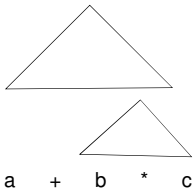
Pushdown Automata

Chapter 12

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.

Example: Bal (the balanced parentheses language)

(((())))

Definition of a 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

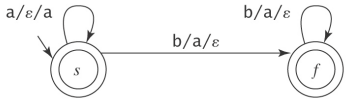
$$\underbrace{(K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*)}_{\text{state, input or } \varepsilon, \text{ string of symbols to pop from top of stack}} \times \underbrace{(K \times \Gamma^*)}_{\text{state, string of symbols to push on top of stack}}$$

Definition of a Pushdown Automaton

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

The initial configuration of M is (s, w, ε).

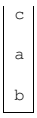
A PDA for $A^nB^n = \{a^nb^n: n \geq 0\}$



Writing it out, we have $M = (K, \Sigma, \Gamma, \Delta, s, A)$, where:

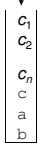
$K = \{s, f\}$, (the states)
$\Sigma = \{a, b\}$, (the input alphabet)
$\Gamma = \{a\}$, (the stack alphabet)
$A = \{s, f\}$, and (the accepting states)
$\Delta = \{((s, a, \varepsilon), (s, a)),$
$((s, b, a), (f, \varepsilon)),$
$((f, b, a), (f, \varepsilon))\}.$

Manipulating the Stack



will be written as cab

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



$c_1c_2\ldots c_n$cab

Yields

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

Let $\vdash_M^*$ be the reflexive, transitive closure of $\vdash_M$.

C_1 ***yields*** configuration C_2 iff $C_1 \vdash_M^* C_2$

Computations

A ***computation*** by M is a finite sequence of configurations $C_0, C_1, \ldots, C_n$ for some $n \geq 0$ such that:

- C_0 is an initial configuration,
- C_n is of the form **(q, ε, γ),** for some state $q \in K_M$ and some string γ in Γ^*, and
- $C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_n$.

Accepting

A computation C of M is an ***accepting computation*** iff:

- $C = (s, w, \varepsilon) \vdash_M^* (q, \varepsilon, \varepsilon)$, and
- $q \in A$.

M ***accepts*** a string w iff at least one of its computations accepts.

Other paths may:
- Read all the input and halt in a nonaccepting state,
- Read all the input and halt in an accepting state with the stack not empty,
- Loop forever and never finish reading the input, or
- Reach a dead end where no more input can be read.

The ***language accepted by*** M, denoted $L(M)$, is the set of all strings accepted by M.

Rejecting

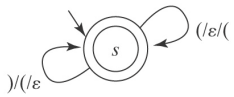
A computation C of M is a ***rejecting computation*** iff:

- $C = (s, w, \varepsilon) \vdash_M^* (q, w', \alpha)$,
- C is not an accepting computation, and
- M has no moves that it can make from (q, ε, α).

M ***rejects*** a string w iff all of its computations reject.

So note that it is possible that, on input w, M neither accepts nor rejects.

A PDA for Balanced Parentheses



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

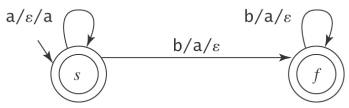
$K = \{s\}$	the states
$\Sigma = \{(,)\}$	the input alphabet
$\Gamma = \{(\}$	the stack alphabet
$A = \{s\}$	

Δ contains:

$((s, (, \varepsilon^{**}), (s, ( ))$
$((s,), ( ), (s, \varepsilon))$

**Important: This does not mean that the stack is empty

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



Writing it out, we have $M = (K, \Sigma, \Gamma, \Delta, s, A)$, where:

$K = \{s, f\}$, (the states)

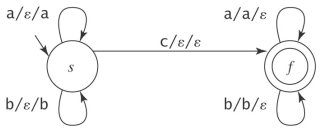
$\Sigma = \{a, b)\}$, (the input alphabet)

$\Gamma = \{a\}$, (the stack alphabet)

