KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BIL 348 OTOMATA TEORİSİ DERS NOTLARI

NUR ALTUN

2013-2014 Bahar Dönemi

~ Otomata Teorisi ~

→ Regular Expressions (RE)

'Σ' = Dilin en küçük yapı taşı alfabeyi temsil edicek.

* $\Sigma = \{a, b\}$ * $\Sigma = \{0, 1\}$

a^* ne anlama gelir? $a^* = \{\lambda, a, aa, aaa, aaaa, \dots\}$
↳ boş string (0)

$a^+ = \{a, aa, aaa, aaaa, \dots\}$ '+' = en az bir tane harf içerir. Boş string yok.

$ab^* = \{a, ab, abb, abbb, \dots\}$ → a ile başlamalı
↳ $a\lambda$

$(ab)^* = \{\lambda, ab, abab, ababab, \dots\}$ ⇒ $(ab)^* \neq a^* b^*$
↳ 1 veya daha fazla a veya b.

$(a+b) = \{a, b\}$
a veya b

$(a+b)(a+b) = \{aa, ab, ba, bb\}$ (kombinasyon)

$(a+b)^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, \dots\}$ → $(a+b)(a+b)(a+b)\dots$

$(a+b)^+ = (a+b)(a+b)^*$ → Boş string yok

Sonsuz sayıda üretilir. Genel bir ifade a ile b'den oluşan tüm stringler tanımlanabilir.

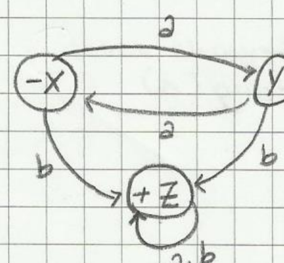
Örnek: 0 dahil çift sayıda harf içeren kelimelerin dili için bir RE yazınız.

$[(a+b)(a+b)]^*$ $(a+b)[(a+b)(a+b)]^*$ → Tek Sayıda

Örnek: a ile başlayan b ile biten tüm kelimelerin dili için RE yazınız.

$a(a+b)^*b$ → Aradaki String Önemsiz

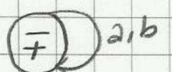
→ FINITE AUTOMATA (FA) (Sonlu Otomata)



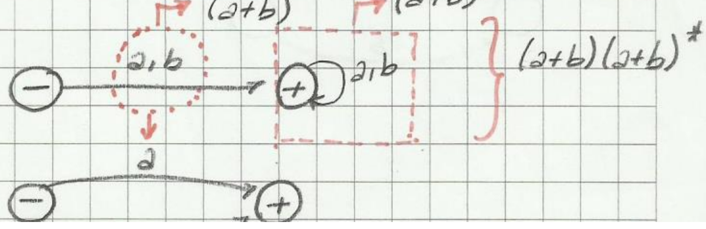
'−' = başlangıç
'+' = bitiş

	a	b
−X	Y	Z
Y	X	Z
+Z	Z	Z

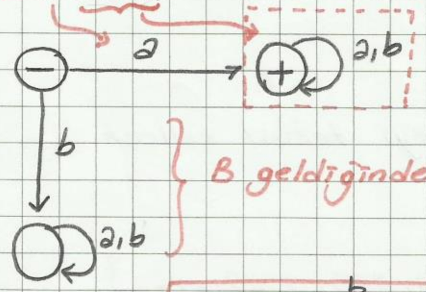
↳ durumlar



$= (a+b)^*$

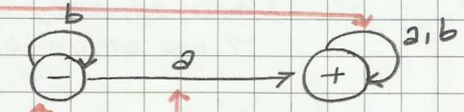


$a(a+b)^*$

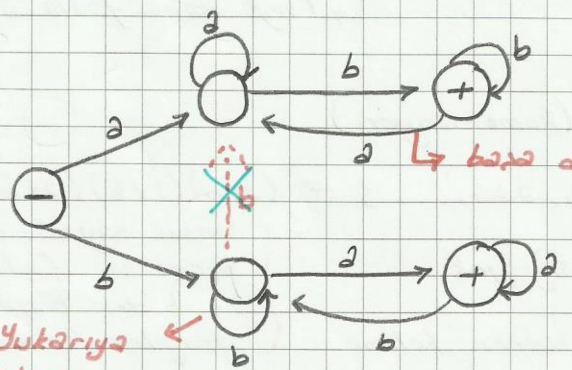


B geldiğinde bir yere gitmeli

$b^*a(a+b)^*$



★ **Örnek:** a ile başlayıp b ile biten veya b ile başlayıp a ile biten tüm kelimelerin dili için FA çiziniz.
(− → + 'ya giden tüm durumları minimum string)



↳ başa dönerse 'ababa' X

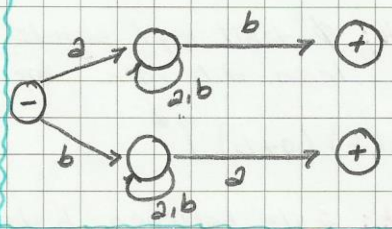
Yukarıya giderse "bbab" X

$(a(a+b)^*b + b(a+b)^*a)$

veya

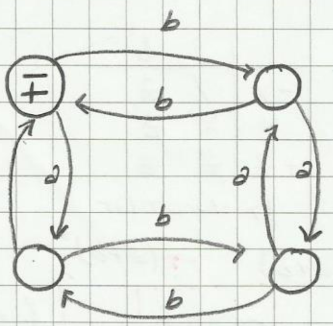
$(a^+b^+)^+ + (b^+a^+)^+$

$[a(a+b)^*b + b(a+b)^*a]$



Non-Deterministic FA

★ **Örnek:** λ dahil çift sayıda a'lardan veya b'lerden oluşan tüm kelimelerin dili için FA çiziniz.

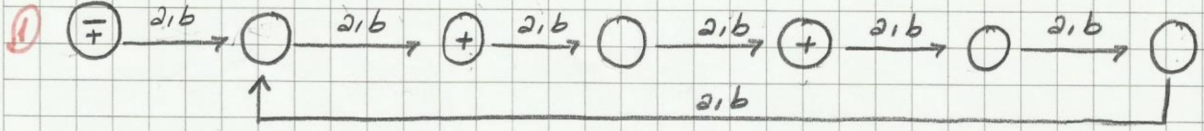


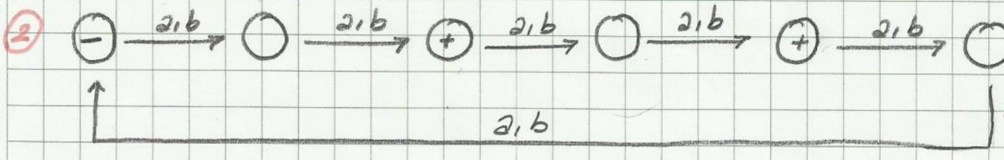
baba

'∓' = Bir durumda varsa boş string (λ)

★Örnek: λ dahil çift sayı uzunluklu ve uzunluğu 6 sayısına tam bölünmeyen tüm kelimelerin dili için FA çiziniz.

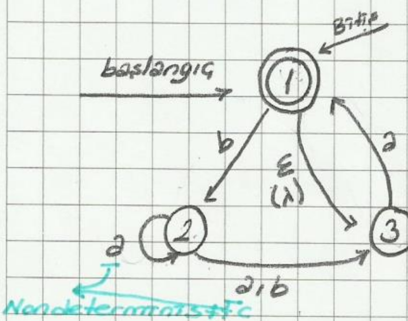
"6'ya tam bölünen yerde (+) olmaz!"





⇒ λ dahil değil ise (2)

⟹ Nondeterministic → Deterministic (NFA to FA)

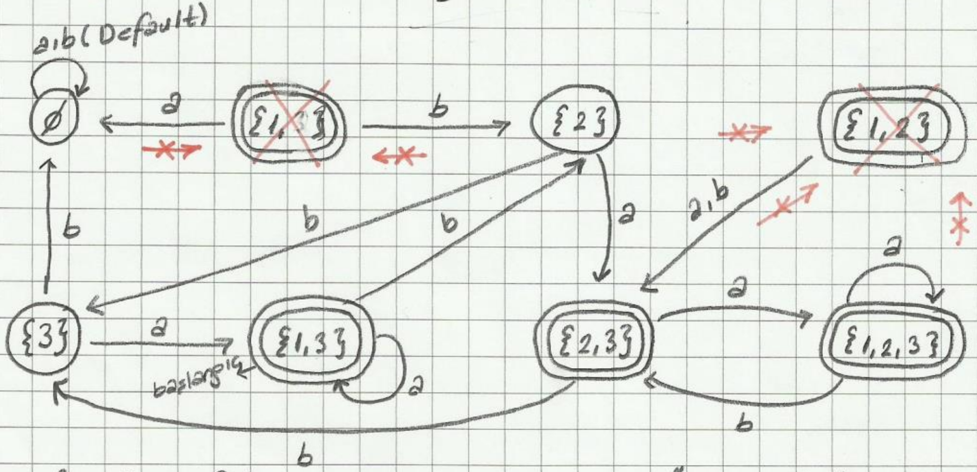


Kaç durum var ise; n ise 2^n durum incelenmeli.

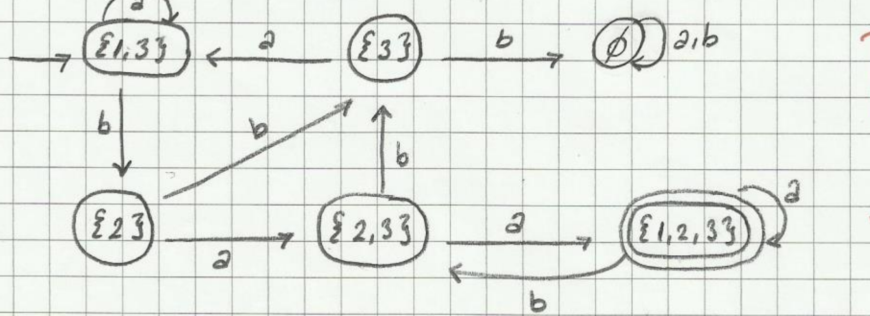
$n=3 \quad 2^3=8$

Başlangıç 1 veya 3 olabilir, ε dolayı.

3 → a geldiğinde ε dolayı geri dönebilir. O yüzden {1,3} olur.



NOT: {1,2} ve {1} durumuna gidilemiyor! "Gereksiz"

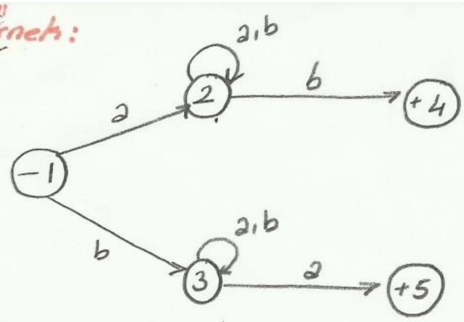


Deterministic

Michael Sipser'ın kitabında 58. sayfada {1} ve {1,2} gereksiz durumları hakkında : "We may simplify this machine by observing that no arrows point at states {1} and {1,2}, so they may be removed" yazıyor. Dolayısıyla {1} ve {1,2} durumlarına başka durumlardan geçiş olmadığı için gereksiz durum oldukları anlaşılıp silinmiştir.

2011-2012 Bahar Dönemi 1. Arasınav 2. sorusunun çözümünde kendilerine başka durumlardan geçiş olmayan {2} ve {2,3} durumlarının silinmesiyle bu durumlardan {3} durumuna olan geçişler ortadan kalktığından ve başka durumlardan da geçiş olmadığından {3} durumu da silinmiştir. {3} silinince de {0} durumuna geçişlerin hepsi ortadan kalkmış olacağından o da silinmiştir. Dolayısıyla kendisine geçiş olmayan durumlar silindiğinde bu durumlardan başka durumlara olan geçişler de silineceğinden yeni gereksiz durumlar çıkabilir.

*** Örnek:**



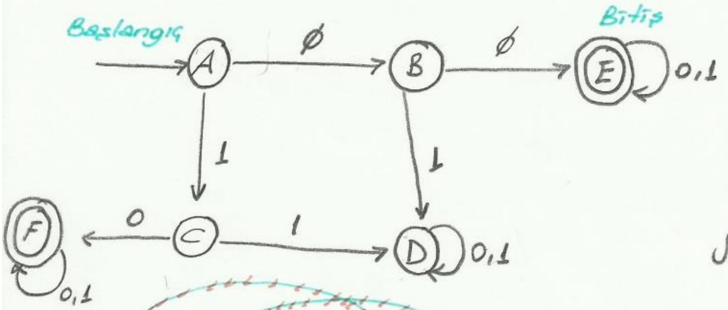
$a(a+b)^* b + b(a+b)^* a$

24.02.2014
Pazartesi.

NonDeterministic FA → FA çizilmişti.
$2^5 = 32$ duruma bakılması gerekir.
Son durumlarda nerelere gidileceği (a,b geldiğinde) belirtilmek zorunda değil.

~ NFA'de $\lambda\ (\varepsilon)$ geçişi vardır.
~ FA'de bütün durumların (a ve b geldiğinde) nereye gittiğini belirtmek durumundayız.

→ MINIMIZING FA



Durum sayısının 1 eksiği ile tablo yapılacak.

Bitiş durumları diğer durumlarla eşdeğer olamaz. İki bitiş durumu, olabilir. (E,F)

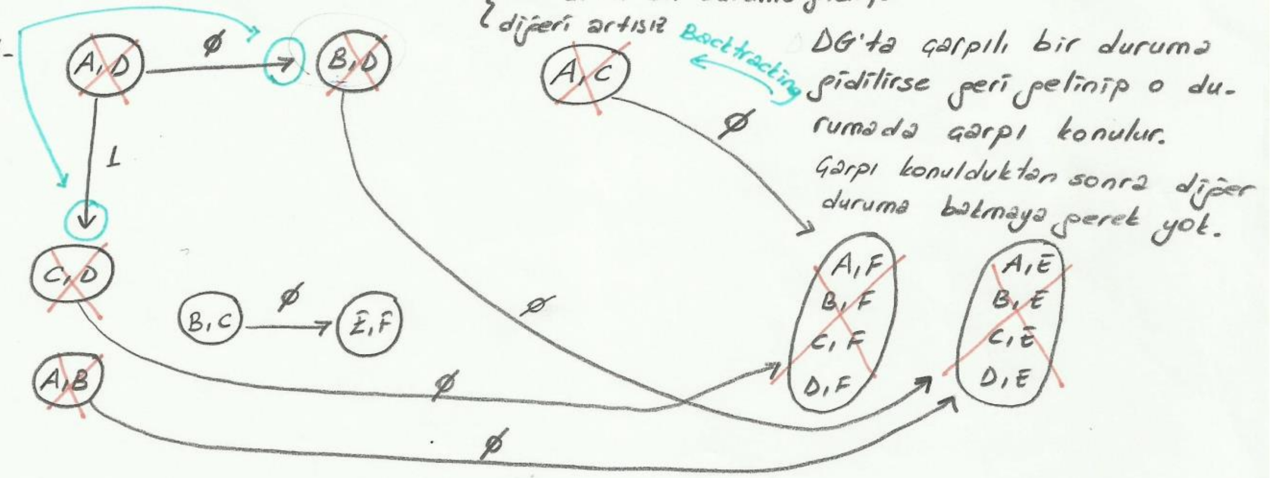
	F	E	D	C	B
A	X	X	X	X	X
B	A,C X	A,B X	A,D X		
C	X	X	A,D X		
D	C,D X	B,D X			
E	B,C				

'X' = Eşdeğer değil
Yoksa eşdeğer

Bir duruma hangi durumdan geçiş yapıldıysa durum kutucuğuna geçiş yapılan durum yazılmalı.

{B,C}
{E,F}
Eşdeğer Durumlar

Dependency Graph

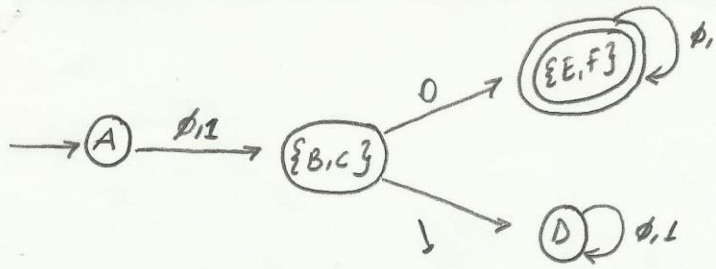


Okun yönüne göre x konulur.

