Pushdown Automata

Machines for Context-Free Languages

Wednesday, November 7, 2007
Reading: Sipser 2.1; Kozen 23

CS235 Languages and Automata

Department of Computer Science
Wellesley College

Goals

- Complete the analogy:

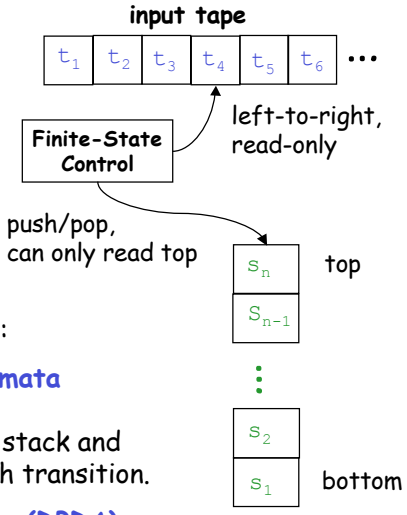
  regular expression : finite automaton
  :: context-free grammar : ???

  ??? = **pushdown automaton**

- Study/design some nondeterministic pushdown automata (NPDAs).
- Discuss the relationship between CFGs and NPDAs.
- Discuss nondeterministic vs. deterministic PDAs.

Pushdown Automata

A pushdown automaton is a finite automaton + a stack of symbols that can be pushed/popped on each transition. The finite-state controller can read only the top symbol of the stack, which can affect the transition.



Comes in many flavors (Sipser/Kozen):

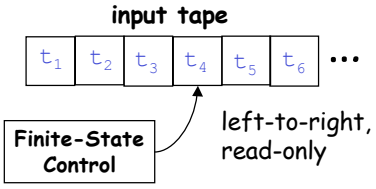
Nondeterministic Pushdown Automata
(Kozen's **NPDA**, Sipser's **PDA**)
= Stoughton's EFA + read top of stack and push/pop stack symbols on each transition.

Deterministic Pushdown Automata (DPDA)
= Stoughton's DFA + stack operations + end-of-input symbol .

Definitions differ slightly from author to author.

Finite Automata Review

A finite automaton accepts/rejects strings that can be viewed as a read-only tape of symbols read by a tape head that can move only left to right, as controlled by a finite-state controller.

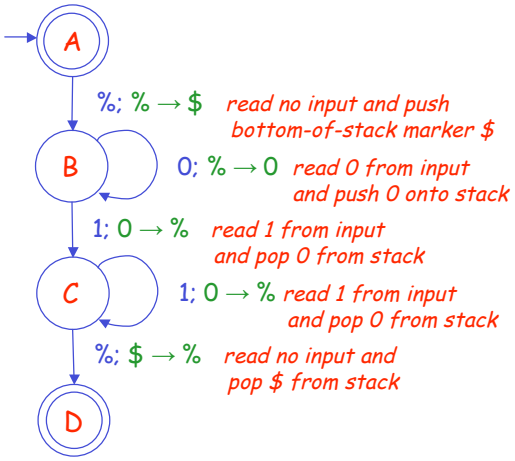


Comes in many flavors (Stoughton):

FA = transitions between states are relations involving arbitrary strings.
EFA = transitions are relations involving strings of length ≤ 1.
NFA = transitions are relations involving strings of length 1.
DFA = transitions are functions involving strings of length 1.

Other authors define such machines in slightly different ways, sometimes with different names.

An NPDA for 0^n1^n



* *Note:* Where we use %; % → $, Sipser uses ε, ε → $.

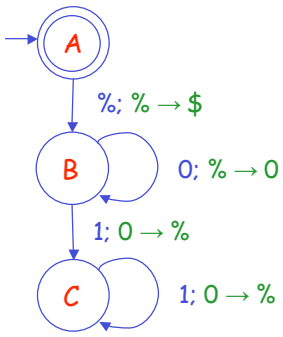
Example Execution

State	Input	Stack
A	000111	%
B	000111	$
B	00111	0$
B	0111	00$
B	111	000$
C	11	00$
C	1	0$
C	%	$
D	%	%

Each row of table is a **machine configuration**. Transitions go from one configuration to another.

An NPDA for $\{0^m1^n \mid m > n\}$

How do we complete the following NPDA to do the right thing? (Hint: use nondeterminism to flush the extra 0s.)



