KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BIL 2010 OTOMATA TEORİSİ DERS NOTLARI

HASRET SUNA DİZDAR

2015-2016 Bahar Dönemi

* Introduction to Computer Theory (D. Cohen)

OTOMATA TEORİSİ

15.02.2016

Regular Expressions (RE)

Diller tanımlanırken alfabeyi "Σ" ile temsil ederiz.

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

$a^* = \{ \underset{\text{boş string}}{\lambda}, a, aa, aaa, \dots \}$

$a^+ = \{ a, aa, aaa, \dots \} \Rightarrow a^+ = aa^*$

$ab^* = \{ a, ab, abb, abbb, \dots \}$

$(ab)^* = \{ \lambda, ab, abab, ababab, \dots \}$

$(a+b)^*$ → a veya b'den oluşabilecek tüm RE'leri temsil eder.

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

$(a+b)(a+b) = \{aa, ab, ba, bb\}$ } sınırlı sayıda üretim yapar.

$[(a+b)(a+b)]^*$ → çift sayıda kelime üreten RE

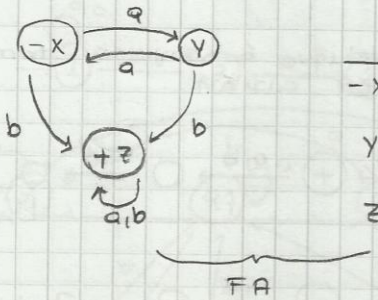
$(a+b)[(a+b)(a+b)]^*$ → tek sayıda kelime üreten RE

$a(a+b)^*b$ → a ile başlayıp b ile biten kelimeler

Finite Automata (FA)

(-X) → başlangıç durumu - ile ifade edilir. (→ (X))

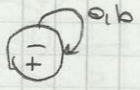
(+Z) → bitiş durumu + ile ifade edilir. ((Z))



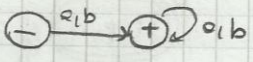
	a	b
-X	Y	Z
Y	X	Z
Z	Z	Z

FA

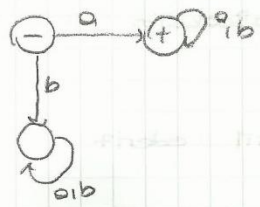
* $(a+b)^*$



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

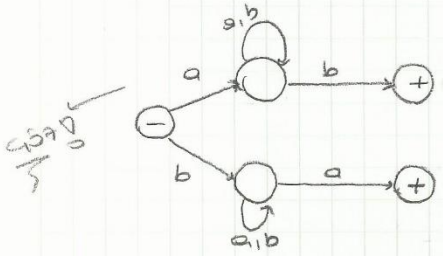


* $a(a+b)^*$

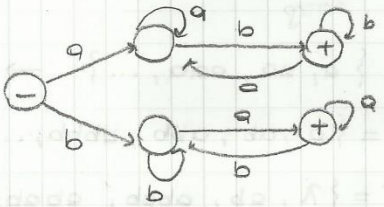


▽ nondeterministic FA yo ekvik yo do bir durum için iki ayrı yere gidiyorsa oluşur

* $a(a+b)^*b + b(a+b)^*a$ → a ile başlayıp b ile biten veya b ile başlayıp a ile biten.

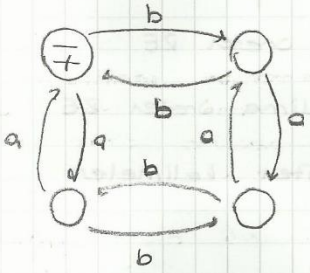


Non-Deterministic FA



Deterministic FA

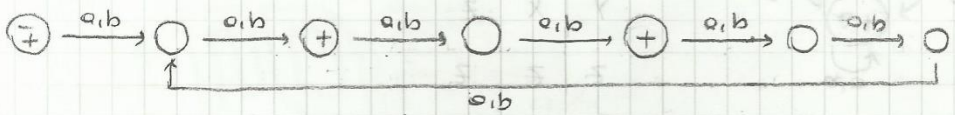
* Çift sayıda a ve b 'den oluşan tüm kelimelerin FA'ini çiziniz.

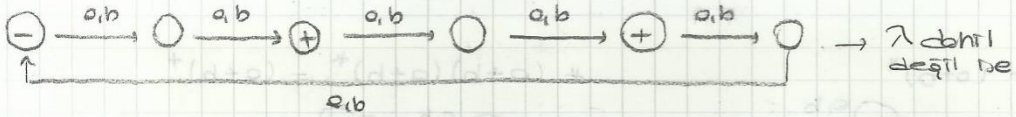


(+) → Boş string (λ)

▽ RE 'den FA'e geçmek zor olduğu için RE → NFA → FA şeklinde bulucaz.

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



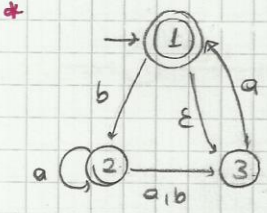


* 6'ya tam bölünen yerde + olmaz.

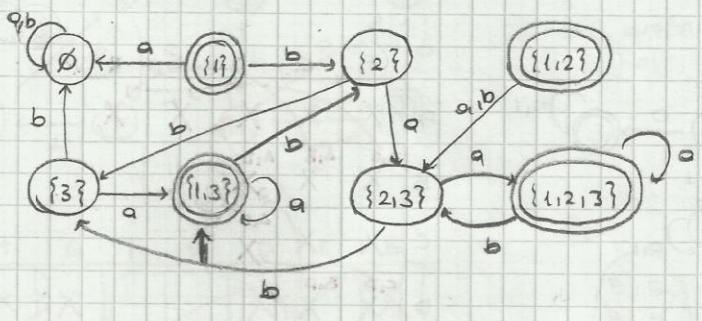
* Çift sayılarda + var fakat 6'nın tam katlarında yok.

NFA to FA

*



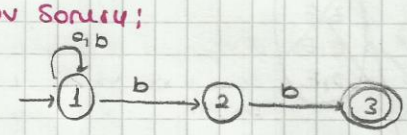
Nondeterministic'tir. Çünkü;
- 2 durumunda a gelince hem kendine hem 3'e gidiyor.
- 3 durumunda b gelince nereye gittiği belli değil.



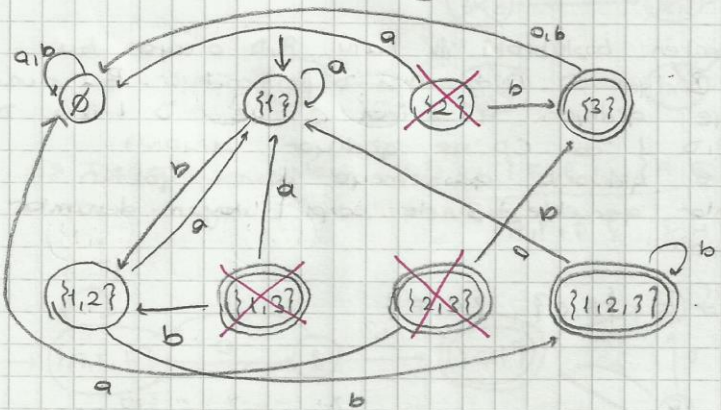
Yukarıdaki FA, deterministic FA'dir. Bu FA'de başlangıç için {1,3} seçildi. Çünkü 1-3 arasında ε geçişi vardır. Bitiş için de 1'in bulunduğu bütün elemanlar seçiliyor. NFA'daki belirsiz durumlar için o ilişkiyi boş kümeye göndermiş. Yani 3'de iken b geldiğinde bir duruma gitmediği için {3} durumunda b geldiğinde boş kümeye gider.

Geçişleri tamamladıktan sonra bir duruma geçiş yoksa bu durumları silebiliriz. Bu durumda {1} ve {1,2} durumlarını silebiliriz. Çünkü onlara herhangi bir durumdan gelen bir ilişki yoktur.

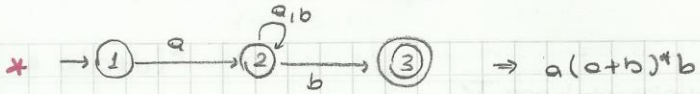
Sınav Sonucu;

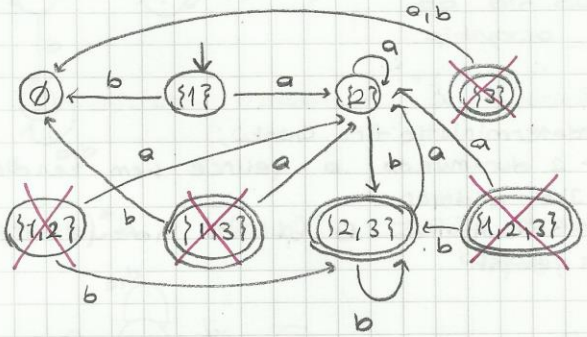


} NFA'i FA'e dönüştürün.



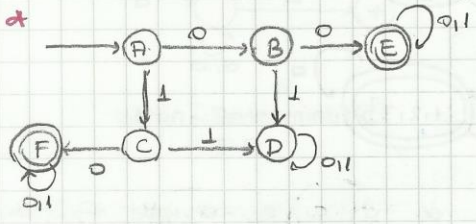
* {2}, {1,3}, {2,3} durumları silinince {3} ve ∅'nin de bağlantıları silindiği için onlarda siliniyor ve 3 durum kalıyor.

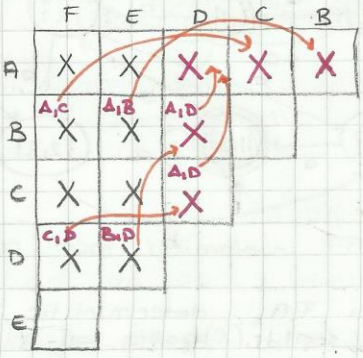




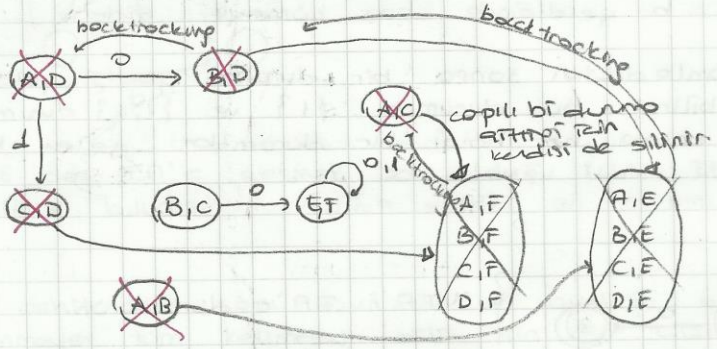
* Başlangıç durumlarına bazko durumlardan ilişki gelmese bile silemeyiz.

MINIMIZING FA





- Çıkış durumları hiçbir durum eş değer olamaz o yüzden onları direk silebiliriz.
- Daha sonra dependency graf çizilir bunu da çarpı olmayan ilk ilişkiden başlayarak yerleştiriyoruz.

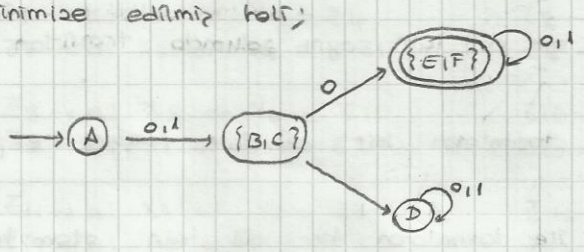


- Dependency graf çizerken boşluktaki ilk kutu A,D oluyor bu durumda A 0'da iken B'ye D 0'da iken D'ye gidiyor. Bu durumda A,D 0'da iken B,D'ye gidiyor. A 1 iken C'ye D 1 iken D'ye gidiyor o yüzden A,D 1'de CD'ye gidiyor diyoruz.
- D,D, A,A, vb bi durumu gidiyorsa göstermeye gerek yoktur.
- Her durum için bunlar gerçekleştirildiğinde çarpı olmayan durumlar eş değer durumlar oluyor.

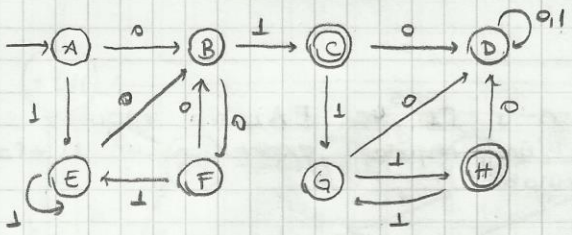
Eş Değer Durumlar = {B,C}, {E,F}

* 6 durumlu tüm FA'lerden rastgele minimize edilebilir, bulup text dosyasına yazdırılacak. (Daha sonra klavyeden girilen sayıda...)

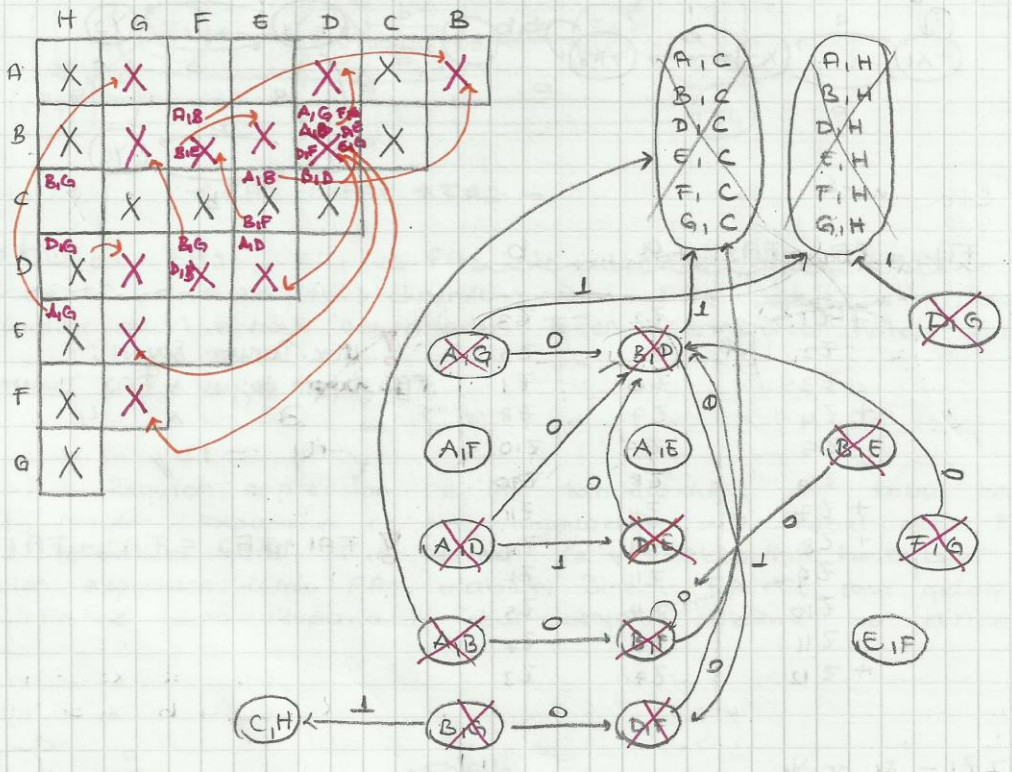
- Minimize edilmiş hali;



*

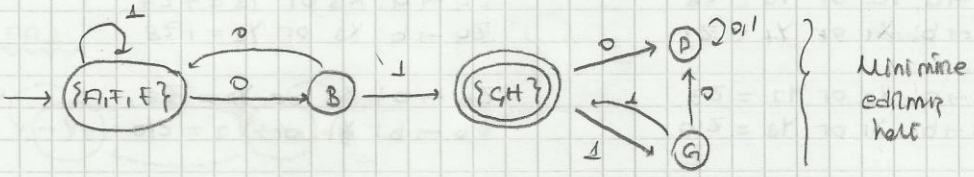


Bu grafın minimize edilmiş hali nasıldır?



Eş Değer Durumlar: {A,F}, {F,E}, {A,E}, {C,H}

{A,F,E}, {C,H}



KLEENE'S THEOREM

Bu teorem 3 bölümden oluşur.

1. FA ile tanımlanan bir dil için aynı zamanda transition graph bulunmaktadır.

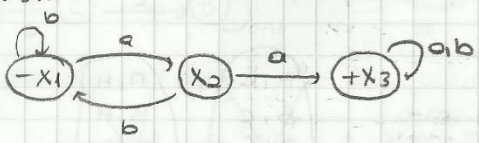
2. Transition graph ile tanımlanan bir dil için regular expression bulunmaktadır.

3. Regular expression ile tanımlanan bir dil için otomata teorisi bulunmaktadır.

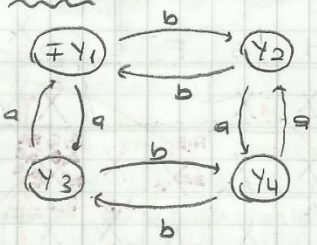
3. Bölüm

Kural 2: FA_1 'in regular expression 'u r_1 ve FA_2 'nin regular expression 'u r_2 'dir. FA_3 'ün regular expression 'u r_1+r_2 'dir. r_1+r_2 den 3. bir FA oluşturulabilir.