{ Biri artılı bir duruma gidiyo
{ diğeri artısız

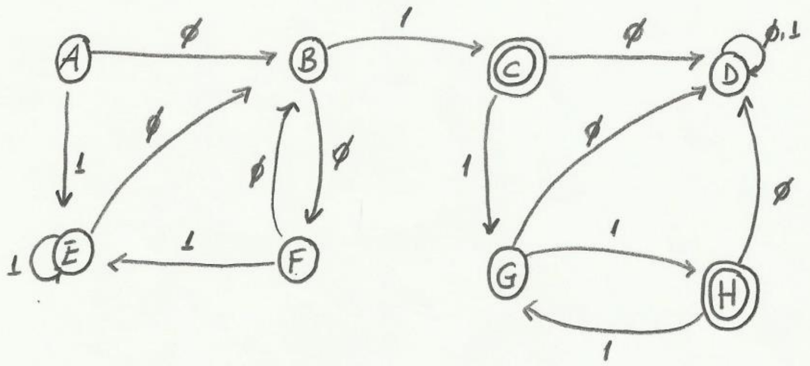
Backtracking

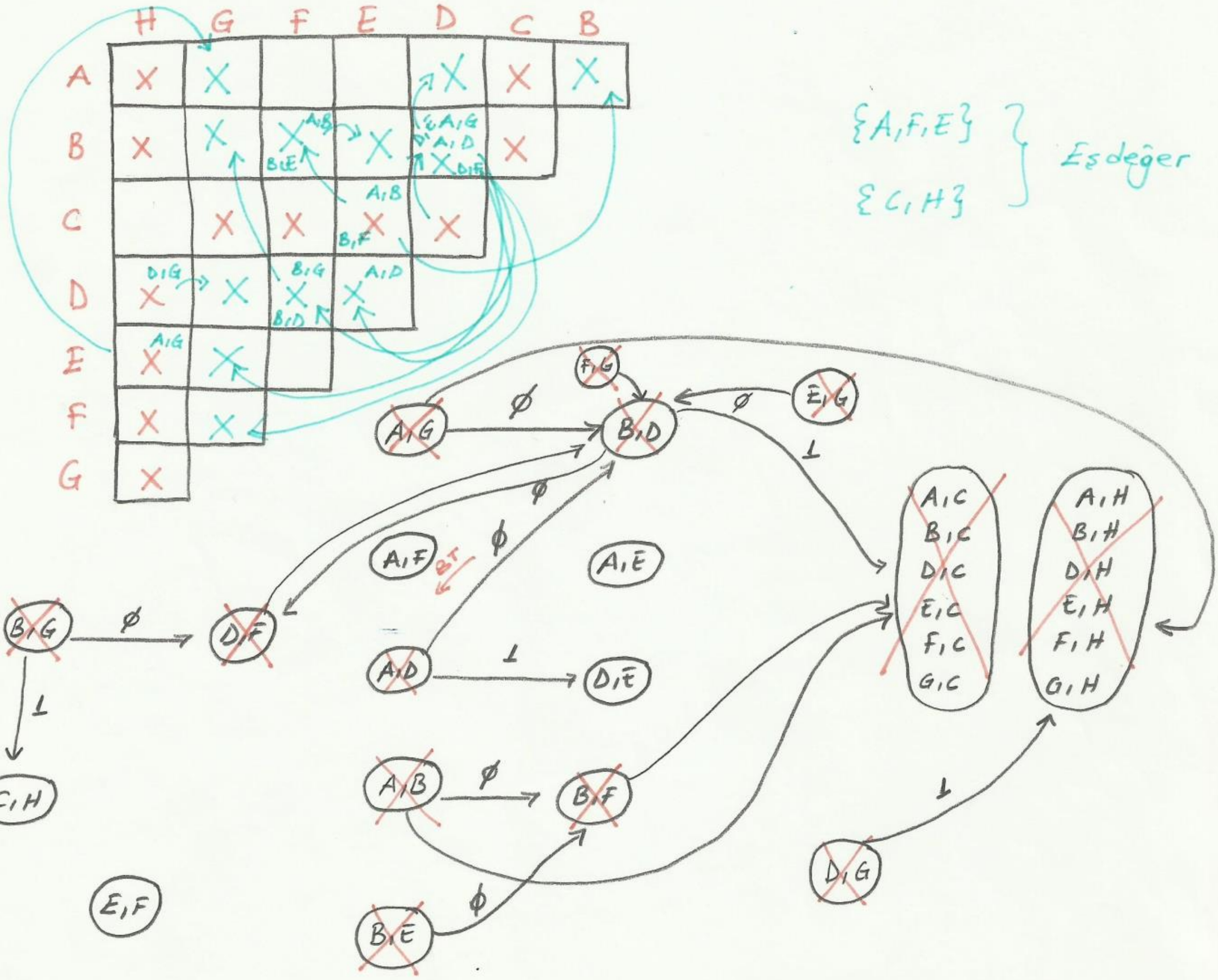
DG'ta çarpılı bir duruma gidilirse geri gelinip o durumada çarpı konulur.
Çarpı konulduktan sonra diğer duruma bakmaya gerek yok.



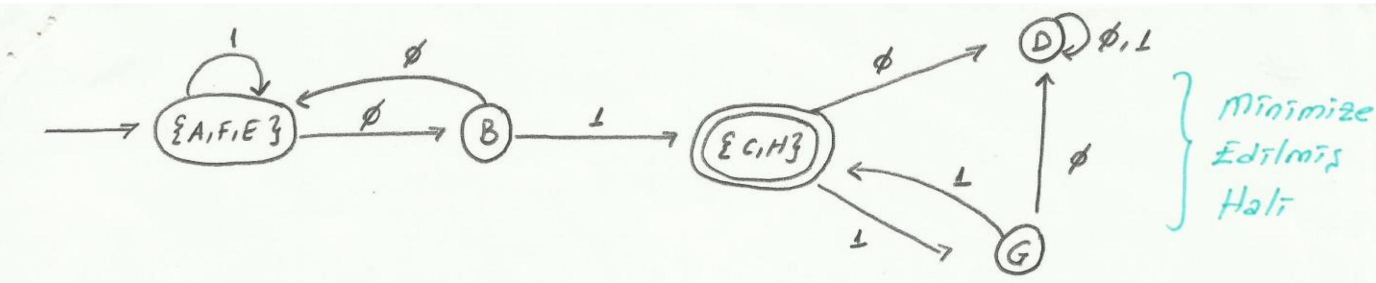
Minimize Edilmiş Hali

Örnek:





{A,F,E}
{C,H}
Eşdeğer



{A,F,E}
B
{C,H}
D
G
ϕ,1
Minimize
Edilmiş
Hali

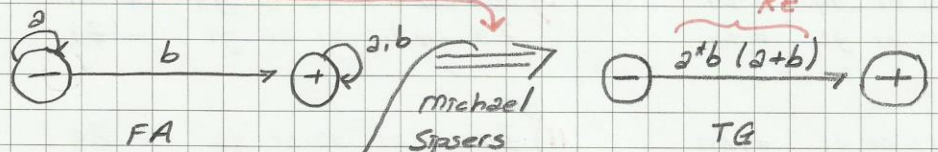
→ KLEENE'S THEOREM

3 bölümden oluşur.

I. Part: Every language that can be defined by a finite automaton can also be defined also by a transition graph.

II. Part: Every language that can be defined by a transition graph can also be defined by a regular expression.

III. Part: Every language than can be defined by a regular expression can also be defined by a finite automata.



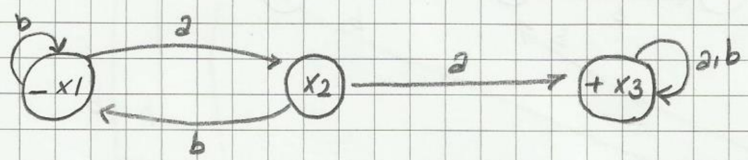
Michael Sipsers Book

Geçiş işlemle:
rindeki herbir FA aynı zamanda
bir TG

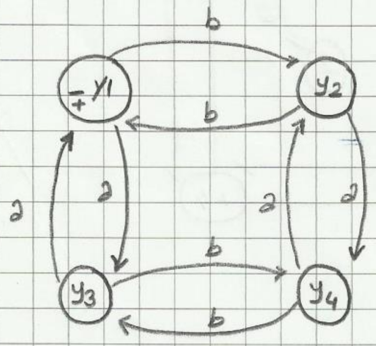
III. Partın Kural 2'si : If there is an FA called by FA_1 that accepts the language defined a regular expressions r_1 and there is an FA called FA_2 that accepts the language defined by the regular expression r_2 then there is an FA that we shall call FA_3 that accepts the language defined by the regular expressions.

$\underbrace{FA_1 + FA_2}_{1} = FA_3$
$(r_1 + r_2)$

Kuralın ispatı yapılacak!



⇒ FA_1



⇒ FA_2

	a	b
$\mp z_1$	z_2	z_3
z_2	z_4	z_5
z_3	z_6	z_1
$+z_4$	z_7	z_8
z_5	z_9	z_{10}
z_6	z_8	z_{10}
$+z_7$	z_4	z_{11}
$+z_8$	z_{11}	z_4
z_9	z_{11}	z_1
z_{10}	z_{12}	z_5
$+z_{11}$	z_8	z_7
$+z_{12}$	z_7	z_3

→ Dependency Graph ile aynı mantık

$\mp z_1 = x_1$ or y_1
$z_1 \to a : x_2$ veya $y_3 = z_2$
$z_1 \to b : x_1$ veya $y_2 = z_3$

\# max. Durum sayısı;
FA_1'deki durum sayısı × FA_2'deki durum sayısı } 3×4 = 12

FABER-CASTELL

2) $Z_2 \xrightarrow{a} : x_3 \text{ or } y_1 = +Z_4$
$Z_2 \xrightarrow{b} : x_1 \text{ or } y_4 = Z_5$

3) $Z_3 \xrightarrow{a} : x_2 \text{ or } y_4 = Z_6$
$Z_3 \xrightarrow{b} : x_1 \text{ or } y_1 = Z_1$

4) $Z_4 \xrightarrow{a} : x_3 \text{ or } y_3 = +Z_7$
$Z_4 \xrightarrow{b} : x_3 \text{ or } y_2 = +Z_8$

5) $Z_5 \xrightarrow{a} : x_2 \text{ or } y_2 = Z_9$
$Z_5 \xrightarrow{b} : x_1 \text{ or } y_3 = Z_{10}$

6) $Z_6 \xrightarrow{a} : x_3 \text{ or } y_2 = Z_8$
$Z_6 \xrightarrow{b} : x_1 \text{ or } y_3 = Z_{10}$

7) $Z_7 \xrightarrow{a} : x_3 \text{ or } y_1 = Z_4$
$Z_7 \xrightarrow{b} : x_3 \text{ or } y_4 = +Z_{11}$

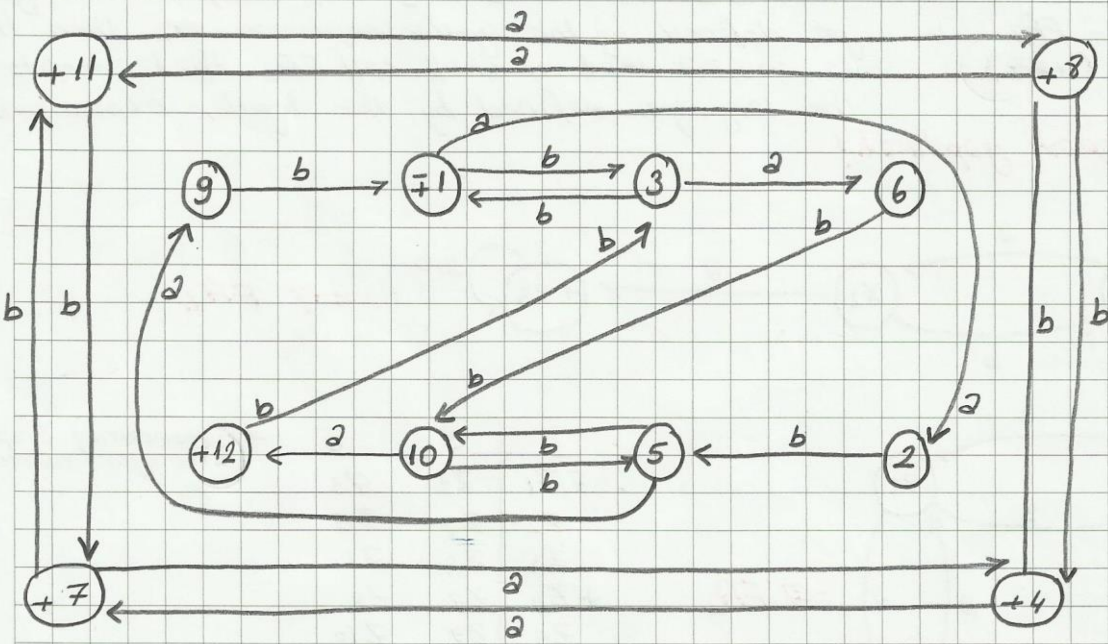
8) $Z_8 \xrightarrow{a} : x_3 \text{ or } y_4 = Z_{11}$
$Z_8 \xrightarrow{b} : x_3 \text{ or } y_2 = Z_4$

9) $Z_9 \xrightarrow{a} : x_3 \text{ or } y_4 = Z_{11}$
$Z_9 \xrightarrow{b} : x_1 \text{ or } y_1 = Z_1$

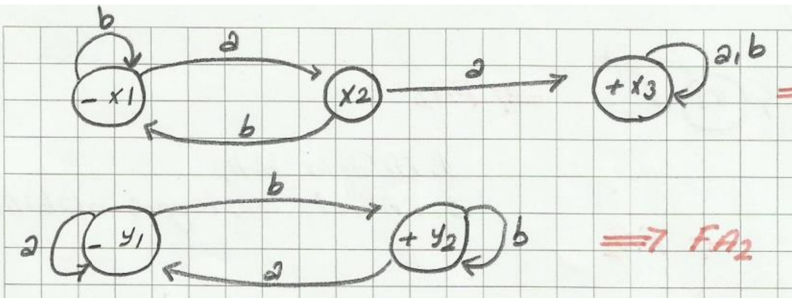
10) $Z_{10} \xrightarrow{a} : x_2 \text{ or } y_1 = +Z_{12}$
$Z_{10} \xrightarrow{b} : x_1 \text{ or } y_4 = Z_5$

11) $Z_{11} \xrightarrow{a} : x_3 \text{ or } y_2 = Z_8$
$Z_{11} \xrightarrow{b} : x_3 \text{ or } y_3 = Z_7$

12) $Z_{12} \xrightarrow{a} : x_3 \text{ or } y_3 = Z_7$
$Z_{12} \xrightarrow{b} : x_1 \text{ or } y_2 = Z_3$



~ ÖDEV: $FA_1 + FA_2 = FA_2 + FA_1$



	a	b
−z1	z2	z3
z2	z4	z3
+z3	z2	z3
+z4	z4	z5
+z5	z4	z5

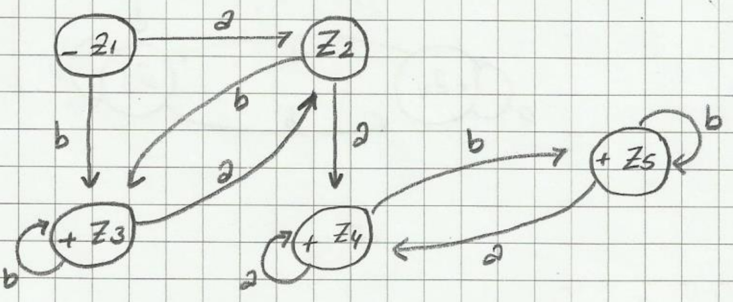
1) $z_1 = x_1$ or y_1
$z_1 \xrightarrow{a}$: x_2 or y_1 = z_2
$z_1 \xrightarrow{b}$: x_1 or y_2 = $+z_3$

2) $z_2 \xrightarrow{a}$: x_3 or y_1 = $+z_4$
$z_2 \xrightarrow{b}$: x_1 or y_2

3) $z_3 \xrightarrow{a}$: x_2 or y_1
$z_3 \xrightarrow{b}$: x_1 or y_2

4) $z_4 \xrightarrow{a}$: x_3 or y_1
$z_4 \xrightarrow{b}$: x_3 or y_2 = $+z_5$

5) $z_5 \xrightarrow{a}$: x_3 or y_1
$z_5 \xrightarrow{b}$: x_3 or y_2



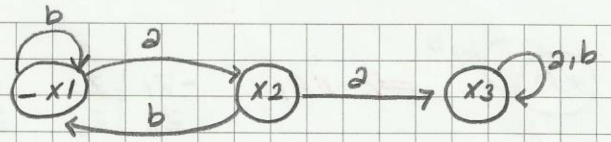
III. Partın Kural 3'ü: If there is an FA1 that accepts the language defined by the regular expressions r1, and there is an FA2 that accepts the language defined by the regular expression r2, then there is an FA3 that accepts the language defined by the concatenation r1r2, the product language.
↳ önce r1'li string, sonra r2'nin kabul ettiği string gelecek. r2'li string sonlandığı anda FA3 oluşacak

$FA_1 * FA_2 = FA_3$
$FA_1 + FA_2 = FA_3$

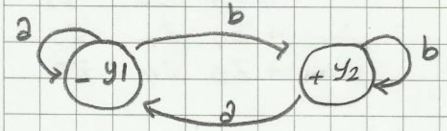
$r_3 = r_1 + r_2$

$r_3 = r_1 r_2 \neq r_2 r_1$

$FA_3 = FA_1 * FA_2$



$\Rightarrow FA_1$



$\Rightarrow FA_2$

1. FA'deki bitiş
2. FA'deki başlangıç olabilir!

$Z_1 = x_1$
$Z_2 = x_2$
$Z_3 = x_3$ or y_1 (bağlantı)

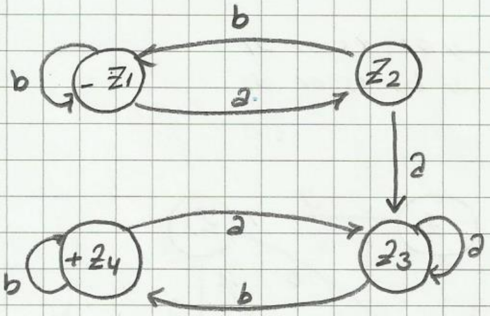
	a	b
$-Z_1$	Z_2	Z_1
Z_2	Z_3	Z_1
Z_3	Z_3	Z_4
$+Z_4$	Z_3	Z_4

1) $Z_1 \to a: x_2$
$Z_1 \to b: x_1$

2) $Z_2 \to a: x_3$ or y_1 (x_3 yazarken y_1 eklenir hep)
$Z_2 \to b: x_1$

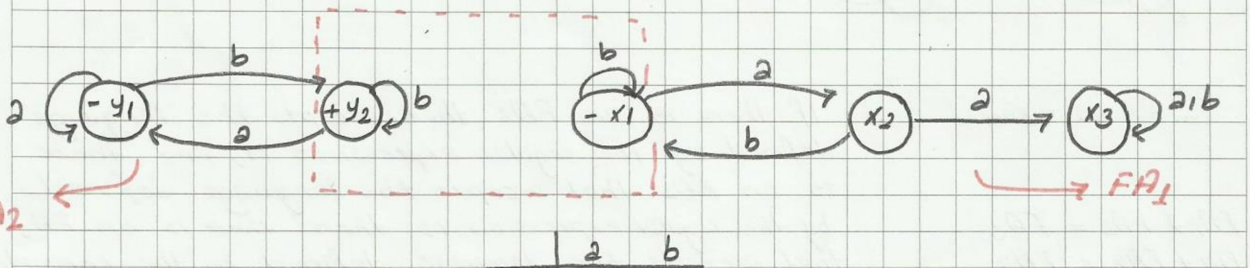
3) $Z_3 \to a: x_3$ or y_1
$Z_3 \to b: x_3$ or y_1 or y_2 $= +Z_4$

