PDAs and CFLs:

Theorem : Herhangi bir bağlamdan bağımsız L dili için, L = L(M) eşitliğini sağlayan bir M NPDA vardır.

PDAs and CFLs:

Proof:

- L bir Context Free Dil ise, bu dili oluşturan bir G Context Free Gramer vardır.
- Bir CFG'yi her zaman Greibach Normal Form'a (GNF) dönüştürebiliriz.
- Greibach Normal Formdaki gramerin soldan türetimlerini (leftmost derivations) gösteren bir NPDA her zaman oluşturulabilir.

Greibach Normal Form:

Greibach Normal Form (GNF) for Context-Free Grammars requires the Context-Free Grammar to have only productions of the following form:

$A \rightarrow ax$

where $a \in T$ and $x \in V^*$. That is,

Nonterminal $\rightarrow$ one Terminal concatenated with a string of 0 or more Nonterminals

Convert the following Context-Free Grammar to GNF:

$S \rightarrow abSb \mid aa$

Greibach Normal Form:

$S \rightarrow abSb \mid aa$

İlk olarak $S \rightarrow aa$ kuralının sonundaki terminal sembol yerine $S \rightarrow aA$ yazıp, yeni bir kural oluşturalım $A \rightarrow a$.

$S \rightarrow abSb$ kuralında da bSb yerine $S \rightarrow aX$ yazalım ve $X \rightarrow bSb$ yeni kural oluşturalım.

Unfortunately, this rule itself needs fixing. Replace the rule with $X \rightarrow bSB$ by creating a new rule, $B \rightarrow b$.

Greibach Normal Form:

So, starting with this set of production rules:

$S \longrightarrow abSb \mid aa$

we now have: (GNF)

$S \longrightarrow aA \mid aX$
$X \longrightarrow bSB$
$A \longrightarrow a$
$B \longrightarrow b$

(other solutions are possible)

CFG to PDA

Bir context-free grammar'i eşdeğer PDA'ya dönüştürmek için:

1. Grameri Greibach Normal Form (GNF)'a dönüştürün.
2. PDA için, S (the Start symbol in the grammar)'i yığına atan bir geçiş kuralı(transition rule) yazın.
3. Gramerdeki her türetim kuralı (production rule) için, eşdeğer geçiş kuralını yazın.
4. Girdi dizgisindeki semboller bittiğinde otomatı kabul durumuna götürecek ve yığın da boş olacak şekilde bir geçiş kuralı yazın.
5. Boş dizgi gramer tarafından tanımlanan dil tarafından kabul ediliyorsa, otomatı başlangıç durumundan kabul durumuna direk götüren bir geçiş kuralı yazılır.

CFG to PDA

How do you write the transition rules? It's really simple:

1. Every rule in the GNF grammar has the following form:

One variable $\rightarrow$ one terminal + 0 or more variables

Example: $\mathbf{A} \rightarrow \mathbf{bB}$

CFG to PDA

2. Her geçiş kuralının sol tarafı, bir sonraki duruma geçmeden önce sağlanması gereken koşulları gösteren bir üçlüdür.
Bu üçlü, o anki durum, girdi dizgisinden o anda okunan sembol, yığının en üstünden o anda çıkarılan semboldür.
Bu nedenle geçiş kuralında bu üçlü şu şekilde yazılır:
o anki durum, gramer kuralındaki terminal ve gramer kuralında soldaki değişken

Our grammar rule: $\mathbf{A \rightarrow bB}$

Geçiş kuralının sol tarafı bu şekilde oluşturulur (precondition): $\delta(\mathbf{q_1, b, A})$

CFG to PDA

3. Geçiş kuralının sağ tarafı *post-condition* (sağlanması istenen koşul) olarak adlandırılır. Post-condition da gidilecek durum ve yığına atılacak sembol(ler)den oluşur. Böylece bu geçiş kuralı için post-condition, gidilecek durum ve gramer kuralının sağ tarafındaki değişken(ler) olacaktır.

Example: $\delta(q_1, b, A) = \{\boldsymbol{(q_1, B)}\}$

CFG to PDA

Eğer gramer kuralının sağ tarafında hiçbir değişken yok ise, yığına hiçbirşey atılmaz. Geçiş kuralında, λ konur.