+ FA_1



FA_2



$FA_3 = FA_1 + FA_2$	a	b
∓ Z1	Z2	Z3
Z2	Z4	Z5
Z3	Z6	Z1
+ Z4	Z7	Z8
Z5	Z9	Z10
Z6	Z8	Z10
+ Z7	Z4	Z11
+ Z8	Z11	Z4
Z9	Z11	Z1
Z10	Z12	Z5
Z11	Z8	Z2
+ Z12	Z7	Z3

! Max. Durum Sayısı;
FA1 Durum Sayısı x FA2 Durum Sayısı
3 x 4
= 12 //

! $FA1 + FA2 = FA2 + FA1$ ✓

$\mp Z1 = X1$ or $Y1$
$Z1 \to a$: $X2$ or $Y3 = Z2$
$Z1 \to b$: $X1$ or $Y2 = Z3$

$Z2 \to a$: $X3$ or $Y1 = +Z4$
$Z2 \to b$: $X1$ or $Y4 = Z5$

$Z3 \to a$: $X2$ or $Y4 = Z6$
$Z3 \to b$: $X1$ or $Y1 = Z1$

$Z4 \to a$: $X3$ or $Y3 = +Z7$
$Z4 \to b$: $X3$ or $Y2 = +Z8$

$Z5 \to a$: $X2$ or $Y2 = Z9$
$Z5 \to b$: $X1$ or $Y3 = Z10$

$Z6 \to a$: $X3$ or $Y2 = Z8$
$Z6 \to b$: $X1$ or $Y3 = Z10$

$z_7 \to a: X_3 \text{ or } Y_1 = z_4$
$z_7 \to b: X_3 \text{ or } Y_4 = + z_{11}$

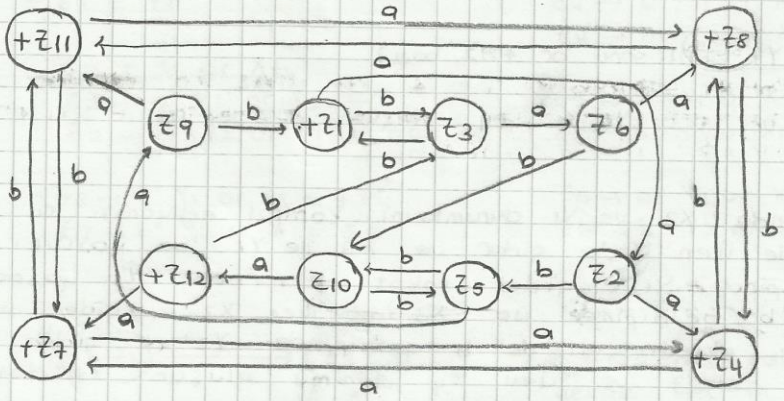
$z_8 \to a: X_3 \text{ or } Y_4 = z_{11}$
$z_8 \to b: X_3 \text{ or } Y_2 = z_4$

$z_9 \to a: X_3 \text{ or } Y_4 = z_{11}$
$z_9 \to b: X_1 \text{ or } Y_1 = z_1$

$z_{10} \to a: X_2 \text{ or } Y_1 = + z_{12}$
$z_{10} \to b: X_1 \text{ or } Y_4 = z_5$

$z_{11} \to a: X_3 \text{ or } Y_2 = z_8$
$z_{11} \to b: X_3 \text{ or } Y_3 = z_7$

$z_{12} \to a: X_3 \text{ or } Y_3 = z_7$
$z_{12} \to b: X_1 \text{ or } Y_2 = z_3$



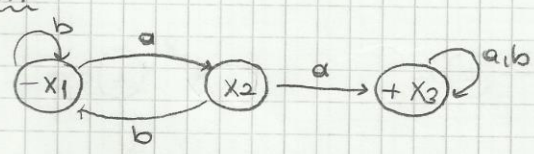
$-FA3 = FA1 + FA2-$

* Oluşan FA3, FA1 ve FA2 'nin toplamı olduğu için FA1 'de ki $(a \to a \to a, \forall b:)$ durumları da, FA2 'deki $(a \to a, b \to b, \forall b.)$ durumları da içermek zorundadır. Eğer içermiyorsa hatalı olur.

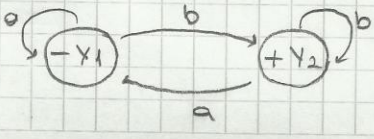
29.02.2016

Kural 3: Regular expression r_1 'in tanımladığı dili kabul eden FA1, regular expression r_2 'nin tanımladığı dili kabul eden FA_2'dir. $r_1 r_2$ şeklindeki yeni FA'in ucuna r_2'yi eklemiş halindeki regular expression için FA3 oluşur. Birinci FA'de sona geldiğimizde ikinci FA'de başta olduğumuzu kabul ediyoruz. Ekleme bu şekilde olur.

FA1



FA2



FA3 = FA1 * FA2	a	b
- z1	z2	z1
z2	z3	z1
z3	z3	z4
+ z4	z3	z4

FA1 * FA2 ≠ FA2 * FA1

$-z_1 = X_1$

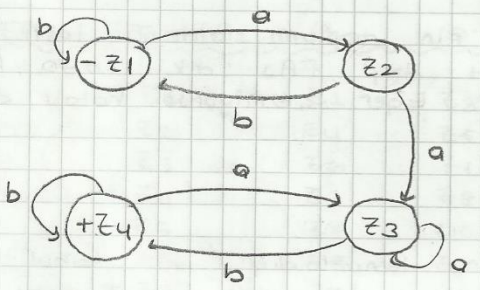
$z_2 = X_2$

$z_3 = X_3$ or Y_1 (FA1 sonu or FA2 bozl)

$z_3 \to a$: X_3 or $Y_1 = z_3$

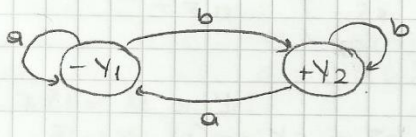
$z_3 \to b$: X_3 or Y_1 or $Y_2 = +z_4$

z_3 durumunda X_3 ve Y_1 durumlarını kontrol ediyoruz. "a" geldiğinde X_3'de iken X_3'e gider ve X_3 ile Y_1'de yazılır. Y_1'de a geldiğinde Y_1'e gider fakat tekrardan Y_1 yazmamıza gerek yoktur. "b" geldiğinde ise X_3'de iken X_3'e gider bu durumda Y_1'de eklenir. Y_1'de b geldiğinde Y_2'ye gider bu yüzden bunu ekleriz ve yeni z_4 durumu oluşur. Y_2'den dolayı da $+z_4$ olur.

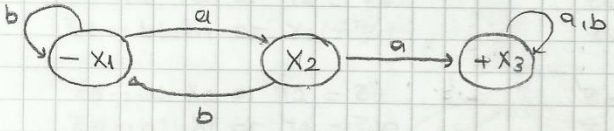


- FA3 = FA1 * FA2 -

FA2



FA1



FA3 = FA2 * FA1	a	b
−z1	z1	z2
z2	z3	z2
z3	z4	z2
+z4	z4	z5
+z5	z6	z5
+z6	z4	z5

$-z_1 = Y_1$

$z_2 = Y_2$ or X_1

$z_2 \to a$: Y_1 or $X_2 = z_3$

$z_2 \to b$: Y_2 or $X_1 = z_2$

$z_3 \to a$: Y_1 or $X_3 = +z_4$

$z_3 \to b$: Y_2 or $X_1 = z_2$

$z_4 \to a$: Y_1 or $X_3 = +z_4$

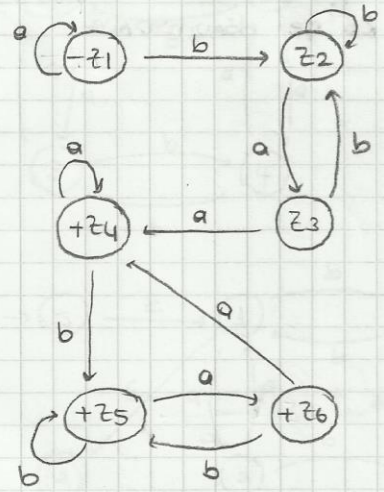
$z_4 \to b$: Y_2 or X_1 or $X_3 = +z_5$

$z_5 \to a$: Y_1 or X_2 or $X_3 = +z_6$

$z_5 \to b$: Y_2 or X_1 or $X_3 = +z_5$

$z_6 \to a$: Y_1 or $X_3 = +z_4$

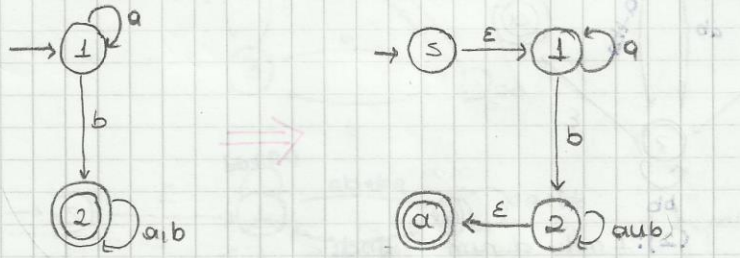
$z_6 \to b$: Y_2 or X_1 or $X_3 = +z_5$



* FA2'nin sonu bi durumda gelirse FA1'in başını da yazmalıyız.

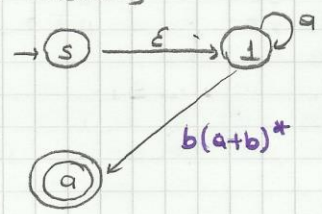
- $FA_3 = FA_2 * FA_1$ -

FA → RE Dönüşümü

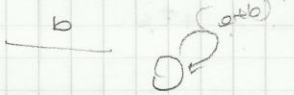


★ Başlangıç durumu yerine s adlı yeni bir başlangıç durumu, bitiş durumu yerine a adlı yeni bir bitiş durumu oluşturup. aralara ε geçişi yapılıyor. Daha sonra yavaş yavaş diğer durumlar siliniyor.

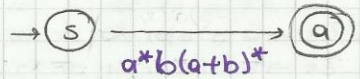
- 2'yi silersek;



* Döngü olduğu için $(2)^{a|b}$ $(a+b)^*$ yapılıyor. b'de çarpım yapılıyor.

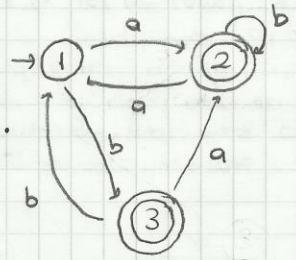


- 1'i silersek;

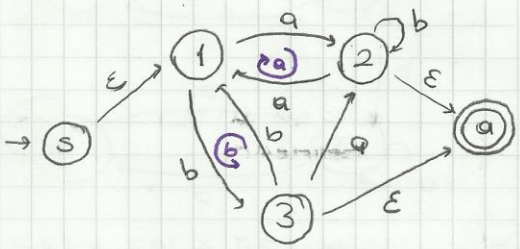


* Silinen durum üzerindeki durumlarda "*" koyulur. Durumlar arası döngüler çarpım durumunda yazılır.

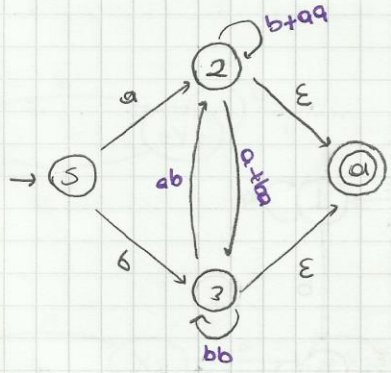
*



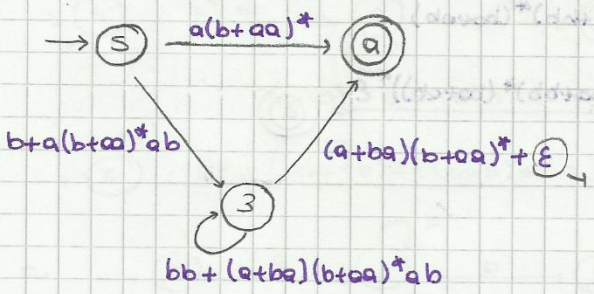
FA'ini RE'ye dönüştürüyoruz.



(1): s, a ve ε geçişleri yapıldı.

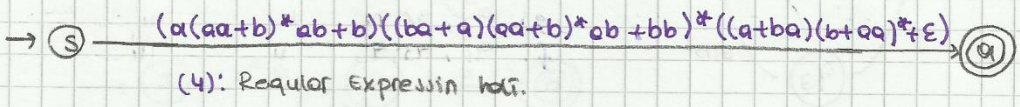


(2): 1 nolu durum silindi.



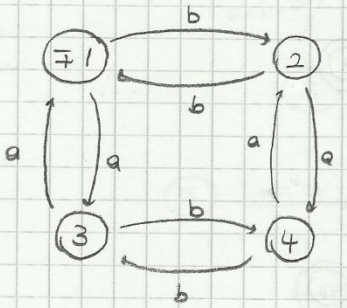
3-a oğlu ε ile

(3): 2 nolu durum silindi.

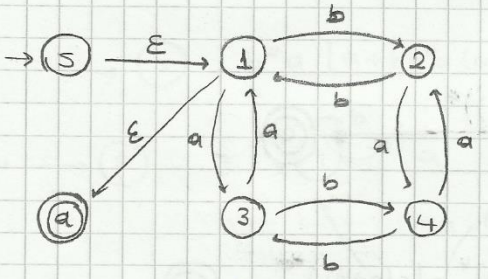


(4): Regular Expressin hali.

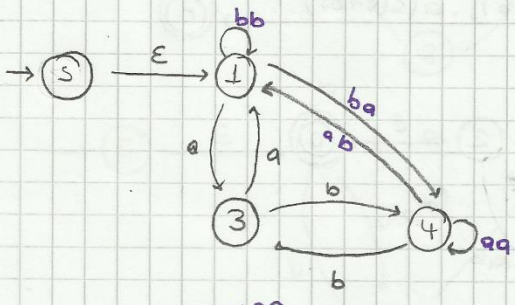
*



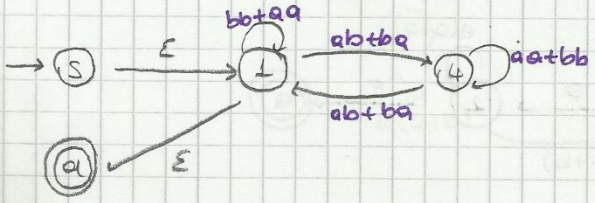
FA'ini RE ye dönüştürünüz.



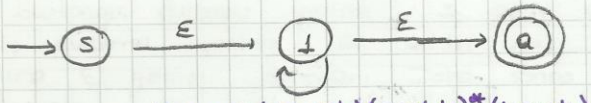
* s ve a eklenir. ε geçişler belirlenir.



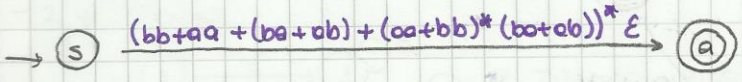
* 2 nolu durum silinir



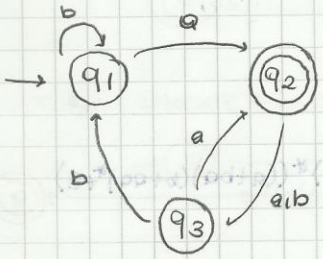
* 3 nolu durum silinir



* 4 nolu durum silinir.

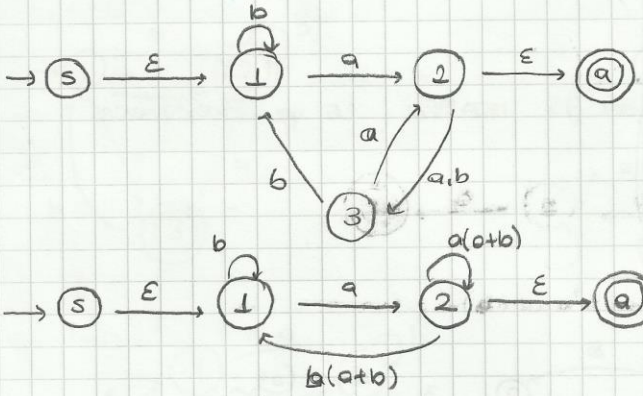


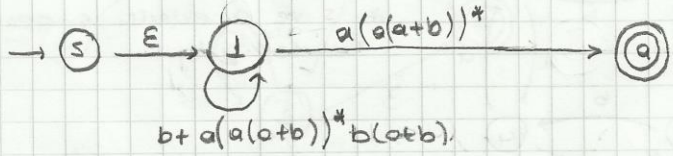
2014
SINAV SORUSU

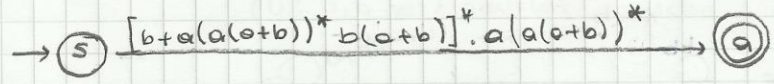


a. q_3, q_2, q_1 sırasıyla silinir.
b. q_3, q_1, q_2 " "
c. q_1, q_3, q_2 " "
d. q_2, q_3, q_1 " "
e. q_2, q_1, q_3 " "
f. q_1, q_2, q_3 " "

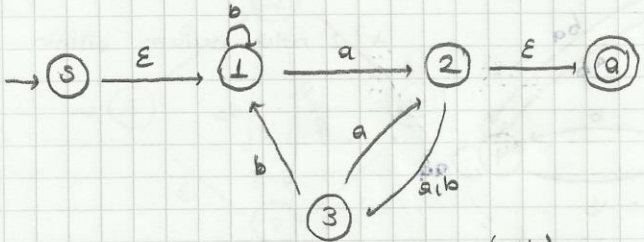
a.

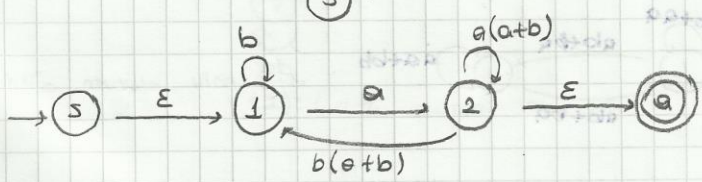


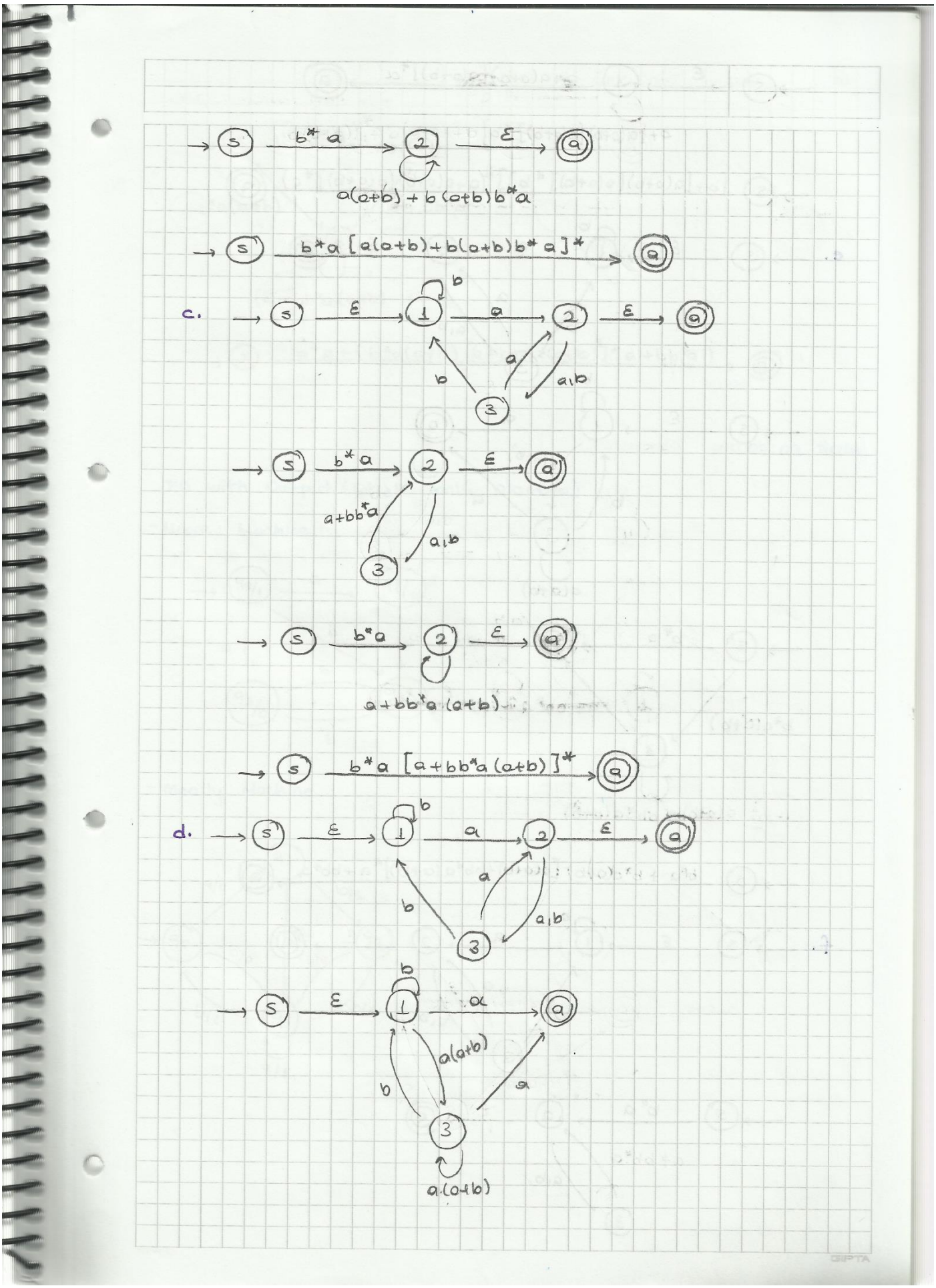


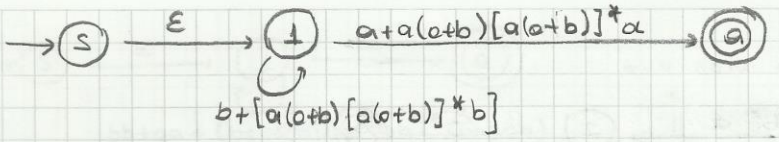


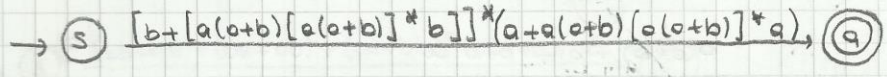
b.