4) $Z_4 \to a: x_3$ or y_1
$Z_4 \to b: x_3$ or y_1 or y_2



$FA_1 * FA_2$

$FA_3 = FA_2 * FA_1$



$-Z_1 = y_1$
$Z_2 = y_2$ or x_1

	a	b
$-Z_1$	Z_1	Z_2
Z_2	Z_3	Z_2
Z_3	Z_4	Z_2
$+Z_4$	Z_4	Z_5
$+Z_5$	Z_6	Z_5
Z_6	Z_4	Z_5

1) $Z_1 \to a: y_1$
$Z_1 \to b: y_2$ or x_1

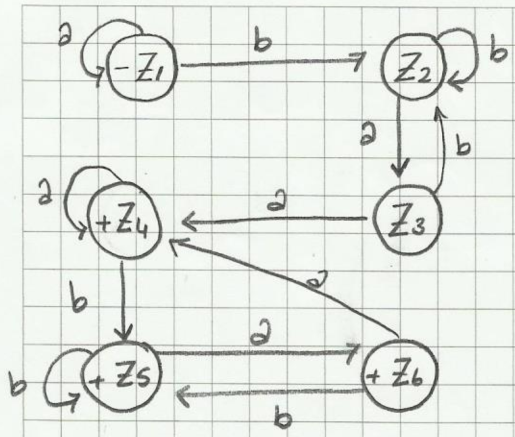
2) $Z_2 \to a: y_1$ or x_2 $= Z_3$
$Z_2 \to b: y_2$ or x_1

3) $Z_3 \to a: y_1$ or x_3 $= +Z_4$
$Z_3 \to b: y_2$ or x_1

4) $Z_4 \to a: y_1$ or x_3

5) $Z_5 \to a: y_1$ or x_2 or x_3 $= +Z_6$
$Z_5 \to b: y_2$ or x_1 or x_3 $= Z_5$

6) $Z_6 \to a: y_1$ or x_3

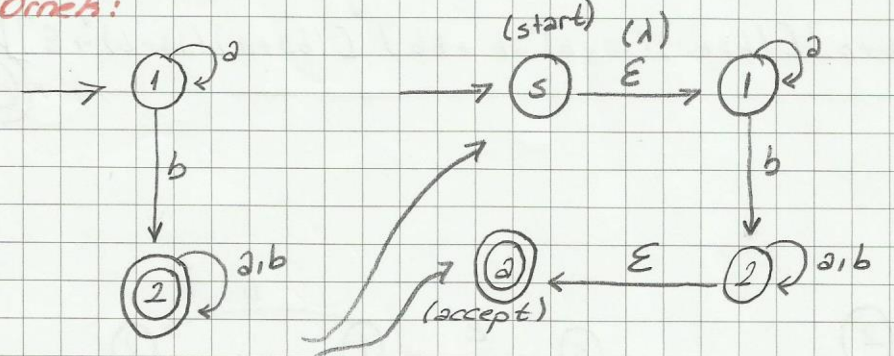


$FA_2 * FA_1$

$FA_1 * FA_2 \neq FA_2 * FA_1$

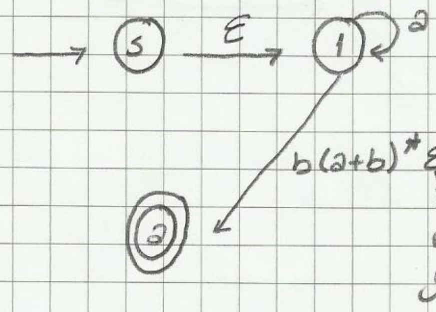
→ FA → RE Dönüşümü

Örnek:



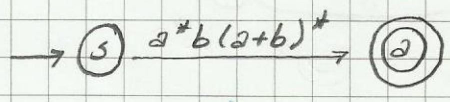
2 tane durum eklemesi yapılır. Başlangıçtan önce bitişten sonra.

2. Durum Yok Edilirken;

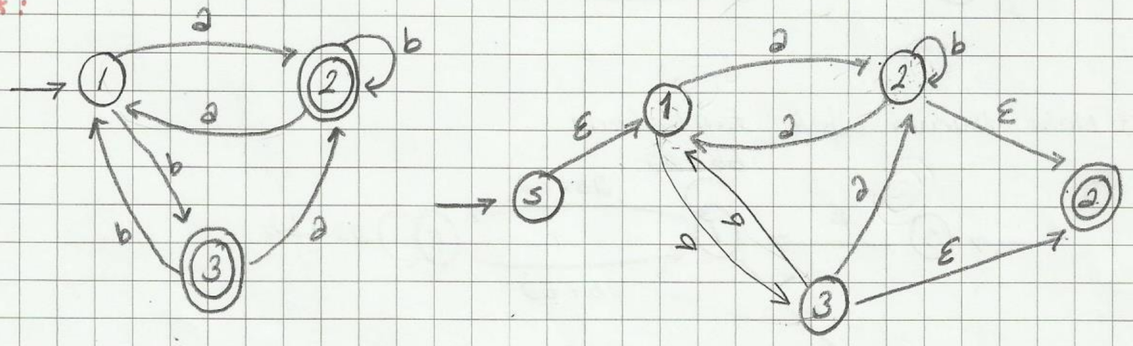


boş string olduğu için yazmaya gerek yok

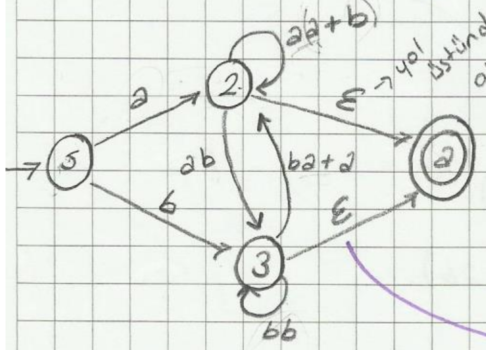
1. Durum Yok Edilirken;

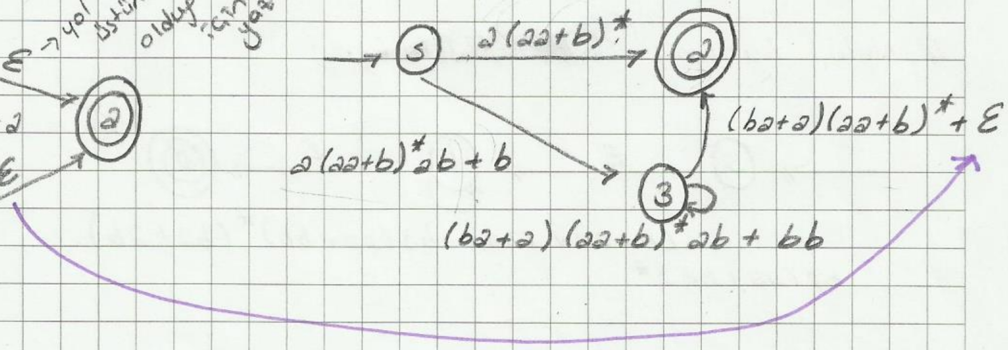


Örnek:



1 Nolu Durum Yok Edilirken; 2 Nolu Durum Yok Edilirken;

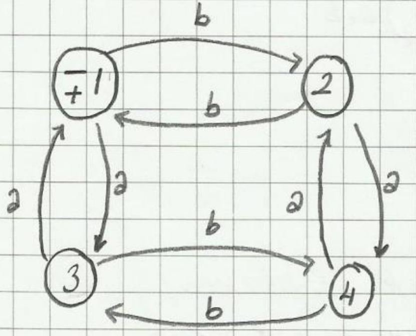


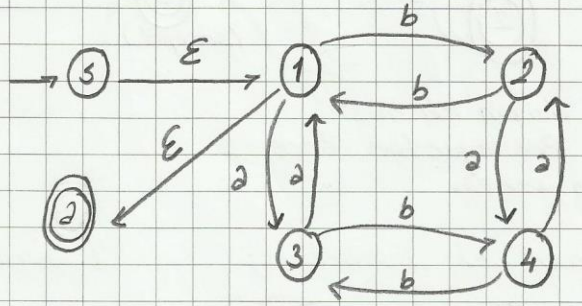


3 Nolu Durum Yok Edilirken;

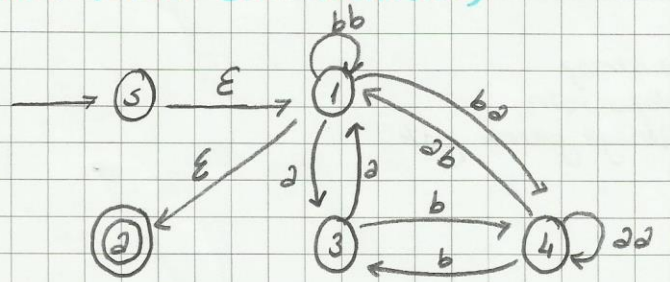
$$\rightarrow (s) \xrightarrow{\left[[a(a+b)^*ab+b][(ba+a)(aa+b)^*ab+bb]^*[(ba+a)(aa+b)^*+\varepsilon]\right]+a(aa+b)^*} ((a))$$

#Örnek:

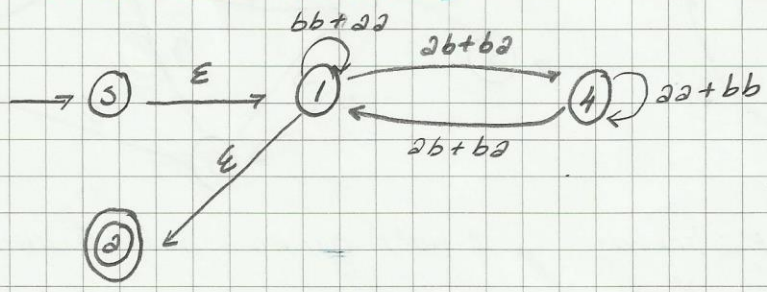




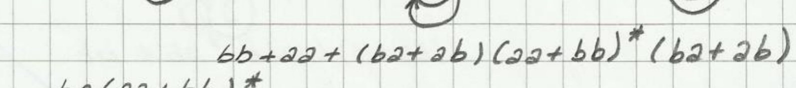
1 Nolu Durum Yok Edilirken;



3 Nolu Durum Yok Edilirken;



4 Nolu Durum Yok Edilirken;



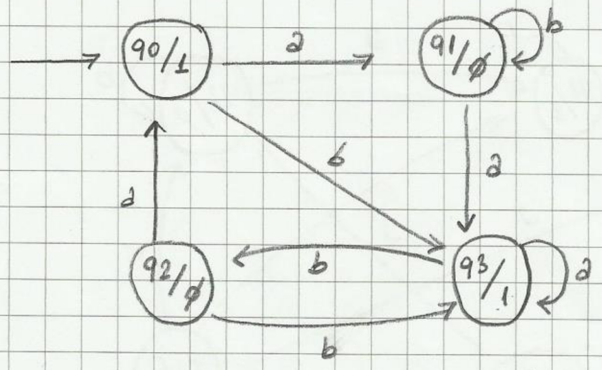
$ba(aa+bb)^*$

1 Nolu Durum Yok Edilirken;

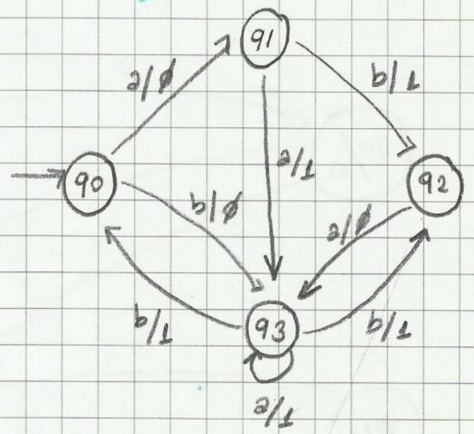
$$\rightarrow (s) \xrightarrow{\varepsilon\,[bb+aa+(ba+ab)+(aa+bb)^{*}(ba+ab)]^{*}\,\varepsilon} ((a))$$

→ FA with Output (Çıkışlı Sonlu Otomata)

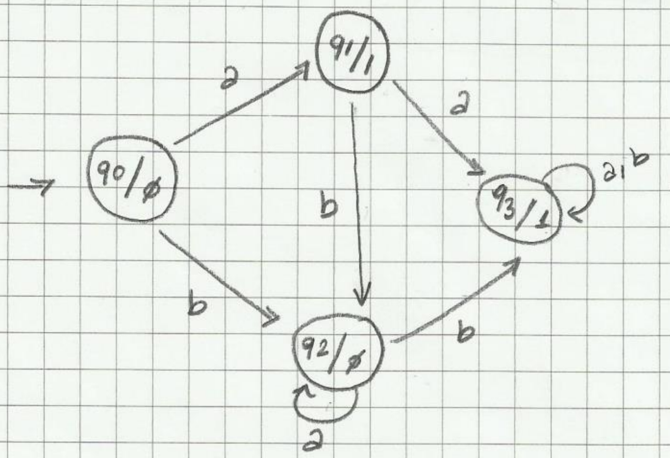
Moore machine

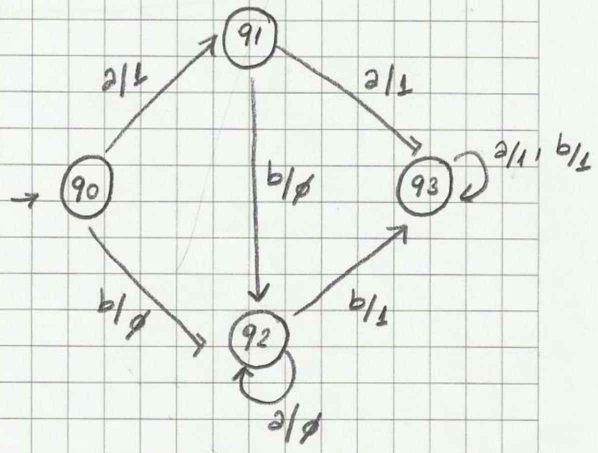


Mealy Machine

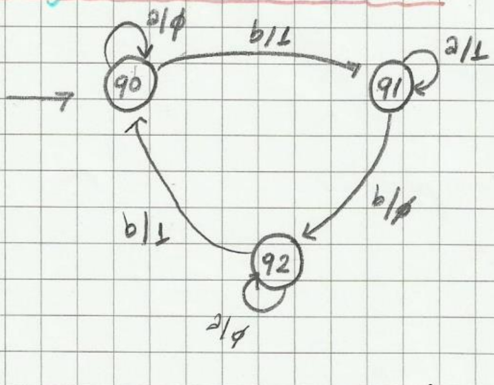


Moore → Mealy Dönüşümü

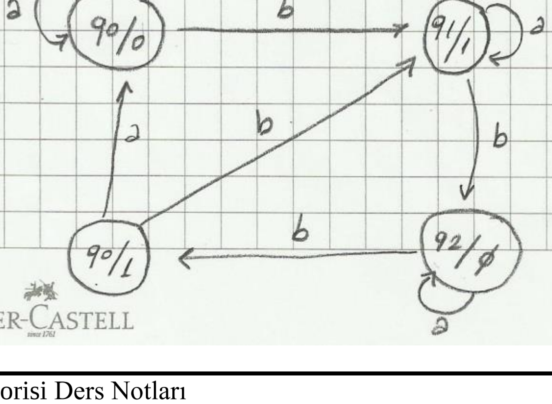




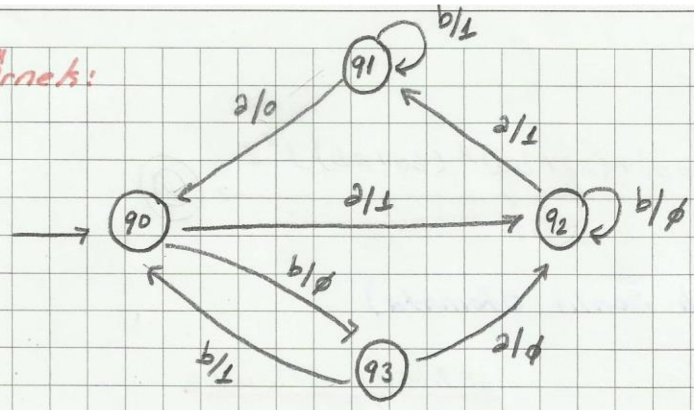
Mealy → Moore Dönüşüm

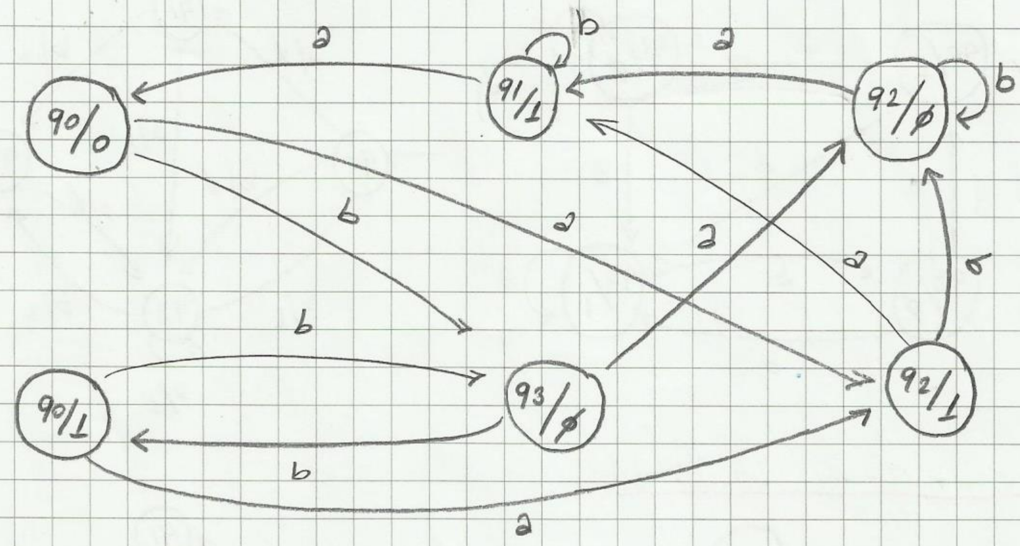


q0'da 2'tane geçiş var bundan dolayı 2 farklı çıkış durumu söz konusu olduğundan durum sayısı zorunlu olarak artar.