State	Input	Stack
A	000011	%
B	000011	$
B	00011	0$
B	0011	00$
B	011	000$
C	11	0000$
C	1	000$
C	%	00$

An NPDA for $\{w \mid w \in \{a,b\}^* \text{ and } \#as = \#bs\}$

Create an NPDA for this language.

State	Input	Stack
	bbaaabba	%

An NPDA for $\{a^i b^j c^k \mid i = j \text{ or } i = k\}$

Create an NPDA for this language (use nondeterminism!)

State	Input	Stack
	aabbbcc	%

An NPDA for $\{ww^R \mid w \in \{0,1\}^*\}$

Create an NPDA for this language (use nondeterminism!)

State	Input	Stack
	100001	%

Formally Defining NPDAs

Definitions differ among authors. This one is similar to Sipser's:

A **Nondeterministic Pushdown Automaton (NPDA)*** in a sextuple:

(1) A set of states Q
(2) An input alphabet Σ (not containing the empty string symbol %)
(3) A stack alphabet Γ
(4) A start state $q_0 \in Q$
(5) A set of accept states $F \subseteq Q$
(6) A transition relation $\delta \subseteq (Q \times (\Sigma \cup \%) \times (\Gamma \cup \%)) \times (Q \times (\Gamma \cup \%))$

This is similar to Stoughton's EFA except:
- The input alphabet is explicit (rather than implicitly being **Sym**)
- The stack alphabet is new
- The transition relation allows pushing/popping a stack symbol (Stoughton's transition relation $T \subseteq (Q \times (\Sigma \cup \%) \times Q)$.)

* Sipser calls it a Pushdown Automaton (PDA)

Defining Acceptance for NPDAs

A **configuration** is a triple in $Q \times \Sigma^* \times \Gamma^*$.

Let p,q in Q; t in $(\Sigma \cup \%)$; x, y in Σ^*; s,s' in $(\Gamma \cup \%)$; and G in Γ^*.

We define the **step relation**, $\Rightarrow$, on configurations:

$$(p, ty, sG) \Rightarrow (q, y, s') \text{ iff } ((p, t, s), (q, s')) \in \delta$$

A NPDA **accepts** a string x iff

$$(q_0, x, \%) \Rightarrow^* (p, \%, G)$$

where q_0 is the start state and $p \in$ the accept states F.

Example for 0^n1^n NPDA

$(A, 000111, \%) \Rightarrow (B, 000111, \$) \Rightarrow (B, 00111, 0\$)$

$\Rightarrow (B, 0111, 00\$) \Rightarrow (B, 111, 000\$) \Rightarrow (C, 11, 00\$)$

$\Rightarrow (C, 1, 0\$) \Rightarrow (C, \%, \$) \Rightarrow (D, \%, \%)$

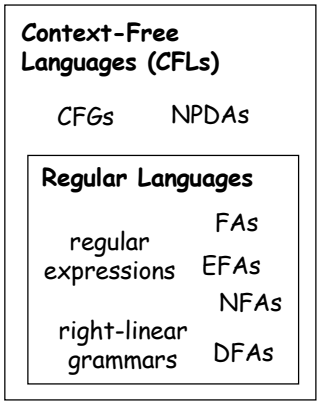
NPDAs are Equivalent to CFGs

It can be shown that:

(1) A CFG can be converted to an NPDA.
I.e.g, every context-free language
(i.e., the language of some CFG)
is the language accepted by some NPDA.

(2) An NPDA can be converted to a CFG.
I.e., the language accepted by an NPDA
is context-free.

For details, see Sipser 2.2 or Kozen 24-25.



Deterministic PDAs (DPDAs)

Like an NPDA except:

- Have a special symbol ($\dashv$) for the end of input.
- Transition relation must be a function (no choices/guessing!)

Effectively a DFA + stack operations, except that some transitions might not read any input (and instead perform stack operations).

For details, see Kozen Supplementary Lecture F.

Where do DPDAs Fit In?

The language of a DPDA is a **Deterministic Context-Free Language (DCFL)**.

DCFLs are a proper subset of CFLs.

I.e., there are CFLs that are not DCFLS. E.g.

$\{a^i b^j c^k \mid i = j \text{ or } i = k\}$

$\{ww^R \mid w \text{ in } \{0,1\}^*\}$

I.e. DPDAs are strictly less powerful than NPDAs.

D