CFG and PDA accept the same languages

Sipser pages 115 - 122

Equivalence of CFGs and PDAs

The equivalence is expressed by two theorems.

Theorem 1. Every context-free language is accepted by some PDA.

Theorem 2. For every PDA M, the language L(M) is context-free.

We will describe the constructions, see some examples and proof ideas.

Lemma 2.21 (page 115 Sipser)

Given a CFG $G=(V,T,P,S)$, we define a PDA
$M=(\{q_{start},q_{loop},q_{accept}\},T, T\cup V\cup\{\$\}, \delta, q_{accept},\{q_{start}\})$,
with δ given by

- $\delta(q_{start},\varepsilon,\varepsilon) = \{(q_{loop},S\$)\}$
- If $A \in V$, then $\delta(q_{loop},\varepsilon,A) = \{ (q_{loop},\alpha) \mid A \to \alpha \text{ is in } P\}$
- If $a \in T$, then $\delta(q_{loop},a,a) = \{ (q_{loop},\varepsilon) \}$
- $\delta(q_{loop},\varepsilon,\$) = \{(q_{accept},\varepsilon)\}$

1. Note that the stack symbols of the new PDA contain all the terminal and non-terminals of the CFG plus \$
2. There is only 3 states in the new PDA, all the rest of the info is encoded in the stack.
3. Most transitions are on ε, one for each production
4. A few other transitions come one for each terminal.
5. The start and accept state each add a transition and use the marker \$

The automaton simulates leftmost derivations of G producing w, accepting by empty stack. For every intermediate sentential form $uA\alpha$ in the leftmost derivation of w (note first that $w = uv$ for some v), M will have $A\alpha$ on its stack after reading u. At the end (case $u=w$ and $v=\varepsilon$) the stack will be empty.

Example

For our old grammar: $S \to SS \mid (S) \mid \varepsilon$

The automaton M will have seven transitions, most from q_{loop} to q_{loop}:

1. $\delta(q_{start}, \varepsilon, \varepsilon) = (q_{loop}, S\$)$
2. $\delta(q_{loop}, \varepsilon, S) = (q_{loop}, SS)$ — $S \to SS$
3. $\delta(q_{loop}, \varepsilon, S) = (q_{loop}, (S)\ )$ — $S \to (S)$
4. $\delta(q_{loop}, \varepsilon, S) = (q_{loop}, \varepsilon\ )$ — $S \to \varepsilon$
5. $\delta(q_{loop}, (, (\ ) = (q_{loop}, \varepsilon)$
6. $\delta(q_{loop},),)\ ) = (q_{loop}, \varepsilon)$
7. $\delta(q_{loop}, \varepsilon, \$) = (q_{accept}, \varepsilon)$

1. Most transitions are on ε, one for each production
2. A few other transitions come one for each terminal
3. Or from the start and accept conditions

Compare

Now compare the leftmost derivation

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$

with the looping part of M's execution on the same string given as input:

```
(q, "(())()" ,S      )  |- [1]
(q, "(())()" ,SS     )  |- [2]
(q, "(())()" ,(S)S   )  |- [4]
(q, "())()"  ,S)S    )  |- [4]
(q, "())()"  ,(S))S  )  |- [4]
(q, "))()"   ,S))S   )  |- [3]
(q, "))()"   ,))S    )  |- [5]
(q, ")()"    ,)S     )  |- [5]
(q, "()"     ,S      )  |- [2]
(q, "()"     ,(S)    )  |- [4]
(q, ")"      ,S)     )  |- [3]
(q, ")"      ,)      )  |- [5]
(q, ε        ,ε      )
```

2.	$\delta(q, \varepsilon, S) = (q, SS)$	$S \rightarrow SS$
3.	$\delta(q, \varepsilon, S) = (q, (S))$	$S \rightarrow (S)$
4.	$\delta(q, \varepsilon, S) = (q, \varepsilon)$	$S \rightarrow \varepsilon$
5.	$\delta(q, (, () = (q, \varepsilon)$	
6.	$\delta(q,),)) = (q, \varepsilon)$	

