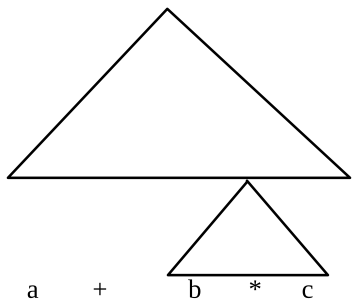
Pushdown Automata

Read K & S 3.3.
Read Supplementary Materials: Context-Free Languages and Pushdown Automata: Designing Pushdown Automata.
Do Homework 13.

Recognizing Context-Free Languages

Two notions of recognition:

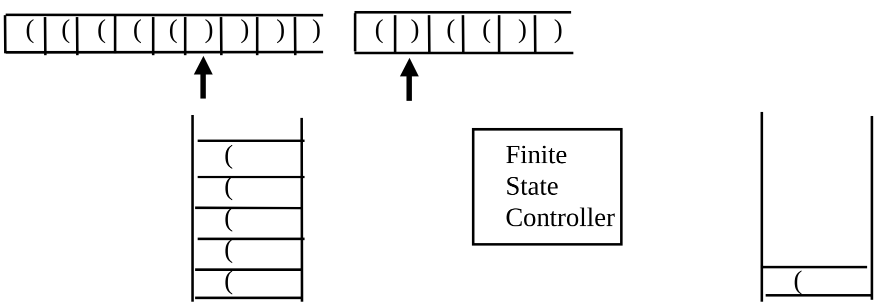
(1) Say yes or no, just like with FSMs
(2) Say yes or no, AND
if yes, describe the structure



Just Recognizing

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.



Definition of a Pushdown Automaton

M = (K, Σ, Γ, Δ, s, F), where:

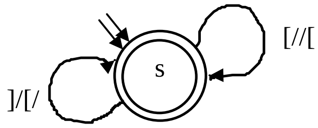
- K is a finite set of states
- Σ is the input alphabet
- Γ is the stack alphabet
- $s \in K$ is the initial state
- $F \subseteq K$ is the set of final states, and
- Δ is the transition relation. It is a finite subset of

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

M accepts a string w iff

$$(s, w, \varepsilon) \vdash_M^* (p, \varepsilon, \varepsilon) \qquad \text{for some state } p \in F$$

A PDA for Balanced Brackets



M = (K, Σ, Γ, Δ, s, F), where:

K = {s}	the states
Σ = {[,]}	the input alphabet
Γ = {[}	the stack alphabet
F = {s}	

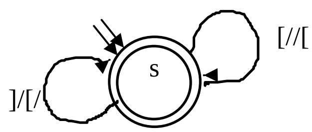
Δ contains:

((s, [, ε), (s, [ ))
((s,], [ ), (s, ε))

Important:

This does not mean that the stack is empty.

An Example of Accepting



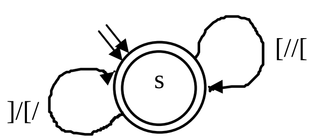
Δ contains:

[1] ((s, [, ε), (s, [ ))
[2] ((s,], [ ), (s, ε))

input = [[[ ] [ ]]]

trans	*state*	*unread input*	*stack*
	s	[[[ ] [ ]]]	ε
1	s	[[ ] [ ]]]	[
1	s	[ ] [ ]]]	[[
1	s	] [ ]]]	[[[
2	s	[ ]]]	[[
1	s	]]]	[[[
2	s	]]	[[
2	s	]	[
2	s	ε	ε

An Example of Rejecting



Δ contains:

[1] ((s, [, ε), (s, [ ))
[2] ((s,], [ ), (s, ε))

input = [[ ]]]

trans	*state*	*unread input*	*stack*
	s	[[ ]]]	ε
1	s	[ ]]]	[
1	s	]]]	[[
2	s	]]	[
2	s	]	ε
none!	s	]	ε

We're in s, a final state, but we cannot accept because the input string is not empty. So we reject.

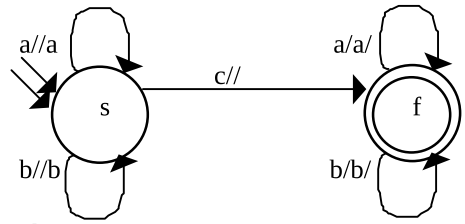
A PDA for $a^n b^n$

First we notice:

- We'll use the stack to count the a's.
- This time, all strings in L have two regions. So we need two states so that a's can't follow b's. Note the similarity to the regular language a*b*.

