Pushdown Automata and Context-Free Grammars

Read K & S 3.4.
Read Supplementary Materials: Context-Free Languages and Pushdown Automata: Context-Free Languages and PDAs.
Do Homework 14.

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

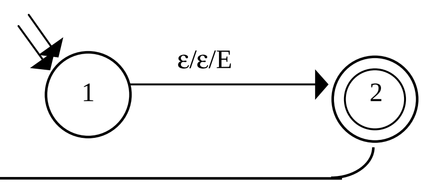
Going One Way

Lemma: Each context-free language is accepted by some PDA.
Proof (by construction by “top-down parse” conversion algorithm):

The idea: Let the stack do the work.

Example: Arithmetic expressions

E → E + T
E → T
T → T * F
T → F
F → (E)
F → id



(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)

The Top-down Parse Conversion Algorithm

Given G = (V, Σ, R, S)
Construct M such that L(M) = L(G)

M = ({p, q}, Σ, V, Δ, p, {q}), where Δ contains:

(1) ((p, ε, ε), (q, S))
push the start symbol on the stack

(2) ((q, ε, A), (q, x)) for each rule A → x in R
replace left hand side with right hand side

(3) ((q, a, a), (q, ε)) for each a ∈ Σ
read an input character and pop it from the stack

The resulting machine can execute a leftmost derivation of an input string in a top-down fashion.

Example of the Algorithm

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

(1)	S → ε
(2)	S → B
(3)	S → aSa
(4)	B → ε
(5)	B → bB

0	(p, ε, ε), (q, S)
1	(q, ε, S), (q, ε)
2	(q, ε, S), (q, B)
3	(q, ε, S), (q, aSa)
4	(q, ε, B), (q, ε)
5	(q, ε, B), (q, bB)
6	(q, a, a), (q, ε)
7	(q, b, b), (q, ε)

input = a a b b a a

trans	***state***	***unread input***	***stack***
	p	a a b b a a	ε
0	q	a a b b a a	S
3	q	a a b b a a	aSa
6	q	a b b a a	Sa
3	q	a b b a a	aSaa
6	q	b b a a	Saa
2	q	b b a a	Baa
5	q	b b a a	bBaa
7	q	b a a	Baa
5	q	b a a	bBaa
7	q	a a	Baa
4	q	a a	aa
6	q	a	a
6	q	ε	ε

Another Example

$L = \{a^n b^m c^p d^q : m + n = p + q\}$

(1)	S → aSd
(2)	S → T
(3)	S → U
(4)	T → aTc
(5)	T → V
(6)	U → bUd
(7)	U → V
(8)	V → bVc
(9)	V → ε

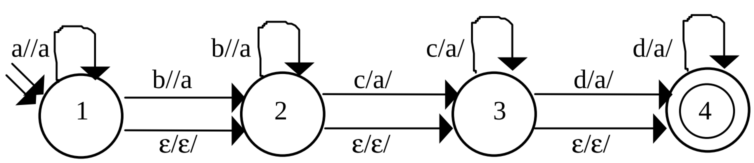
0	(p, ε, ε), (q, S)
1	(q, ε, S), (q, aSd)
2	(q, ε, S), (q,T)
3	(q, ε, S), (q,U)
4	(q, ε, T), (q, aTc)
5	(q, ε, T), (q, V)
6	(q, ε, U), (q, bUd)
7	(q, ε, U), (q, V)
8	(q, ε, V), (q, bVc
9	(q, ε, V), (q, ε)
10	(q, a, a), (q, ε)
11	(q, b, b), (q, ε)
12	(q, c, c), (q, ε)
13	(q, d, d), (q, ε)

input = a a b c d d

The Other Way—Build a PDA Directly

$L = \{a^n b^m c^p d^q : m + n = p + q\}$

(1)	S → aSd
(2)	S → T
(3)	S → U
(4)	T → aTc
(5)	T → V
(6)	U → bUd
(7)	U → V
(8)	V → bVc
(9)	V → ε



input = a a b c d d

Notice Nondeterminism

Machines constructed with the algorithm are often nondeterministic, even when they needn't be. This happens even with trivial languages.

