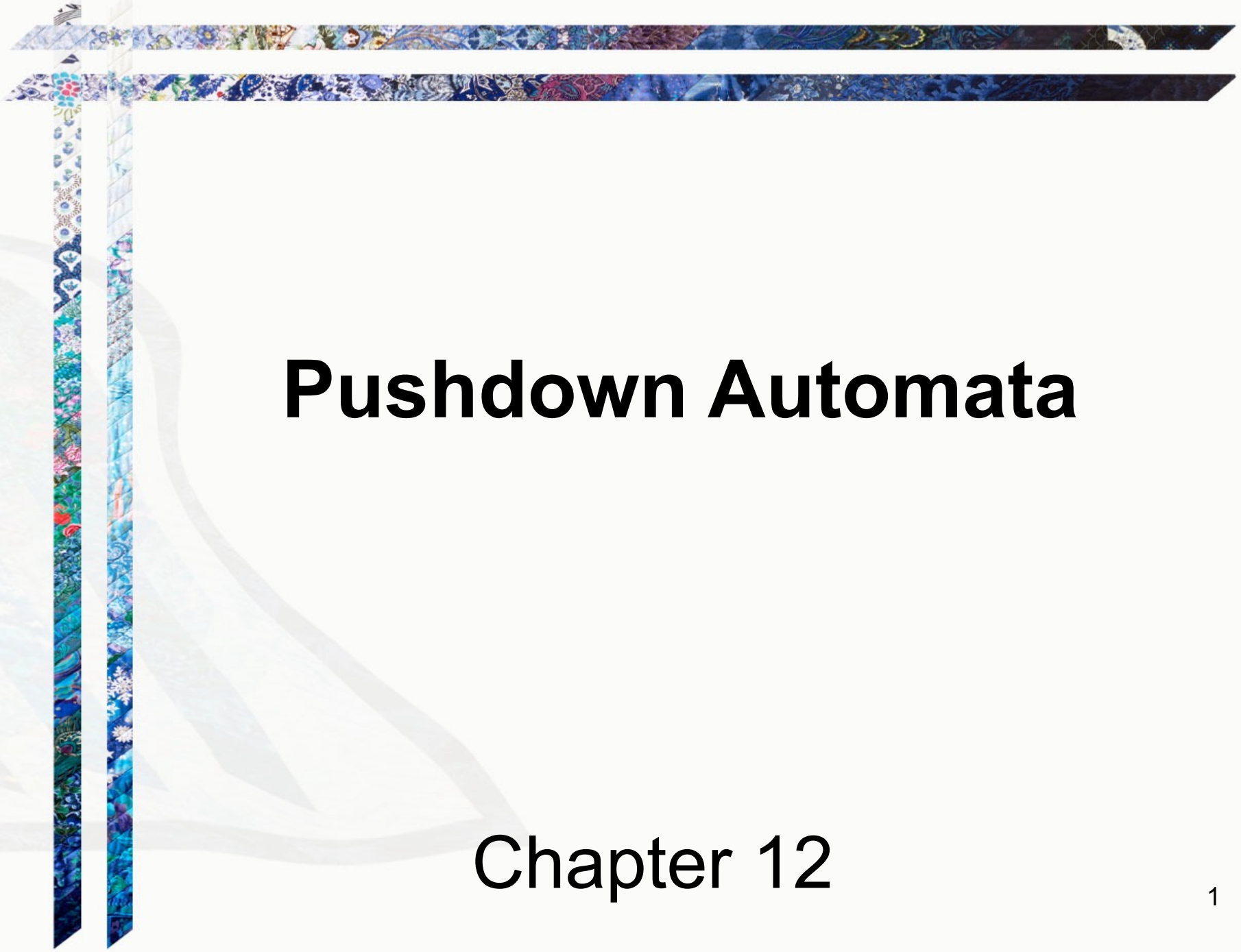


Pushdown Automata

Chapter 12

Recognizing Context-Free Languages

We need a device similar to an FSM except that it needs more power.

The insight: Precisely what it needs is a stack, which gives it an unlimited amount of memory with a restricted structure.

Example: Bal (the balanced parentheses language)

((()))

Before Defining PDA

- It's defined as nondeterministic
  - DFSM = NDFSM = Regular language
  - NDPDA = Context-free language > DPDA

  - In contrast to regular languages, where nondeterminism is a convenient design tool.
  - Some context-free languages do not have equivalent DPDA to recognize them.

Before Defining PDA

Alternative **equivalent** PDA definitions

- Our version is sometimes referred to as "Generalized extended PDA" (GPDA), a PDA which writes an entire string to the stack or removes an entire string from the stack in one step.
  - In some definition, *M* may pop only a single symbol but it may push any number of them.
  - In some definition, *M* may pop and push only a single symbol.
- In our version, *M* accepts *w* only if, when it finishes reading w, it is **in an accepting state** and its **stack is empty**.
  - **Finite state acceptance**: when it finishes reading *w,* it is in an accepting state, regardless of the content of the stack.
  - **Empty stack acceptance**: when it finishes reading *w*, the stack is empty, regardless of the state *M* is in.
- We do not use "bottom of stack marker" but some do.
- All of these are provably **equivalent** as they recognize the same *L.*

Note: JFLAP uses a stack marker, either finite state or empty stack acceptance. So, the examples in the book may not run well in JFLAP.

Before Defining PDA

- Alternative **non-equivalent** variants
  - Variation 1 (Tag systems or Post machines): FSM + a first-in, first-out (FIFO) queue (instead of a stack)
  - Variation 2: FSM + two stacks

- Both are more powerful, equivalent to Turing machines.