Örnek:


q0
q1
q2
q3
a/0
b/1
a/1
a/1
b/φ
b/φ
b/1
a/φ


q0/0
q1/1
q2/φ
q0/1
q3/φ
q2/1
a
b
a
b
b
a
a
a
b
b
b
a

→ CONTEXT-FREE GRAMMARS (CFG)

CFG oluşturulurken bazı semboller kullanılacak.

terminal (sonlanan): $\{a, b, \lambda\}$

Nonterminal (sonlanmayan): $\{S, A, B, X, Y, Z, \ldots\}$

(start)

$\left.\begin{array}{l} S \rightarrow aS \\ S \rightarrow \lambda \end{array}\right\} S \rightarrow aS \mid \lambda$ (CFG gösterimi)

$\mid$ → veya

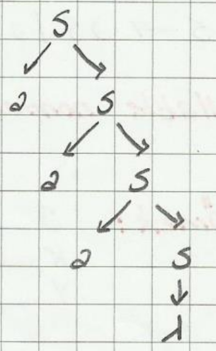
Her biri production

*** Bu grammar hangi stringleri üretir?**

$S \rightarrow aS \rightarrow a\lambda = a$

$S \rightarrow aS \rightarrow aaS \rightarrow aa\lambda = aa$

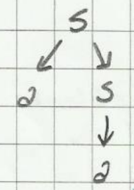
$S \rightarrow aS \rightarrow aaS \rightarrow aaaS \rightarrow aaa\lambda = aaa$



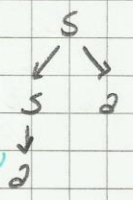
*** $(a+b)^*$ için** $S \rightarrow aS \mid bS \mid a \mid b \mid \lambda$

* Herhangi bir string için birden fazla parse ağacı çizebiliyorsak, o gramere "ambiguos (belirsiz) gramer" denir.

 # Örnek: $S \rightarrow aS \mid Sa \mid a$ gramerinde aa için oluşturalım.



$S \rightarrow aS \rightarrow aa$



$S \rightarrow Sa \rightarrow aa$

parse ağacı

2 farklı şekilde aa üretebildik (ambiguos)

CFG → λ yok etme (Chamsky Normal Form) → CYK Parsing Algoritması

CYK parsing algoritması stringin gramer kabul edip etmediğine bakar. Bunun için belli bir forma getirir. λ'ları yok eder.

→ λ Yok Etme Algoritması

1) Delete all λ productions

2) For each production X → something with at least one nullable non-terminal on the right hand side (RHS) do the following each possi nonempty subset of nullable nonterminals on the RHS: Generate X → something where the new RHS is the some old RHS expect with entire current subset of nullable nonterminals removed.

sağda birden fazla nullable varsa

* $X \rightarrow Y$ ($X \rightarrow Y \rightarrow \lambda$ → X dolaylı olarak null olabilir)

$Y \rightarrow a \mid \lambda$ (Y doğrudan null olabilir)

* $S \rightarrow aS \mid a$ ⟹ S nullable olamaz.

nullable nonterminal: null olabilecek nonterminallerdir. Birkaç adım sonra da null olan nonterminaldir.

#Örnek:

$$\left.\begin{array}{l} S \rightarrow a \mid Xb \mid aYa \\ X \rightarrow Y \mid X \mid \lambda \\ Y \rightarrow X \mid a \end{array}\right\}$$

nullables: X, Y

$Y \Rightarrow X \Rightarrow \lambda$ → nullable olabilir

1. adımda (λ'lar silinir);

$S \rightarrow a \mid Xb \mid aYa$
$X \rightarrow Y \mid X$ → λ silinmiştir
$Y \rightarrow X \mid a$

Original Production	New Production
$S \rightarrow Xb$	$S \rightarrow b$
$S \rightarrow aYa$	$S \rightarrow aa$
$X \rightarrow Y$	nothing
$X \rightarrow X$	nothing
$Y \rightarrow X$	nothing

$S \rightarrow b$, $S \rightarrow aa$ } Mevcut gramere eklenir.

$$\left.\begin{array}{l} S \rightarrow a \mid Xb \mid aYa \mid b \mid aa \\ X \rightarrow Y \mid X \\ Y \rightarrow X \mid a \end{array}\right\}$$ Gramer'in son hali

Örnek:

$S \to X \mid XY \mid Z$
$X \to Z \mid \lambda$
$Y \to Wa \mid a$
$Z \to WX \mid aZ \mid Zb$
$W \to XYZ \mid bXa \mid \lambda$

nullables : X, W, Z, S

Orginal Production	New Production
$S \to X$	Nothing
$S \to XY$	$S \to Y$
$S \to Z$	Nothing
$X \to Z$	Nothing
$Y \to Wa$	$Y \to a$
$Z \to WX$	$Z \to X$, $Z \to W$
$Z \to aZ$	$Z \to a$
$Z \to Zb$	$Z \to b$
$W \to XYZ$	$Z \to YZ$, $W \to XY$, $W \to Y$
$W \to bXa$	$W \to ba$

$S \to X \mid XY \mid Z \mid Y$
$X \to Z$
$Y \to Wa \mid a$
$Z \to WX \mid aZ \mid Zb \mid X \mid W \mid a \mid b$
$W \to XYZ \mid bXa \mid YZ \mid XY \mid Y \mid ba$

Gramerin son hali

→ CNF (Chomsky Normal Form)

Each of production has the one of two forms:

1) Nonterminal → string of exactly two nonterminals (pipelarla ayrılmış nonterminal olacak. AX | XY | BY | AB gibi)

2) Nonterminal → one terminal ($S \to a$, $A \to a$ gibi)
$S \to a \mid b$ de olabilir.
$S \to a\cancel{A} \mid \cancel{Bb}$ yazak

Bu iki formdan birini sağlamalı. CNF'ye gelmeden λ'lar yok edilmeli. Aynı zamanda gramerin sağ tarafından büyük küçük harf yan yana olmaz.

*** Nonterminaller 2, terminaller 1 tane olmak zorundadır.**

Örnek:

$S \to aXX$
$X \to aS \mid bS \mid a$

⇒ $X \to a$'dan dolayı CNF'ye dönüştürür. Bu durumda diğerleri CNF'ye uydurulur.

$S \to AXX$ → uymuyor	$S \to AR$
$X \to AS \mid BS \mid a$	$R \to XX$
$A \to a$	$X \to AS \mid BS \mid a$
$B \to b$	$A \to a$

Örnek: $S \rightarrow A\underbrace{X\overbrace{X\,\boxed{BA}}^{R_2}}_{R_1} \rightarrow R_3$

$S \rightarrow AR_1$
$R_1 \rightarrow XR_2$
$R_2 \rightarrow XR_3$
$R_3 \rightarrow BA$

} ikişerli Alt parçalara bölünebilir.

* $S \rightarrow a \mid Xb \mid aYa \mid b \mid aa$

$X \rightarrow Y \mid X \mid \lambda$ → silinir.

$Y \rightarrow X \mid a$

Unit olduğu için bunlarıda yok etmemiz gerekir. Tek başına nonterminal olamaz.

Bundan sorumlu değiliz ?

nullable : X, Y

AR (R→YA)

$S \rightarrow a \mid XB \mid AYA \mid b \mid AA$
$X \rightarrow Y \mid X$
$Y \rightarrow X \mid a$
$A \rightarrow a$
$B \rightarrow b$

→ CYK Parsing Algoritması

Herhangi bir stringin gramer tarafından kabul edilip edilmediğine bakıyor. Bunun için tablo oluşturulur.

$S \rightarrow BS \mid AX \mid b$
$X \rightarrow BX \mid AS \mid a$
$A \rightarrow a$
$B \rightarrow b$

* Yukarıdaki gramerin "abbab" stringini kabul edip etmediğine bakalım

	I	II	III	IV	V
a	X,A	X	X	S	S
b	S,B	S	X	X	
b	S,B	X	X		
a	X,A	X			
b	S,B				

→ S buraya yazılırsa gramer stringi kabul eder.

a, b'ler hangi productionda varsa ilk sütuna onları yazıyoruz.

II. sütun için;

X,A
S,B } XS, XB, AS, AB → X'te var

S,B
S,B } SS, SB, BS, BB → S'de var

S,B
X,A } SX, SA, BX, BA → X'te var

X,A
S,B } XS, XB, AS, AB → X'te var

III. sütun için;

X, A
S } XS, AS → X'te var

X
S, B } XS, XB

1. satır

S, B
X } SX, BX → X'te var

S
X, A } SX, SA

2. satır

S, B
X } SX, BX → X'te var

X
S, B } XS, XB

3. satır

IV. sütun için;

X, A
X } XX, AX → S'de var

XX

X
XA } XX, XA

1

S, B
X } SX, BX → X'te var

SX

X
S, B } XS, XB

2

V. sütun için;

X, A
X } XX, AX → S'te var

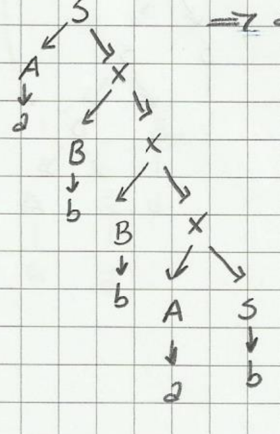
XX
XX

S
S, B } SS, SB

*** Bu bize parse ağacı çizmede yardımcı olacak.**

→ Parse Ağacı

Tersten gidilir. (Çaprazlamalar sonucu oluşan S ve X'lere bakılır)



⟹ abbab

Bu stringin tek bir parse ağacı vardır. Örneğin elde edilen son S sadece bir şekilde elde edilmiş alternatifi yok.

Örnek: $S \rightarrow SS \mid AC \mid BD \mid AB \mid BA \mid b$

$C \rightarrow SB$

$D \rightarrow SA$

$A \rightarrow a$

$B \rightarrow b$

bbba kelimesi için kaç tane parse ağacı çıkar?

	I	II	III	IV
b	S,B	S,C	S,C	S,D
b	S,B	S,C	S,D	
b	S,B	S,D		
a	A			

II. sütun için;

$\left.\begin{matrix} S,B \\ S,B \end{matrix}\right\}$ $SS \rightarrow S$, $SB \rightarrow C$, BS, BB

$\left.\begin{matrix} S,B \\ A \end{matrix}\right\}$ $SA \rightarrow D$, $BA \rightarrow S$

III. sütun için;

$\left.\begin{matrix} S,B \\ S,C \end{matrix}\right\}$ $SS \rightarrow S$, SC, BS, BC

$\left.\begin{matrix} S,C \\ S,B \end{matrix}\right\}$ $SS \rightarrow S$, $SB \rightarrow C$, CS, CB

$\left.\begin{matrix} S,B \\ S,D \end{matrix}\right\}$ $SS \rightarrow S$, SD, BS, $BD \rightarrow S$

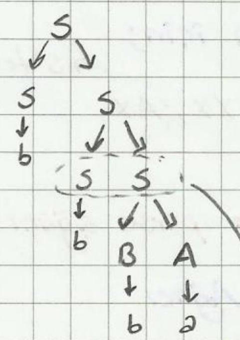
$\left.\begin{matrix} S,C \\ A \end{matrix}\right\}$ $SA \rightarrow D$, CA

IV. sütun için;

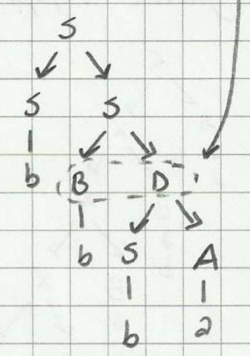
$\left.\begin{matrix} S,B \\ S,D \end{matrix}\right\}$ SS * $\rightarrow S$, SD, BS, BD ** $\rightarrow S$

$\left.\begin{matrix} S,C \\ S,D \end{matrix}\right\}$ SS *** $\rightarrow S$, SD, CS, CD

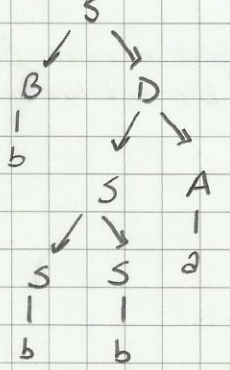
$\left.\begin{matrix} S,C \\ A \end{matrix}\right\}$ $SA \rightarrow D$, CA

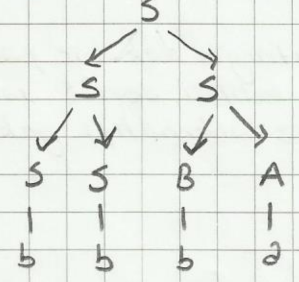


alternatif:



**

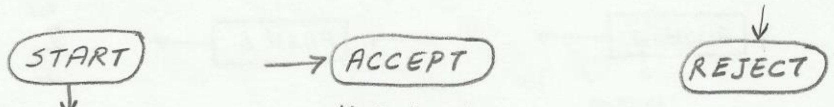




→ PUSH DOWN AUTOMATA (PDA)

24.03.2014

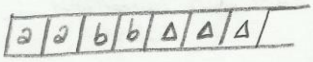
PDA'nın alt bileşenleri;



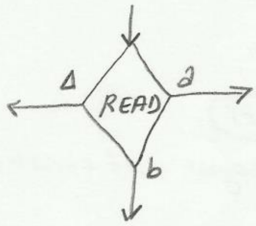
→ stringi kabul etmeme durumu

Kabul Etme Durumu

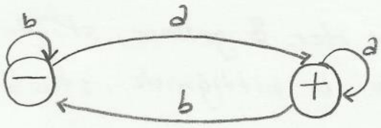
TAPE



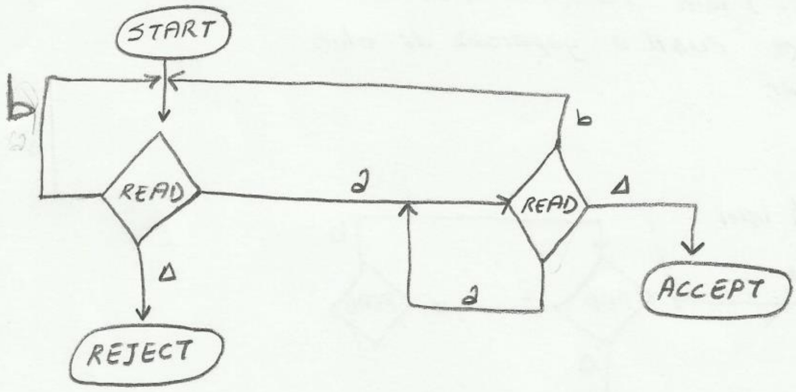
Δ'lar tape'in geri kalan kısmı boş demek.



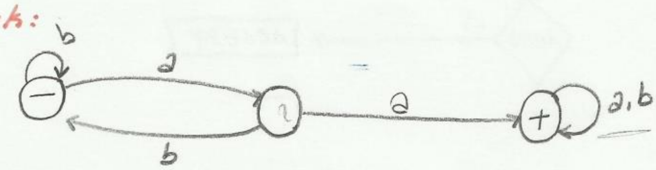
#Örnek:

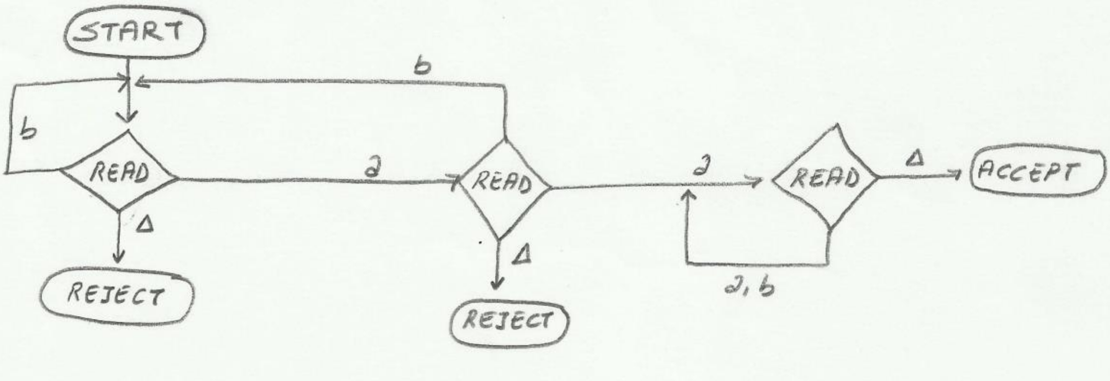


2 durum var o yüzden iki tane READ olacak.

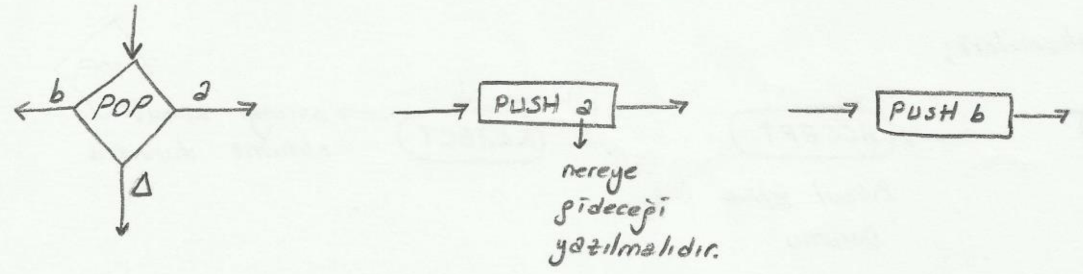


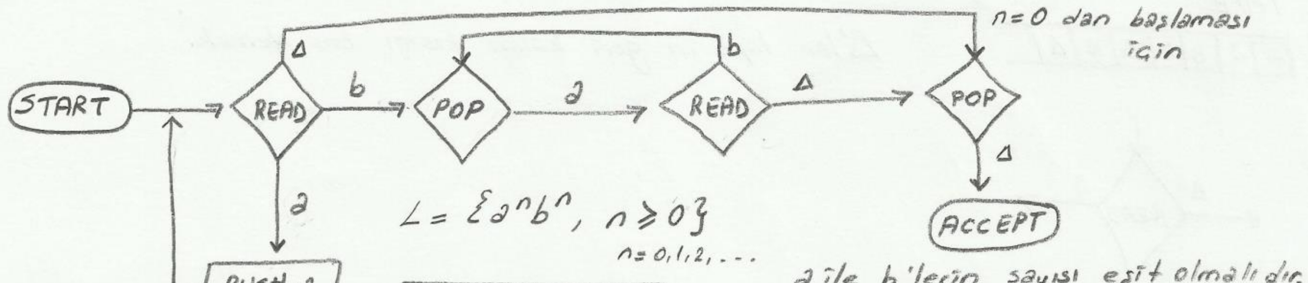
#Örnek:





★ Kullanacağımız PDA'lara stack ekleyeceğiz.