Example: $L = a^n b^n$

A grammar for L is:

[1] $S \rightarrow aSb$
[2] $S \rightarrow \varepsilon$

A machine M for L is:
(0) ((p, ε, ε), (q, S))
(1) ((q, ε, S), (q, aSb))
(2) ((q, ε, S), (q, ε))
(3) ((q, a, a), (q, ε))
(4) ((q, b, b), (q, ε))

But transitions 1 and 2 make M nondeterministic.

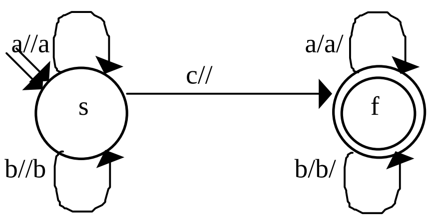
A **nondeterministic transition group** is a set of two or more transitions out of the same state that can fire on the same configuration. A **PDA is nondeterministic** if it has any nondeterministic transition groups.

A directly constructed machine for L:

Going The Other Way

Lemma: If a language is accepted by a pushdown automaton, it is a context-free language (i.e., it can be described by a context-free grammar).
Proof (by construction)

Example: $L = \{wcw^R : w \in \{a, b\}^*\}$



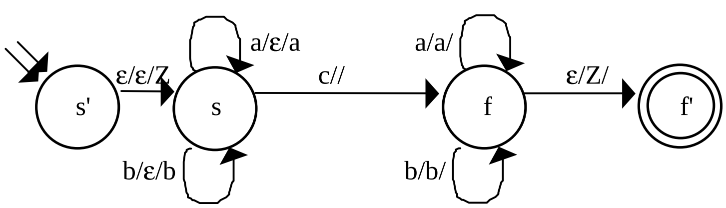
Δ contains:
((s, a, ε), (s, a))
((s, b, ε), (s, b))
((s, c, ε), (f, ε))
((f, a, a), (f, ε))
((f, b, b), (f, ε))

M = ({s, f}, {a, b, c}, {a, b}, Δ, s,{f}), where:

First Step: Make M Simple

A PDA M is simple iff:
1. there are no transitions into the start state, and
2. whenever ((q, x, β), (p, γ) is a transition of M and q is not the start state, then $\beta \in \Gamma$, and $|\gamma| \leq 2$.

Step 1: Add s' and f':

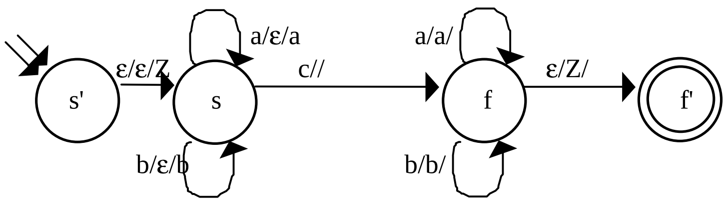


Step 2:
(1) Assure that $|\beta| \leq 1$.

(2) Assure that $|\gamma| \leq 2$.

(3) Assure that $|\beta| = 1$.