Definition of a (Nondeterministic) Pushdown Automaton

$M = (K, \Sigma, \Gamma, \Delta, s, A)$, where:

K is a finite set of states

Σ is the input alphabet

Γ is the stack alphabet

$s \in K$ is the initial state

$A \subseteq K$ is the set of accepting states, and

Δ is the transition relation. It is a finite subset of

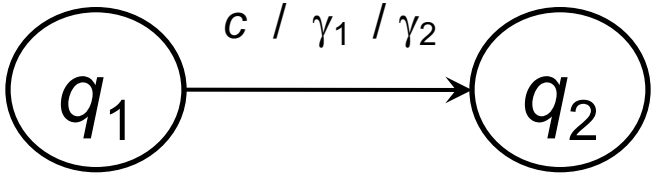
$$\underbrace{(K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*)}_{\text{state, input or } \varepsilon, \text{ string of symbols to pop from top of stack}} \times \underbrace{(K \times \Gamma^*)}_{\text{state, string of symbols to push on top of stack}}$$

Transition

$(K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*) \times (K \times \Gamma^*)$

K	$\Sigma \cup \{\varepsilon\}$	Γ^*	K	Γ^*
state	input or ε	string of symbols to pop from top of stack	state	string of symbols to push on top of stack

$((q_1, c, \gamma_1), (q_2, \gamma_2))$



- If c matches the input and γ_1 matches the current top of the stack, the transition From q_1 to q_2 can be taken.
- Then, c will be removed from the input, γ_1 will be popped from the stack, and γ_2 will be pushed onto it.
- M cannot peek at the top of the stack without popping
- If $c = \varepsilon$, the transition can be taken without consuming any input
- If $\gamma_1 = \varepsilon$, the transition can be taken without checking the stack or popping anything. **Note: it's not saying "the stack is empty".**
- If $\gamma_2 = \varepsilon$, nothing is pushed onto the stack when the transition is taken.

Configuration

To describe the execution of machine *M* on input *w*, we need a few definitions.

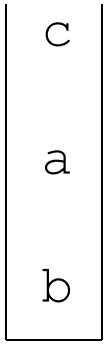
A configuration of *M* is an element of $K \times \Sigma^* \times \Gamma^*$.

- It captures the three things that decide *M*'s future behavior:
  - Current state
  - Input that is still left to read
  - Contents of its stack

- It provides a "snapshot" of the system at a particular execution/processing step.

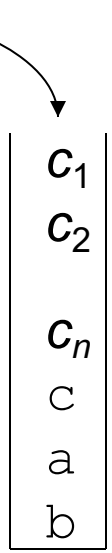
The ***initial configuration*** is (s, w, ε)

Manipulating the Stack



will be written as `cab`

If $c_1c_2\ldots c_n$ is pushed onto the stack: rightmost first



$c_1c_2\ldots c_n$`cab`

The Yields Relations

During execution, when a state transition occurs, the system moves from one configuration to another.

Let c be any element of $\Sigma \cup \{\varepsilon\}$,
Let γ_1, γ_2 and γ be any elements of Γ^*, and
Let w be any element of Σ^*.

Then: $(q_1, cw, \gamma_1\gamma) \vdash (q_2, w, \gamma_2\gamma)$ iff $((q_1, c, \gamma_1), (q_2, \gamma_2)) \in \Delta$.

After the transition, state q_1 -> q_2 , remaining string cw -> w, stack content $\gamma_1\gamma$ -> $\gamma_2\gamma$

Let $\vdash^*$ be the reflexive, transitive closure of $\vdash$
C_1 ***yields*** configuration C_2 iff $C_1 \vdash^* C_2$

Nondeterminism

If M is in some configuration (q_1, s, γ) it is possible that:

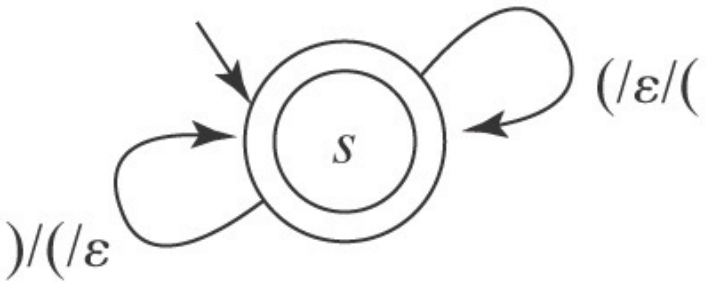
- Δ contains exactly one transition that matches.
- Δ contains more than one transition that matches.
- Δ contains no transition that matches.

Accepting

- *M* accepts a string *w* iff there exists *some path* that accepts it.
  - Recall a path is a maximal sequence of execution/processing steps (described by configurations) from the start
- Predefined accepting conditions: (1) all symbols in w have been processed/consumed. (2) in an accepting state. (3) stack is empty.

  More formally, accepting configuration for PDA: $(q, \varepsilon, \varepsilon)$ where $q \in A$
- *M* halts upon acceptance. Recall if one path halts and accepts, M halts and accepts. Other paths may:
  - Read all the input and halt in a nonaccepting state (reject)
  - Read all the input and **halt in an accepting state** with the **stack not empty** (reject)
  - Reach a dead end where no more input can be read (reject)
  - Loop forever and never finish reading the input (infinite path)
- The ***language accepted by*** *M*, denoted $L(M)$, is the set of all strings accepted by *M*.
- *M* rejects a string *w* iff all paths reject it.
- It is possible that, on input $w \notin L(M)$, *M* neither accepts nor rejects. In that case, no path accepts and some path does not reject.