Example: **A** $\rightarrow$ **a**[no variable here]
would be represented in transition rule form as:

$\delta(q_1, a, A) = \{\mathbf{(q_1, \lambda)}\}$

CFG to PDA

How do you know which state to move to? It's simple:

1. We always *start off* with this special transition rule:

   $\boldsymbol{\delta(q_0, \lambda, \#) = \{(q_{1,}\ S\#)\}}$

This rule says:

a. q_0 durumundan başlanır.

b. Yığının en üstündeki eleman alınır. Eğer # (the empty stack symbol) ise, o halde

c. Girdi dizgisinden bir şey okunmadan q_1 durumuna geçilir ve yığına tekrar # atılır ve daha sonra S (the Start symbol in the grammar) yığına atılır.

CFG to PDA

2. We always *end up* with this special transition rule:

   $\boldsymbol{\delta(q_1, \lambda, \#) = \{(q_2, \#)\}}$

This rule says:

a. q_1 durumundan başlanır.
b. Yığının en üstündeki eleman alınır. Eğer bu # (the empty stack symbol) ise bu durumda,
c. Girdi dizgisinden birşey okunmadan q_2 durumuna geçilir ve yığına # atılır.

q_1 durumunda olmak için yığına daha önce birşey atmış olmamız gerekmektedir. q_1 durumunda iken yığının en üstündeki elemanı alırsak ve boş yığın sembolünü (#) bulursak, bu bize dizginin işlenmesinin bittiğini gösterir. Böylece q_2 durumuna gidebiliriz.

CFG to PDA

3. Diğer geçiş kurallarının hepsi bizi q_1 durumunda bırakır.

CFG to PDA

Örnek: GNF gramerini ele alalım: G = (V, T, S, P)

V = {S, A, B, C},

T = {a, b, c,},

S = S,

ve P :

$S \longrightarrow aA$

$A \longrightarrow aABC \mid bB \mid a$

$B \longrightarrow b$

$C \longrightarrow c$

Bu grameri bir PDA'ya dönüştürelim.

CFG to PDA

Gramer kuralı:	*PDA geçiş kuralı:*
(always)	1. $\delta(q_0, \lambda, Z) = \{(q_1, SZ)\}$
$S \rightarrow aA$	2. $\delta(q_1, a, S) = \{(q_1, A)\}$
$A \rightarrow aABC \mid a$	3. $\delta(q_1, a, A) = \{(q_1, ABC), (q_1, \lambda)\}$
$A \rightarrow bB$	4. $\delta(q_1, b, A) = \{(q_1, B)\}$
$B \rightarrow b$	5. $\delta(q_1, b, B) = \{(q_1, \lambda)\}$
$C \rightarrow c$	6. $\delta(q_1, c, C) = \{(q_1, \lambda)\}$
(always)	7. $\delta(q_1, \lambda, Z) = \{(q_f, \lambda)\}$

Böylece eşdeğer PDA şu şekilde tanımlanabilir:
M = $(\{q_0, q_1, q_2\}, T, V \cup \{Z\}, \delta, q_0, Z, \{q_2\})$, burada δ yukarda verilen geçiş kuralları kümesidir.

CFG to PDA

Bu gramer gerekirci midir?

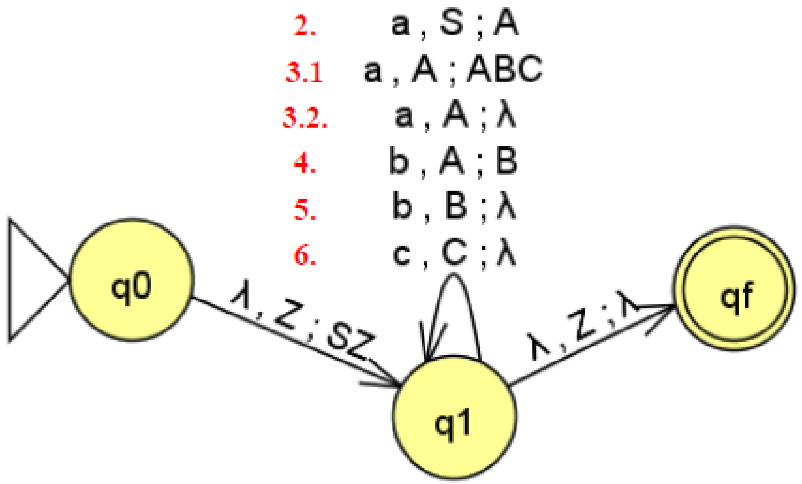
1. $\delta(q_0, \lambda, Z) = \{(q_{1,} SZ)\}$
2. $\delta(q_1, a, S) = \{(q_{1,} A)\}$
3. **$\boldsymbol{\delta(q_1, a, A) = \{(q_{1,} ABC), (q_{1,} \lambda)\}}$**
4. $\delta(q_1, b, A) = \{(q_{1,} B)\}$
5. $\delta(q_1, b, B) = \{(q_{1,} \lambda)\}$
6. $\delta(q_1, c, C) = \{(q_{1,} \lambda)\}$
7. $\delta(q_1, \lambda, Z) = \{(q_{2,} Z)\}$