Making M Simple

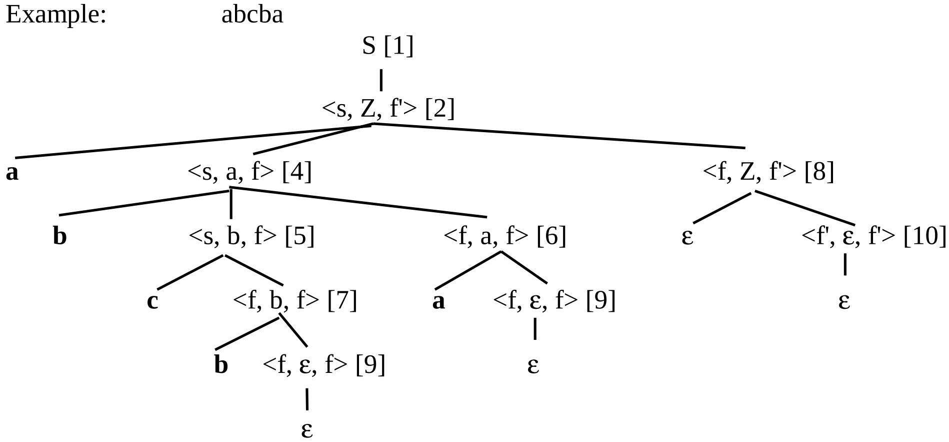


M = ({s, f, s', f'}, {a, b, c}, {a, b, Z}, Δ, s',{f'}), Δ=

	((s', ε, ε), (s, Z))
((s, a, ε), (s, a))	((s, a, Z), (s, aZ))
	((s, a, a), (s, aa))
	((s, a, b), (s, ab))
((s, b, ε), (s, b))	((s, b, Z), (s, bZ))
	((s, b, a), (s, ba))
	((s, b, b), (s, bb))
((s, c, ε), (f, ε))	((s, c, Z), (f, Z))
	((s, c, a), (f, a))
	((s, c, b), (f, b))
((f, a, a), (f, ε))	((f, a, a), (f, ε))
((f, b, b), (f, ε))	((f, b, b), (f, ε))
	((f, ε, Z), (f', ε))

Second Step - Creating the Productions

The basic idea -- simulate a leftmost derivation of M on any input string.
Example: abcba



If the nonterminal $\langle s_1, X, s_2\rangle \Rightarrow^* w$, then the PDA starts in state s_1 with (at least) X on the stack and after consuming w and popping the X off the stack, it ends up in state s_2.

Start with the rule:

$S \rightarrow \langle s, Z, f'\rangle$ where s is the start state, f' is the (introduced) final state and Z is the stack bottom symbol.

Transitions $((s_1, a, X), (s_2, YX))$ become a set of rules:

$\langle s_1, X, q\rangle \rightarrow a \langle s_2, Y, r\rangle \langle r, X, q\rangle$ for $a \in \Sigma \cup \{\varepsilon\}$, $\forall q,r \in K$

Transitions $((s_1, a, X), (s_2, Y))$ becomes a set of rules:

$\langle s_1, X, q\rangle \rightarrow a \langle s_2, Y, q\rangle$ for $a \in \Sigma \cup \{\varepsilon\}$, $\forall q \in K$

Transitions $((s_1, a, X), (s_2, \varepsilon))$ become a rule:

$\langle s_1, X, s_2\rangle \rightarrow a$ for $a \in \Sigma \cup \{\varepsilon\}$

Creating Productions from Transitions

	S → <s, Z, f'>	[1]
((s', ε, ε), (s, Z))		
((s, a, Z), (s, aZ))	<s, Z, f'> → a <s, a, f> <f, Z, f'>	[2]
	<s, Z, s> → a <s, a, f> <f, Z, s>	[x]
	<s, Z, f> → a <s, a, s> <s, Z, f>	[x]
	<s, Z, s> → a <s, a, s> <s, Z, f>	[x]
	<s, Z, s'> → a <s, a, f> <f, Z, s'>	[x]
((s, a, a), (s, aa))	<s, a, f> → a <s, a, f> <f, a, f>	[3]
((s, a, b), (s, ab))	…	
((s, b, Z), (s, bZ))	…	
((s, b, a), (s, ba))	<s, a, f> → b <s, b, f> <f, a, f>	[4]
((s, b, b), (s, bb))	…	
((s, c, Z), (f, Z))	…	
((s, c, a), (f, a))	<s, a, f> → c <f, a, f>	
((s, c, b), (f, b))	<s, b, f> → c <f, b, f>	[5]
((f, a, a), (f, ε))	<f, a, f> → a <f, ε, f>	[6]
((f, b, b), (f, ε))	<f, b, f> → b <f, ε, f>	[7]
((f, ε, Z), (f', ε))	<f, Z, f'> → ε <f', ε, f'>	[8]
	<f, ε, f> → ε	[9]
	<f' ε, f'> → ε	[10]

Comparing Regular and Context-Free Languages

Regular Languages

- regular exprs.
  - or
- regular grammars
- recognize
- = DFSAs

Context-Free Languages

- context-free grammars
- parse
- = NDPDAs