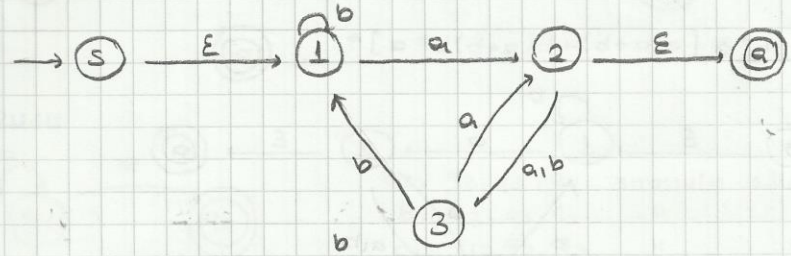


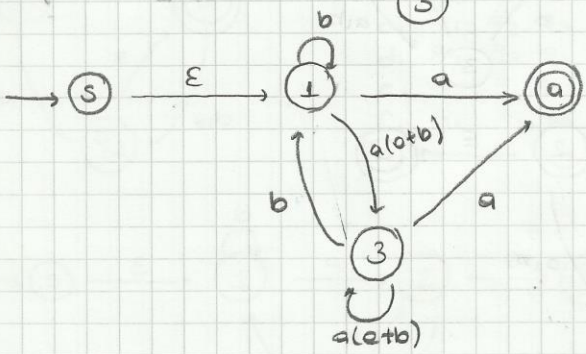

s
b* a
2
ε
a
a(a+b) + b(a+b)b*a
b*a [a(a+b)+b(a+b)b* a]*
c.
1
b
a
a,b
3
a+bb*a
d.

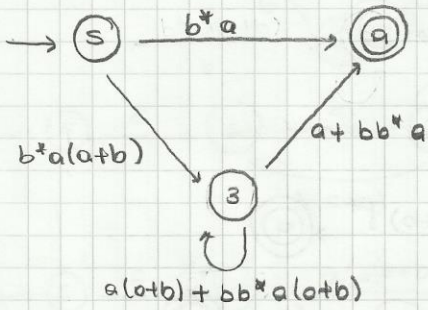

S
ε
1
a+a(a+b)[a(a+b)]*a
2
b+[a(a+b)[a(a+b)]*b]

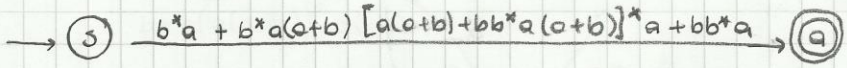

S
[b+[a(a+b)[a(a+b)]*b]]*(a+a(a+b)[a(a+b)]*a)
a

e.

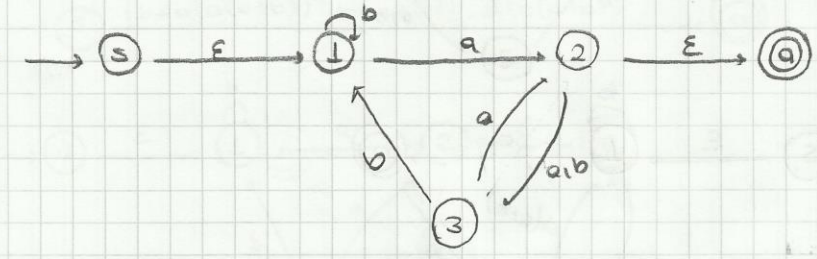

S
ε
1
b
a
2
ε
a
b
a,b
3

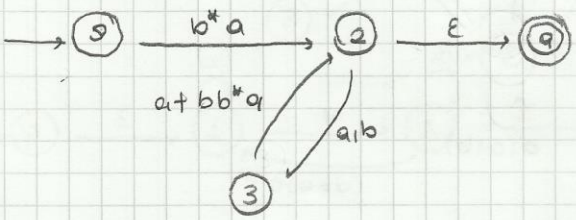

S
ε
1
b
a
a(a+b)
b
a
3
a(a+b)

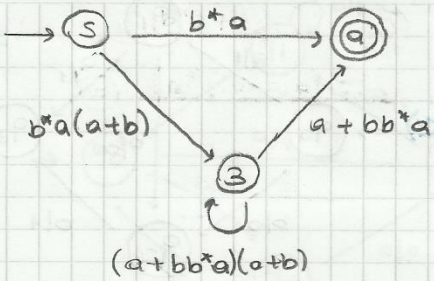

S
b*a
a
b*a(a+b)
a+bb*a
3
a(a+b)+bb*a(a+b)


S
b*a+b*a(a+b)[a(a+b)+bb*a(a+b)]*a+bb*a
a

f.


S
ε
1
b
a
2
ε
a
b
a,b
3


S
b*a
2
ε
a
a+bb*a
a,b
3

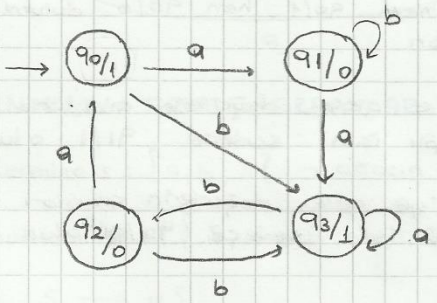


$$\rightarrow (S) \xrightarrow{b^*a + (b^*a(a+b)[(a+bb^*a)(a+b)]^*a + bb^*a)} (a)$$

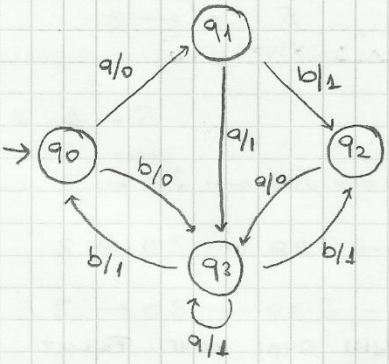
07.03.2016

FA with Output (Çıktılı Sonlu Otomata)

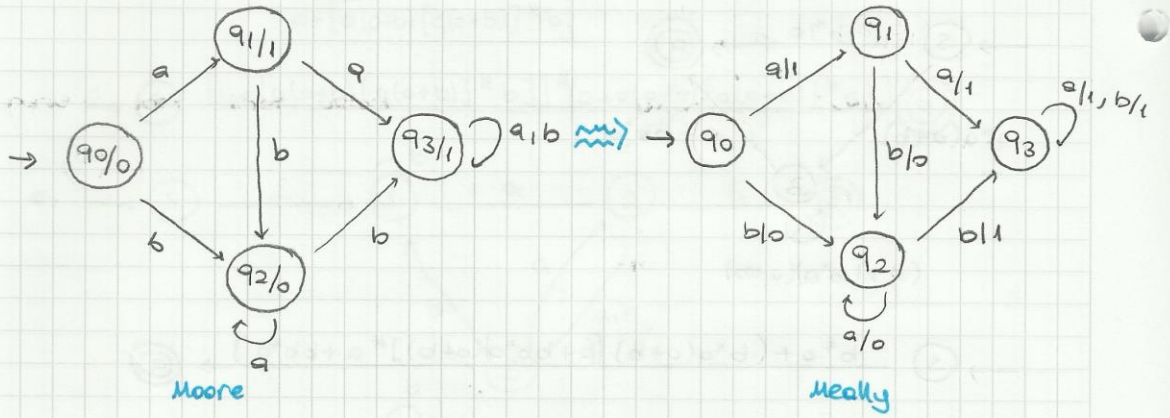
- Moore Machine



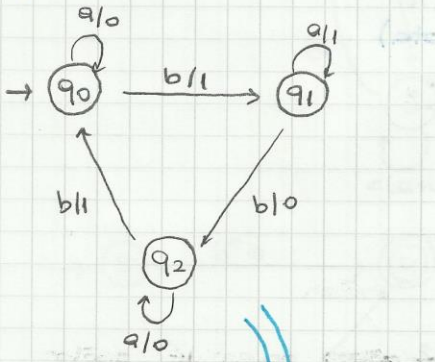
- Mealy Machine



Moore → Mealy Dönüşümü



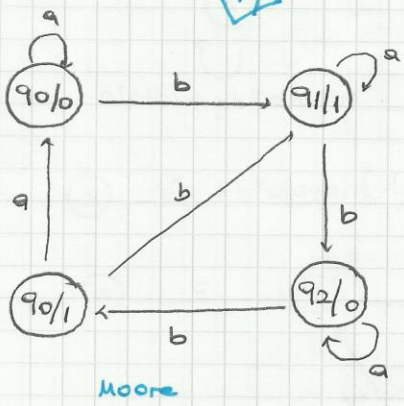
Meally → Moore Dönüşümü



* q_0'a gelen değerler 0 ve 1 olduğu için hem $q_0/1$ hem $q_0/0$ durumları oluyor

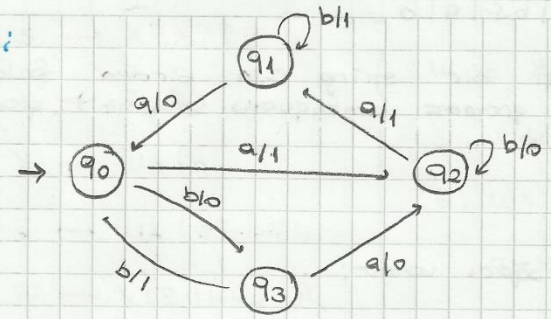
* q_1'e gelen değerlerin çıkışları 1 olduğu için sadece $q_1/1$ olur.

* q_2'ye gelen değerlerin çıkışları 0 olduğu için sadece $q_2/0$ olur

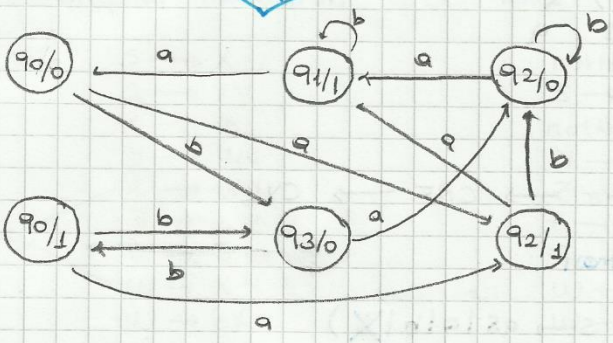


Moore → Meally dönüşümünde durum sayısı aynı kalır. Fakat Meally → Moore dönüşümünde durum sayısı artabilir.

ÖRN:



Moore tarzı hâlini bulunuz.



CONTEXT – FREE GRAMMAR (CFG)

terminals : $\{a, b, \lambda\}$ → ağacın birimini soğlar.

nonterminals : $\{S, A, B, X, Y, Z, \ldots\}$ → ağaca devamlılık soğlar.

$S \longrightarrow aS$ (start)

$S \longrightarrow \lambda$

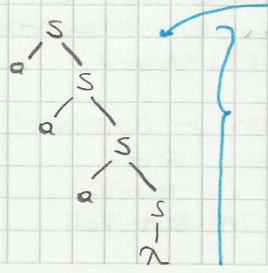
$S \longrightarrow aS \mid \lambda$ (veya)

ör $S \longrightarrow \lambda$

$S \longrightarrow aS \longrightarrow a\lambda \longrightarrow a$

$S \longrightarrow aS \longrightarrow aaS \longrightarrow aa\lambda \longrightarrow aa$

$S \longrightarrow aS \longrightarrow aaS \longrightarrow aaaS \longrightarrow aaa\lambda \longrightarrow aaa$



parse tree

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

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

- **Ambiguous Grammer:** Herhangi bir string için birden fazla parse ağacı çizebiliyorsak o gramere ambiguous (belirsiz) grammer denir.

$$S \rightarrow aS|Sa|a$$

* "aa" için 2 tane parse ağacı vardır:

CFG ⟶ λ Yok Etme ⟶ CNF ⟶ CYK

- **λ Yok Etme Algoritması**

* λ'lar silinir. ($S \rightarrow aS|a|b|\cancel{X}$)

* Production'ların sağ tarafında λ olabilecek nonterminaller varsa onu silinir.

$X \rightarrow Y$
$Y \rightarrow a|\lambda$

} X ve Y nulloble nonterminal:
$Y \rightarrow \lambda$
$X \rightarrow Y \rightarrow \lambda$

* $S \rightarrow a|Xb|aYa$
$X \rightarrow Y|X|\cancel{X}$
$Y \rightarrow X|a$

} X ve Y nulloble nonterminal:
$X \rightarrow \lambda$
$Y \rightarrow X \rightarrow \lambda$
S nulloble olmaz çünkü X ve Y'nin yanında a,b var.

Orginal Production	New Production
$S \rightarrow Xb$	$S \rightarrow b$
$S \rightarrow aYa$	$S \rightarrow aa$
$\cancel{X} \rightarrow Y$	nothing
$X \rightarrow X$	nothing
$Y \rightarrow X$	nothing

$S \rightarrow a|Xb|aYa|b|aa$ → Grammer'in son hali

* $S \rightarrow X \mid XY \mid Z$

$X \rightarrow Z \mid \lambda$

$Y \rightarrow Wa \mid a$

$Z \rightarrow WX \mid aZ \mid Zb$

$W \rightarrow XYZ \mid bXa \mid \lambda$

W ve X doğrudan nullable.

S, X'den dolayı nullable.

Z, WX 'den dolayı nullable.

W, X, S, Z nullable.

Original Production

$S \rightarrow X$
$S \rightarrow XY$
$X \rightarrow Z$
$Y \rightarrow Wa$
$Z \rightarrow WX$
$Z \rightarrow aZ$
$Z \rightarrow Zb$
$W \rightarrow XYZ$
$W \rightarrow bXa$

Subset

New Production

nothing
$S \rightarrow Y$
nothing
$Y \rightarrow a$
$Z \rightarrow W$ $Z \rightarrow X$ nothing
$Z \rightarrow a$
$Z \rightarrow b$
$W \rightarrow YZ$ $W \rightarrow XZ$ $W \rightarrow Y$
$W \rightarrow ba$

Gramer'in son hali;

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

* Orginal production'a içinde λ olabilecek (nullable) nonterminalleri yazıp düzeltiyoruz. Düzelterken de nullable nonterminalleri siliyoruz.

* Subset; sağ tarafındaki tüm değerler nonterminal ise nullable nonterminallerden önce birinci sonra ikinci en sonunda hepsini siliyoruz ve kalanı yazıyoruz.

- CNF 'ye Dönüştürme

Aşağıdaki iki formdan biri olmak zorundadır:

* Nonterminal → string of exactly two nonterminals
$S \rightarrow AX \mid XY \mid BY \mid AB$

* Nonterminal → one terminal
$S \rightarrow a$
$S \rightarrow b$
$S \rightarrow a \mid b$

* $S \to aXX$
  $X \to aS \mid bS \mid a$
  $A \to a$
  $B \to b$

} S'deki küçük a'yı büyük A'ya, X'deki küçük a ve b'yi büyük A ve B'ye dönüştürüyoruz.

$S \to AXX$ } S uymuyor. Çünkü 3 tane nonterminal yan yana gelmez.
$X \to AS \mid BS \mid a$
$A \to a$
$B \to b$

Son hali = $S \to AR$
$R \to XX$
$X \to AS \mid BS \mid a$
$A \to a$
$B \to b$

} CNF'ye uyar.

* $S \to AXXBA$ (R_2: XXBA, R_1: AX, R_3: BA)

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

} S' bu hale gelince CNF'ye uyar.

- CYK Parsing Algoritması

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

"abbab"

	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				

I. sütun: küçük a ve b'lerin bulunduğu nonterminaller ilgili hücreye yazılıyor.

II. sütun: solundaki ve alt çaprazındaki değerler çaprazlanır.

X, A } XS, XB, (AS), AB
B, S
→ X (tablodaki değerine bakılıyor neye denk geliyorsa yazılır.)

S, B } SS, SB, (BS), BB
S, B
→ S

S, B } SX, SA, (BX), BA
X, A
→ X

X, A } XS, XB, (AS), AB
S, B
→ X

III. sütun: 2 tane çaprazlama yapılır. Solundaki I. sütun değerini ve alt çaprazını, ikincisi ise solundaki değer ile birinci sütundaki iki alt çaprazını çaprazlar.

X, A } XS, (AS) → X
S

ve

X } XS, XB
S, B

S, B } SX, (BX) → X
X

ve

S } SX, AS
X, A

S, B } SX, (BX) → X
X

ve

X } XS, XB
S, B

IV. sütun: 3 tane çaprazlama yapılır. I. sütunun ilk elemanı ile ilk alt çaprazı, II. sütunun ilk elemanı ile II. sütunun iki alt çaprazı ve III. sütunun üçüncü elemanı ile I. sütun sağ alt çaprazı.

X, A } XX, (AX) → S
X

ve

X } XX
X

ve

X } XX, XA
X, A

S, B } SX (BX) → X
X

ve

S } SX
X

ve

X } XS, XB
S, B

V. sütun: 4 tane çarpıtlara verdim. I. sütunun ilk elemanı ile IV. sütunun ilk alt çarpımı, II. sütunun ilk elemanı ile III. sütunun iki alt çarpımı, III. sütunun ilk elemanı ile II. sütunun üç alt çarpımı, IV sütunun ilk elemanı ile I. sütunun dört alt çarpımı.

X, A } XX, (AX) → S
X

ve

X } XX
X

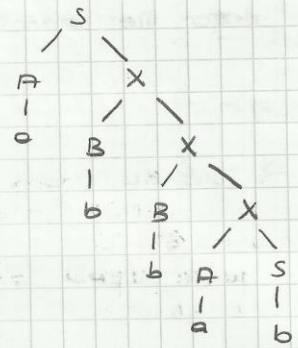
ve

X } XX
X

ve

S } SS, SB
S, B

Parse Tree



* S → SS | AC | BD | AB | BA | b

C → SB

D → SA

A → a

B → b

"bbba" bulunuz.