Note we write q for q_{loop} for brevity

Transitions simulate left-most derivation

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$$

2.	$\delta(q, \Lambda, S) = (q, SS)$	$S \to SS$
3.	$\delta(q, \Lambda, S) = (q, (S))$	$S \to (S)$
4.	$\delta(q, \Lambda, S) = (q, \varepsilon)$	$S \to \varepsilon$
5.	$\delta(q, (, ( ) = (q, \varepsilon)$	
6.	$\delta(q,),)) = (q, \varepsilon)$	

```
(q, "(())()" ,S      )  |- [1]
(q, "(())()" ,SS     )  |- [2]
(q, "(())()" ,(S)S   )  |- [4]
(q, "())()"  ,S)S    )  |- [4]
(q, "())()"  ,(S))S  )  |- [4]
(q, "))()"   ,S))S   )  |- [3]
(q, "))()"   ,))S    )  |- [5]
(q, ")()"    ,)S     )  |- [5]
(q, "()"     ,S      )  |- [2]
(q, "()"     ,(S)    )  |- [4]
(q, ")"      ,S)     )  |- [3]
(q, ")"      ,)      )  |- [5]
(q, ε        ,ε      )
```

Note there is an entry in δ for each terminal and non-terminal symbol. The stack operations mimic a top down parse, replacing Non-terminals with the rhs of a production.

Note we write q for q_{loop} for brevity

Proof Outline

To prove that every string of L(G) is accepted by the PDA M, prove the following more general fact:

If $S \Rightarrow_{left\text{-}most}^{*} \alpha$ then (q_{loop}, uv, S) |-* (q_{loop}, v, β)

where $\alpha = u\beta$ is the "leftmost factorization" of α (u is the longest prefix of α that belongs to T^*, i.e. all terminals).

For example: if α = abcWdXa then u = abc, and β = WdXa, since the next symbol after abc is W$\in$V (a non-terminal or ε)

$S \Rightarrow_{lm}^{*}$ abcW... then $(q_{loop}$, abcV,S) |-* $(q_{loop}$,V, W...)

The proof is by induction on the length of the derivation of α.

We also need to prove that every string accepted by M belongs to L(G). Again, to make induction work, we need to prove a slightly more general fact:

If $(q_{loop}, w, A) \vdash^{*} (q_{loop}, \varepsilon, \varepsilon)$, then $A \Rightarrow^{*} w$

For all Stacks A, letting A = Start we have our proof.

This time we induct on the length of execution of M that leads from the ID $(q_{loop}, w, A\$)$ to $(q_{loop}, \varepsilon, \$)$.

Grammar from a PDA

lemma 2.27 Sipser pg 119

Assume the $M = (Q,\Sigma,\Gamma,\delta,q_0,F)$ is given, and that it accepts by empty stack.

Alter it so that it has the following additional properties

1. It has a single accept state
2. Each transition either
   1. Pushes a symbol onto the stack
   2. Or pops a symbol off the stack
   3. But not both

Why can we do this? (hint add new states)

Symbols of the CFG

For every pair of states p,q $\in$ Q

Make a variable (non-terminal) A_{pq}

A symbol A_{pq} should derive a string if that string cause the PDA to move from state p (with an empty stack) to state q (with an empty stack).

Such strings can do the same, starting and ending with the same arbitrary stack. Why?

Productions of the CFG

Consider a string x that moves the PDA from p to q on empty stack.

1. The first move must be a push (why?)
2. The last move must be a pop (why?)

(p,x,ε) |- (_,_,Z) |- ... |- (_,_,T) -| (q,ε,ε)

There are 2 cases (Z=T)=True or (Z=T)=False

1. (Z=T)=True

   Stack is only empty at the beginning and at the end.

   (p,ay,ε) |- (r,y,Z) |- ... |- (s,b,T) -| (q,ε,ε)

$$A_{pq} \rightarrow a\ A_{rs} b$$

2. (Z=T)=False

   the stack is empty in the middle, at some state r

   (p,x,ε) |- ... (r,_, ε) |- ... -| (q,ε,ε)

$$A_{pq} \rightarrow A_{pr}\ A_{rq}$$

Given $M = (Q,\Sigma,\Gamma,\delta,s,\{f\})$

Construct $G = (V,\Sigma,R,S)$

$V = \{ A_{pq} \mid p,q \in Q\}$

$S = A_{sf}$

$\Sigma = \Sigma$

R = cases

1. For each $p \in Q$ — $A_{pp} \rightarrow \varepsilon$
2. For each $p,q,r \in Q$ — $A_{pq} \rightarrow A_{pr}\, A_{rq}$
3. For each $p,q,r,s \in Q$