Görüyoruz ki 3 numaralı kural aynı önkoşula ve iki farklı post-condition'a sahiptir. Bu nedenle bu PDA nondeterministic'tir.

CFG to PDA

aaabc dizgisinin, başlangıç önkoşulu ile başlayarak, bu PDA tarafından nasıl işlendiğine bakalım:

$$
\begin{aligned}
(q_0, aaabc, Z) \quad & \overset{1}{\vdash} (q_1, aaabc, SZ) \\
& \overset{2}{\vdash} (q_1, aabc, AZ) \\
& \overset{3.1}{\vdash} (q_1, abc, ABCZ) \\
& \overset{3.2}{\vdash} (q_1, bc, BCZ) \\
& \overset{5}{\vdash} (q_1, c, CZ) \\
& \overset{6}{\vdash} (q_1, \lambda, Z) \\
& \overset{7}{\vdash} (q_2, \lambda, \lambda)
\end{aligned}
$$



CFG'de buna ait türetim:

$$S \Rightarrow aA \Rightarrow aaABC \Rightarrow aaaBC \Rightarrow aaabC \Rightarrow aaabc$$

Simple example: $G = (\{S, A\}, \{a, b\}, S, \{S \to aSA \mid a, A \to aA \mid b\})$

production	transition
(always)	$\delta(q_0, \lambda, Z) = \{(q_1, SZ)\}$
$S \to aSA \mid a\lambda$	$\delta(q_1, a, S) = \{(q_1, SA), (q_1, \lambda)\}$
$A \to aA$	$\delta(q_1, a, A) = \{(q_1, A)\}$
$A \to b\lambda$	$\delta(q_1, b, A) = \{(q_1, \lambda)\}$
(always)	$\delta(q_1, \lambda, Z) = \{(q_f, \lambda)\}$

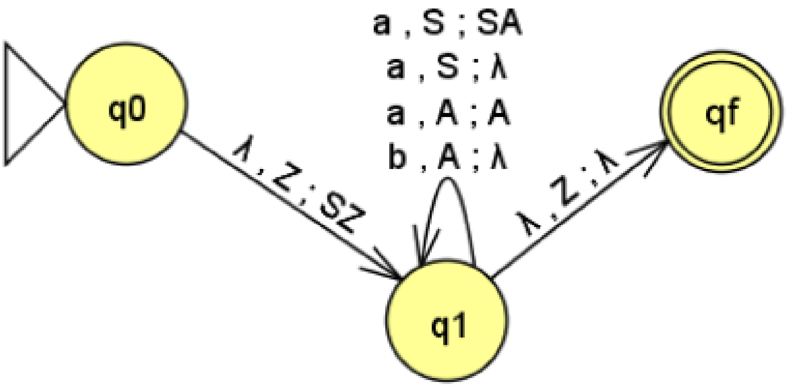
- GNF grammar

$$S \to aSA \mid a$$
$$A \to aA$$
$$A \to b$$

- Derivation of $w = aaabb$

$$S \Rightarrow aSA \Rightarrow$$
$$aaSAA \Rightarrow aaaAA \Rightarrow$$
$$aaabA \Rightarrow aaabb$$

- Equivalent npda
  - recall: (input, popped, pushed)