	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

S, B } (SS), (SB), SB, BB
S, B

S, B } SS, SB, SB, BB
S, A

S, B } (SA), (BA)
A

III. Sütun

S, B } (SS), SC, BS, BC
S, C
S, C
S, B

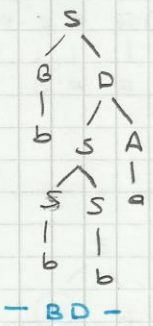
S, B } (SS), (SB), CS, CB
S, D
S, C
A

S, B } (SS), SD, BS, (BD)
S, D
S, C
A

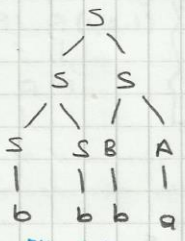
(SA), CA

IV. satır

S, B → (SS)^S, SD, BS, (BD)^S
S, D
S, C → (SS)^S, SD, CS, CD
S, D
S, C → (SA)^D, CA
A

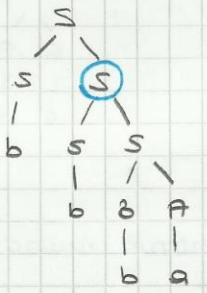


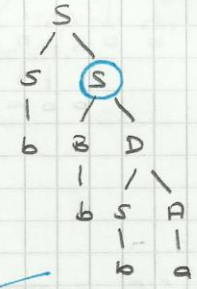
– BD –



– İlk SS –

* 3. sütunun 1. çarpımlarından sonucu elde edilen ağaçlardır, son SS' diyelim nereden üretilmiş onun a-b karşılığı bulunana kadar tüm çarpımlardan kontrol edilir.





– İkinci SS –

* Yuvarlak içindeki S iki şekilde elde edildiği için 2 ayrı parse ağacı ortaya çıkmış olur.

* 2014 Sınav Sorusu

S → AA

A → ASI AB | BA | a

B → b

"babaaa" kelimesinin parse ağacını bulunuz.

	I	II	III	IV	V	VI
b	B	A	A	S	A	S
a	A	A	S	A	S	
b	B	A	S	A		
a	A	S	A			
a	A	S				
a	A					

II. sütun

BA → A

AB → A

BA → A

AA → S

AA → S

III. sütun

BA → A (3.1.1), AB → A (3.1.2)

AA → S, AA → S

BS, AA → S

AS → A, SA

IV. sütun

BS, AA → S, AA → S

AS → A, AS → A, SA

BA → A (4.3.1), AS → A, SA (4.3.2)

V. sütun

BA → A, AS → A,

AS → A, SA

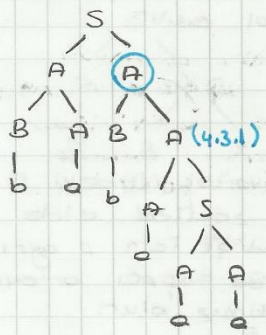
AA → S, AA → S

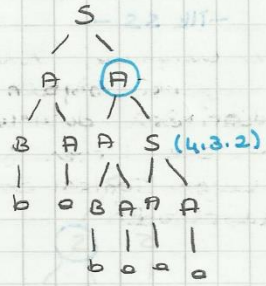
SS, AA → S

VI sütun

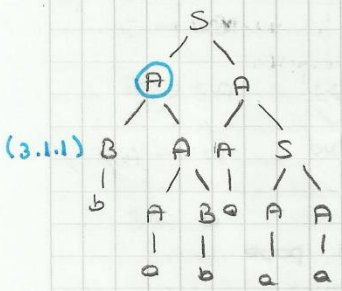
BS, AA → S (6.1.2), AA → S (6.1.3), SS, AA → S (6.1.5)

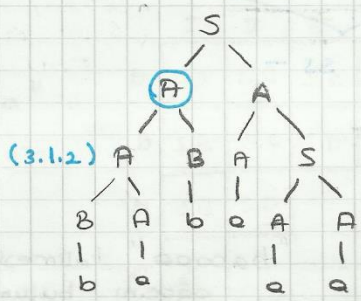
6.1.2 için;





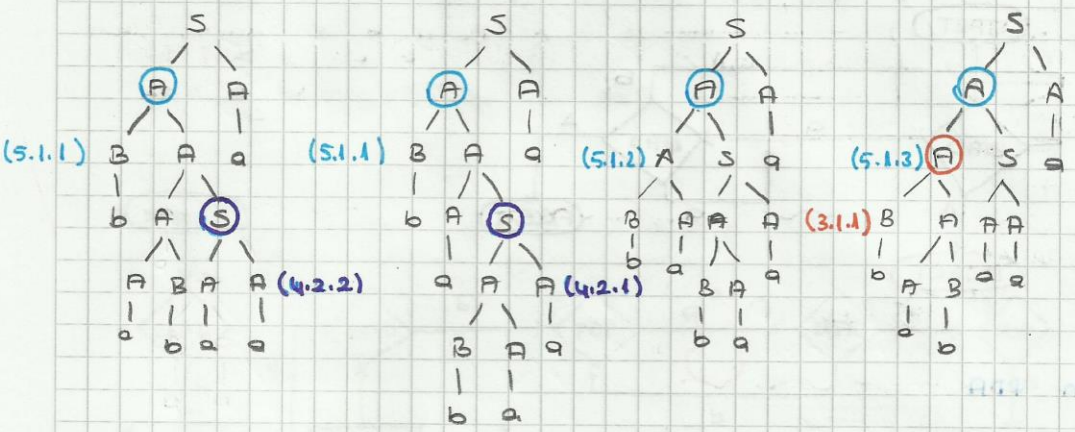
6.1.3 için;

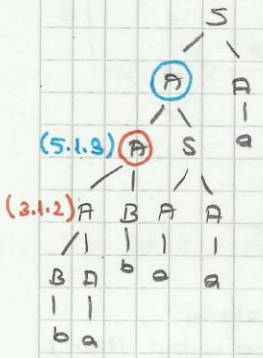




Alt kelimeler denilen oluşturulan tablodaki diğer S durumlarıdır. Onlarda tek tek kontrol edilir ve bu incelemeyle ambiguous olup olmadığına böyle karar verilir.

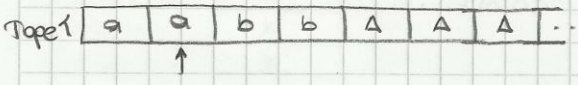
6.1.5 İçin:





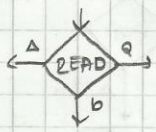
∇ Son durumda oluşan bütün durumları incelyoruz. Böylece parse ağaçlarımız ortaya çıkar. Bazı durumlar 2'ye ayrılabiliyor böylece birden fazla ağaç ortaya çıkıyor. Toplam da 9 tane parse ağacı çıkmaktadır.

PUSH DOWN AUTOMATA (PDA)

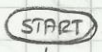


} Δ'lar tape'in devamının boş olduğunu gösterir.

PDA'nın alt bileşenleri:

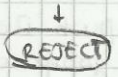


} Tape'in içeriğini okur.

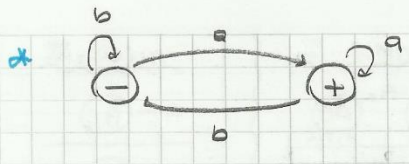


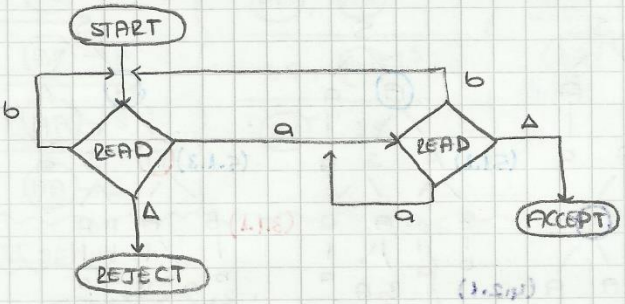
} PDA'nın başlangıcı

} String'i kabul ettiği durumdur.



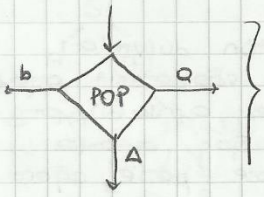
} String'i kabul etmediği durumdur.



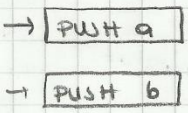


Yığın PDA

* PDA'lara stack eklediğimizde kullanacağımız bileşenler;



Yığından elemanları çekmek için kullanılır.

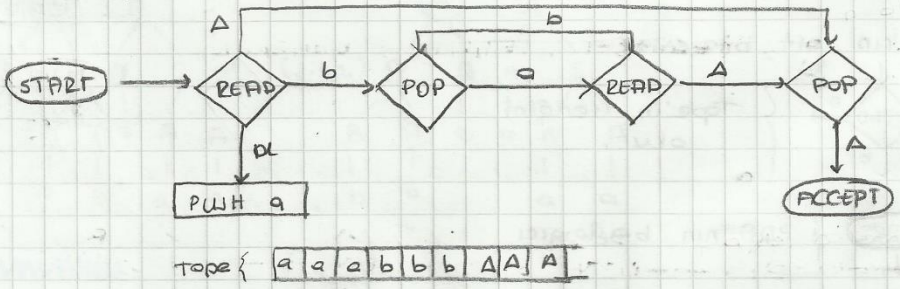


Yığına eleman itmek için kullanılır. a ve b nereye gideceklerini göstermek için yazılmıştır. (okuma kafası itilir)

* $L = \{a^n b^n , n \geq 0\}$ PDA'yı çizin.

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

a sayısı b sayısına eşit



PDA'da ACCEPT durumu 1 tanedir. REJECT durumlarını yazmıyoruz.

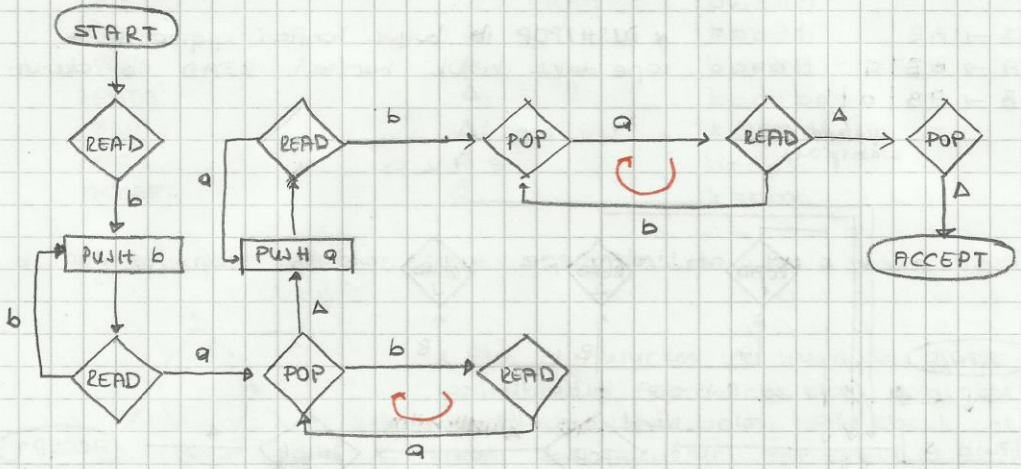
* Öncelikle tape'deki a'lar okunur. a okundukça push ile yığına atılır. a'lar bitip b okuduğunda (POP a) yapıp yığındaki a değerlerinden biri silinir. b okundukça a silinir. En son Δ bu stringe geldiğinde stack tamamen Δ haline gelmiş olur.

* POP yapmadan accept'e gidebiliyorsak öncesinde push yapmayabiliriz. Fakat POP yapmadan accept'e gidemiyorsak öncesinde mutlaka push yapmalıyız.

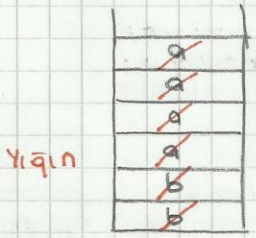
21.03.2016

* $L = \{ b\ a\ b,\ j = i + k,\ i = 1,2,3,\ldots,\ k = 1,2,3,\ldots \}$
  (i) (j) (k)

i	i	k	k
b	a	a	b



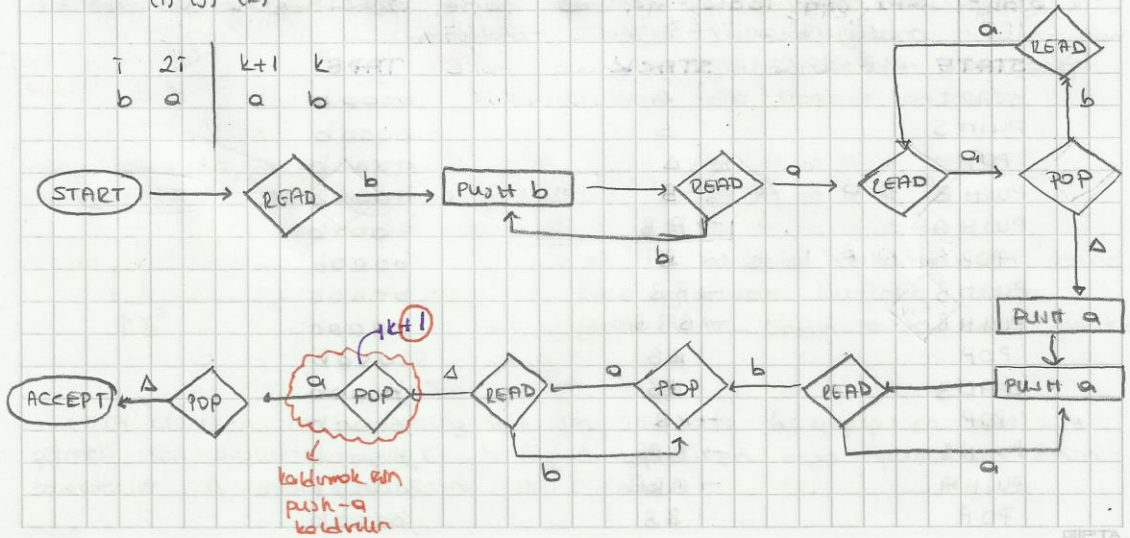
Tape: b b a a a a a b b b Δ ...



* Sonunda yığında, tapede boş kalır.

* $L = \{ b\ a\ b,\ j = 2i + k + 1,\ i = 1,2,3,\ldots,\ k = 1,2,3,\ldots \}$
  (i) (j) (k)

i	2i	k+1	k
b	a	a	b



Tape | b | a | a | a | a | a | a | b | b | b | Δ | ...

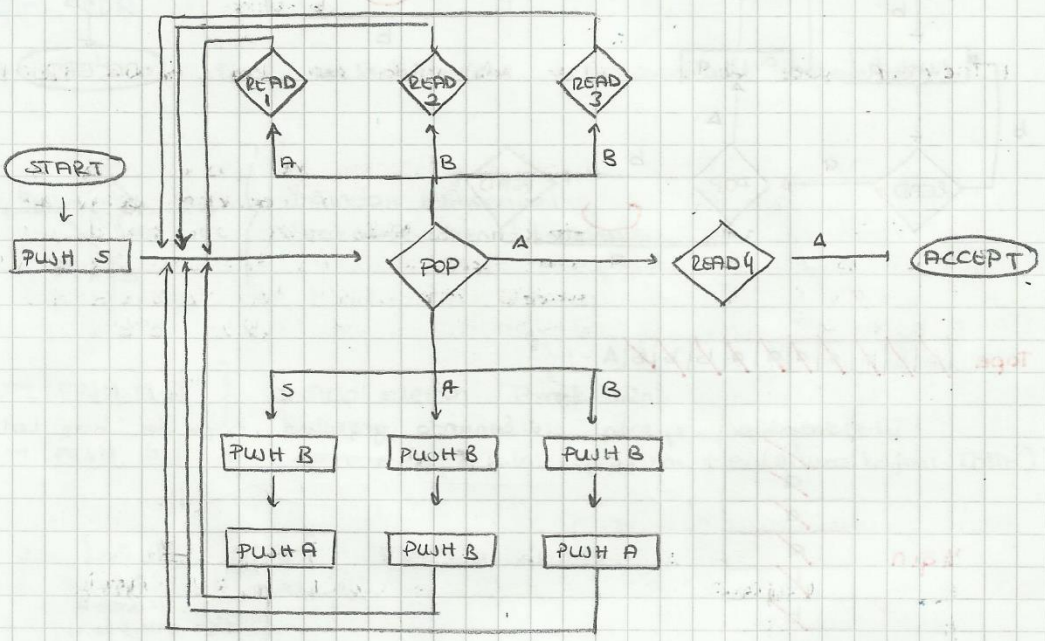
* Fazladan 2 tane push a vardır. Silinen ilk 2 a, b'yi pop yapmak için kullanılır.

* Sondaki POP-a'yı kaldırmak için PUSH-a'dan birini siliyt.

* Okudukça tape'den: Siliyt. Push ile yığına atıyt. Pop ile yığından siliyt.

* S → AB
  A → BB | a
  B → AB | a | b
  (grammer: ambiguous)

* PUSH/POP ile büyük harfleri yığına atıyt, tape'deki küçük harfleri READ ile okuruz.

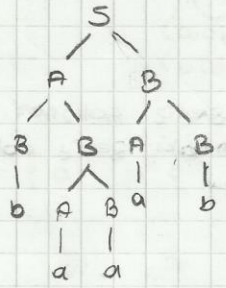


*POP'da 2 yol olduğu için non-deterministiktir. PDA non-deterministik olduğu için yo tablo yo da parse ağacı bize verilmelidir.

STATE	STACK	TAPE
START	Δ	baaaab
PUSH S	S	baaaab
POP	Δ	baaaab
PUSH B	B	baaaab
PUSH A	→ A B	baaaab
POP	B	baaaab
PUSH B	→ B B	baaaab
PUSH B	→ B B B	baaaab
POP	B B	baaaab
READ 3	B B	baaaab
POP	B	baaaab
PUSH B	→ B B	baaaab
PUSH A	→ A B B	baaaab
POP	B B	baaaab

STATE	STACK	TAPE
READ1	BB	baaab
POP	B	baaab
READ2	B	baaab
POP	Δ	baaab
PUSH B	→B	baaab
PUSH A	→AB	baaab
POP	B	baaab
READ1	B	baaab
POP	Δ	baaab
READ3	Δ	baaab
POP	Δ	baaab
READ4	Δ	baaab
ACCEPT	Δ	baaab

* Tablodan parse ağacı, parse ağacından da bu tablo çıkarılır.

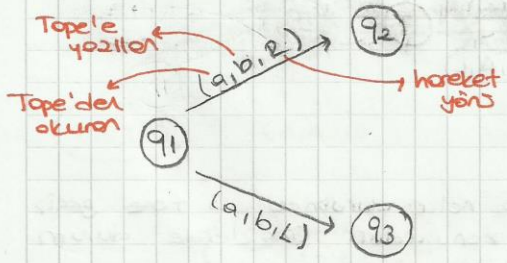


* Ağaç oluştururken her push'da ağaca ekleme yapılmaktadır. Ağaca çocuklar yazılırken stack'in üstünde olan sol çocuk altında olan sağ çocuklar. POP yapıldığında ilk sıradaki POP yapılır. Yani ilk sıradakine çocuk ekleyir. (POP yapılana çocuk eklenir.)