$A = \{s, f\}$, and (the accepting states)

$\Delta = \{((s, a, \varepsilon), (s, a)),$
$((s, b, a), (f, \varepsilon)),$
$((f, b, a), (f, \varepsilon))\}.$

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



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

$K = \{s, f\}$ the states

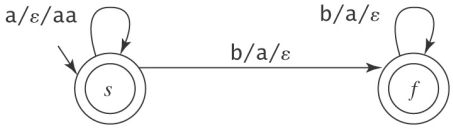
$\Sigma = \{a, b, c\}$ the input alphabet

$\Gamma = \{a, b\}$ 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 $\{a^n b^{2n}: n \geq 0\}$



Nondeterminism

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

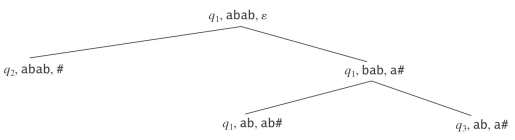
- Δ contains exactly one transition that matches.
- Δ contains more than one transition that matches.
- Δ contains no transition that matches.

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.

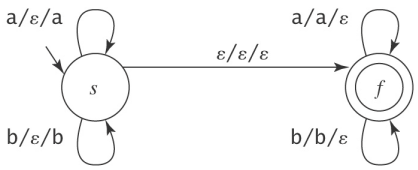
But many useful PDAs are not deterministic.



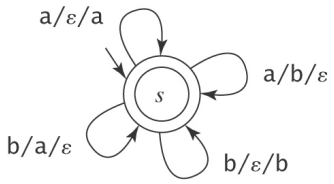
A PDA for PalEven $=\{ww^R: w \in \{a, b\}^*\}$

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

A PDA:



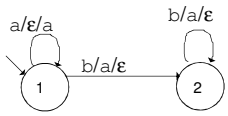
A PDA for $\{w \in \{a, b\}^* : \#_a(w) = \#_b(w)\}$



More on Nondeterminism Accepting Mismatches

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

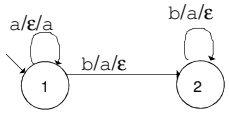
Start with the case where $n = m$:



More on Nondeterminism Accepting Mismatches

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

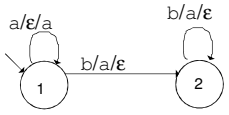
Start with the case where $n = m$:



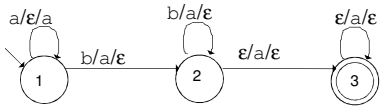
- If stack and input are empty, halt and reject.
- If input is empty but stack is not ($m > n$) (accept):
- If stack is empty but input is not ($m < n$) (accept):

More on Nondeterminism Accepting Mismatches

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

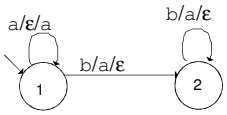


- If input is empty but stack is not ($m < n$) (accept):

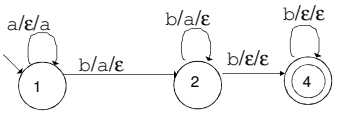


More on Nondeterminism Accepting Mismatches

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

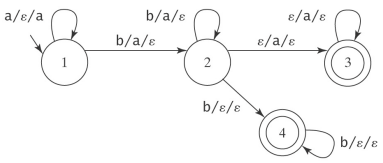


- If stack is empty but input is not ($m > n$) (accept):



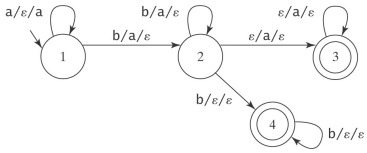
Putting It Together

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



- Jumping to the input clearing state 4:
  Need to detect bottom of stack.
- Jumping to the stack clearing state 3:
  Need to detect end of input.