A PDA for wcw^R

A PDA to accept strings of the form wcw^R:



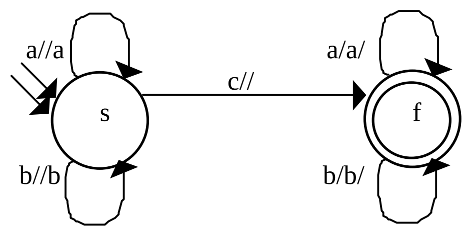
M = (K, Σ, Γ, Δ, s, F), where:

K = {s, f}	the states
Σ = {a, b, c}	the input alphabet
Γ = {a, b}	the stack alphabet
F = {f}	the final states

Δ contains:

((s, a, ε), (s, a))
((s, b, ε), (s, b))
((s, c, ε), (f, ε))
((f, a, a), (f, ε))
((f, b, b), (f, ε))

An Example of Accepting



Δ contains:

[1] ((s, a, ε), (s, a))
[2] ((s, b, ε), (s, b))
[3] ((s, c, ε), (f, ε))
[4] ((f, a, a), (f, ε))
[5] ((f, b, b), (f, ε))

input = b a c a b

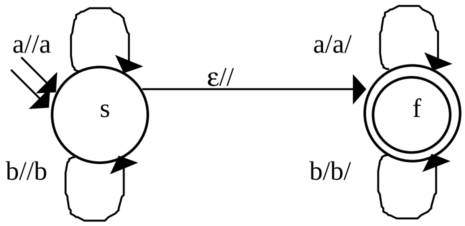
trans	***state***	***unread input***	***stack***
	s	b a c a b	ε
2	s	a c a b	b
1	s	c a b	ab
3	f	a b	ab
5	f	b	b
6	f	ε	ε

A Nondeterministic PDA

$L = ww^R$

$S \to \varepsilon$
$S \to aSa$
$S \to bSb$

A PDA to accept strings of the form ww^R:



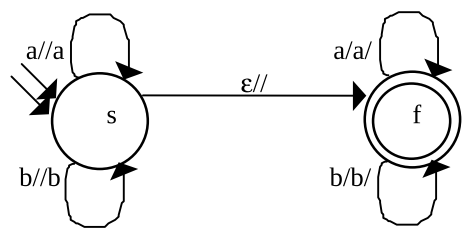
$M = (K, \Sigma, \Gamma, \Delta, s, F)$, where:

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

Δ contains:

$((s, a, \varepsilon), (s, a))$
$((s, b, \varepsilon), (s, b))$
$((s, \varepsilon, \varepsilon), (f, \varepsilon))$
$((f, a, a), (f, \varepsilon))$
$((f, b, b), (f, \varepsilon))$

An Example of Accepting



[1]	$((s, a, \varepsilon), (s, a))$	[4]	$((f, a, a), (f, \varepsilon))$
[2]	$((s, b, \varepsilon), (s, b))$	[5]	$((f, b, b), (f, \varepsilon))$
[3]	$((s, \varepsilon, \varepsilon), (f, \varepsilon))$		

input: a a b b a a

trans	***state***	***unread input***	***stack***
	s	a a b b a a	ε
1	s	a b b a a	a
3	f	a b b a a	a
4	f	b b a a	ε
none			

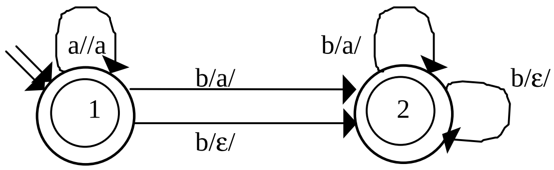
trans	***state***	***unread input***	***stack***
	s	a a b b a a	ε
1	s	a b b a a	a
1	s	b b a a	aa
2	s	b a a	baa
3	f	b a a	baa
5	f	a a	aa
4	f	a	a
4	f	ε	ε

$L = \{a^m b^n : m \leq n\}$

A context-free grammar for L:

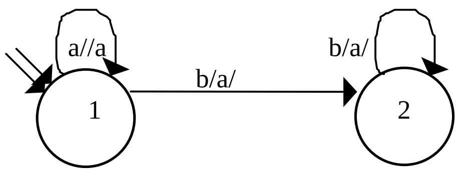
$S \rightarrow \varepsilon$
$S \rightarrow Sb$ /* more b's
$S \rightarrow aSb$

A PDA to accept L:

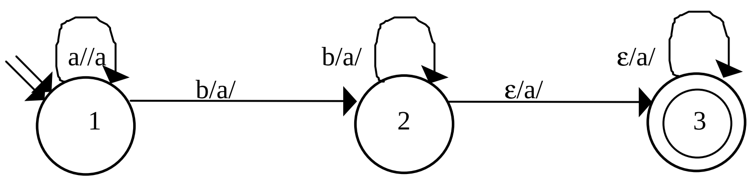


Accepting Mismatches

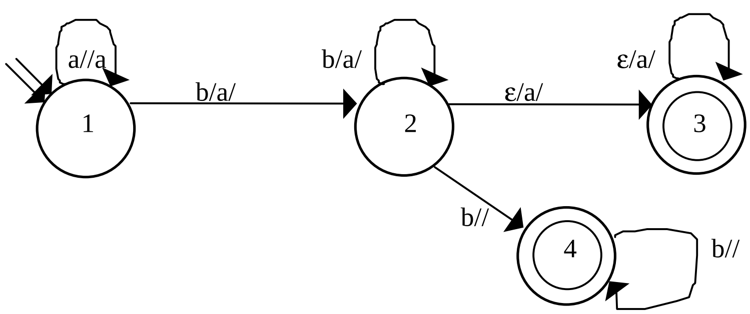
$L = \{a^m b^n\ m \neq n;\ m, n > 0\}$



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

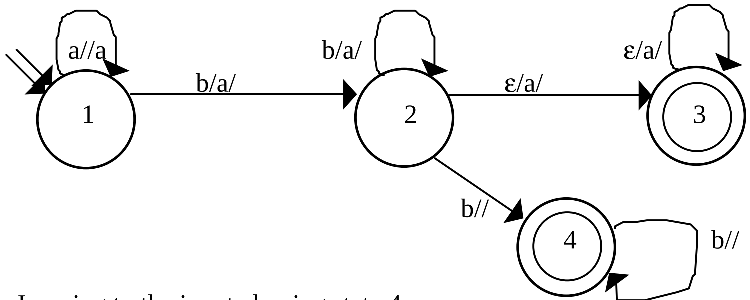


- If stack is empty but input is not (m < n) (accept):

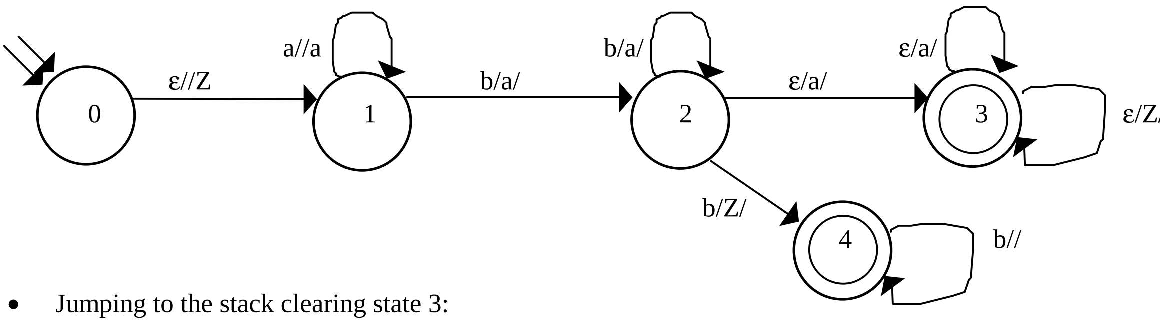


Eliminating Nondeterminism

A PDA is **deterministic** if, for each input and state, there is at most one possible transition. Determinism implies uniquely defined machine behavior.



- Jumping to the input clearing state 4:
  Need to detect bottom of stack, so push Z onto the stack before we start.



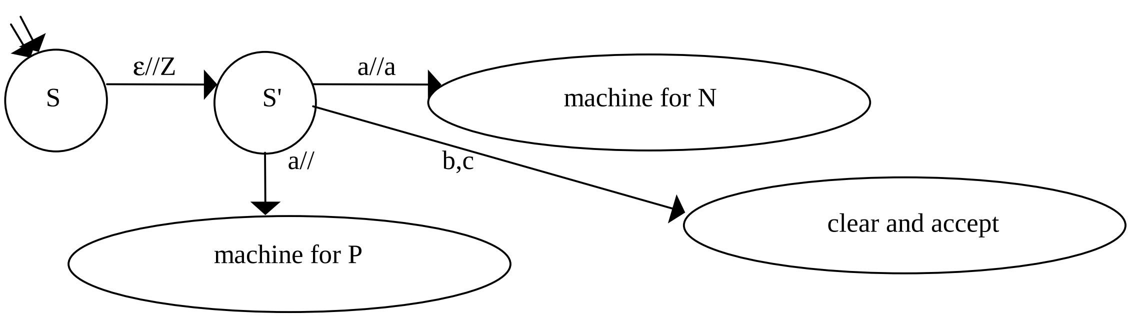
- Jumping to the stack clearing state 3:
  Need to detect end of input. To do that, we actually need to modify the definition of L to add a termination character (e.g., $)