$L = \{a^n b^n,\ n \geq 0\}$

$n = 0, 1, 2, \ldots$

a ile b'lerin sayısı eşit olmalıdır.

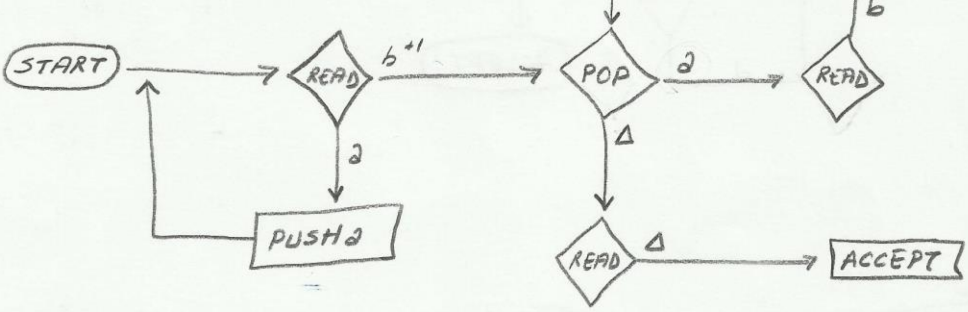
★ Tapeden önce a'ları okur. B gelene kadar stacke iter. B gelince, stacten a çeker. Kaç tane a itmişse o kadar b okuyor. Tape de b bittiğinde, stackte a'da kalmaz.

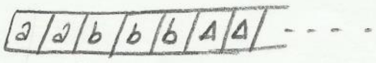
 # Örnek: $L = \{a^n, b^{2n},\ n = 0, 1, 2, 3, \ldots\}$ için PDA'sını çiz.

PUSH a olan yerde 2 kere PUSH a yaparsak da olur.

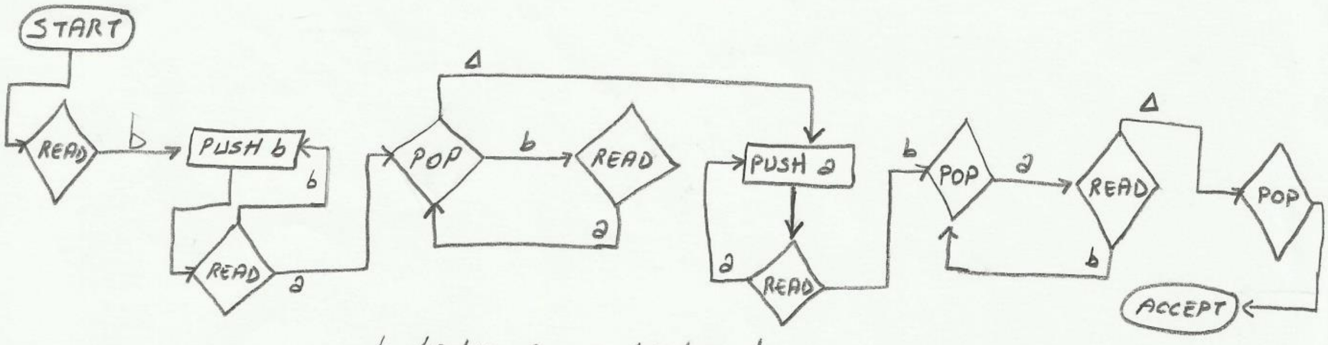
Notlarda diğer çözümüde var.

 # Örnek: $L = \{a^n\ b^{n+1}\quad n = 1, 2 \ldots\}$ için





 # Örnek: $L = \{ b^i a^j b^k ,\ j = i+k ,\ i = 1,2,3\ldots ,\ k = 1,2,3\ldots \}$



b kadar a , a kadar b

bbaa   aaabbb

↳ Toplamı b'lerin toplamına eşit

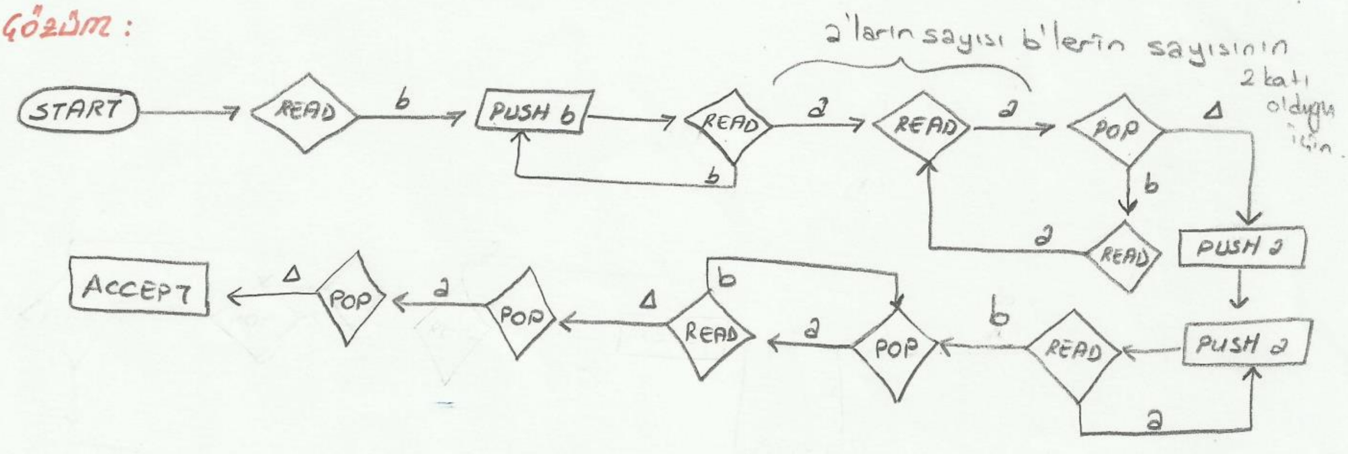
 # Örnek: $L = \{ b^i a^j b^k ,\ j = 2i+k+1 ,\ i>0 ,\ k>0 \}$ dili için PDA çiziniz.

baaaaab $\boxed{b^i a^{2i}}$ $\boxed{a^{k+1} b^k}$

minimumu $\Rightarrow i = 1 ,\ k = 1$

$b^1 a^2 a^2 b^1 \Rightarrow baaaab$

Çözüm:



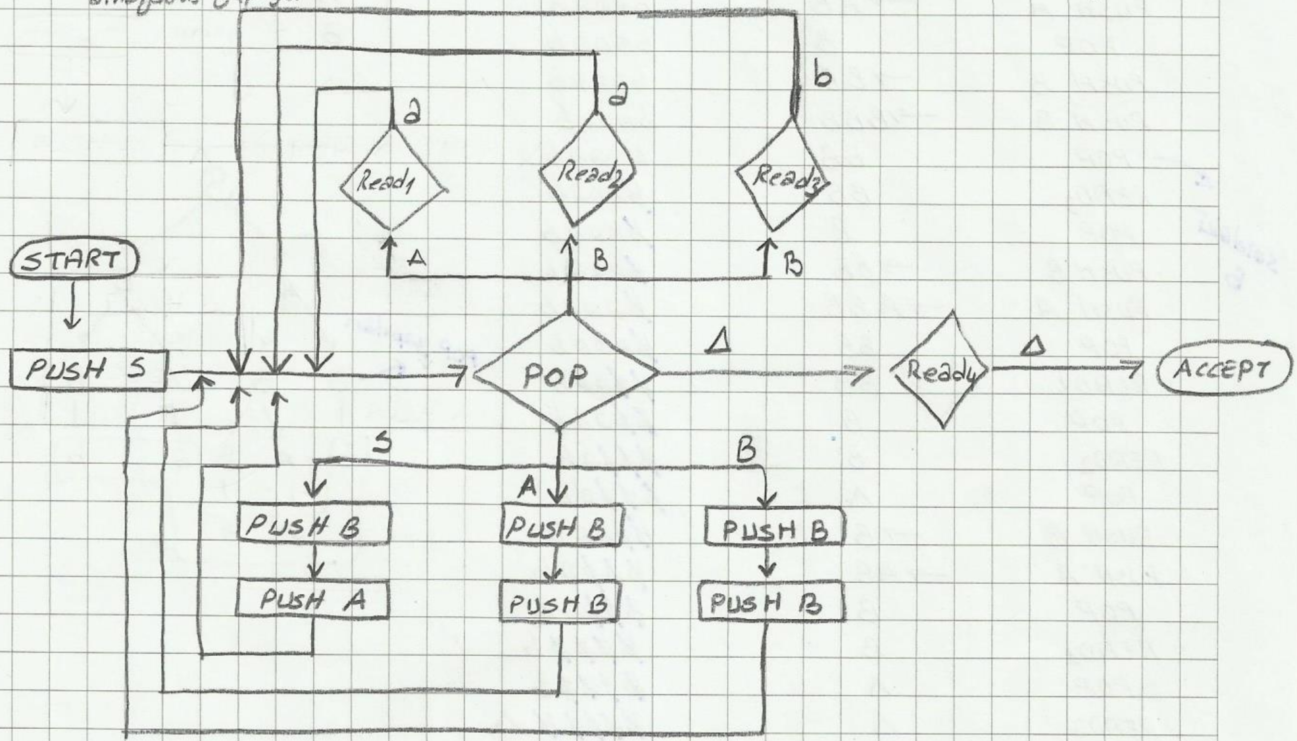
CFG = PDA (Herhangi bir CFG'nin eşdeğer bir PDA'sını çizebilir-miyiz?)

\# $S \rightarrow AB$
$A \rightarrow BB \mid a$
$B \rightarrow AB \mid a \mid b$

gramer'i ambiguous yapıyor

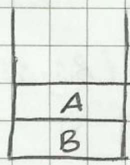
PDA çizerken öncelikle nondeterministic olduğuna dikkat ediniz.

(Büyük harfler PUSH/POP küçük harfler READ yapılır)

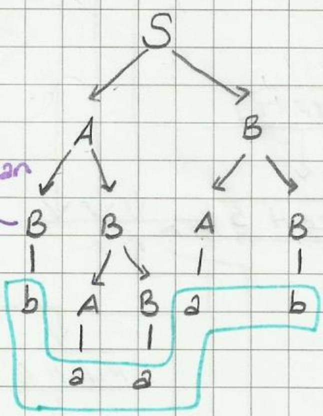


Örnek: "baaab"

STATE	STACK	TAPE
START	Δ	baaab
PUSH S	S	baaab
POP	Δ	baaab
PUSH B	B	baaab
PUSH A	→ AB	baaab
POP	B	baaab
PUSH B	→ BB	baaab
PUSH B	→ BBB	baaab
POP (soldaki B)	BB	baaab
READ$_3$	BB	~~b~~aaab
POP	B	~~b~~aaab
PUSH B	→ BB	~~b~~aaab
PUSH A	→ ABB	~~b~~aaab
POP	BB	~~b~~aaab
READ$_1$	BB	~~ba~~aab
POP	B	~~ba~~aab
READ$_2$	B	~~baa~~ab
POP	Δ	~~baa~~ab
PUSH B	→ B	~~baa~~ab
PUSH A	→ AB	~~baa~~ab
POP	B	~~baa~~ab
READ$_1$	B	~~baaa~~b
POP	Δ	~~baaa~~b
READ3	Δ	~~baaab~~ Δ ↑ → kafa
POP	Δ	~~baaab~~
READ$_4$	Δ	~~baaab~~
ACCEPT	Δ	~~baaab~~

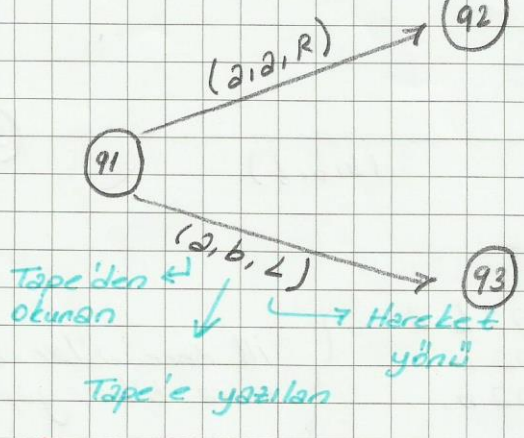


POP yapılan



→ TURING MACHINE (TM)

FA ile PDA'nın birleştirilmiş hali gibi düşünebiliriz. Tape kullanılması PDA'ya benzer özelliğidir. Tape'in üzerine hem yazıp hem de okuyabiliyoruz.



! Tape'de ilk okunan stringten sonra sağa gitmek zorundayız. Sonrakilerde her iki yöne gidebiliriz.

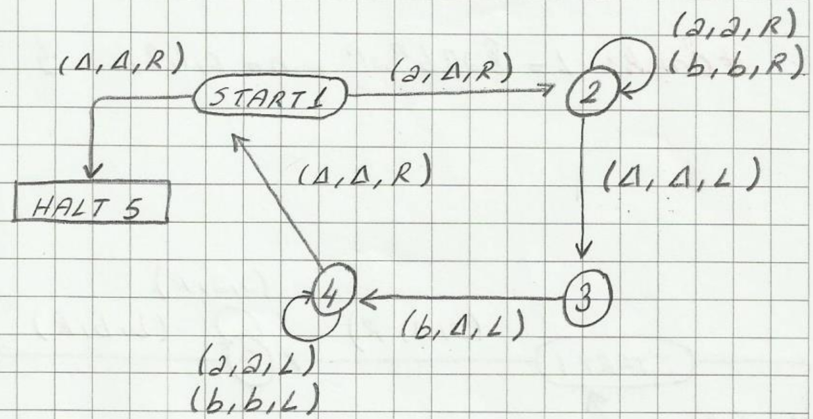
 # Örnek: $L = \{a^n b^n,\ n = 0,1,2,3,\ldots\}$ için TM geliştirin.

TM'de stack yok. Ama tape'e yazabildiğimiz için PDA'daki stack işlevi görür.

a a a b b b Δ
Δ a a b b b Δ
Δ a a b b Δ Δ
Δ Δ a b b Δ Δ
Δ Δ a b Δ Δ Δ
Δ Δ Δ b Δ Δ Δ
Δ Δ Δ Δ Δ Δ Δ

