Context-Free Languages

CS 712
Dept. of Computer Science
Univ. of New Hampshire

Context-Free Languages are generated by

Context-Free Grammars

Context-Free Grammars have productions of the form: $A \to w$

A: one nonterminal

w: string of terminals and nonterminals

we know $\{a^n b^n \mid n \geq 0\}$ is not regular
but it is context-free.

$$S \to \lambda$$
$$S \to aSb$$

$S \to \lambda$

$S \to aSb \to ab$

$S \to aSb \to aaSbb \to aabb$

$S \to aSb \to aaSbb \to aaaSbbb \to aaabbb$

therefore the set of regular languages is a proper subset of the set of context-free languages.

remember: every regular language is context-free because it can be generated by a regular grammar and every regular grammar is context free

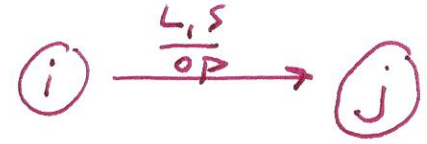
Pushdown Automata (PDA)

- a finite automaton with a stack

  stack is initialized to contain a single symbol

  stack has usual push/pop plus nop

  arcs of the finite automaton are labeled

  with $\frac{L, S}{op}$ 

  instruction written as a tuple

  (i, L, S, op, j)

  can actually be a sequence of operations

  if PDA is in state i, and either L is next in the input or L is λ, and the symbol on top of the stack is S,

  then execute op

  move to state j

  if $L \neq \lambda$, then go to next input

A string is accepted by the PDA if the stack is empty and the input string is consumed

(i.e. no requirement that PDA be in any particular state)

PDA is deterministic if there is at most one move possible from each state

there is nondeterminism if a state has

1) two arcs labelled with same input & stack symbols

2) two arcs labelled with same stack symbol and one arc is labelled with λ for the input symbol

Theorem The language generated by any context-free grammar can be accepted by a PDA.

Proof by construction

Given CFG, construct a PDA with a single state:

i) initialize stack to contain start symbol

ii) for each terminal symbol a, create the instruction (ϕ, a, a, pop, ϕ)

iii) For each production $A \to B_1 B_2 \cdots B_n$, $n \neq \phi$, create the instruction

$(\phi, \lambda, A, <pop, push\ B_n, push\ B_{n-1}, \ldots, push\ B_1, >, \phi)$

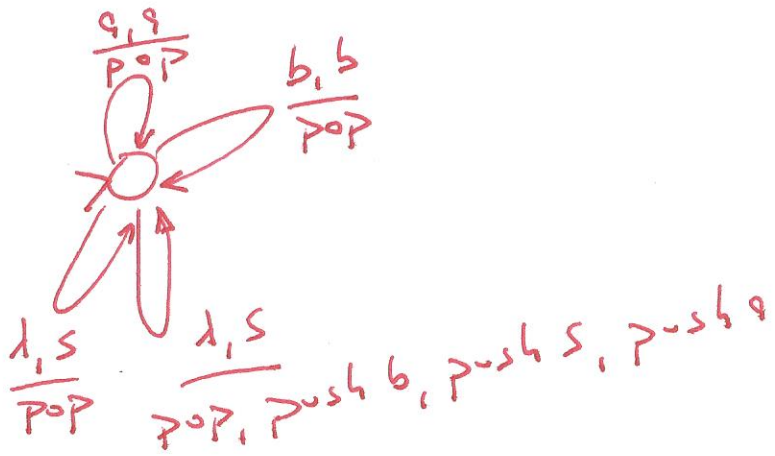
iv) For each production $A \to \lambda$, create the instruction $(\phi, \lambda, A, pop, \phi)$

The PDA accepts the language because each state transition with a nonterminal on top of the stack corresponds to one derivation step.