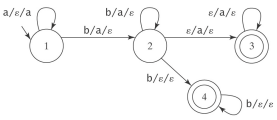
Reducing Nondeterminism



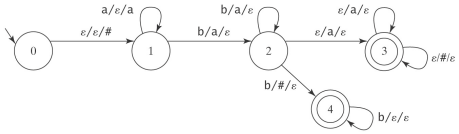
- Jumping to the input clearing state 4:
  Need to detect bottom of stack, so push # onto the stack before we start.
- Jumping to the stack clearing state 3:
  Need to detect end of input. Add to *L* a termination character (e.g., $)

Reducing Nondeterminism

Using a bottom of stack marker

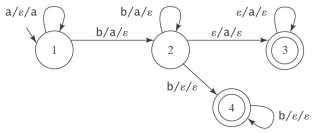


- Jumping to the input clearing state 4:

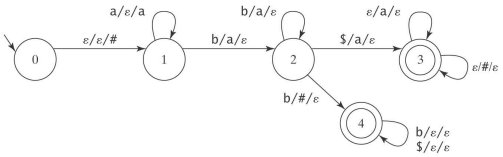


Reducing Nondeterminism

Using an end-of string Marker



- Jumping to the stack clearing state 3:



Nondeterminism and Halting

Recall that a computation C of a PDA $M = (K, \Sigma, \Gamma, \Delta, s, A)$ on a string w is an accepting computation iff:

$$C = (s, w, \varepsilon) |\text{-}_M^* (q, \varepsilon, \varepsilon), \text{ for some } q \in A.$$

We'll say that a computation C of M ***halts*** iff at least one of the following conditions holds:

- C is an accepting computation, or
- C ends in a configuration from which there is no transition in Δ that can be taken.

We'll say that M ***halts*** on w iff every computation of M on w halts. If M halts on w and does not accept, then we say that M rejects w.

Nondeterminism and Halting

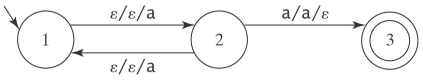
1. There are context-free languages for which no deterministic PDA exists.
2. It is possible that a PDA may
   - not halt,
   - not ever finish reading its input.

   Example is in next slide
3. There exists no algorithm to minimize a PDA. It is undecidable whether a PDA is minimal.

Nondeterminism and Halting

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

Let $\Sigma = \{a\}$ and consider $M =$



$L(M) = \{a\}$: (1, a, ε) |- (2, a, a) |- (3, ε, ε)

On any other input except a:
- *M* will never halt.
- *M* will never finish reading its input unless its input is ε.

(1, aa, ε) |- (2, aa, a) |- (1, aa, aa) |- (2, aa, aaa) |- (1, aa, aaaa) |- (2, aa, aaaaa) |-...

Solutions to the Problem

- For NDFSMs:
  - Convert to deterministic, or
  - Simulate all paths in parallel.
- For NDPDAs:
  - Formal solutions that usually involve changing the grammar.
  - Practical solutions that:
    - Preserve the structure of the grammar, but
    - Only work on a subset of the CFLs.

Alternative Equivalent Definitions of a PDA

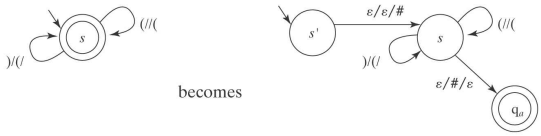
Accept by accepting state at end of string (i.e., we don't care about the stack).

From M (in our definition) we build M' (in this one):

1. Initially, let $M' = M$.
2. Create a new start state s'. Add the transition:
   $((s', \varepsilon, \varepsilon), (s, \#))$.
3. Create a new accepting state q_a.
4. For each accepting state a in M do,
   4.1 Add the transition $((a, \varepsilon, \#), (q_a, \varepsilon))$.
5. Make q_a the only accepting state in M'.

Example

The balanced parentheses language



What About These?

- FSM plus FIFO queue (instead of stack)?
- FSM plus two stacks?

Comparing Regular and Context-Free Languages

Regular Languages	Context-Free Languages
• regular exprs. • or	
• regular grammars grammars	• context-free
• recognize	• parse
• = DFSMs	• = NDPDAs