A PDA for Balanced Parentheses



$M = (K, \Sigma, \Gamma, \Delta, s, A)$, where:

$K = \{s\}$ the states

$\Sigma = \{(,)\}$ the input alphabet

$\Gamma = \{(\}$ the stack alphabet

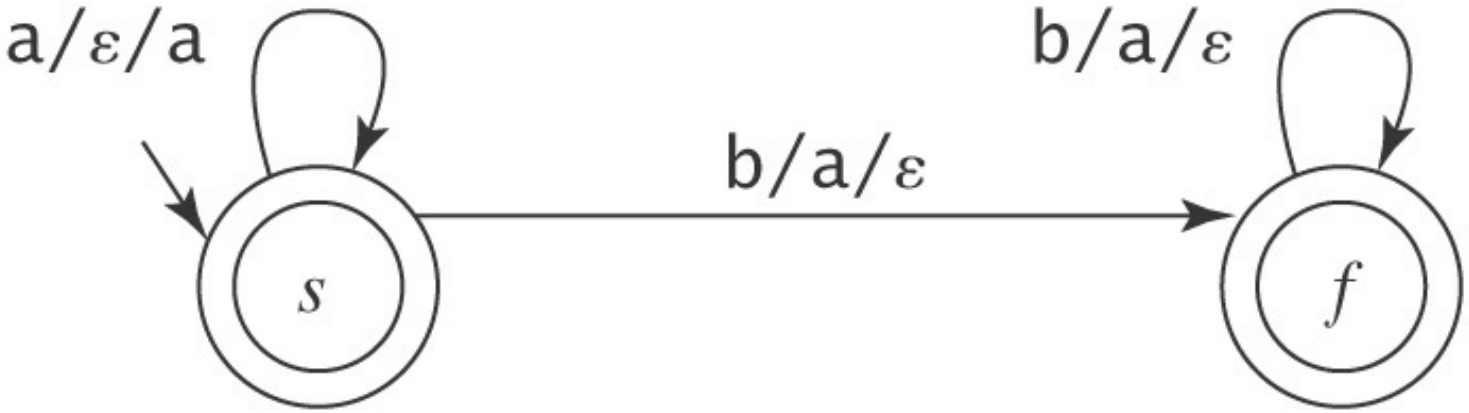
$A = \{s\}$

Δ contains:

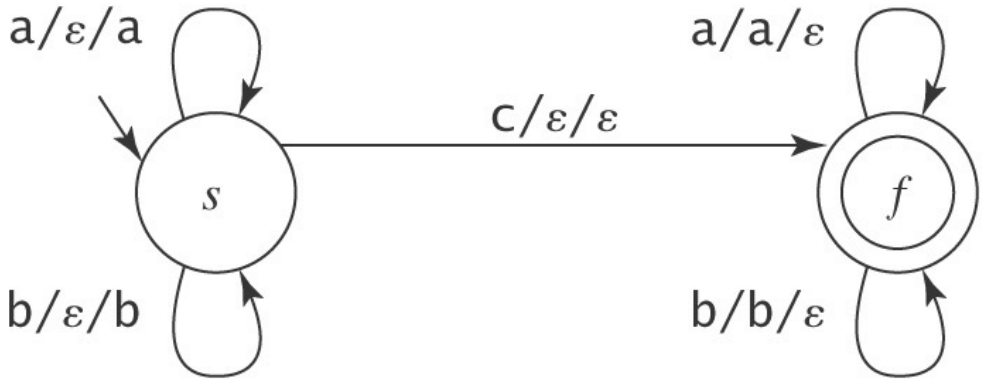
($(s, (, \varepsilon), (s, ( )$)

($(s,), ( ), (s, \varepsilon)$)

A PDA for $A^nB^n = \{a^m b^n : n \geq 0\}$



A PDA for $\{wcw^R: w \in \{a, b\}^*\}$

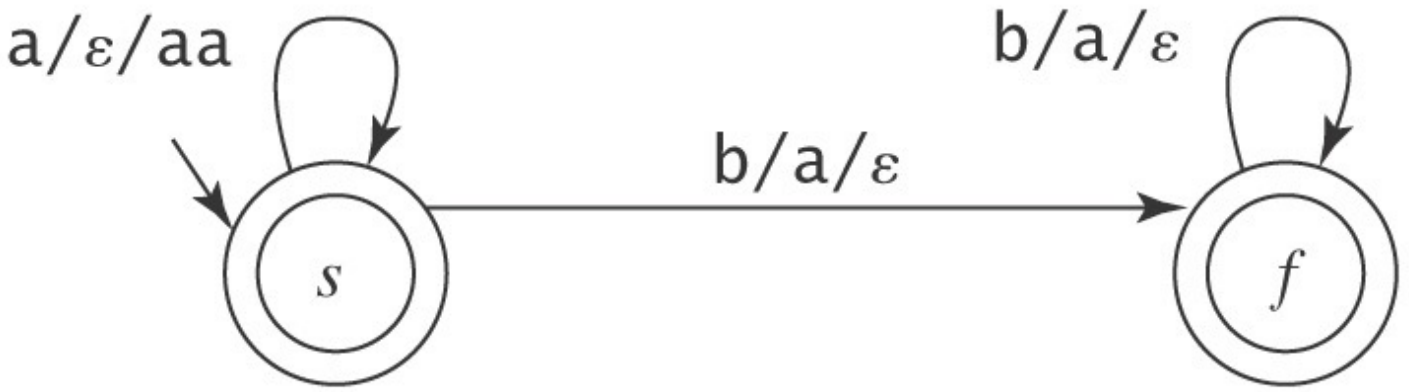


$M = (K, \Sigma, \Gamma, \Delta, s, A)$, where:

$K = \{s, f\}$	the states
$\Sigma = \{a, b, c\}$	the input alphabet
$\Gamma = \{a, b\}$	the stack alphabet
$A = \{f\}$	the accepting states

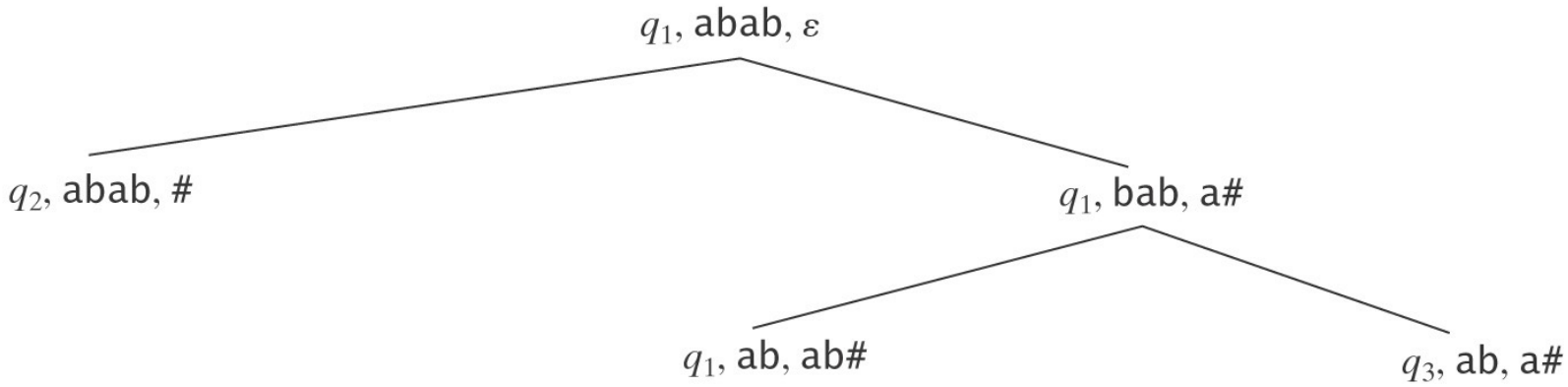
Δ contains: $((s, a, \varepsilon), (s, a))$
$((s, b, \varepsilon), (s, b))$
$((s, c, \varepsilon), (f, \varepsilon))$
$((f, a, a), (f, \varepsilon))$
$((f, b, b), (f, \varepsilon))$

A PDA for $\{a^n b^{2n}: n \geq 0\}$



Exploiting Nondeterminism

- A PDA *M* is ***deterministic*** iff:
  - Δ_M contains no pairs of transitions that compete with each other, and
  - whenever *M* is in an accepting configuration it has no available moves.
  - Unfortunately, unlike FSMs, there exist NDPDA s for which no equivalent DPDA exists.
- **Previous examples are DPDA**, where each machine followed only a single computational path.
- But many useful PDAs are not deterministic, where from a single configuration there exist multiple competing moves.
  - Easiest way to envision the operation of a NDPDA is as a tree