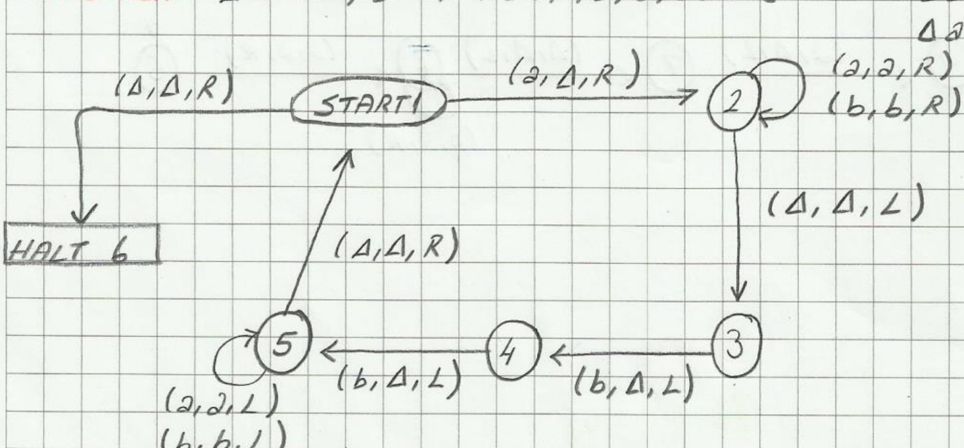
a	a	a	b	b	b	Δ

Δ a a b b Δ Δ
Δ Δ a b Δ Δ Δ
Δ Δ Δ Δ Δ Δ Δ

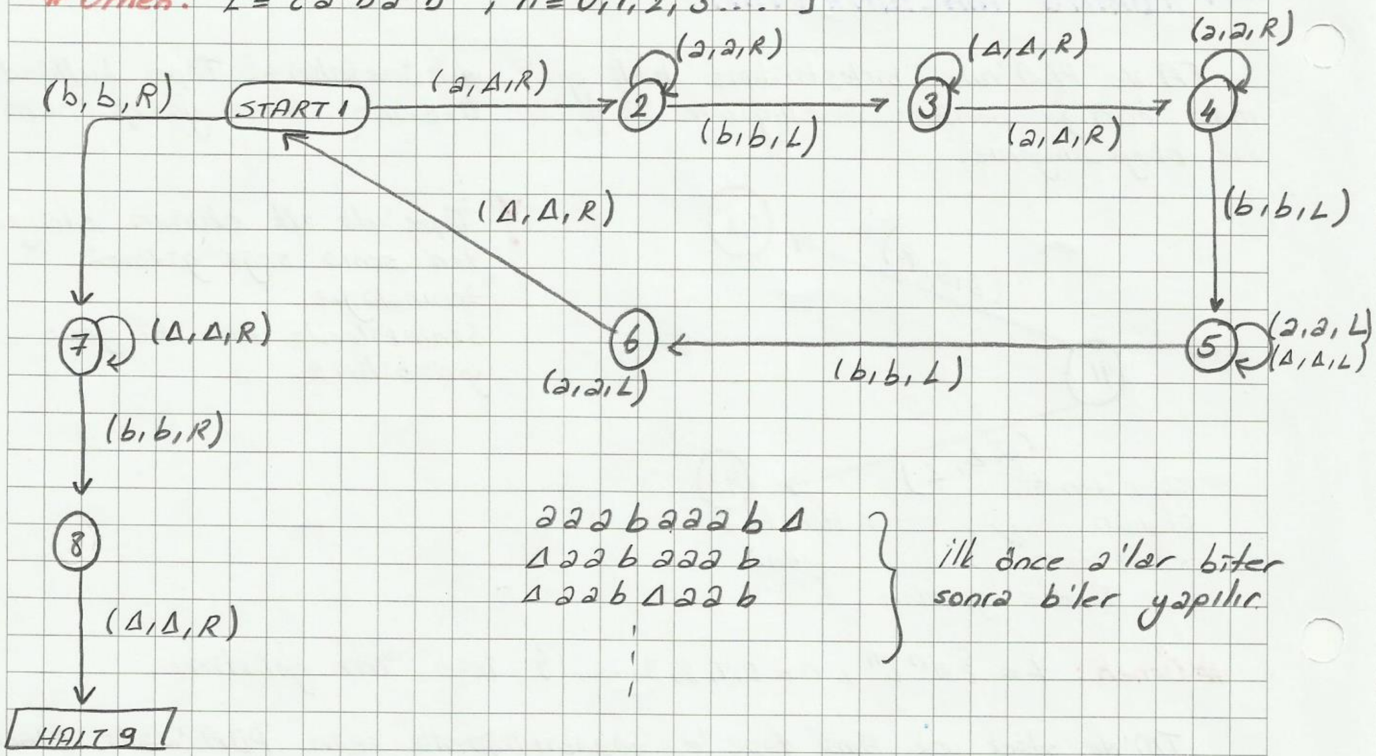


 # Örnek: $L = \{a^n, b^{2n},\ n = 0,1,2,3,\ldots\}$

a a a b b b b b b Δ
Δ a a b b b b Δ Δ Δ



Örnek: $L = \{a^n b a^n b \;,\; n = 0, 1, 2, 3 \ldots\}$

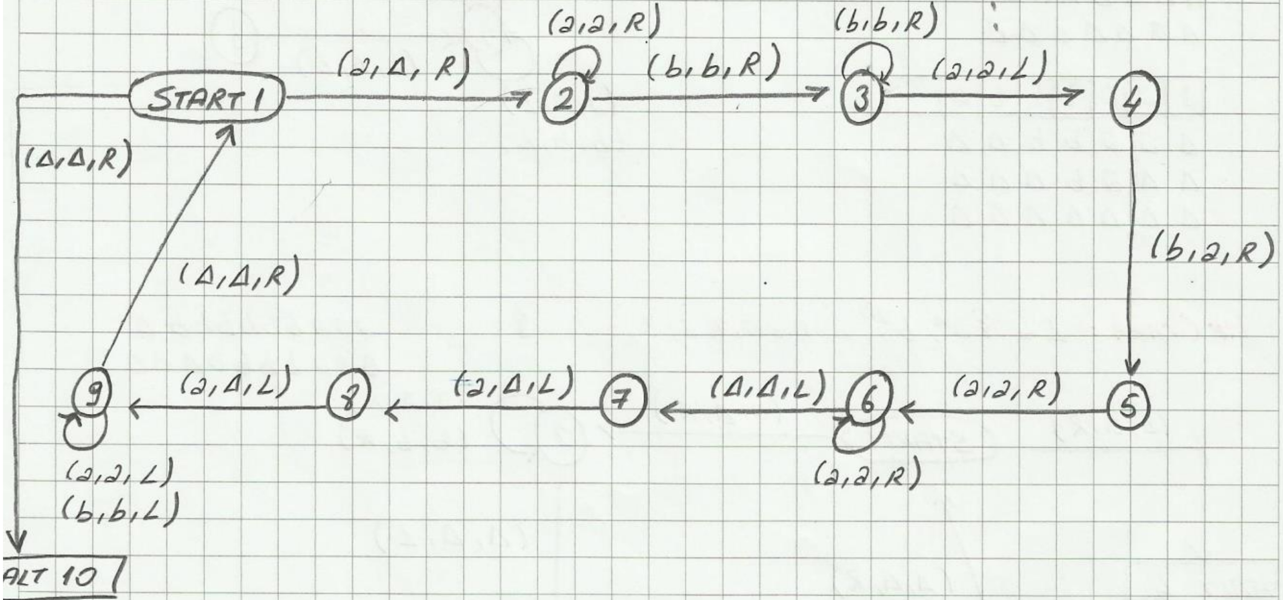


a a a b a a a b Δ
Δ a a b a a a b
Δ a a b Δ a a b
⋮

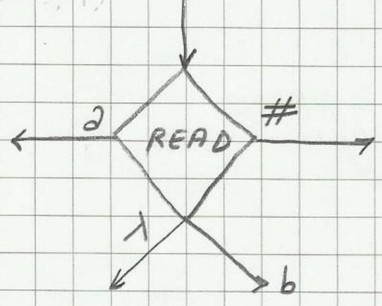
ilk önce a'lar biter sonra b'ler yapılır

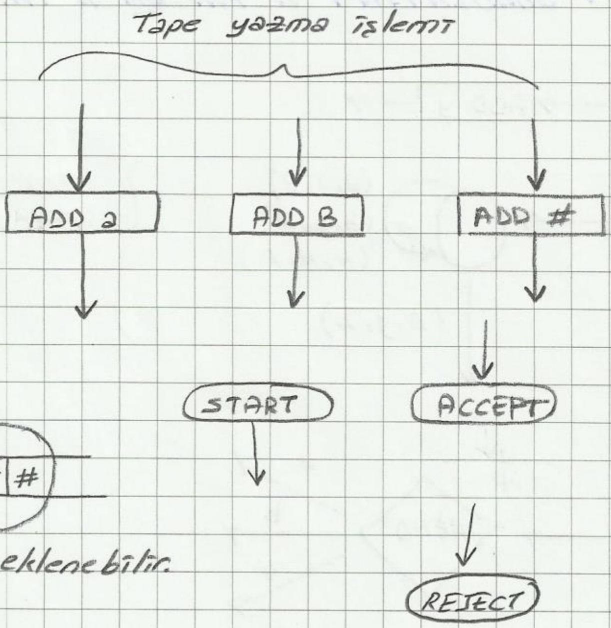
Örnek: $L = \{a^n b^n a^n \;,\; n = 0, 1, 2, \ldots\}$

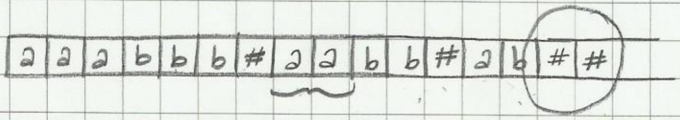
a a a b b b a a a
Δ a a b b a a a a
Δ a a b b a a Δ Δ
Δ Δ a b a a a Δ Δ
Δ Δ a b a Δ Δ Δ Δ
⋮



→ POST MACHINES (PM)

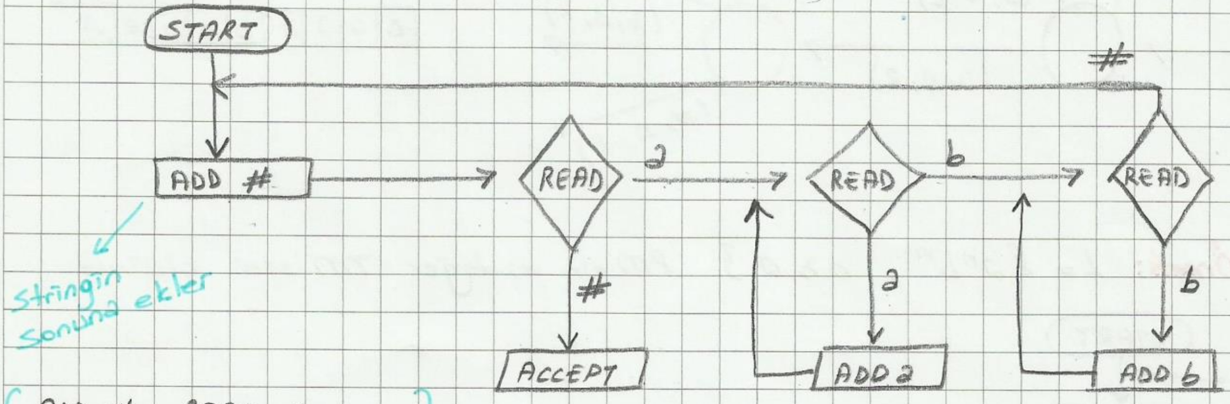






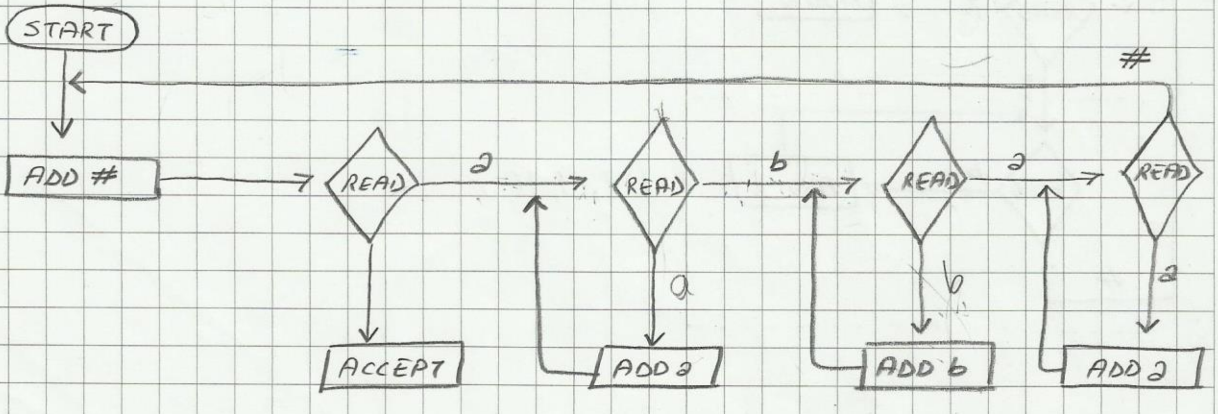
Mevcut stringin peşine # eklenebilir.

Örnek: $L = \{a^n, b^n \quad n = \emptyset, 1, 2, \ldots\}$

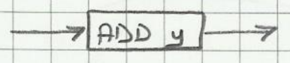


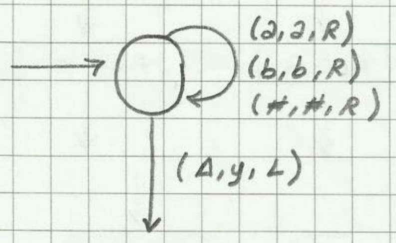
{ ADD #, READ #
peşpeşe olursa ACCEPT
yapılır }

Örnek: $L = \{a^n b^n a^n, \ n \geq 0\}$

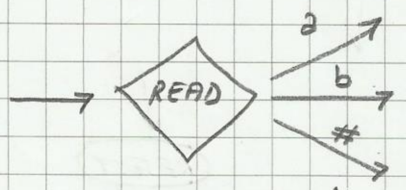


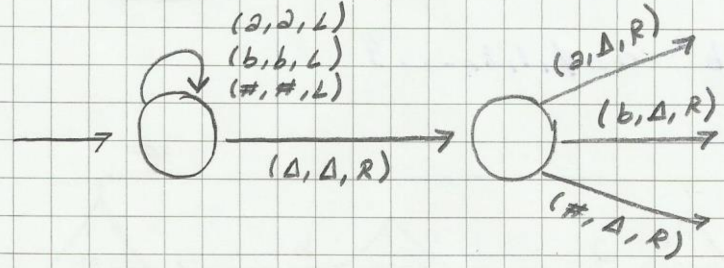
→ SIMULATING A PM ON A TM





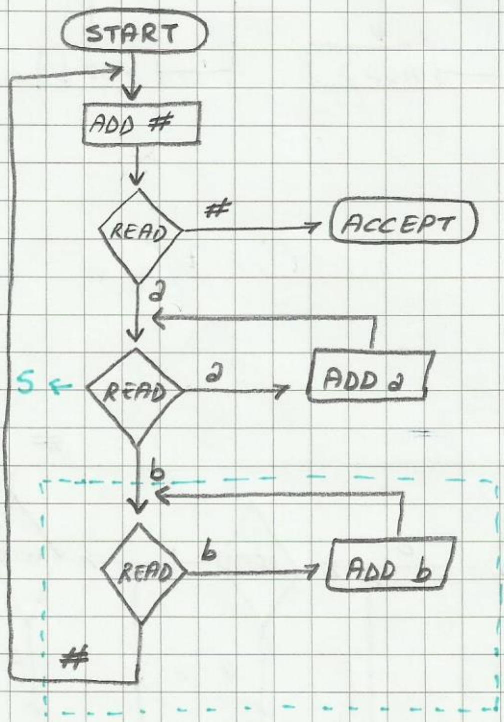
PM'deki ADD işleminin TM'deki karşılığı





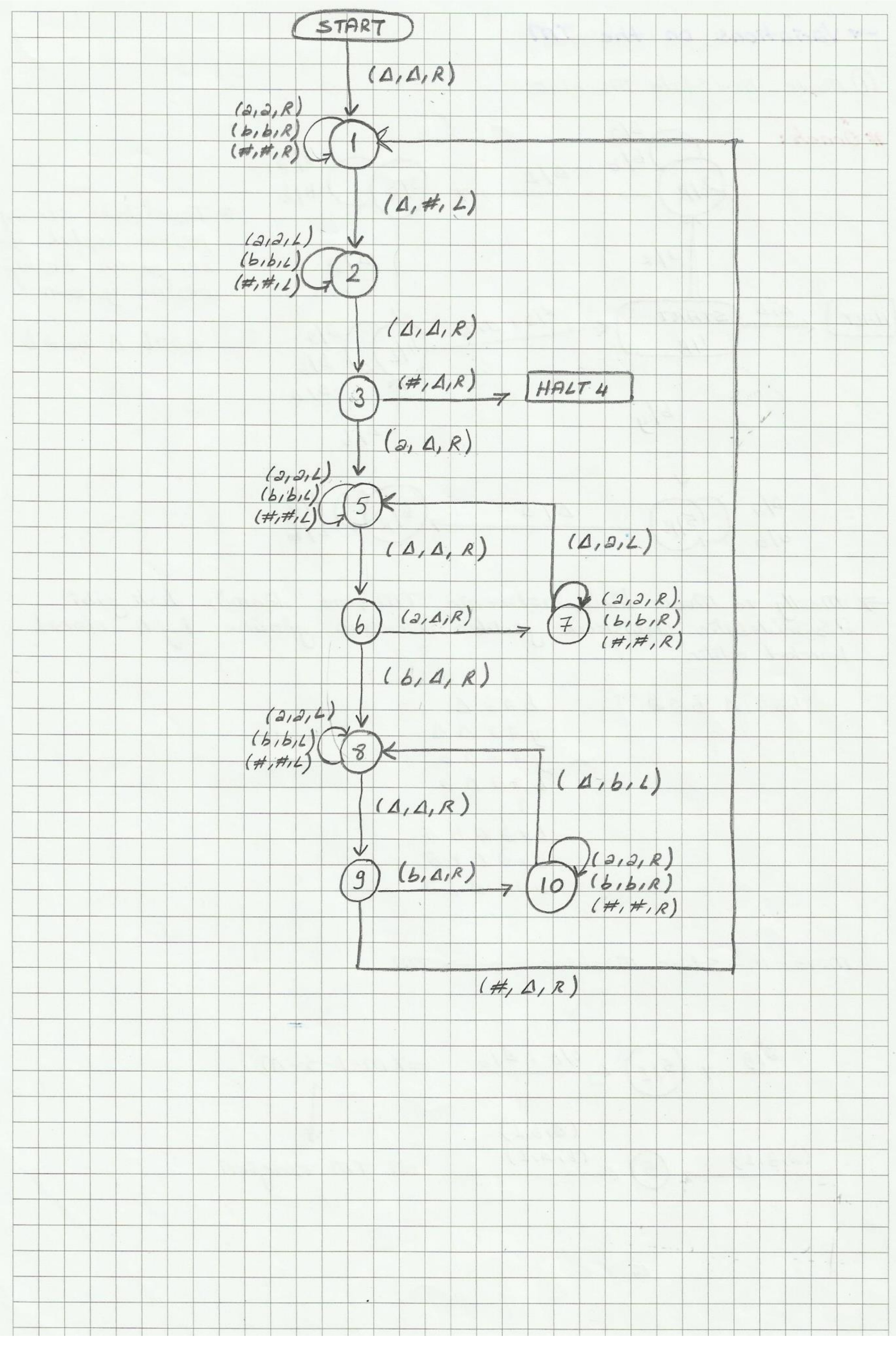
Δ	a	a	a	b	b	b	Δ	Δ

#Örnek: $L = \{a^n b^n,\ n \geq 0\}$ PM'nin eşdeğer TM'sini çiziniz



(8, 9, 10)

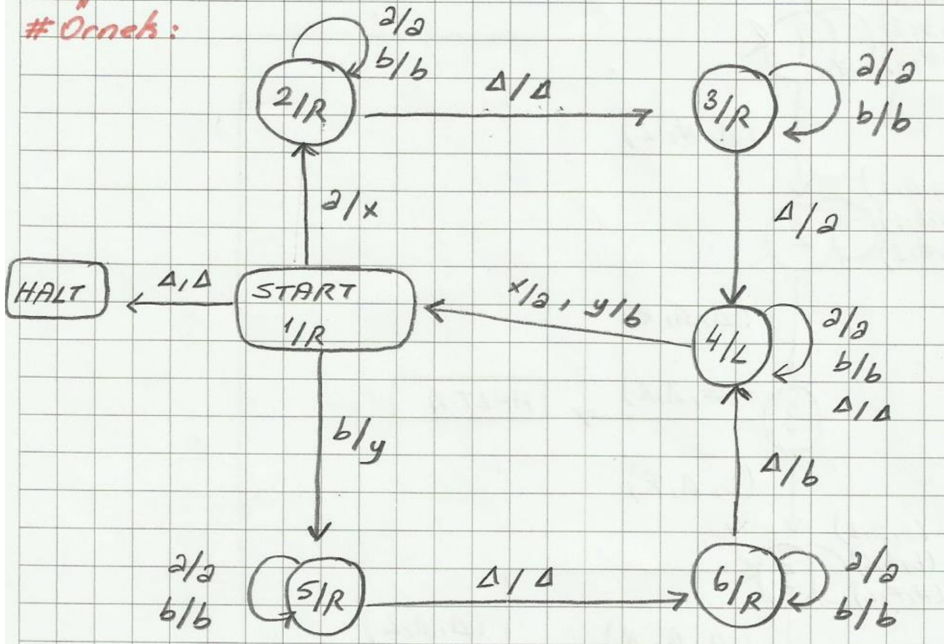
Δ	~~a~~ Δ	a	a	b	b	b	#


START
(Δ,Δ,R)
(a,a,R)
(b,b,R)
(#,#,R)
1
(Δ,#,L)
(a,a,L)
(b,b,L)
(#,#,L)
2
(Δ,Δ,R)
3
(#,Δ,R)
HALT 4
(a,Δ,R)
(a,a,L)
(b,b,L)
(#,#,L)
5
(Δ,a,L)
(Δ,Δ,R)
6
(a,Δ,R)
7
(a,a,R)
(b,b,R)
(#,#,R)
(b,Δ,R)
(a,a,L)
(b,b,L)
(#,#,L)
8
(Δ,b,L)
(Δ,Δ,R)
9
(b,Δ,R)
10
(a,a,R)
(b,b,R)
(#,#,R)
(#,Δ,R)

→ Variations on the TM

① Move-In-State Machine

Örnek:



* Tape'deki stringin peşine boşluk yapıp kopyasını boşluğun peşine yazar.