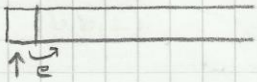
11.04.2016

TURING MACHINE (TM)

Görsel olarak FA'e çok benzerdir. TAPE kullanması da PDA ile benzer özelliğidir. Durumlar ve durumlar arası geçişi olan bir yapıya sahiptir.



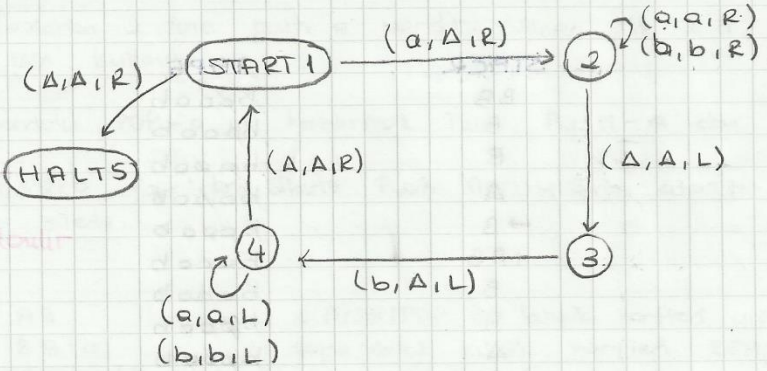
* TAPE kullanma açısından;
  - TAPE'e yazılır.
  - TAPE üzerinde hem sağa hem sola hareket edilebilir.

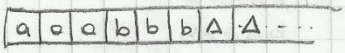


* İlk karakter okunduktan sonra sola gidilmez. Mutlaka sağa gidilmelidir. Ondan sonra sola gidilebilir.

* TM çizerken her bir string için TM ilk harfi Δ yapıp en sona gitmeli son bozukluktan önceki harfi Δ yapıp en başa gelmelidir. Bütün karakterler Δ olduktan sonra HALT olur.

* $L = \{ a^n b^n , n = 0, 1, 2, \ldots \}$ için TM geliştiriniz.



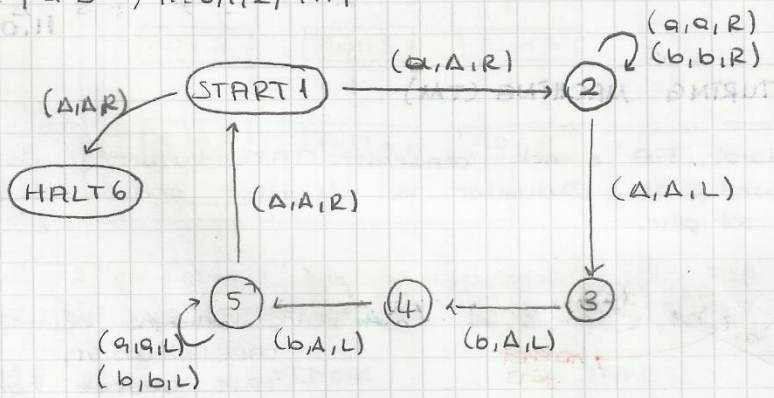


* TM'de yığın yoktur.

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

1 ile 2 arasındaki geçiş çalışmaz.
1 ile HALT arasındaki geçiş çalışır.
String kabul eder.

* $L = \{ a^n b^{2n} , n = 0, 1, 2, \ldots \}$

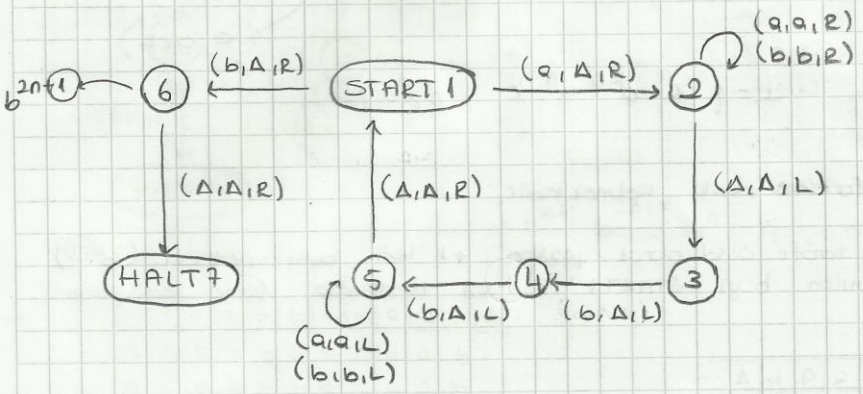


a a b b b b Δ
Δ a b b Δ Δ Δ
Δ Δ b b Δ Δ Δ
Δ Δ Δ Δ Δ Δ Δ

* 5 nolu durumda 3 tane geçiş söz konusudur. Geçişlerde durum belirtmek zorundayız.

* $L = \{ a^n, b^{2n+1} \mid n = 0, 1, 2, \ldots \}$

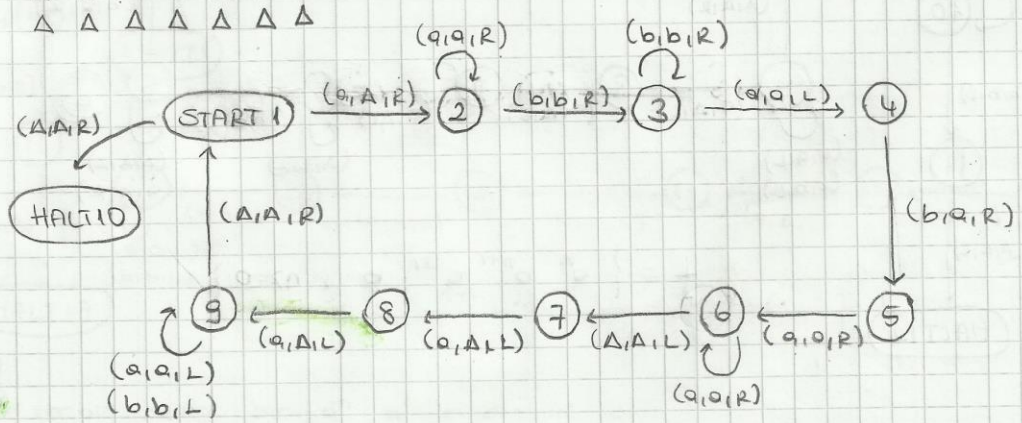
q	a	b	b	b	b	b	Δ ...
Δ	a	b	b	b	Δ	Δ	Δ
Δ	Δ	b	b	b	Δ	Δ	Δ
Δ	Δ	b	Δ	Δ	Δ	Δ	Δ
		↑					



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

a^n a^{2n} → 1 tane b'yi a yapıyor ve a^{2n} diye basitleştiriyor.

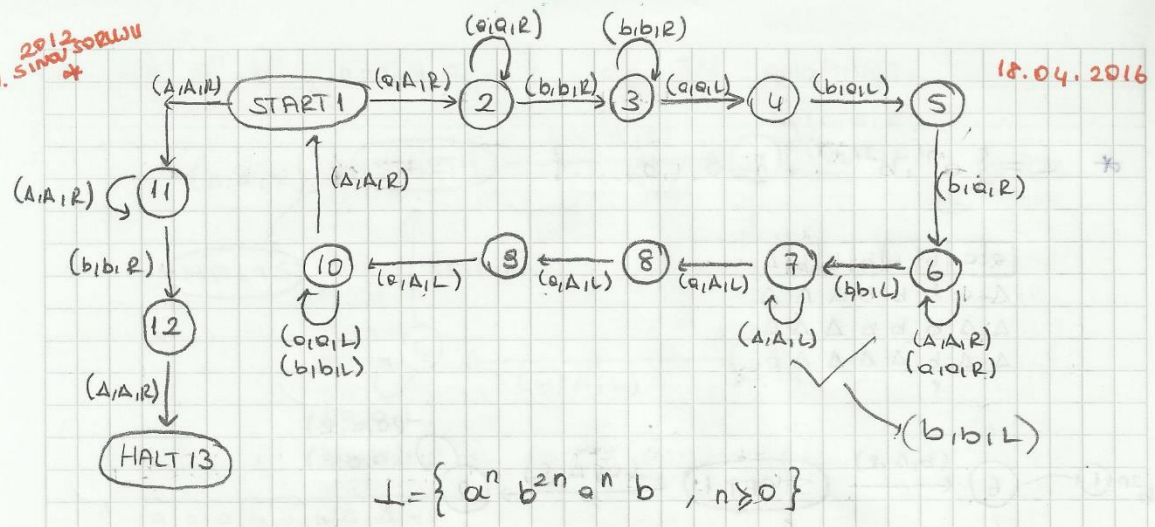
a	a	b	b	a	a	Δ
Δ	a	b	b	a	a	Δ
Δ	a	b	a	a	a	Δ
Δ	a	b	a	Δ	Δ	Δ
Δ	Δ	a	a	Δ	Δ	Δ
Δ	Δ	Δ	Δ	Δ	Δ	Δ



* 5-6 arasındaki durum aslında a'ya çevrilmiş b'yi geçme izlemidir. Onu geçtikten sonra diğer a^n dizgisi yerleştirilir.

2012 V. SINAV SORULU ✱

18.04.2016



$$\perp = \{ a^n b^{2n} a^n b \;,\; n \geq 0 \}$$

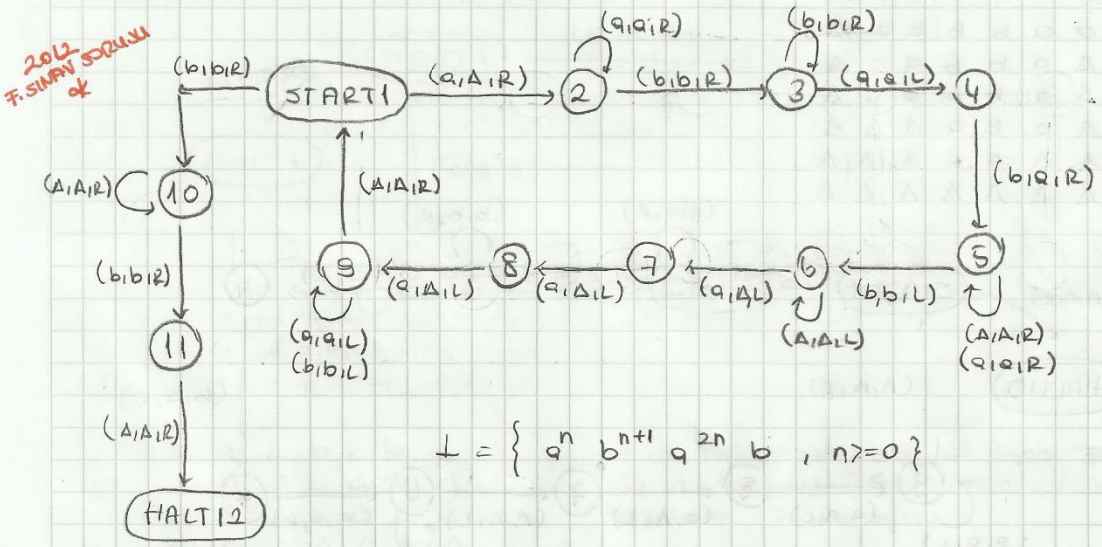
✱ Döngüler a^n gibi ifadelere denk gelmektedir.

✱ START1'in sol tarafı üstel olarak yazılan +1'leri belirtmektedir. (b^{n+1}) Burada tek kullanılan b'yi START1'in sol tarafında ifade ediyoruz.

a	a	b	b	b	b	a	a	b	Δ

Δ a b b b b a (a'yı görünce solo gitti 2 b'yi değiştirdi.)
Δ a b b a a a a b
Δ a b b a Δ Δ Δ b (Pezpeze 3 a'yı boşluk yapıyoruz.)
Δ Δ a a a Δ Δ Δ b
Δ Δ Δ Δ Δ Δ Δ Δ b
↳ 11, 12 arası

2012 F. SINAV SORULU ✱

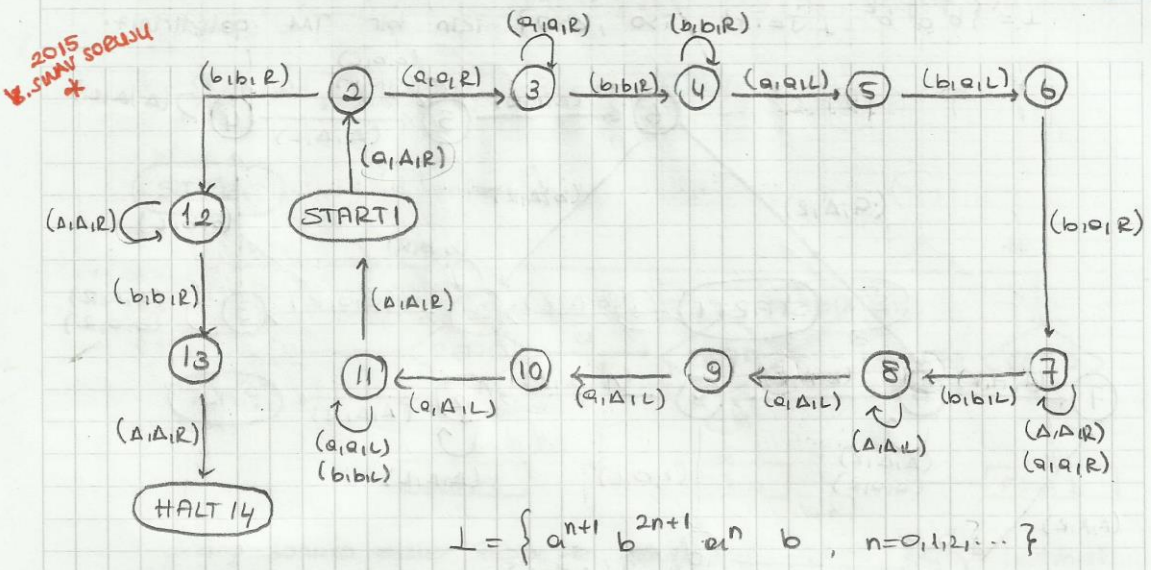


$$\perp = \{ a^n b^{n+1} a^{2n} b \;,\; n >= 0 \}$$

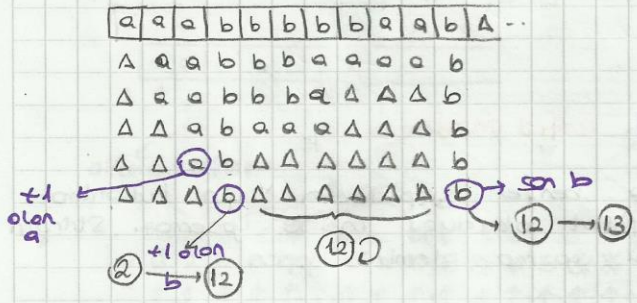
✱ Kaç tane b'yi a'ya çeviriyorsak b'nin üstü ona göre değişiyor. 2 tane b a'ya çeviriliyorsa dili b^{2n} oluyor.

✱ Diyelim ki 1 tane b'yi a'ya çeviriyoruz ve döngü sonunda 3 tane a'yı A'a çeviriyorsak $b^n a^{2n}$ olur. Eğer 2 tane b'yi a'ya çevirmiş olsaydık $b^{2n} a^n$ olurdu ama üst toplamının 3 olmasıdır.

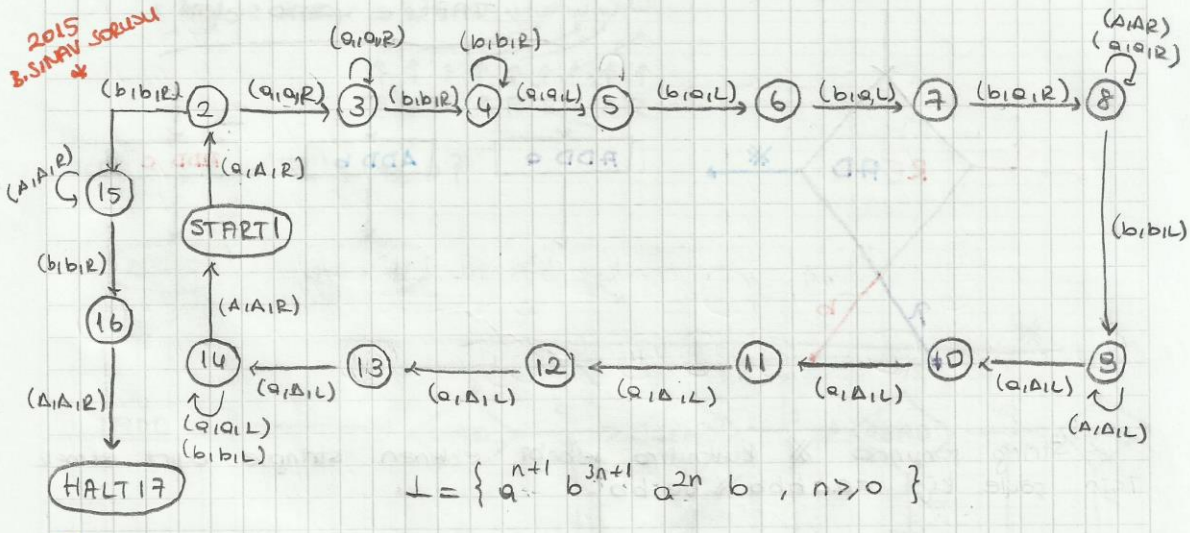
2015 B. SINAV SORULU *



$$\perp = \{ a^{n+1}\, b^{2n+1}\, a^{n}\, b \;,\; n = 0,1,2,\ldots \}$$



2015 B. SINAV SORULU *



$$\perp = \{ a^{n+1}\, b^{3n+1}\, a^{2n}\, b \;,\; n \geq 0 \}$$

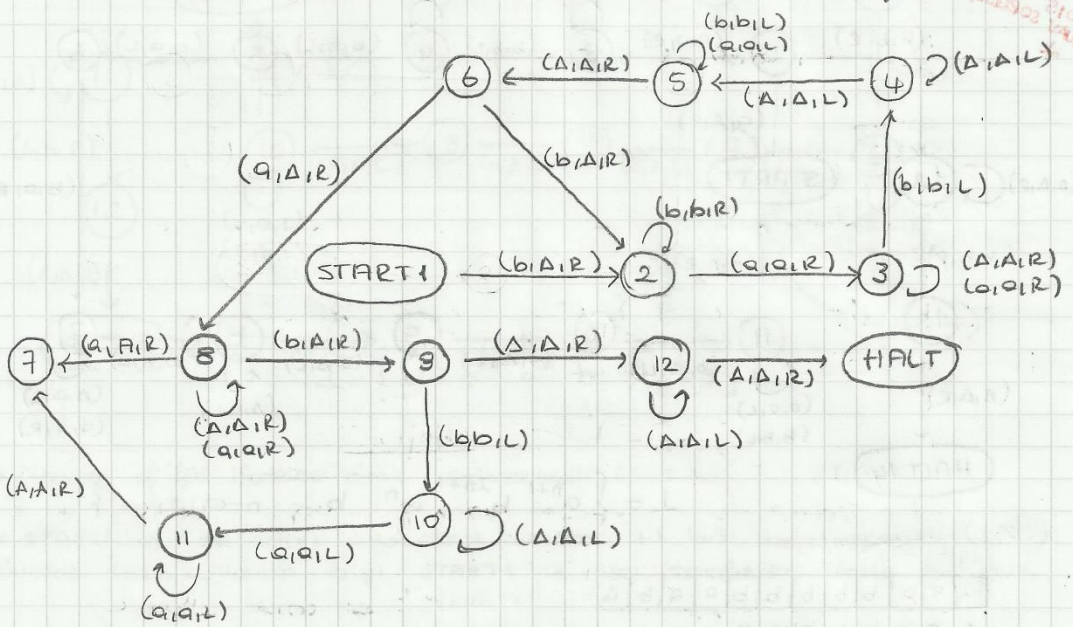
* Her bir döngü bir a^n, b^n 'e denktir.