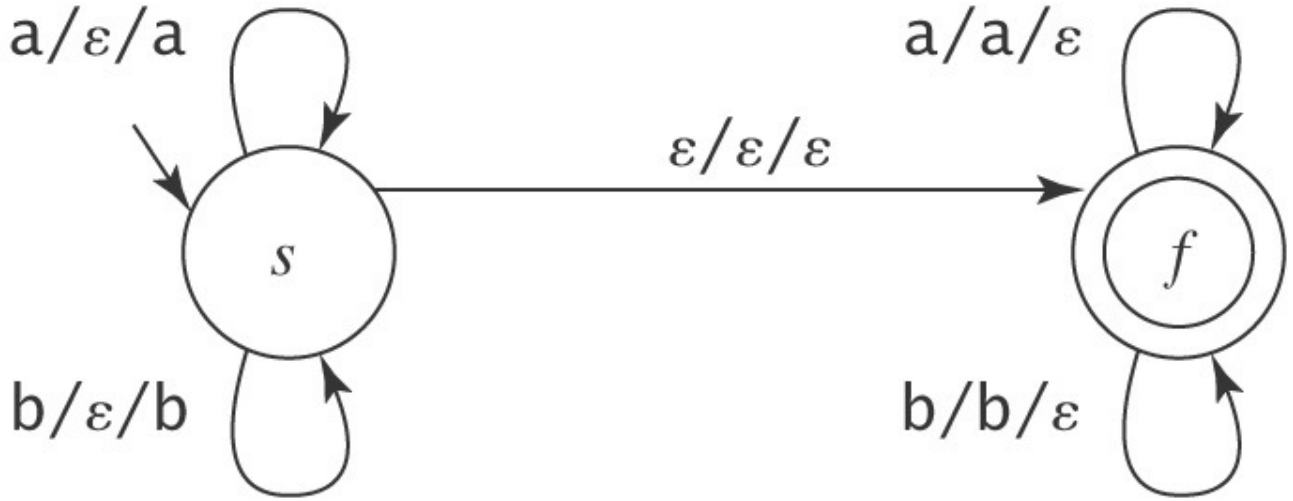


A PDA for PalEven ={ww^R: $w \in$ {a, b}*}

$S \rightarrow \varepsilon$
$S \rightarrow aSa$
$S \rightarrow bSb$

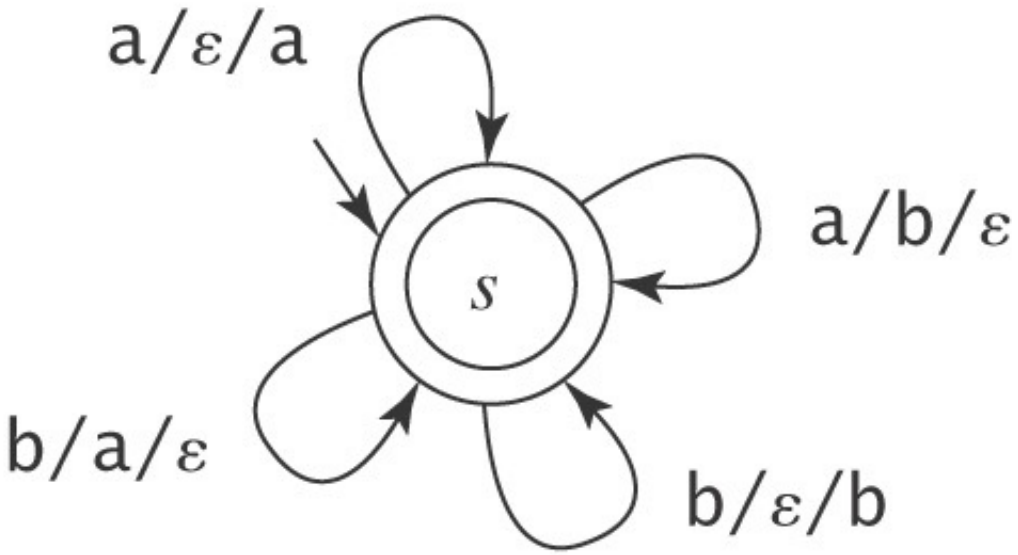
Even length palindromes

A PDA:



A PDA for $\{w \in \{a, b\}^* : \#_a(w) = \#_b(w)\}$

Equal numbers of a's and b's



evidence of nondeterminism?

PDA for $A^nB^nC^n$?

Consider $A^nB^nC^n = \{a^n b^n c^n : n \geq 0\}$.

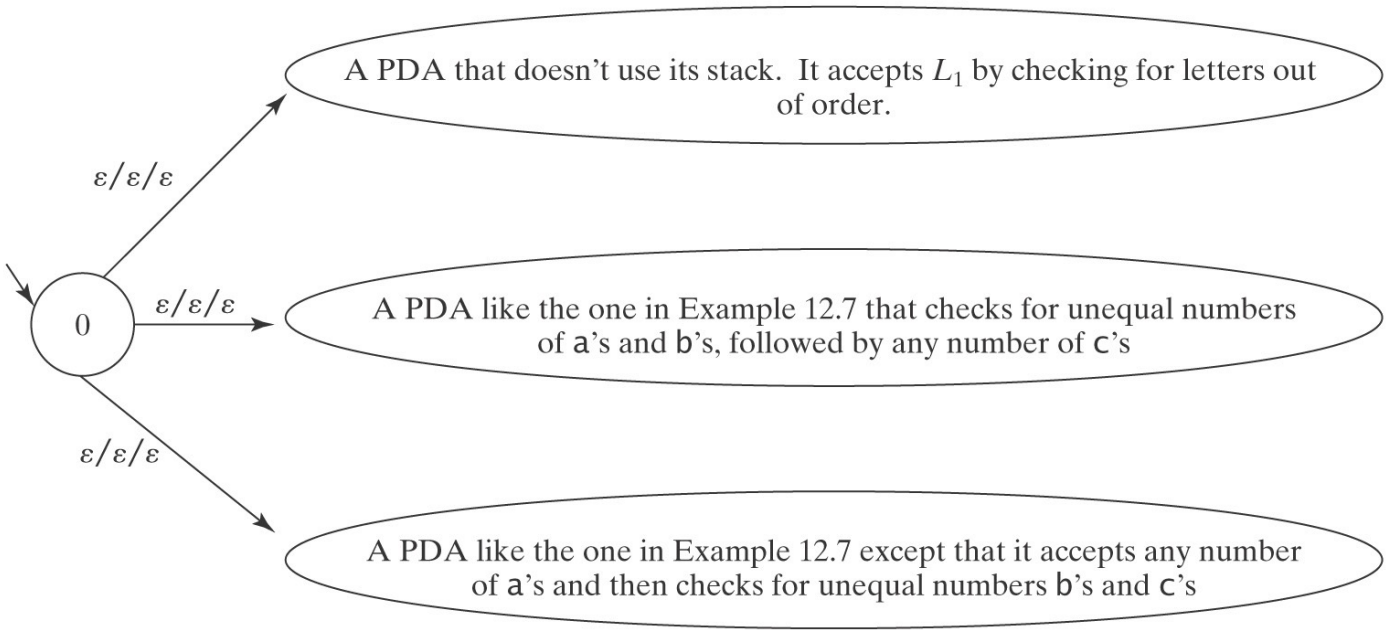
PDA for it?

Now consider $L = \neg A^nB^nC^n$. L is the union of two languages:

1. $\{w \in \{a, b, c\}^* :$ the letters are out of order$\}$, and

2. $\{a^i b^j c^k : i, j, k \geq 0$ and $(i \neq j$ or $j \neq k)\}$ (in other words, unequal numbers of a's, b's, and c's).