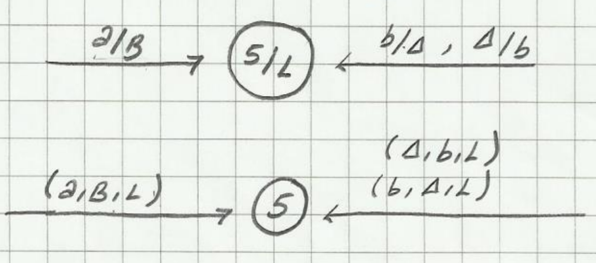
baab Δ baab

→ Mealy ve Moore türü makinenin TM'sine dönmüş hali gibi düşünülebilir. Geçişlerde gidilen durumun yönüne bağlı olarak hareket edilir.

baa Δ baa

baa Δ
yaa Δ Δ
yaa Δ b
baa Δ b
↑
bxa Δ b
bxa Δ ba
baa Δ ba

Move-In-State Machine ⟶ TM

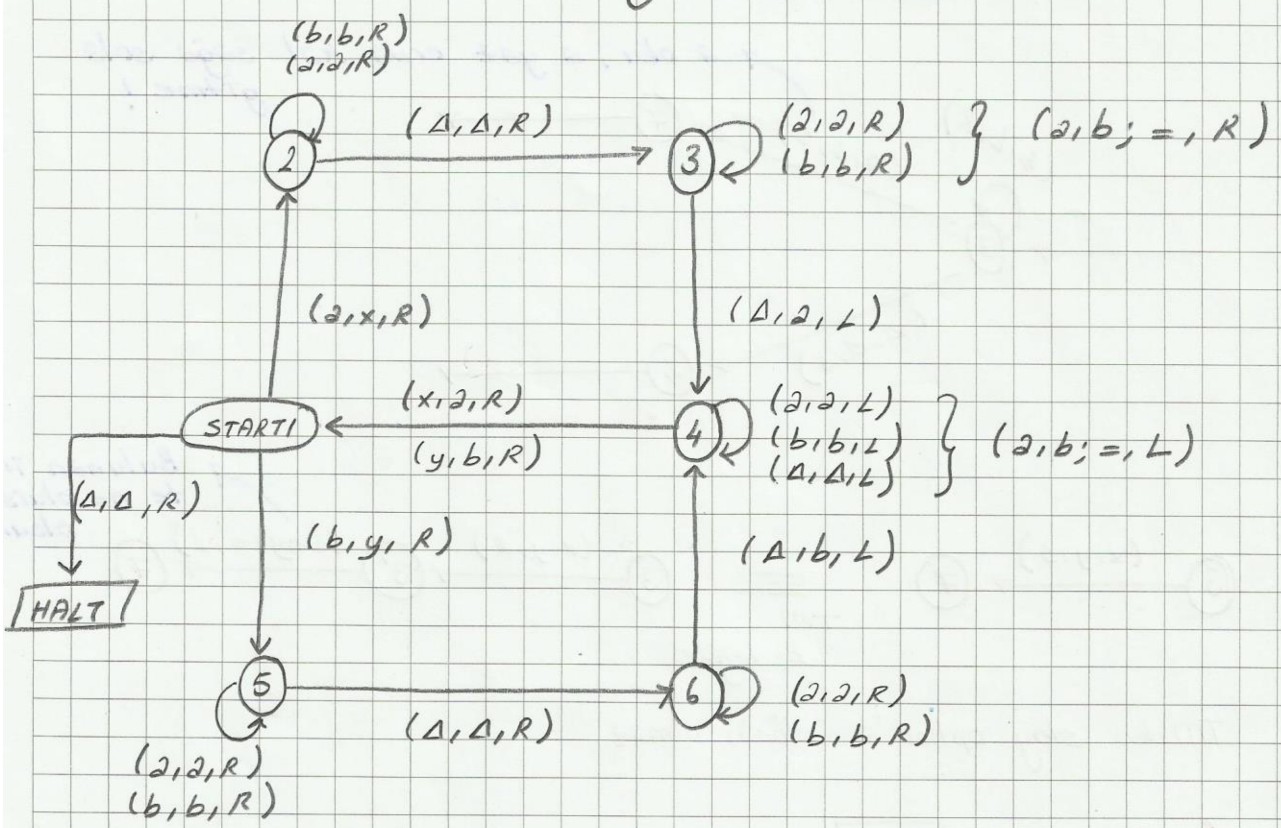


=> M-I-S-M

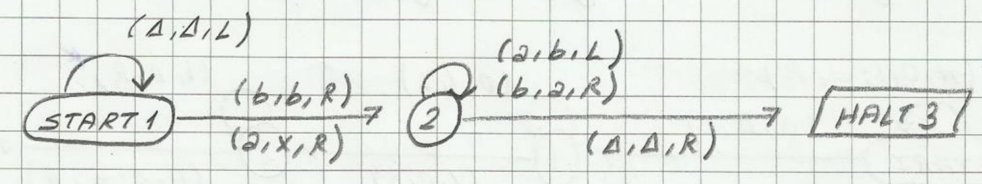
↓

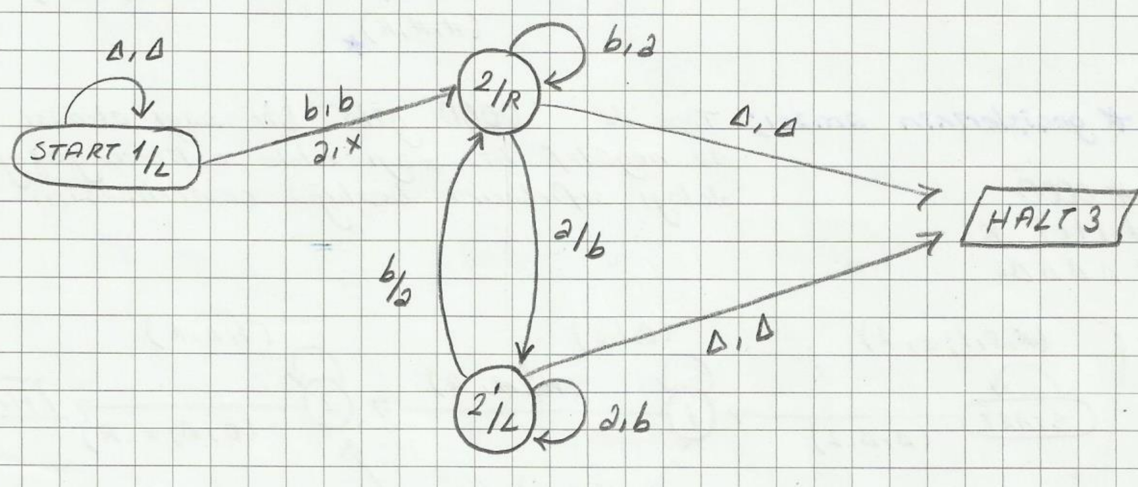
=> TM eşdeğeri

Örnek: Bir önceki MISM örneğinin TM'sini çizelim.



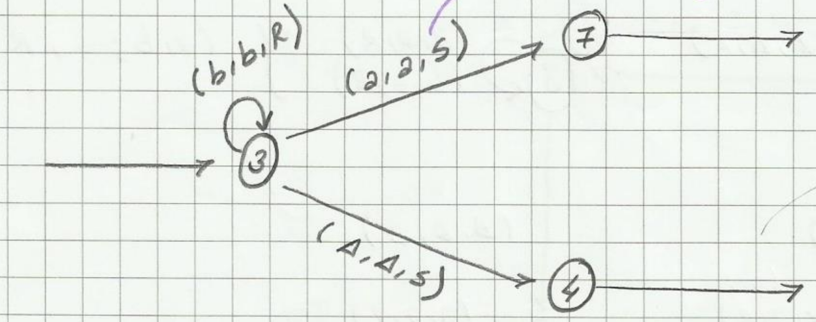
Örnek: TM → MOVE-IN-STATE M.



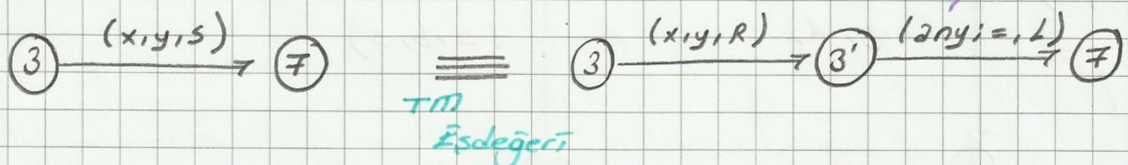


STAY-OPTION MACHINE

a oku, a yaz orda kal sağa sola gitme!



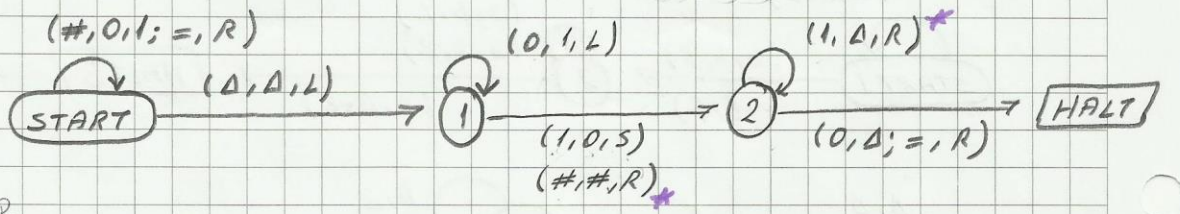
Bulunan indiste ne olursa olsun.



TM'den stay option'a dönüş olmaz.

 # Örnek: 10100$\boxed{1000}$ 8 } Bir sayının Binary decrementer (--)
10100$\underline{0111}$ 7 } Bir eksiği

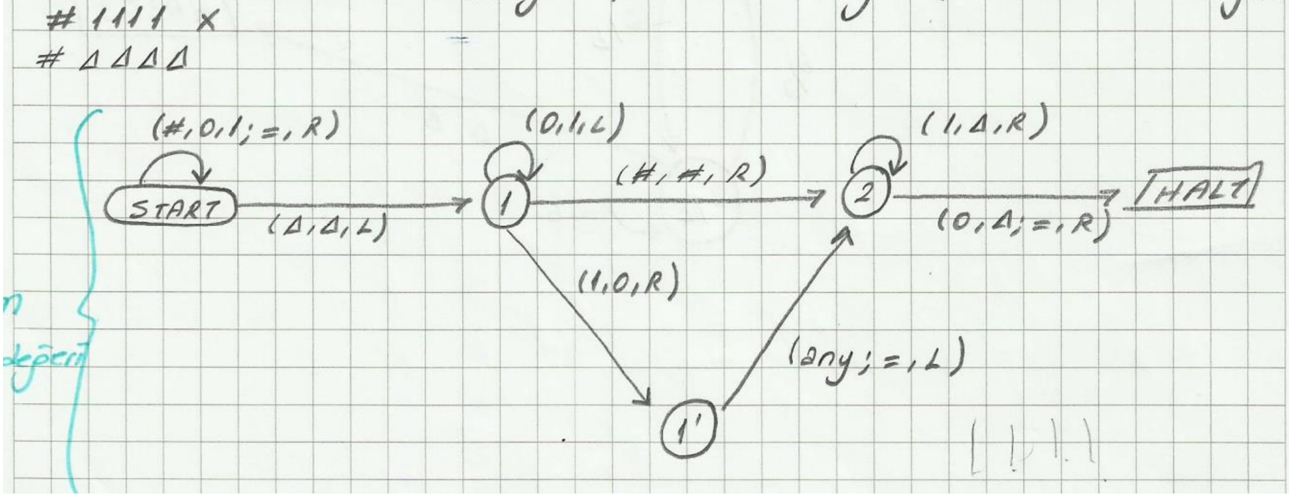
Tape'de a, b yerine 0, 1 olduğu varsayılır ve en başta '#' olduğu varsayılır.



e e R.

*** geçişlerinin amacı;** Tape'de '0000' gibi bir sayı olması durumunda negatif bir sayı elde edilemeyeceğinden dolayı sıfırların boşluğa çevrilmesini sağlar.

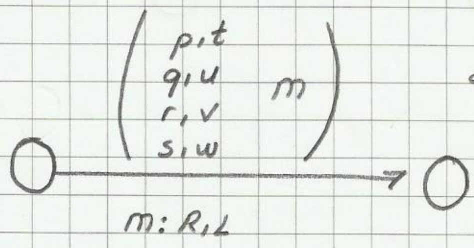
\# 0000
\# 1111 x
\# ΔΔΔΔ



(X,Y,3R) ⟶ X oku Y yaz 3 kere sağa git

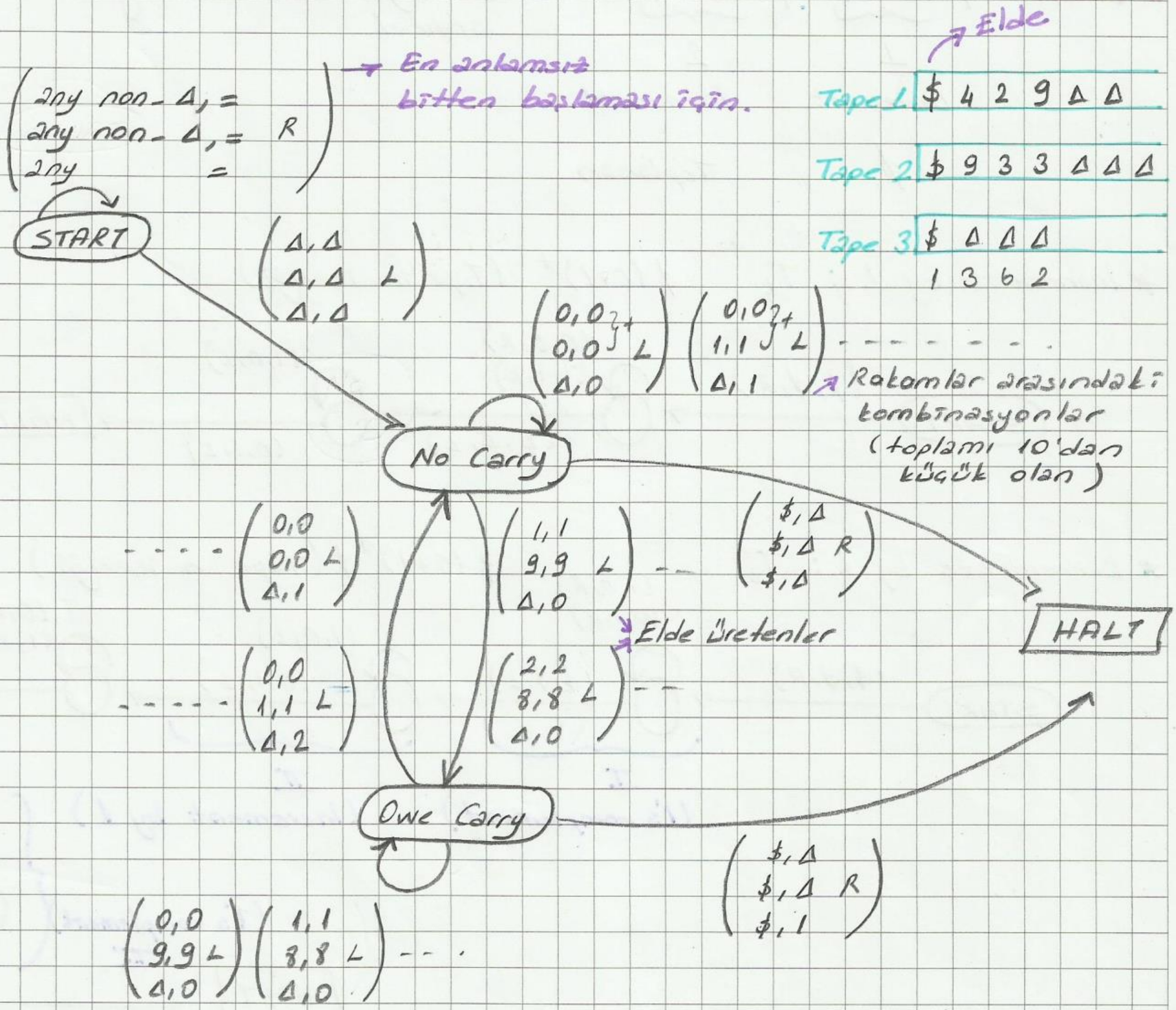
k-TRACK TM

Birden fazla tape mevcut. Aynı anda her ikisine okuma yazma yapılabilir.



dört tape var.

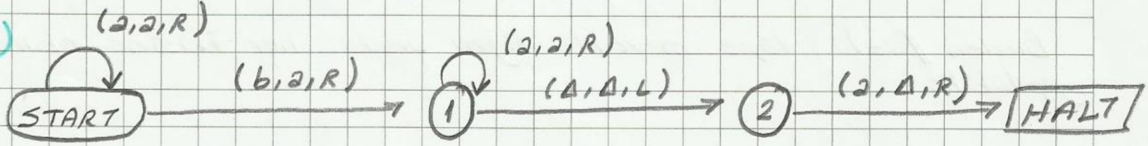
#Örnek: 1. ve 2. tape'deki sayılar toplanacak 3. tape yazılacak. Elde ihtimaline karşı tapelerin ilk indislerinde boşluk tutmakta fayda var.