L = {$a^n b^m c^p$: n,m,p ≥ 0 and (n ≠ m or m ≠ p)}

S → NC /* n ≠ m, then arbitrary c's
S → QP /* arbitrary a's, then p ≠ m
N → A /* more a's than b's
N → B /* more b's than a's
A → a
A → aA
A → aAb
B → b
B → Bb
B → aBb

C → ε | cC /* add any number of c's
P → B' /* more b's than c's
P → C' /* more c's than b's
B' → b
B' → bB'
B' → bB'c
C' → c | C'c
C' → C'c
C' → bC'c
Q → ε | aQ /* prefix with any number of a's

L = {$a^n b^m c^p$: n,m,p ≥ 0 and (n ≠ m or m ≠ p)}



Another Deterministic CFL

$L = \{a^n b^n\} \cup \{b^n a^n\}$

A CFG for L: A NDPDA for L:

$S \rightarrow A$
$S \rightarrow B$
$A \rightarrow \varepsilon$
$A \rightarrow aAb$
$B \rightarrow \varepsilon$
$B \rightarrow bBa$

A DPDA for L:

More on PDAs

What about a PDA to accept strings of the form ww?

Every FSM is (Trivially) a PDA

Given an FSM M = (K, Σ, Δ, s, F)
and elements of Δ of the form
(p, i, q)
old state, input, new state

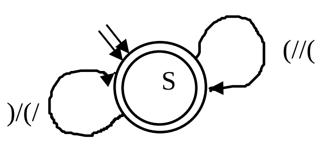
We construct a PDA M' = (K, Σ, Γ, Δ, s, F)
where Γ = ∅ /* stack alphabet
and
each transition (p, i, q) becomes
((p, i, ε), (q, ε))
old state, input, don't look at stack new state don't push on stack

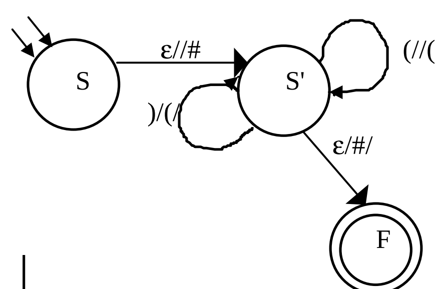
In other words, we just don't use the stack.

Alternative (but Equivalent) Definitions of a NDPDA

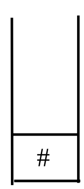
Example: Accept by final state at end of string (i.e., we don't care about the stack being empty)
We can easily convert from one of our machines to one of these:
1. Add a new state at the beginning that pushes # onto the stack.
2. Add a new final state and a transition to it that can be taken if the input string is empty and the top of the stack is #.

Converting the balanced parentheses machine:





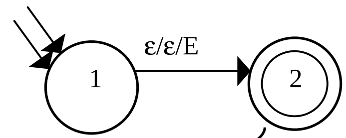
The new machine is nondeterministic:
( ) ( )
⇑
The stack will be:



What About PDA's for Interesting Languages?

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow id$

Arithmetic Expressions



(1) (2, ε, E), (2, E+T)
(2) (2, ε, E), (2, T)
(3) (2, ε, T), (2, T*F)
(4) (2, ε, T), (2, F)
(5) (2, ε, F), (2, (E))
(6) (2, ε, F), (2, id)
(7) (2, id, id), (2, ε)
(8) (2, (, ( ), (2, ε)
(9) (2,),)), (2, ε)
(10) (2, +, +), (2, ε)
(11) (2, *, *), (2, ε)

Example:
a + b * c

But what we really want to do with languages like this is to extract structure.

Comparing Regular and Context-Free Languages

Regular Languages

- regular expressions
  - or -
- regular grammars
- recognize
- = DFSAs

Context-Free Languages

- context-free grammars
- parse
- = NDPDAs