   $T \in \Gamma$ $a,b \in \Sigma_\varepsilon$

   $(r,T) \in \delta(p,a,\varepsilon)$ $(q,\varepsilon) \in \delta(s,b,T)$

   $A_{pq} \rightarrow a\, A_{rs} b$

$(p,ay,\varepsilon) \vdash (r,y,Z) \vdash \dots \vdash (s,b,T) \dashv (q,\varepsilon,\varepsilon)$

Claim 2.30

If A_{pq} generates x, then x can bring the PDA from p with empty stack to q with empty stack

$A_{pq} \Rightarrow^* x$ implies $(p,x,\varepsilon) \vdash^* (q,\varepsilon,\varepsilon)$

Proof by induction on the number of steps in the derivation $A_{pq} \Rightarrow^* x$

Claim 2.31

If x can bring the PDA from p with empty stack to q with empty stack then A_{pq} generates x

$(p,x,\varepsilon) \mid\!-^* (q,\varepsilon,\varepsilon)$ implies $A_{pq} \Rightarrow^* x$

Proof by induction on the length of

$$(p,x,\varepsilon) \mid\!-^* (q,\varepsilon,\varepsilon)$$

The following is additional material for the curious.

It gives a second construction not described in Sipser.

It is not required.

An another algorithm for CFG from a PDA

Assume the M = $(Q,\Sigma,\Gamma,\delta,q_0,F)$ is given, and that it accepts by empty stack. Consider execution of M on an accepted input string.

If at some point of the execution of M the stack is $Z\zeta$ (Z is on top, ζ is the rest of stack)

In terms of instantaneous descriptions

(state$_i$, input, $Z\zeta$) $|-$ . . .

Then we know that eventually the stack will be ζ.

Why? Because we assume the input is accepted, and M accepts by empty stack, so eventually Z must be removed from the stack

$$(state_i,\ \alpha X,\ \ Z\zeta)\ \ \vdash^*\ \ (state_j,\ X,\ \ \zeta)$$

The sequence of moves between these two instants is the “net popping” of Z from the stack.

During this sequence of moves, the stack may grow and shrink several times, some input will be consumed (the α), and M will pass through a sequence of states, from $state_i$ to $state_j$.

Net Popping

Net popping is fundamental for the construction of a CFG G equivalent to M.

We will have a variable (Non-terminal) [qZp] in the CFG G for every triple in (q,Z,p) $\in Q \times \Gamma \times Q$ from the PDA. Recall

1. Q is the set of states
2. Γ Is the set of stack symbols

We want the rhs of a production whose lhs is [qZp] to generate precisely those strings $w \in \Sigma^*$ such that M can move from q to p while reading the input w and doing the net popping of Z. A production like [qZp] -> ?

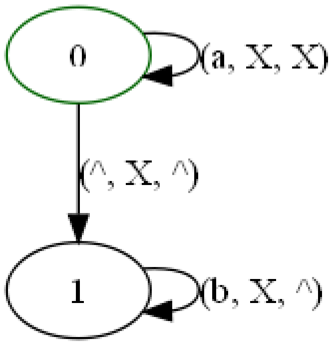
This can be also expressed as (q,w,Z) |-* (p, Λ , Λ)

Productions of G correspond to transitions of M.

If $(p,\zeta) \in \delta(q,a,Z)$, then there is one or more corresponding productions, depending on complexity of ζ.

1. If $\zeta = \Lambda$, we have $[qZp] \to a$
2. If $\zeta = Y$, we have $[qZr] \to a[pYr]$ for every state r
3. If $\zeta = YY'$ we have $[qZs] \to a[pYr][rY's]$, for every pair of states r and s.
4. You can guess the rule for longer ζ.

Example



Q = {0,1}

S = {a,b}

Γ = {X}

δ(0,a,X) = { (0,X) }

δ (0,Λ,X) = { (1,Λ) }

δ (1,b,X) = { (1,Λ) }

$Q_{0 = 0}$

Z_0 = X

F = {}, accepts by empty stack

Non-terminals

(q,Z,p) $\in$ Q$\times\Gamma\times$Q

(0,'X',0)

(0,'X',1)

(1,'X',0)

(1,'X',1)

Productions, At least one from each element in delta

(p,z) $\in$ δ(q,a,Z)

(0,a,X,0,X)

(1,b,X,1,Λ)

(0,Λ,X,1,Λ)]

0X0 -> a 0X0

0X1 -> a 0X1

1X1 -> b

0X1 -> Λ