→ COMPUTERS

Tm kullanarak 2 sayıyı toplama, çıkarma, max. olanın bulunması.

#Örnek: (TM ADDER)



$\underbrace{a^*}_{\text{toplanan1}} b \underbrace{a^*}_{\text{toplanan 2}}$

aaa b aaaaa

3 + 5

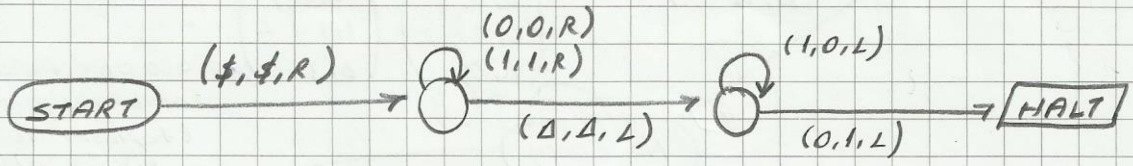
 # Örnek: $\underbrace{\$ (0+1)^*}_{1} \underbrace{\$ (0+1)^*}_{2}$

Bir sayı azalırken diğeri artıcak.

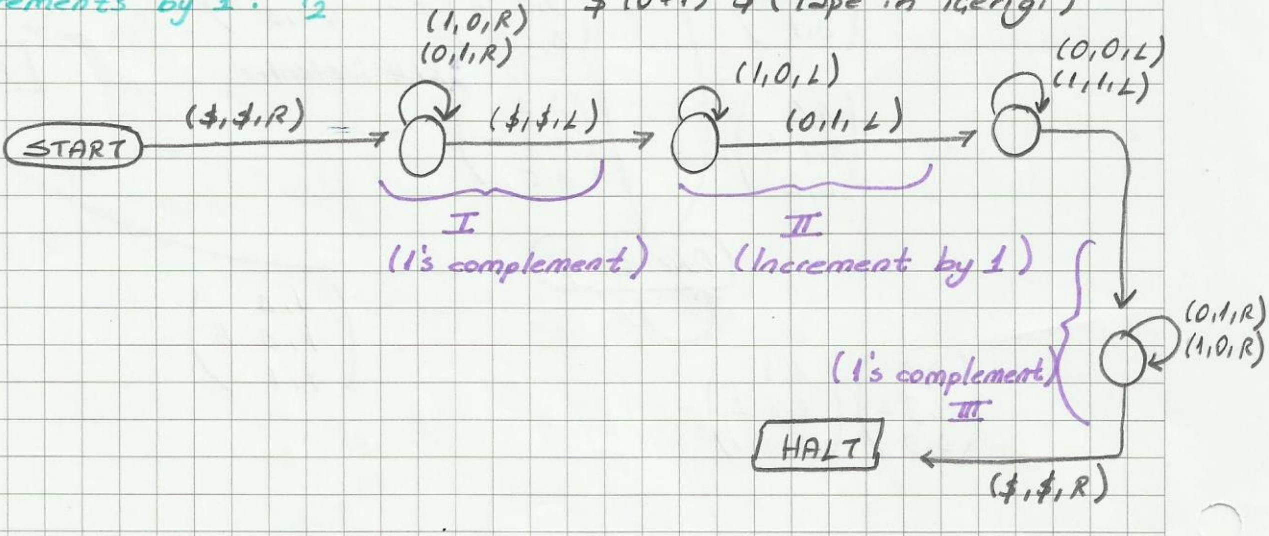
1: Decrement — ϕ

2: Increment — Toplanan

★ Increments by 1 : T_1 $\$(0+1)^*$ (Tape'in içeriği)



★ Decrements by 1 : T_2 $\$ (0+1)^* \$$ (Tape'in içeriği)



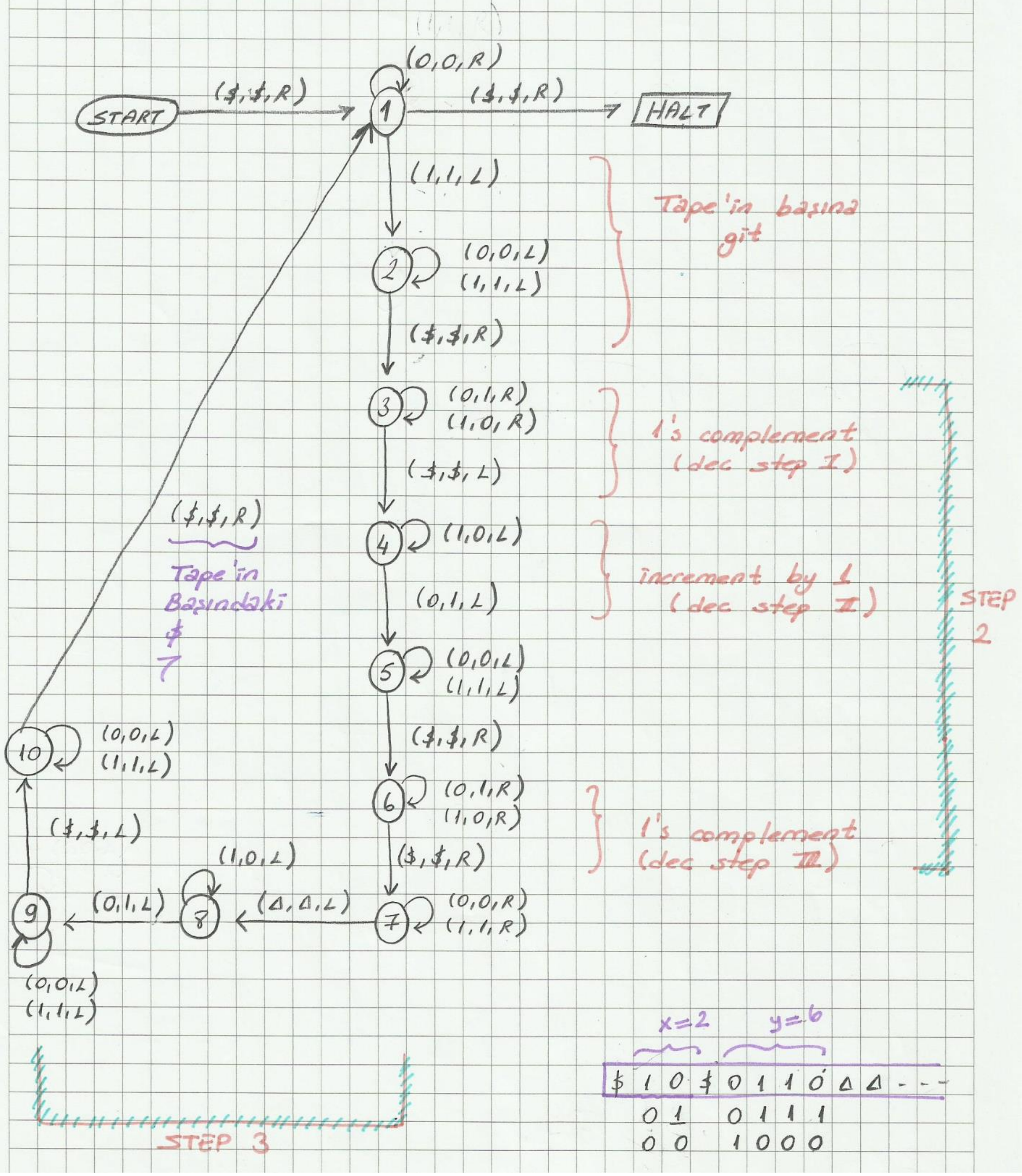
The algorith to calculate x+y in binary:

Step 1: Check the x-part to see whethes it is ϕ. If yes, halt. If no, proceed.

Step 2: Subtract 1 from the x-part using T_2.

Step 8: Add 1 to the y-part using T_1.

Step 4: Go to step 1.



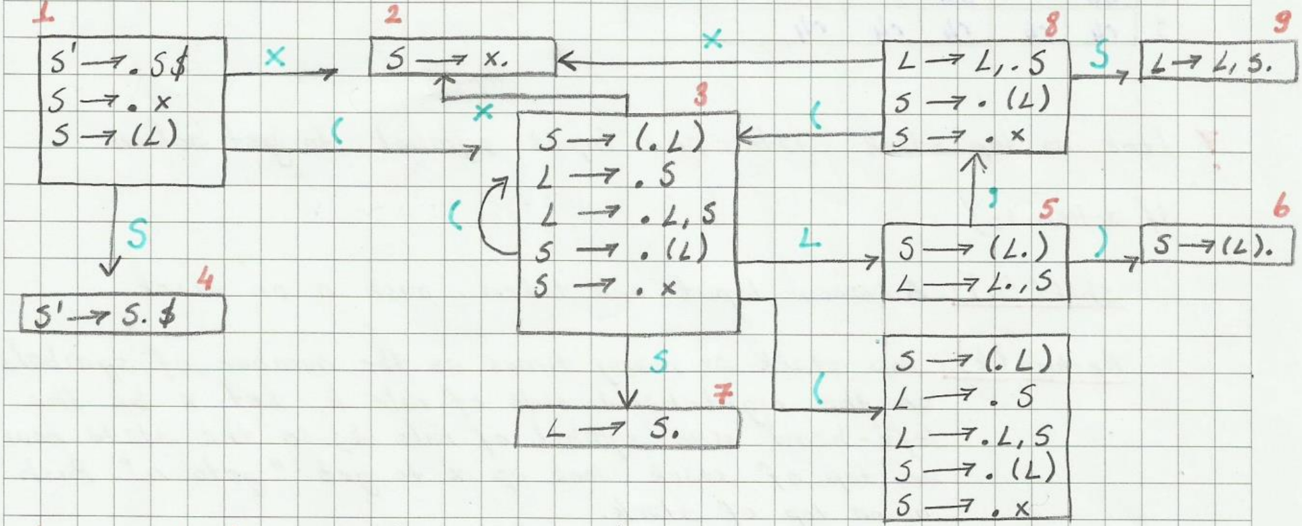
→ LR(∅) PARSING

LR → Left To Right

$S' \rightarrow S\$$ (S ile üretilen string bittiğinde \$'a rastlarsak accept yapabilmek için bu production'ı fazladan ekliyoruz)

her biri bir rule:
1. $S \rightarrow (L)$
2. $S \rightarrow x$
3. $L \rightarrow S$
4. $L \rightarrow L, S$

S, L : nonterminals
x, ( ) : terminals



1. numaralı durum; default olarak S' yazılır. Ardından S içeren terminaller yazılır.

2. numaralı durum; Eğer nokta en sona geldiyse buradan başka bir duruma geçiş olmaz.

3. numaralı durum; Noktadan sonra L geldiği için gramerde L olanlar yazılır.

4. numaralı durum; .\$ olduğunda daha fazla ilerletilmez. Accept yapıldığından.

s → shift (geçişlerde terminal karakter olduğunda kullanılır)
g → go to (geçişlerde nonterminal karakter olduğunda kullanılır)
r → reduce (sonlanan ifadelerde kendisinden başkasına geçiş olmayan durumlarda kullanılır)

Tabloda satırlar durum, sütunlar önce terminal sonra nonterminal olarak yazılır.

Tablo durumlar yazılırken "." sembolü productionın sonuna gelmiş ise reduce yazılır.

	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2				
9	r4	r4	r4	r4	r4		

! Look up top stack state, and Input symbol, to get action.

If action is;

shift(n): Advance Input one token, push n on stack

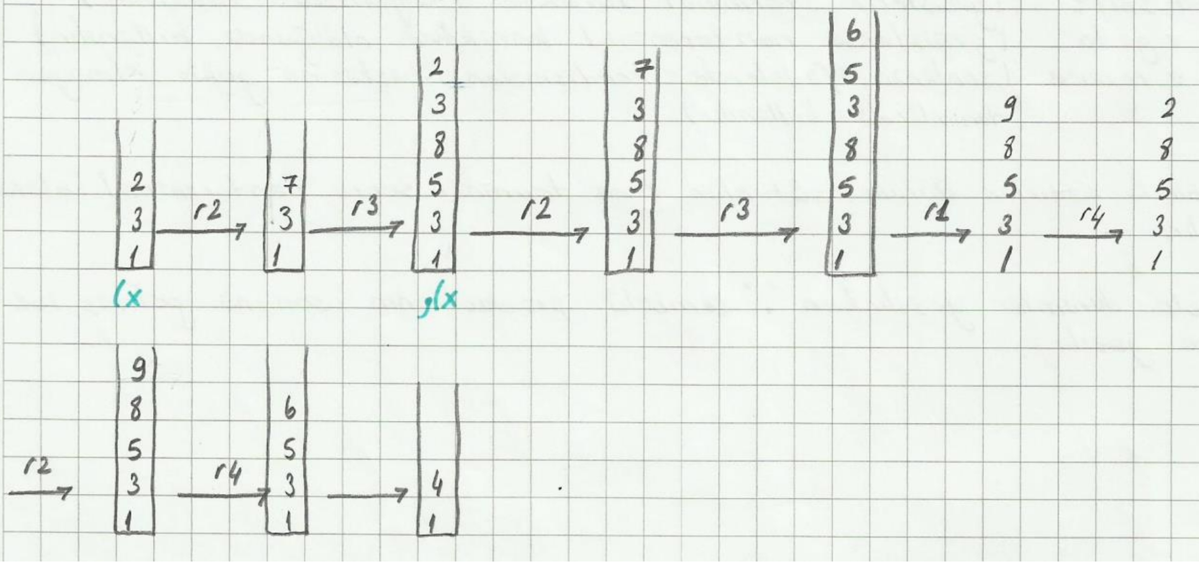
Reduce(k): Pop stack as many times as the number of symbols on the right-hand side of rule k; Let x be the left-hand side symbol of rule k; in the state now on top of stack look up x to get "goto n". Push n on top of stack.

Accept: Stop parsing, report success.

Error: Stop parsing, report failure.

#Örnek: (x, (x), x) $ ifadesini parsing tablosunun kabul edip etmediğine bakalım.

İlk olarak yığında ① numaralı durumun olduğunu varsayıyoruz.



→ LR(1) PARSING

LR(∅)'ın yetersiz kaldığı iki durum var. (Dezavantaj)

1) Shift / reduce conflict

2) Reduce / Reduce conflict

LR(1)
↓
Look-ahead symbol

LR(0)'da o anda kafanın olduğu yere bakılırken LR(1)'de bir sonraki karakterinde ne olduğuna bakılır.

①

→ E → T. + E   (shift)
   E → T.   (reduce)

②

→ E → E - T.
   E → T.

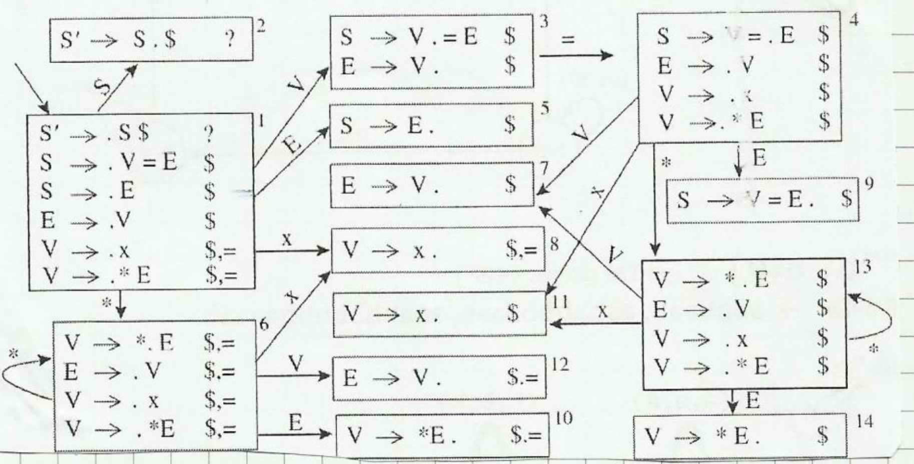
} Her ikisinde de reduce yapmak istiyor

FIRST() } sets
FOLLOW()

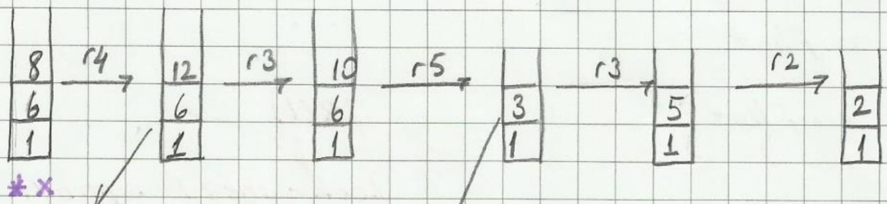
Look-ahead karakterini oluşturmak için 2 tane küme oluşturmamız gerekiyor. FIRST SET ve FOLLOW SETS

LR(1) yukarıdaki karmaşıklık durumlarında ne yapılması gerektiğine çözüm getirir.

0 $S' \rightarrow S\,\$$
1 $S \rightarrow V = E$
2 $S \rightarrow E$
3 $E \rightarrow V$
4 $V \rightarrow x$
5 $V \rightarrow *\,E$

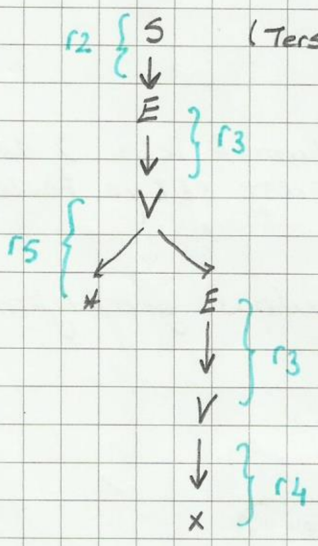


#Örnek: * x $ LR(1)'e göre parse edelim.



6'dayken
r4'de bakarız
r4'de V var
tabloda V
ile çakışma
yaparız g12

Okunması beklenen
karakter $ olduğu
için reduce yapılır.



(Tersten gidilerek pars ağacı çıkarılabilir)

	x	*	=	$	S	E	V
1	s8	s6			s2	s5	s3
2				a			
3			s4	r3			
4	s11	s13				s9	s7
5				r2			
6	s8	s6				s10	s12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				s14	s7
14				r5			