PDA for $\neg A^n B^n C^n$



Example 12.7 is in the following slides (grayed out)

Chapter 11 slide 35: unequal a's and b's

Are the Context-Free Languages Closed Under Complement?

$\neg A^nB^nC^n$ is context free.

If the CF languages were closed under complement, then

$$\neg\neg A^nB^nC^n = A^nB^nC^n$$

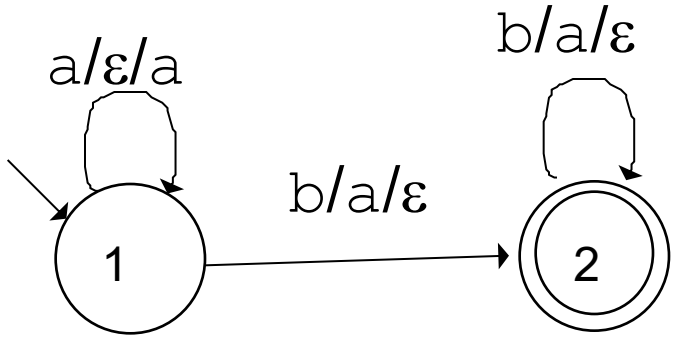
would also be context-free.

But we will prove that it is not.

More on Nondeterminism
Accepting Mismatches

$L = \{a^m b^n : m \neq n; m, n > 0\}$

Start with the case where $n = m$:



Note: m, n > 0. thus ε is not in ¬ L, and
state 1 is not double circled.

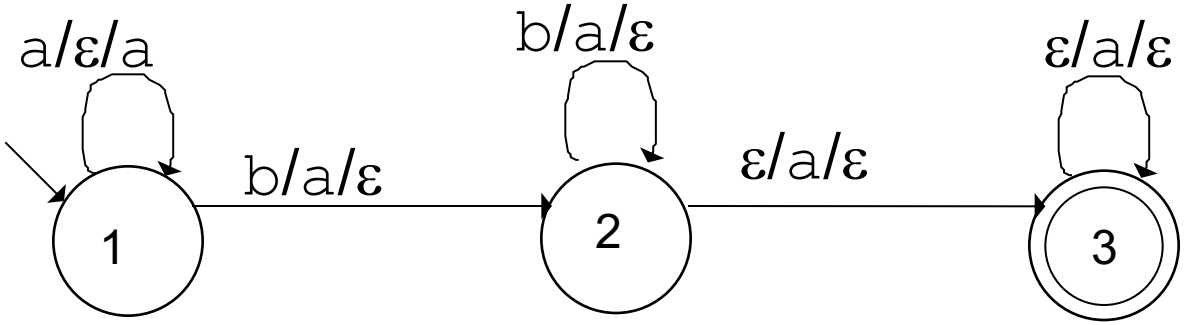
Hard to build a machine that looks for something negative, like ≠
Idea: break L into two sublanguages:

$\{a^m b^n : 0 < n < m\}$ and $\{a^m b^n : 0 < m < n\}$

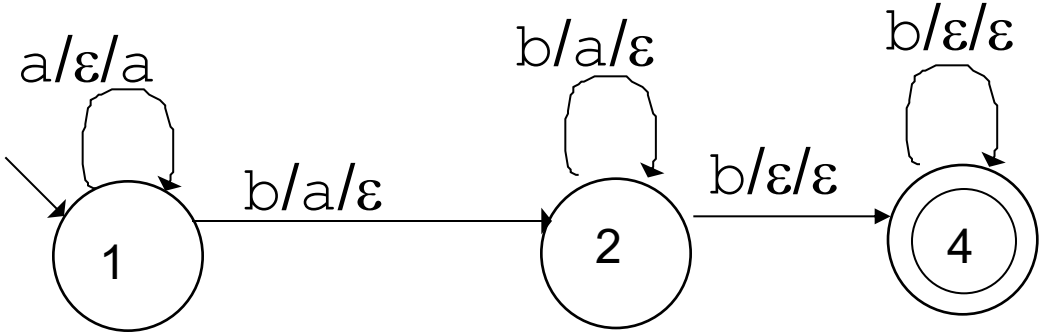
- If stack and input are empty, halt and reject.
- If input is empty but stack is not ($m > n$) (accept)
- If stack is empty but input is not ($m < n$) (accept)

More on Nondeterminism Accepting Mismatches

- If input is empty but stack is not ($m > n$) (accept):