- Acceptance of $w = aaabb$

$$(q_0, aaabb, Z) \vdash (q_1, aaabb, SZ) \vdash$$
$$(q_1, aabb, SAZ) \vdash (q_1, abb, SAAZ) \vdash$$
$$(q_1, bb, AAZ) \vdash (q_1, b, AZ) \vdash$$
$$(q_1, \lambda, Z) \vdash (q_f, \lambda, \lambda)$$

CFG’den PDA oluşturmak için alternatif yaklaşım

G = (V, T, S, P) bir CFG (context-free grammar) olsun. Bu durumda L(M) = G eşitliğini sağlayan bir M yığın yapılı otomat (PDA) vardır.

Bir CFG’den, önce GNF’ye dönüştürmeden, bir NPDA oluşturabilir miyiz? Evet

Alternative Approach to Constructing a PDA from a CFG

Bu yaklaşımda da türetim kurallarının başı ve sonu GNF yöntemi ile aynıdır.
Bu nedenle PDA'mızdaki türetim kurallarında her zaman aşağıdaki 2 türetim kuralı olacaktır:

$$\delta\,(q_0, \lambda, \#) = \{(q_1, S\#)\}$$

ve

$$\delta\,(q_1, \lambda, \#) = \{(q_2, \#)\}$$

Alternative Approach to Constructing a PDA from a CFG

Diğer türetim kuralları gramer kurallarından elde edilir:

• Yığının en üstündeki eleman çıkarılırsa (POP) ve bu bir değişken(nonterminal) ise girdi dizgisinden hiçbirşey okunmaz. Bu değişkeni içeren gramer kuralının sağ tarafı yığına atılır.

• Yığının en üstündeki eleman çıkarılırsa (POP) ve bu bir terminal ise, girdi dizgisinin bir sonraki karakteri okunur. Yığına hiçbirşey atılmaz.

Constructing a PDA from a CFG

Given $G = (V, T, S, P)$,

construct $M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F,)$, with

$Q = \{q_0, q_1, q_2\}$

$\Gamma = V \cup \Sigma \cup \{\#\} \mid \# \notin V \cup \Sigma$

$F = q_2$

$\delta(q_0, \lambda, \#) = \{(q_1, S\#)\}$

For $A \in V$, $\delta(q_1, \lambda, A) = \{(q_1, \alpha)\}$, where $A \to \alpha$

For $a \in \Sigma$, $\delta(q_1, a, a) = \{(q_1, \lambda)\}$

$\delta(q_1, \lambda, \#) = \{(q_2, \#)\}$

Constructing a PDA from a CFG

Dil:

$L = \{x \in \{a, b\}^* \mid n_a(x) > n_b(x)\}$

Context-free grammar:

$S \rightarrow a \mid aS \mid bSS \mid SSb \mid SbS$

Constructing a PDA from a CFG

$S \rightarrow a \mid aS \mid bSS \mid SSb \mid SbS$

M = (Q, Σ, Γ, q_0, z, A, δ), daha önce tanımladığımız PDA olsun. Türetim kuralları şöyle olacaktır:

Rule #	**State**	**Input**	**Top of Stack**	**Move(s)**
1	q_0	λ	#	$(q_1, S\#)$
2	q_1	λ	S	(q_1, a), (q_1, aS), (q_1, bSS), (q_1, SSb), (q_1, SbS)
3	q_1	a	a	(q_1, λ)
4	q_1	b	b	(q_1, λ)
5	q_1	λ	#	$(q_2, \#)$

CFG to PDA

baaba dizgisinin işlenmesi:

(q_0, baaba, #) |- (q_1, baaba, S#) rule 1

|- (q_1, baaba, bSS#) rule 2, 3rd alternative

|- (q_1, aaba, SS#) rule 4

|- (q_1, aaba, aS#) rule 2, 1st alternative

|- (q_1, aba, S#) rule 3

|- (q_1, aba, SbS#) rule 2, 5th alternative

|- (q_1, aba, abS#) rule 2, 1st alternative

|- (q_1, ba, bS#) rule 3

|- (q_1, a, S#) rule 4

|- (q_1, a, a#) rule 2, 1st alternative

|- (q_1, λ, #) rule 3

|- (q_2, λ, #) rule 5