example

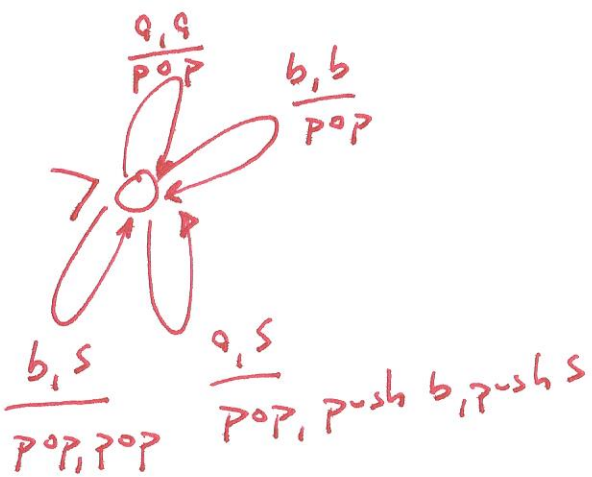
$S \to \lambda$

$S \to aSb$



input	top ← stack	derivation
aaabbb	S	S
aaabbb	aSb	→ aSb
aabbb	Sb	
aabbb	aSbb	→ aaSbb
abbb	SSbb	
abbb	aSbbb	→ aaaSbbb
bbb	Sbbb	
bbb	bbb	→ aaabbb
bb	bb	
b	b	

deterministic PDA to match the same language



input	top ← stack
aabb	S
abb	bSb
bb	Sbb
b	b

but in general PDA relies on nondeterminism to "choose" the correct production when a nonterminal is on top of the stack

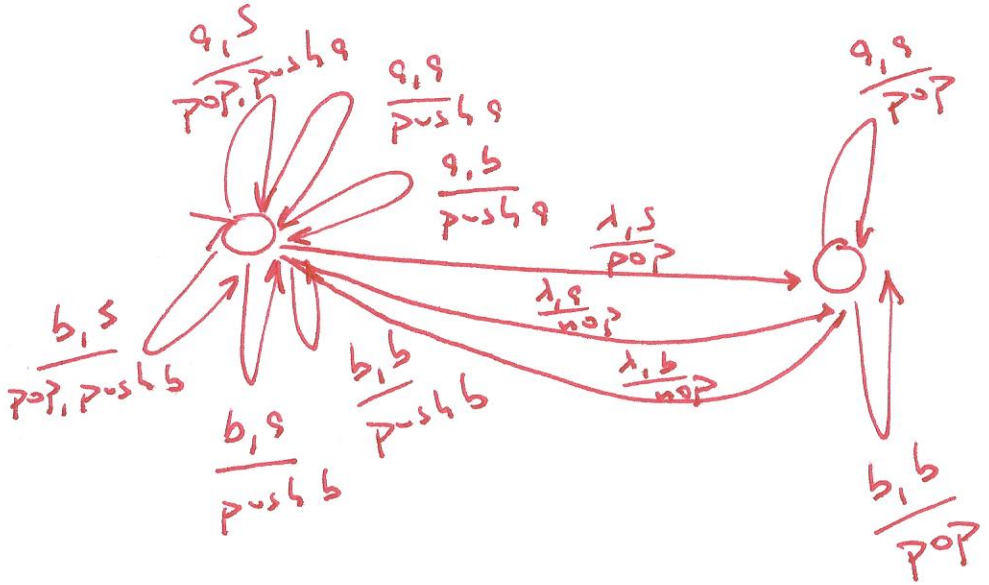
also parse is top-down with leftmost derivation

i.e. leftmost nonterminal in sentential form is rewritten

intermediate step in a derivation

nondeterminism is sometimes necessary

consider the even palindromes $\{ww^R / w \in \{a,b\}^*\}$



$S \to \lambda$

$S \to aSa$

$S \to bSb$

deterministic PDA cannot tell when the middle of the string has been reached

LL(1) parsing

- → leftmost derivation
- → read input left to right

top down

must always choose correct production
with 1 symbol lookahead

only parses a subset of the CFL's

can be implemented with a table

Select [A, a] = rhs of production with A as lhs

- a → current input
- A → top of stack