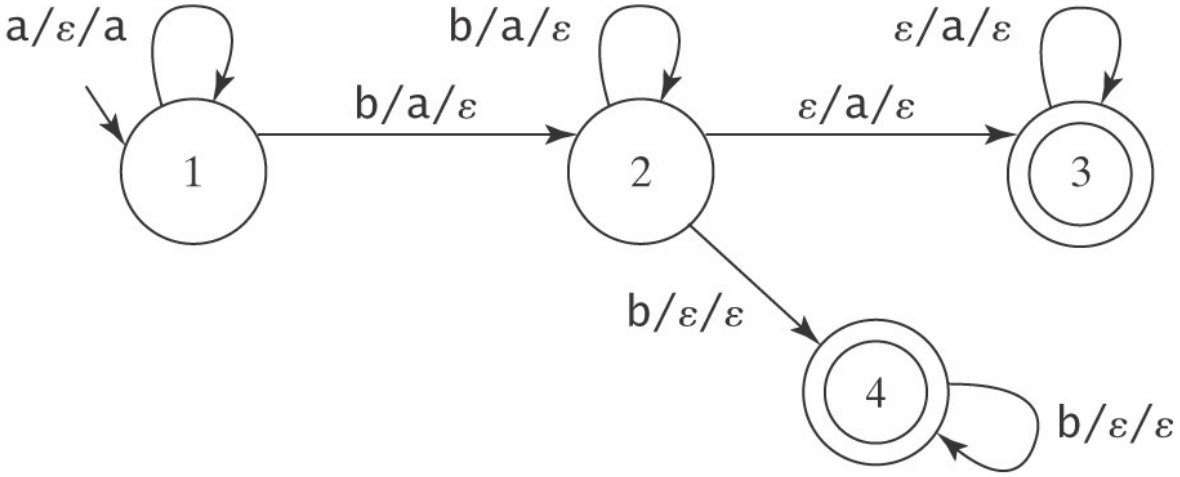


- If stack is empty but input is not ($m < n$) (accept):



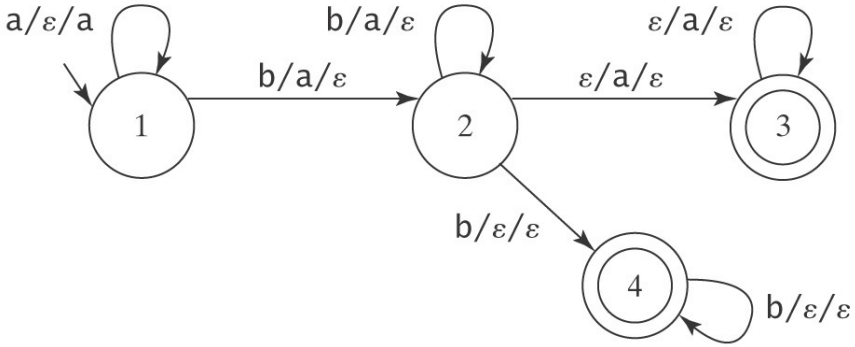
Putting It Together

$L = \{a^m b^n : m \neq n;\ m, n > 0\}$

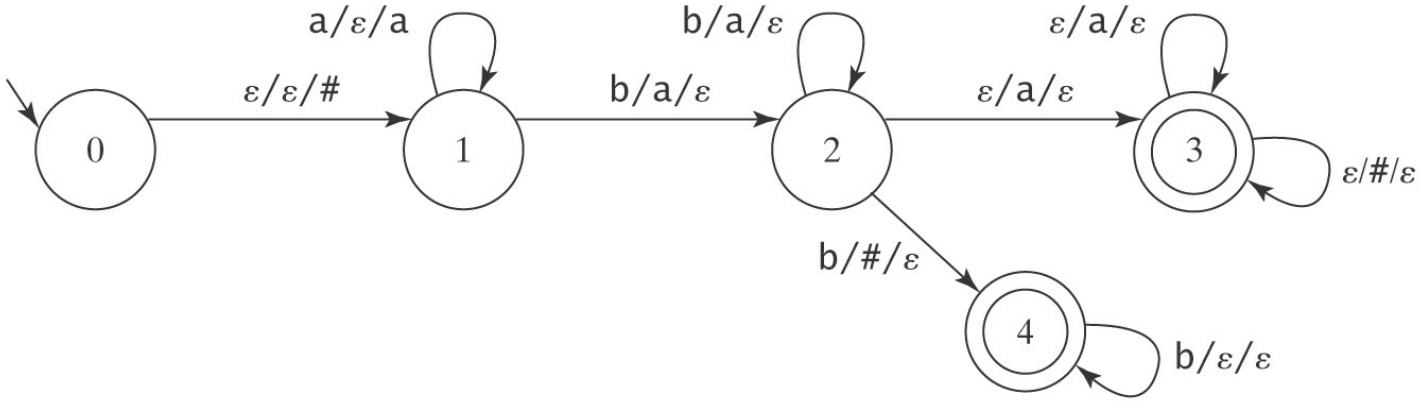


Node 2 is nondeterministic.