* Sondaki tek b harfi sonuna kadar sağa gitmemizi engeller. Döngünün devamlılığını sağlamış olur.

* Sayı üzerindeki en son döngüden sonraki b sondaki b için yazılmıştır. Mesela 2015 B.SS'de 15 - 16 arasındaki (b,b,R) diğindeki son b için yazılmıştır.

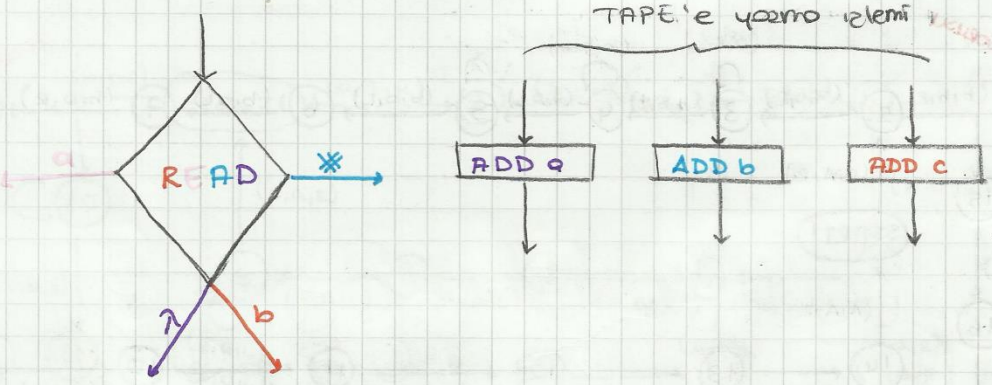
2010 F. SINAV SORUSU

$L = \{ b^i a^j b^k \;,\; j = i+k \;,\; i>0 \;,\; k>0 \}$ için bir TM geliştiriniz.



POST MACHINES (PM)

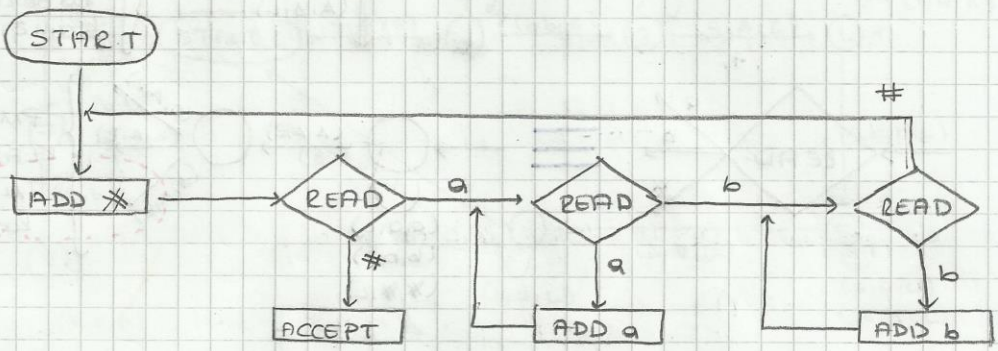
Gösterim olarak PDA'ya benzer yapıdadır. Yığın kullanılmaz, TAPE'e yazma yetkisi var fakat okuduğu tndiye yazamaz. Stringin sonuna yazılabilir. ADD durumu yazma işlemini yapar.



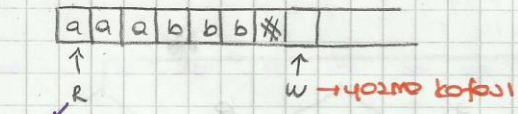
* String sonunda ※ kullanılma sebebi eklenen stringle ayırt etmek için şöyle ki; aaabbb※aabb.

* ADD fonksiyonu okunan yere değil string'in sonuna yazıyor. Hem okuma hem yazma kafası vardır.

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

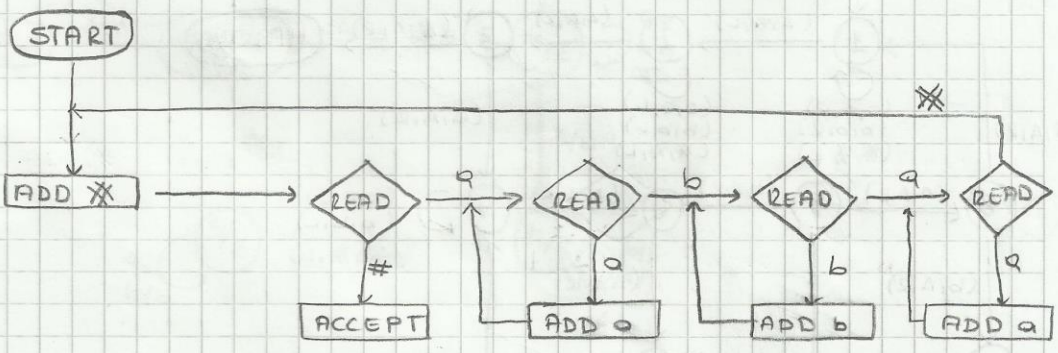


String sonuna ekler. Peşpeşe gelen iki stringi ayırt etmek için kullanılır.



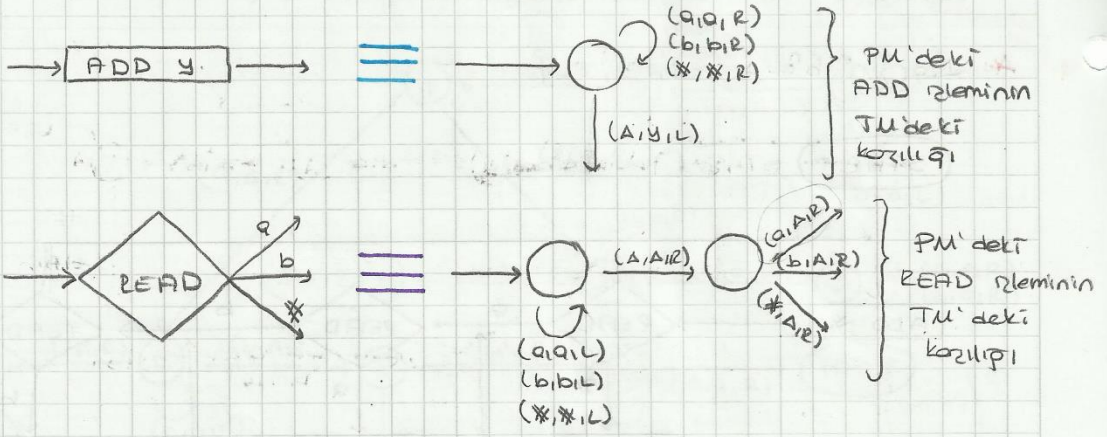
* 1'er tane a'yı, b'yi sildi. TM olaydı Δ yazacaktı.

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

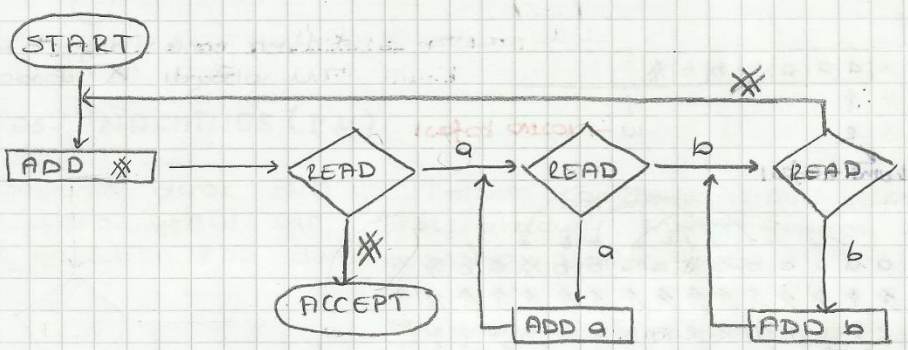


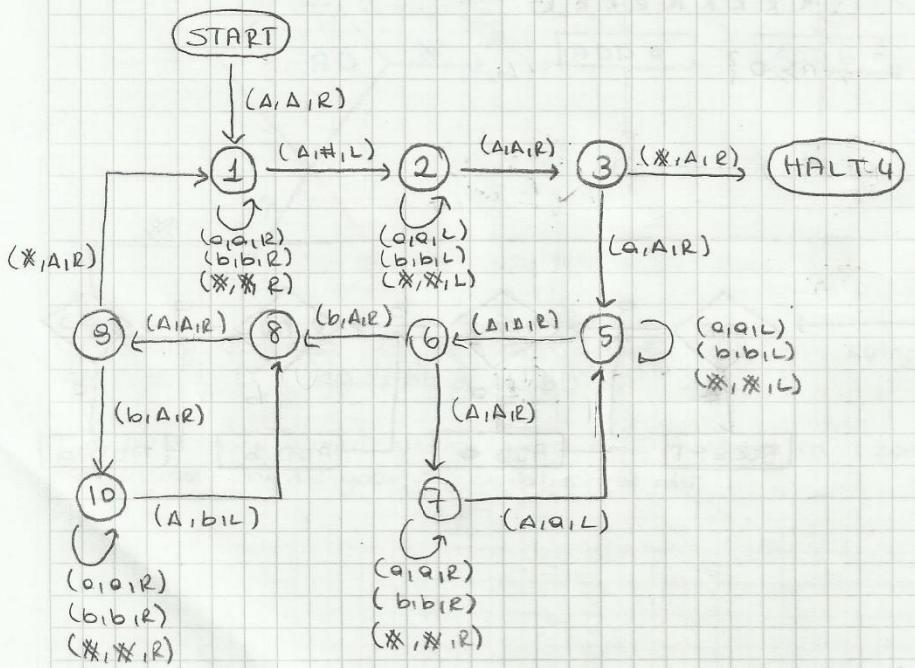
??

SIMULATING A PM ON A TM

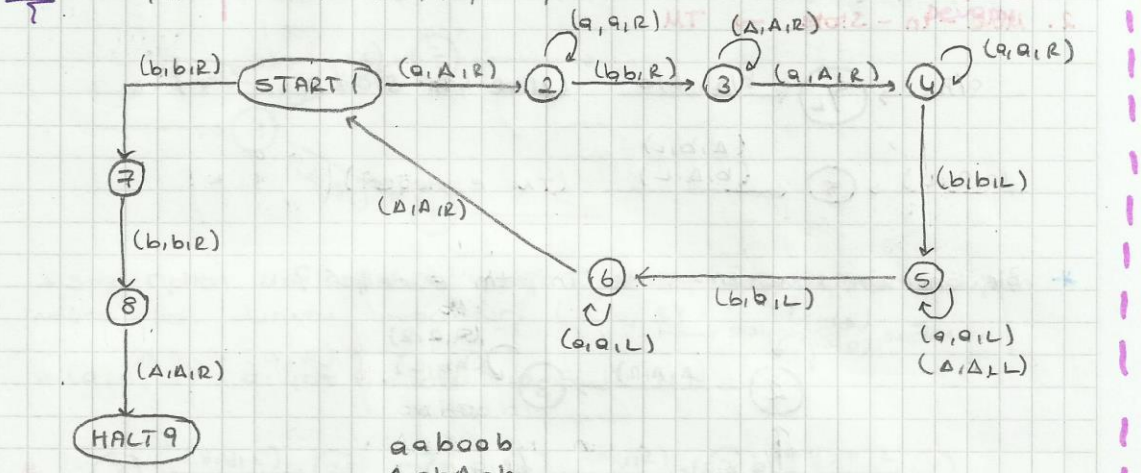


* $L = \{a^n b^n , n \geq 0\}$ PM'nin eşdeğer TM'sini çiziniz.





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

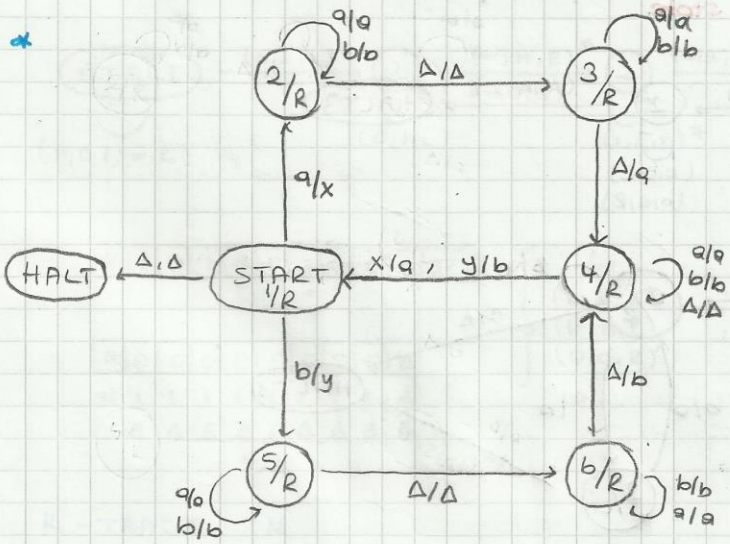


aaboob
ΔobAob
ΔΔbΔΔb
↳ START a geldiğinde b'nin üzerindeyiz
b'leri okuyup HALT edicez

25.04.2016

Variations on the TM

1. Move - In - State Machine

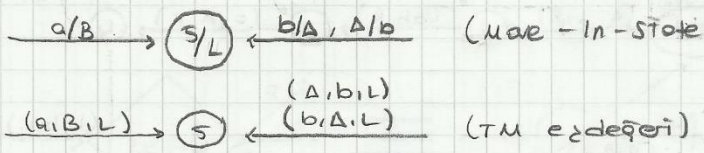


b	a	a	Δ	Δ	Δ	...
y	a	a	Δ	b		
b	a	a	Δ	b	Δ	
b	x	a	Δ	b	a	
b	a	a	Δ	b	a	Δ
b	a	x	Δ	b	a	a
b	a	a	Δ	b	a	a

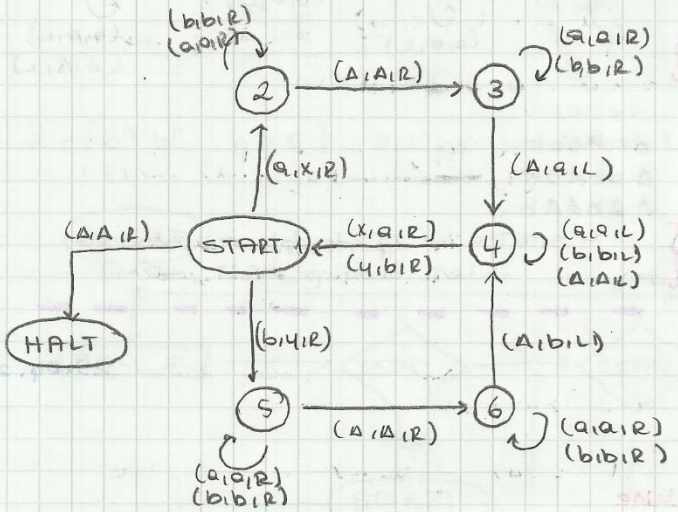
↳ 1. string sonu

* Durum içerisindeki R, L'ler sayesinde tape'de sağa ya da sola gidiyoruz. Bir durumdan diğer duruma geçerkeki şeyler ise tape'de olan olmak ise tape'e yazılacak olan diye kabul edilir. (a/b)

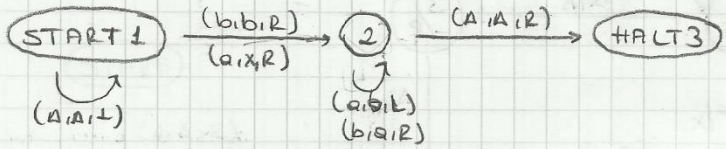
2. Move-In-State → TM

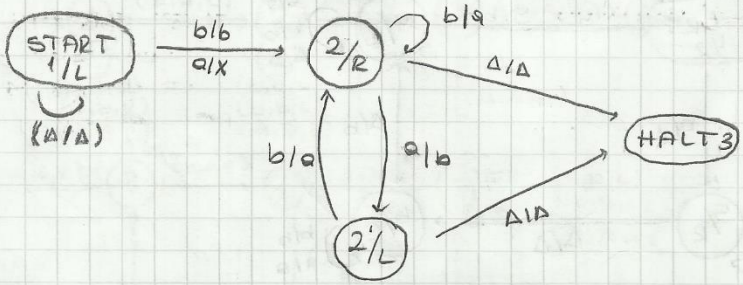


* Bir önceki move-in-state'in tm eşdeğeri?



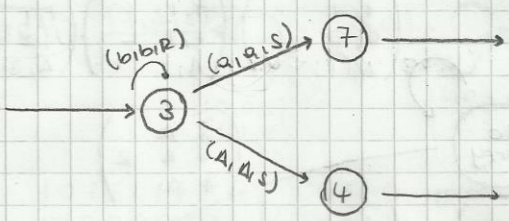
3. TM → Move-In-state





* Bir durum üzerinde hem L/R varsa 2 ayrı durum açılıyor.

STAY-OPTION MACHINE



* Stay-Option Machine'de sağa ya da sola gitme olduğun indişte kal durumu mevcuttur. (a,a,S) sayesinde sağlanır.

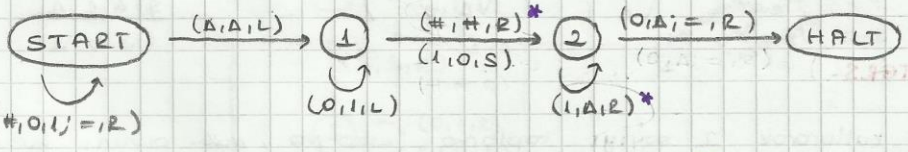
* (a,a,S) : a yaz, a oku ve orda kal.

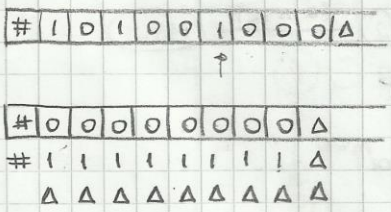
* 

* TM'den Stay-Option'ı çıkarsak yok. TM eşdeğerdir. TM'de mutlaklık sağa veya sola gidilir.