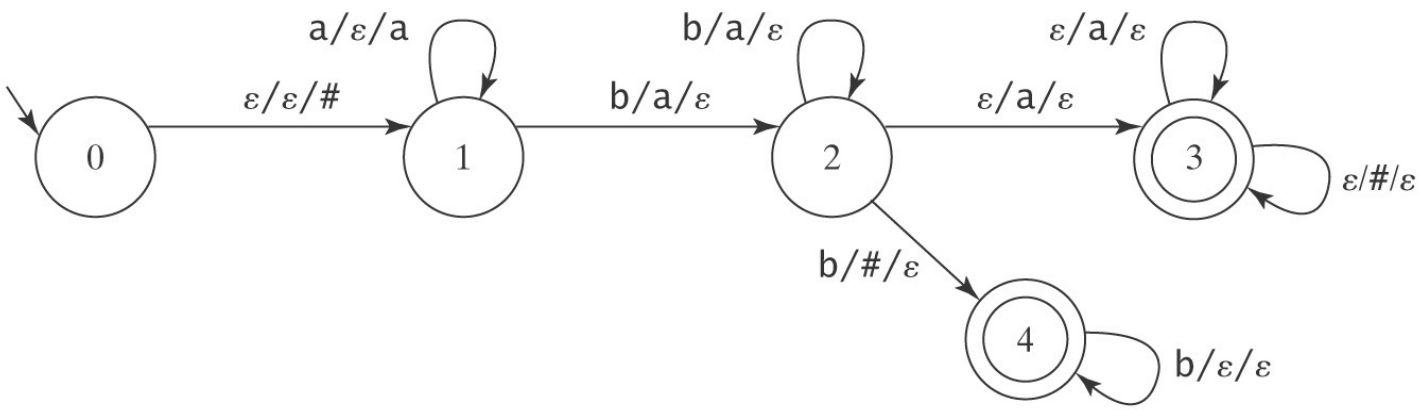
Reducing Nondeterminism



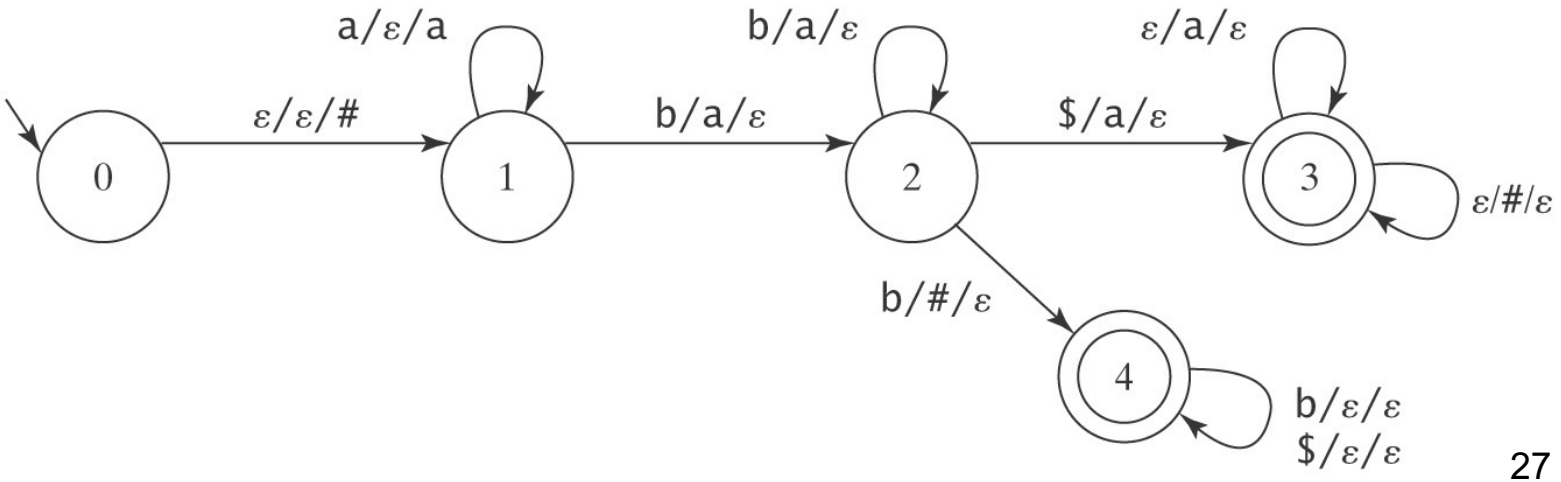
- Use a bottom-of-stack marker: #



Continue Reducing Nondeterminism



- Use an end-of-string marker: $



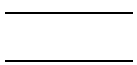
PDAs and Context-Free Grammars

Theorem: The class of languages accepted by PDAs is exactly the class of context-free languages.

Recall: context-free languages are languages that can be defined with context-free grammars.

Restate theorem:

Can describe with context-free grammar



Can accept by PDA

Proof

***Lemma**:* Each context-free language is accepted by some PDA.

Lemma: If a language is accepted by a PDA *M*, it is context-free (i.e., it can be described by a context-free grammar).

Proof by construction

Nondeterminism and Halting

1. There are context-free languages for which no deterministic PDA exists.

2. It is possible that a PDA may
   - not halt,
   - not ever finish reading its input.

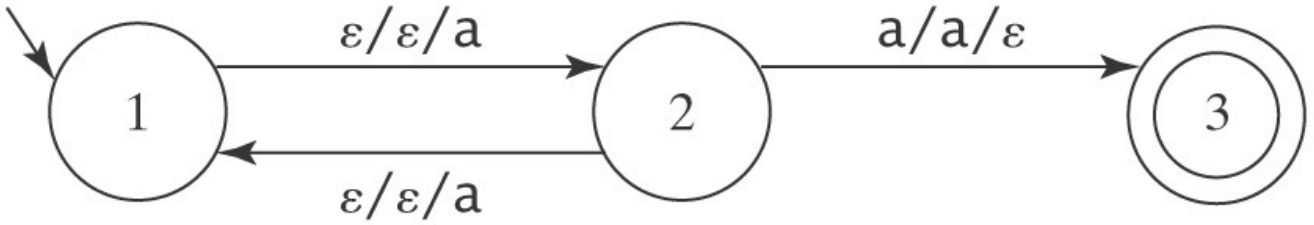
However, for an arbitrary PDA M, there exists M' that halts and $L(M') = L(M)$

Nondeterminism and Halting

It is possible that a PDA may not halt.

- Note: the same situation for NDFSM.
- But we can find an equivalent PDA that halts

Let Σ = {a} and consider *M* =



The path (1, a, ε) -> (2, a, a) -> (3, ε, ε) causes *M* to accept a.

L(*M*) = {a}. On any other input except a:

- *M* will never halt because of one path never ends and none of the paths accepts.

Question: for aa, how many rejecting paths?

Comparing Regular and Context-Free Languages

Regular Languages	**Context-Free Languages**
regular expressions or regular grammars	context-free grammars
= DFSMs	= NDPDAs