* 1 0 1 0 0 1 0 0 0 } 8 } Bir sayının bir eksiği
1 0 1 0 0 0 1 1 1 } 7 } Binary decrementer (--)

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



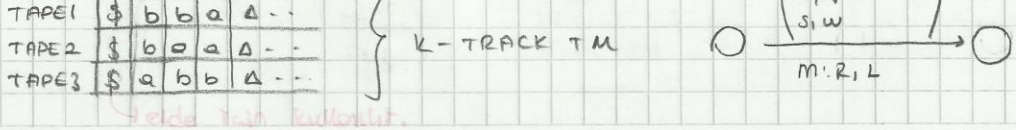


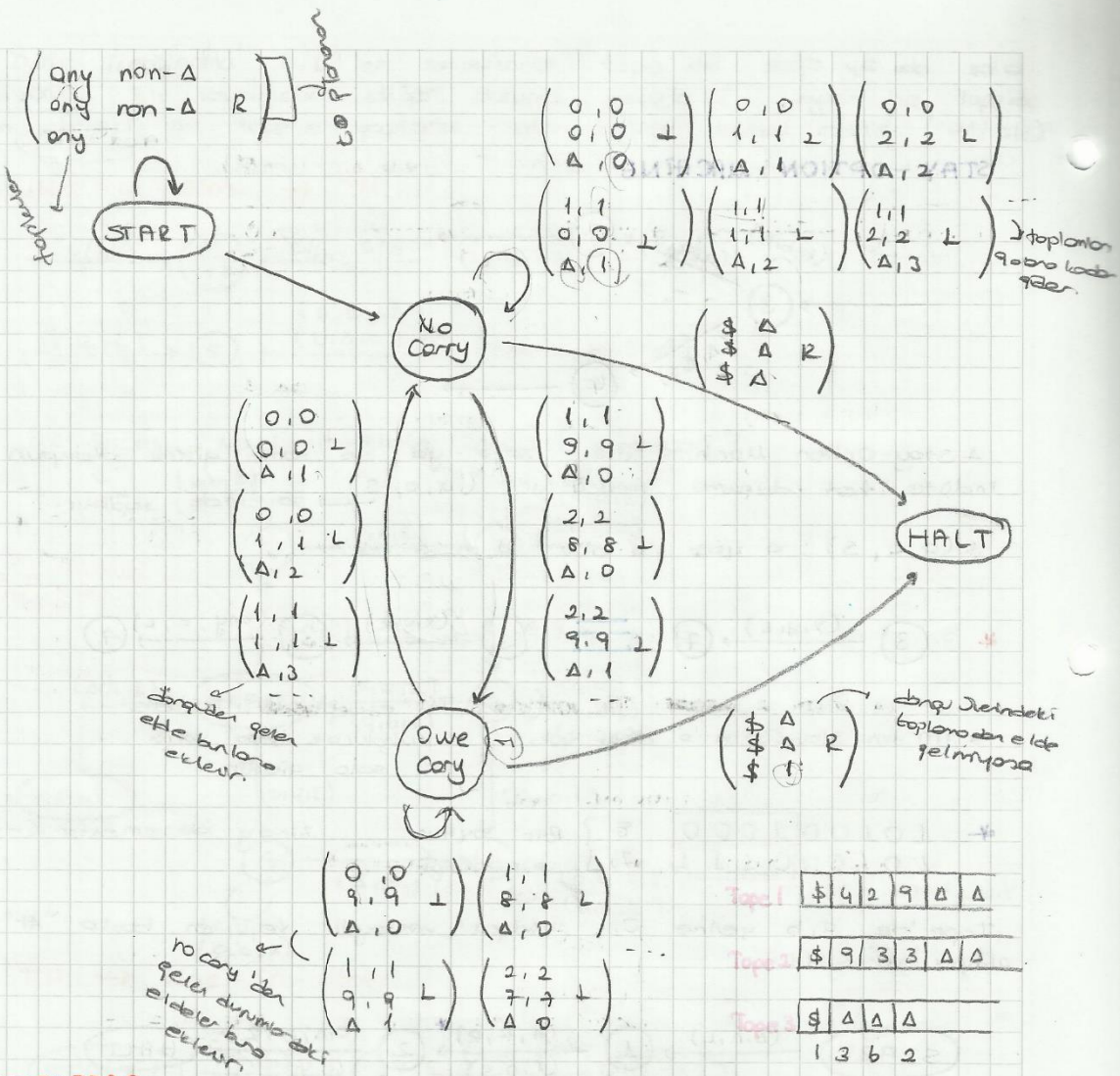
(1,1,R)
(#,#,R)
(0,0,R)

▽ Gezicilerin amacı, tape'de '0000' gibi bir sayı olması durumunda negatif bir sayı elde edilemeyeceğinden dolayı sıfırların boşluğa gelinmesini sağlar.

K-TRACK TM

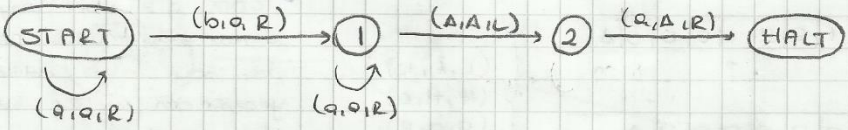
* Birden fazla tape vardır. Aynı anda her ikisinde de okuma yazma yapabiliriz.





COMPUTERS

* TM kullanarak 2 sayıyı toplama, çıkarma, max. olanı bulunması.



$a^* b a^* \Rightarrow \underbrace{a^n}_{\text{toplanan1}} b \underbrace{a^m}_{\text{toplanan2}} \Longrightarrow a^n \underbrace{a}_{\downarrow} a^{m-1}$

aaa b aaaaa

3 + 5

b, a'ya geçmişti

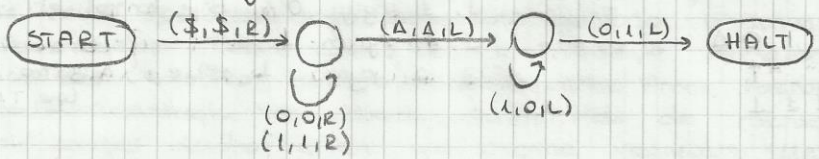
o yüzden m-1 oldu.

* $\$(0+1)^* \quad \$(0+1)^*$ bir sayı azaltırken diğeri artacak.

Decrement — Increment

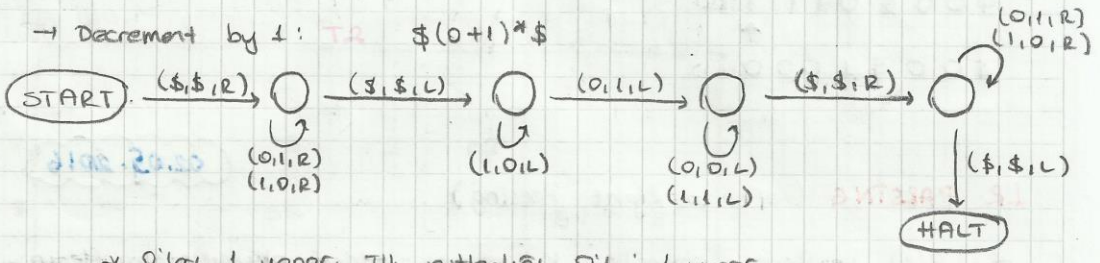
Toplama

→ Increment by 1 : $(0+1)* (Tape'nin içeriği)



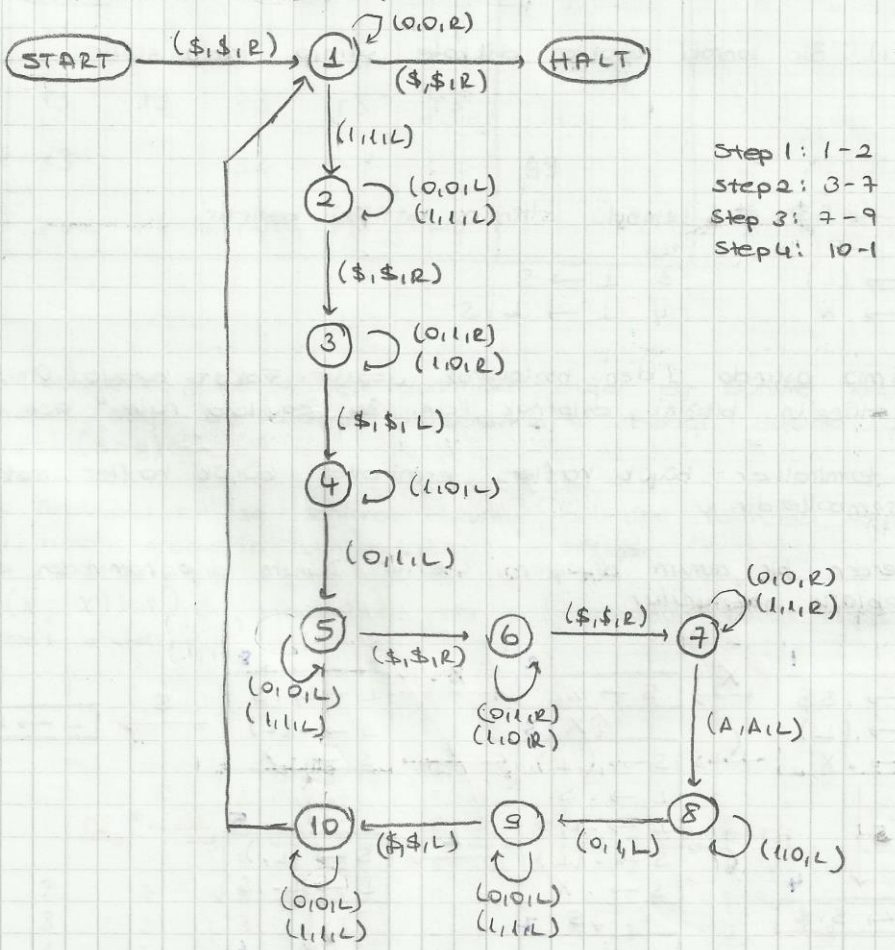
10111111 } 1'leri 0 yapar. Tlk rastladığı
11000000 } 0'ı, 1 yapar.

→ Decrement by 1 : $(0+1)*$



* 0'ları 1 yapar. Tlk rastladığı 0'ı ; 1 yapar.

* Tki algoritmanın birleştirilmiş hali:



Step 1: 1-2
Step 2: 3-7
Step 3: 7-9
Step 4: 10-1

$	1	0	$	0	1	1	0	Δ	Δ

$ 0 1 $
↑
$ 1 0 $
$ 0 1 $ 0 1 1 0 Δ
← ↑
$ 0 1 $ 0 1 1 1
↑
$ 1 1 $
$ 0 0 $ 0 1 1 1
↑
$ 0 0 $ 0 1 1 1 Δ
↑
$ 0 0 $ 1 0 0 0

Kurallar

1. 1. sayıyı 0 mı diye kontrol et.
2. 1. sayıdan 1 çıkar. } Ekleme ve
3. 2. sayıya 1 ekle. } çıkarma için 2 bant TM.

02.05.2016

LR PARSING (Left-to-Right Parsing)

* LR(k) şeklinde gösterilir. k sayısı stringi oluşturan harflerin sayısının her biri demektir.

* LR(0): Bir sonraki karakteri dikkate almıyor.

* LR(1): Bir sonraki karakter dikkate alınıp tablo çizilir.

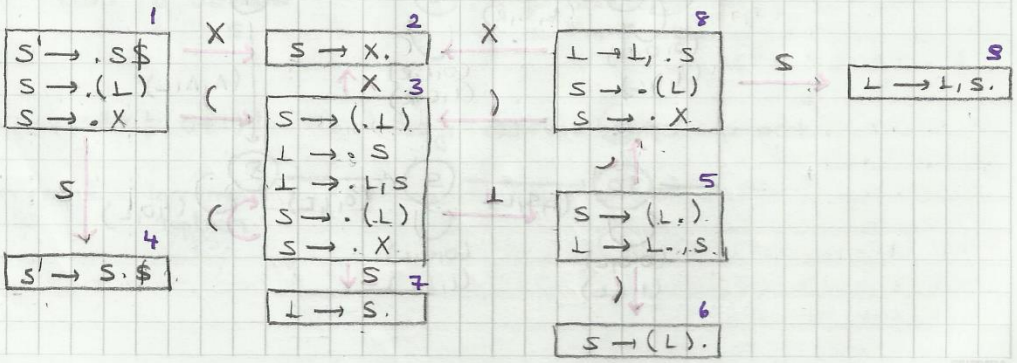
0 $S' \rightarrow S\$$ # $ sembolü stringin bittiğini belirtir.

1 $S \rightarrow (L)$
2 $S \rightarrow X$
3 $L \rightarrow S$
4 $L \rightarrow L, S$

* Gramerimiz aslında 1'den başlayarak yazılır. Fakat buraya 0'da eklediğ string'in bitişini anlatmak için. Bu sayılara "rule" adı verilir.

* Non-terminaller büyük harfler, terminaller küçük harfler, matematiksel ifadeler, sembollerdir.

* Gramerden bir durum diyagramı çıkarılır. Durum diyagramından da parsing tablosu oluşturulur.



* Noktanın konumuna göre durum oluşturuyoruz. Nokta sona geldiyse veya noktadan sonra $ sembolü geliyorsa o durumu daha fazla ilerletmiyoruz. Fakat nokta en başta veya noktadan sonra non-terminal varsa durumu ilerletiyoruz. Amaç, noktayı sona gelene kadar ilerletmek. Yeni durum oluştururken de noktadan sonraki ifadeyi bir sonraki duruma giden yol üzerine yazıyoruz. Mesela $1 \rightarrow 1 . S$ durumunda iken virgül (,) geldiğinde değip $1 \rightarrow 1 , . S$ durumunu açıyoruz. Durum diyagramı oluştuktan sonra da tablo oluşturuyoruz.

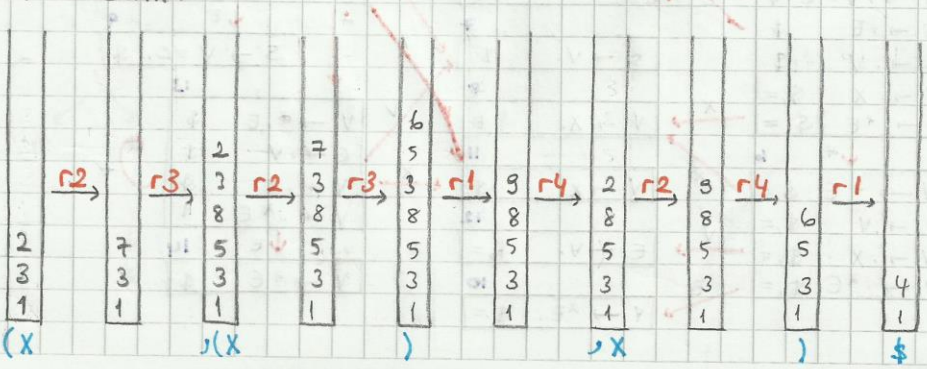
	(	)	X	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

($'in olduğu durum)

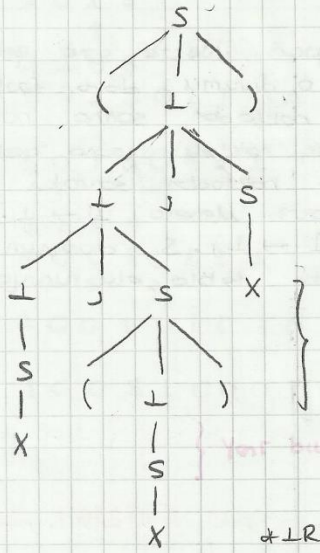
* Tablodaki s ve g bir durumdan diğer bir duruma geçişi gösterir. Yanındaki sayı ise durum diyagramında hangi duruma geçtiğini belirtir. (s=g)

* Tablodaki r ise sonlanan durumu belirtir. Yanındaki sayı gramerdeki hangi gramerin sonlandığını belirtiyor.

* (X, (X), X) $ ifadesini parsing tablosunun kabul edip etmediğini kontrol edelim.



* Arkadaki stack tablosuna göre parse ağacı çıkarabiliriz.



* Stackteki rule'lara göre işlem yapılır. En son rule r1 olduğu için gramerdeki rule1 direkt yazılır. Tüm non-terminaller terminal olduğunda parse ağacı biter.

Sonunda hangi ⊥ harfini S yapacağımızı şöyle belirleriz: Hangi dal üzerinde işlem o dal sonlanmadan diğer dala geçmiyoruz.

* LR(0)'da durum tablosunda noktadan sonrasında non-terminal varsa o non-terminal'in gramer kuralıklarını da yazılır. Yine non-terminal varsa devam edilir.

* LR(0) 'ın yetersiz kaldığı iki durum vardır.
  - Shift/Reduce Conflict
  - Reduce/Reduce Conflict

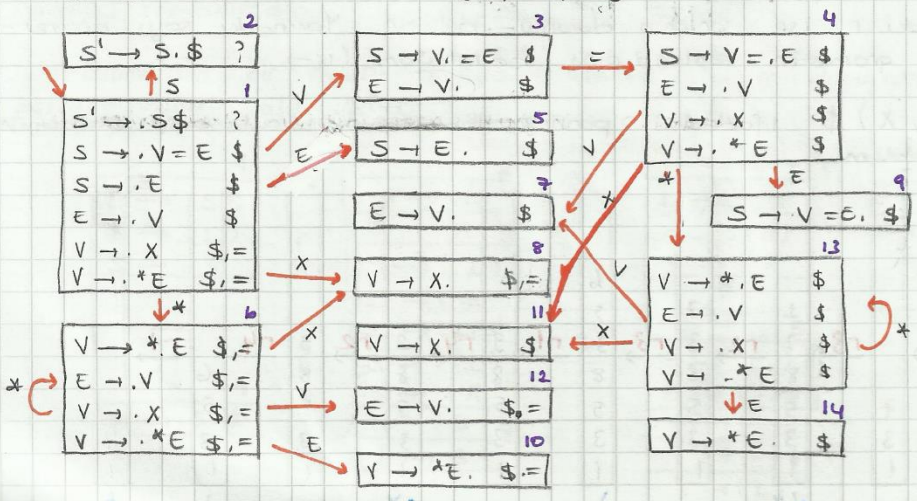
1. $E \rightarrow T. + E$ } shift
   $E \rightarrow T.$ } reduce

2. $E \rightarrow E - T.$ } Reduce
   $E \rightarrow T.$

* LR(1) karakterini oluşturmak için 2 tane küme oluşturmamız gerekiyor. FIRST SET ve FOLLOW SETS

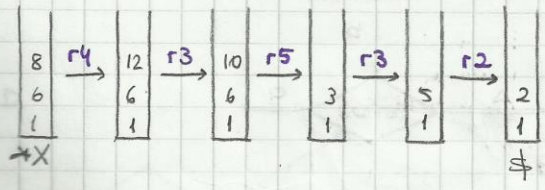
* LR(1) karmaşıklık durumunda ne yapılması gerektiğine çözüm getirmektedir.

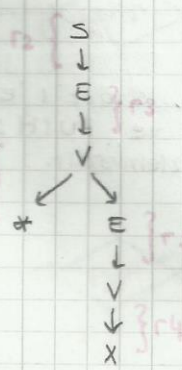
0 $S' \rightarrow S\$$
1 $S \rightarrow V = E$
2 $S \rightarrow E$
3 $E \rightarrow V$
4 $V \rightarrow X$
5 $V \rightarrow * E$



	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			

* * X $ LR(1)'e göre parse edelim.





* Tersten gidilerek parse ağacı çıkarılabilir.

Durumları stack'e atma algoritması;

Shift: Tablodaki s harfleri shifti belirtmektedir. sn görduğümüzde n olan sayıyı stack'e atarız.

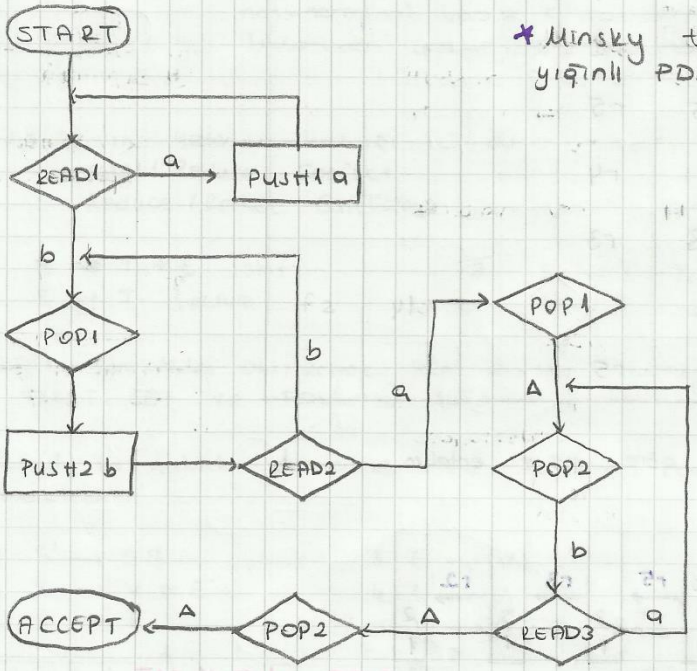
Reduce: Eğer durum grafiğinde gidilecek yerim kalmadıysa gidilecek durumu olana kadar silerim ve o durumdan farklı bir yol seçip ilerlerim. Reduce durumlarının sayısı ise tablodaki durumların karşılıklarından bulunur. rn'dir.

Accept: Bu durum diyagramındaki $S' \rightarrow .S\$$ durumudur. Bu duruma geldiğimizde accept yapmış oluruz. Parsing durdurulur.

Error: Parsing durdurulur. Hatalar rapor edilir.

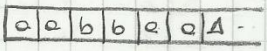
09.05.2016

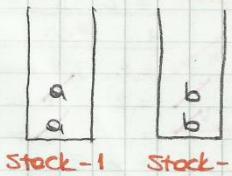
MINSKY TEOREMİ



* Minsky teoremine göre 2 yığınlı PDA = TM'dir.

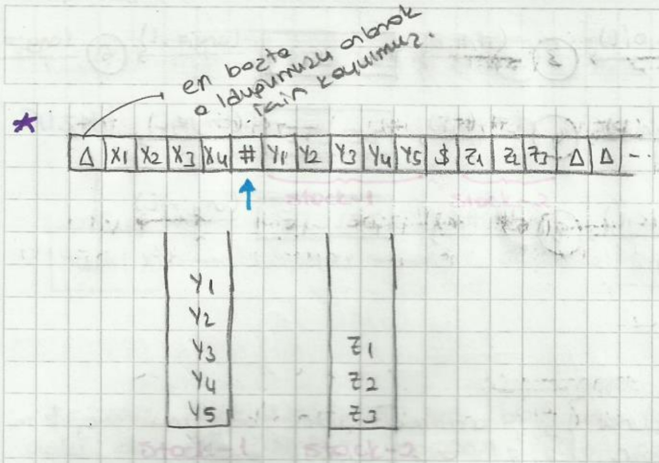
$L = \{a^n b^n a^n \;,\; n > \phi\}$





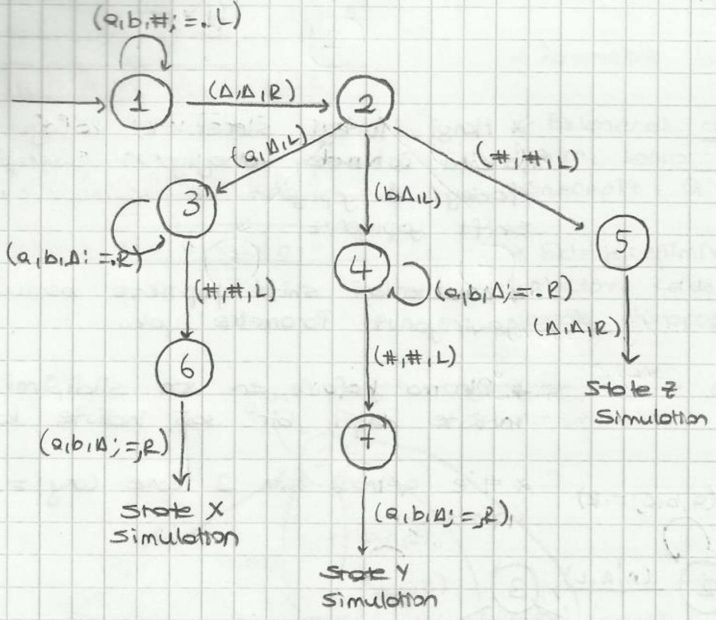
* POP1 ve PUSH 1, Stack-1'e ait işlemlerdir. POP 2 ve PUSH 2, Stack-2'ye ait işlemlerdir.

★



* Her bir işlemde okuma-yazma kafasının #'de olduğunu varsayacağız.

* İki yığın tek bir tape üzerinde simüle edilir.



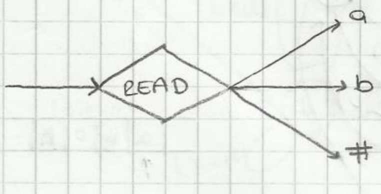
* # üzerinde olduğumuz için tape başına gideriz. (1 durumu)

* Tape'den okuduğu durumu Δ yapıyoruz. (2 durumu)

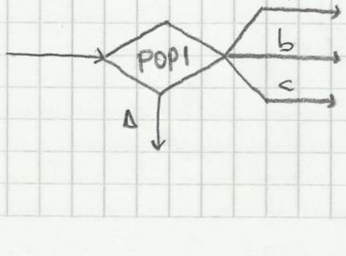
* Döngülerle sağa gidiyoruz. (3,4 durumu)

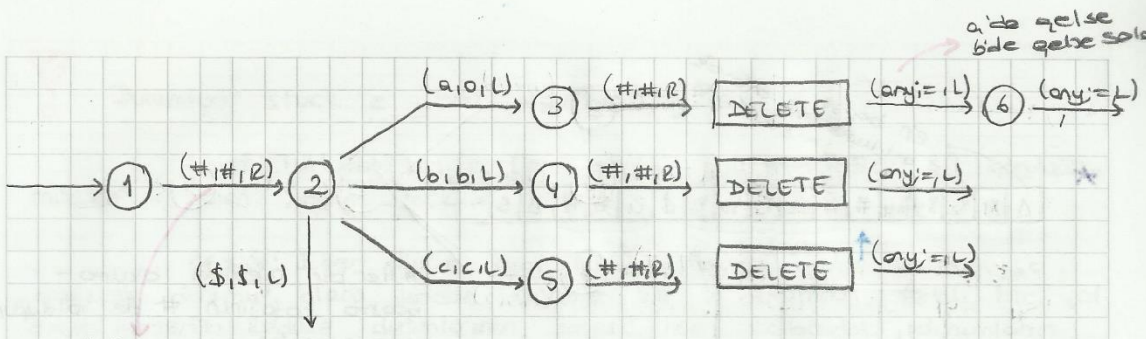
* # görünce sola gidiyoruz sonra sağa. (3→6, 4→7 durumları)

READ



POP1





* Yığında a karakteri olduğunu { 2,3 / 2,4 / 2,5 } arasında yığın boş mu değil mi varsayarak 2-3 arası çalışır. onu kontrol eder.

* (any; =, L), kafa neyin üzerinde olursa olsun sola git diyerek en bazda (#) götürür.

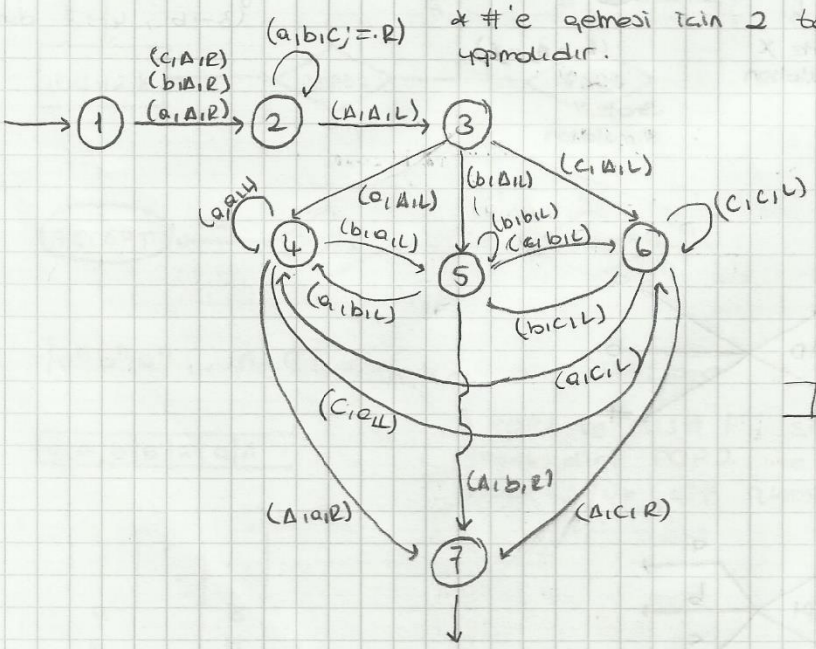
DELETE

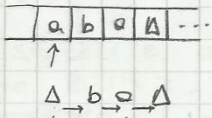
* Hangi ifadeyi sileceksek, kafayı o ifadenin üstünde varsayıyoruz. Sileceğimiz ifadeyi Δ yapıyoruz ve sağındaki ifadeleri shift yapıyoruz

* karakterleri shift yaptıkça boşluğa çeviriyoruz. Parametre yok.

* Okuma kafası en son sildiğimiz indiste değil bir sağ indiste kalır

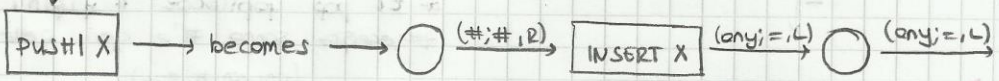
* #'e gelmesi için 2 tane (any; =, L) yapmalıdır.





Δ → b → a → Δ
b ← a ← Δ ←

PUSH 1

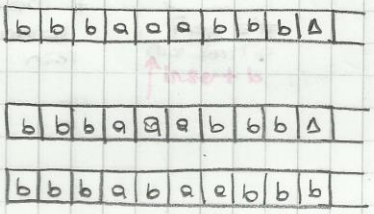


#'nin üzerine gele-bilmek için 2 kez (any;=,L) yapıyoruz.

* Sonsuz büyüklükte yığın var boş da olsa dolu da olsa INSERT yapıyoruz. Yı'in önüne Yo INSERT yaptık. Sonra yine Yı'e kafa işaret eder.

INSERT

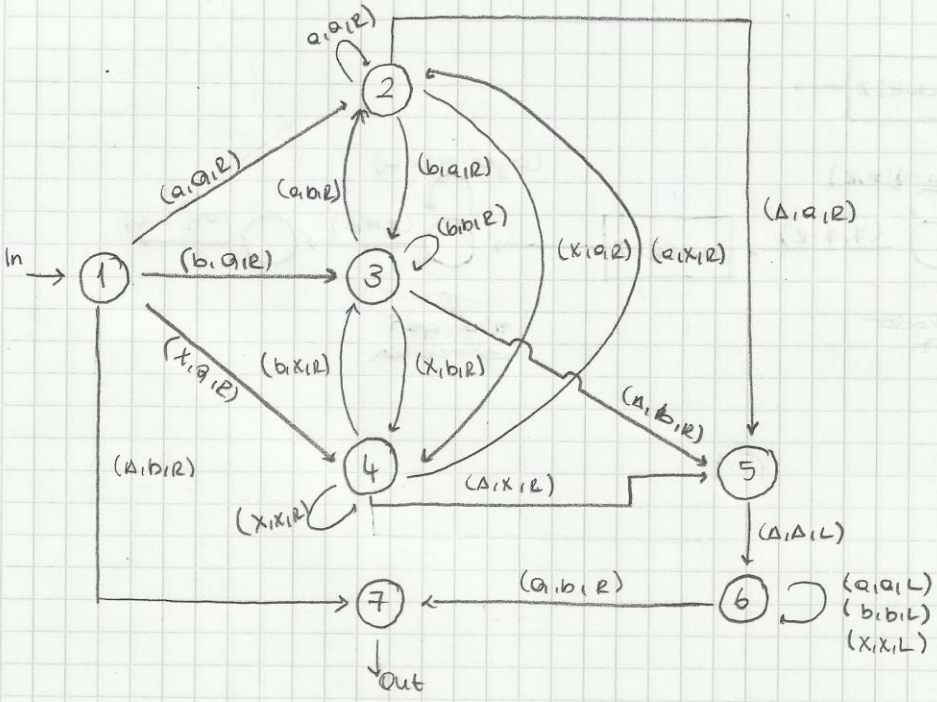
$b^n a^n b^n$

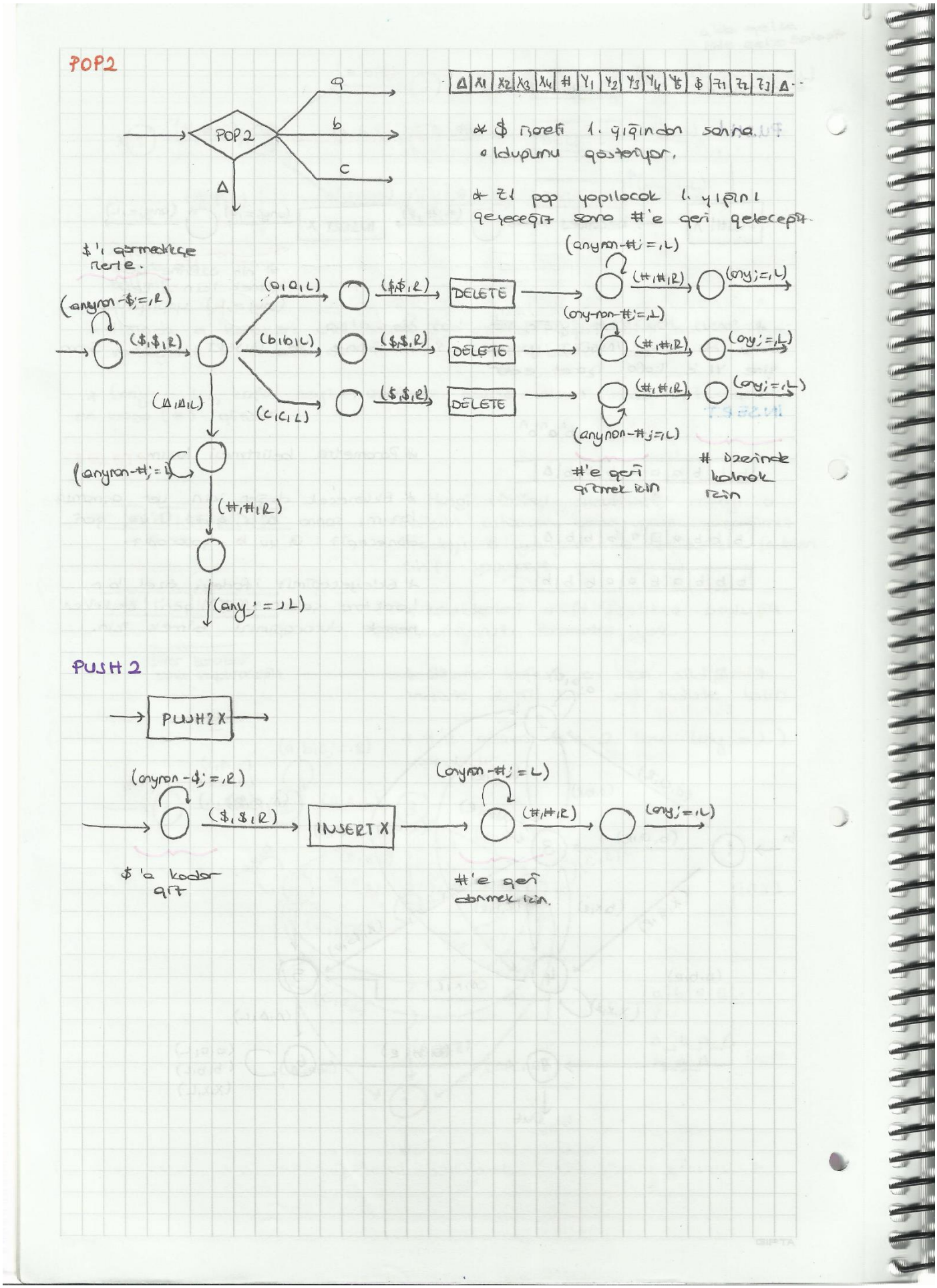


* Parametre belirtmesi lazım.

* Eklenecek değer için yer açmamız lazım, sonra Δ'a gidip @'yu geri döneceğiz. @'yu b yapacağız.

* Ekleyeceğimiz ifadeyi özel bir karaktere çekeriz (@) geri getirken nerede duracağımızı bilmek için.




POP2
a
b
c
Δ
POP2
Δ | X1 | X2 | X3 | X4 | # | Y1 | Y2 | Y3 | Y4 | Y5 | $ | Z1 | Z2 | Z3 | Δ
* $ işareti 1. yığından sonra olduğunu gösteriyor.
* Z1 pop yapılacak 1. yığını geleceği sonra #'e geri gelecek
$'ı görmekçe kadar.
(anynon-$;=,R)
($,$,R)
(a,a,L)
(b,b,L)
(c,c,L)
($,$,R)
DELETE
(anynon-#;=,L)
(#,#,R)
(any;=,L)
(Δ,Δ,L)
(anynon-#;=,L)
(#,#,R)
(any;=,L)
#'e geri gitmek için
üzerinde kalmak için
PUSH 2
PUSH2X
(anynon-$;=,R)
($,$,R)
INSERT X
(anynon-#;=L)
(#,#,R)
(any;=,L)
$'a kadar git
#'e geri dönmek için.