KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BIL 348 OTOMATA TEORİSİ DERS NOTLARI

2012-2013 Bahar Dönemi

18.02.2012 / 13:00

REGULAR EXPRESSIONS (RE)

$\Sigma = \{a, b\}$, $\Sigma = \{1, 0\}$ binler üzerinden matematiksel ifadeler yazacağız.

a^* ne tür stringler üretir?

$a^* = \{ (\lambda), a, aa, aaa, aaaa, \dots \}$

↳ boş string

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

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

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

$(ab)^* \neq a^* b^*$

⟹ önce a'lar geliyor. a'lar bittikten sonra b'ler geliyor.

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

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

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

⟹ λ veya $(a+b)(a+b)(a+b)\dots$ leri sonsuz sayıda üretir.

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

a b a b a b

b a b

* in kendisi λ demek.

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

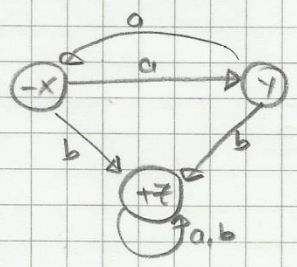
$+$ yı sağlıyor.

$+ \rightarrow$ $a+b$ den en az biri gelecek demek.

Örn: "a ile başlayan b ile biten tüm kelimelerin dili" için bir RE yazın.

$a(a+b)^*b$ → aradaki string ne olursa olsun.

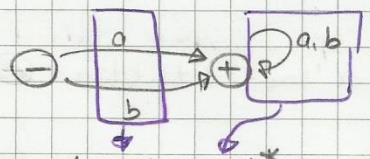
FINITE AUTOMATA (FA)



	a	b
-x	y	z
y	x	z
+z	z	z

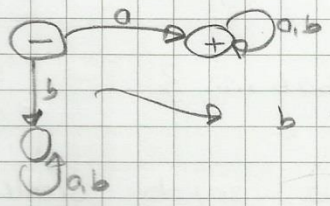
Başlangıç durumu (-), bitiş durumu (+) ile gösterilir.

*



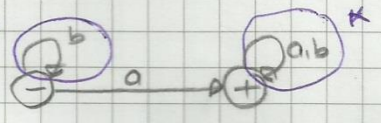
*

$a(a+b)^*$ için FA çizin.



b için en güvenli yol boş stringe gitmek.

*

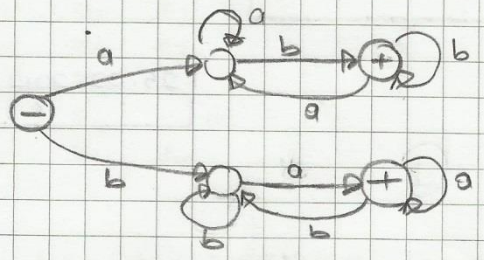


RE si nedir?

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

* Döngü, * ı çağrıştırır.

ÖRN: "a ile başlayıp b ile biten veya b ile başlayıp a ile biten tüm kelimelerin dili" için bir FA çizin.



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

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

RE ile üretilecek min string (+) ya eşitmek zorunda.

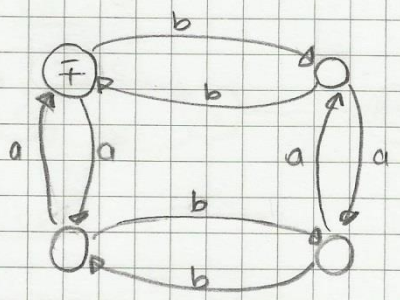
$$\left(a(a+b)^*b + b(a+b)^*a\right)^+ = \left[a(a+b)^*b + b(a+b)^*a\right]\left[a(a+b)^*b + b(a+b)^*a\right]$$

$$= abba$$

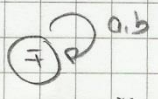
* olsaydı + da olacağı için yine sağlamazdı.

a ile başlayıp a ile bitti. Koşulu sağlamıyor.

ÖRN: " λ dahil çift sayıda a'lar veya b'lerden oluşan tüm kelimelerin dili" için FA çizin.



baba
babbba

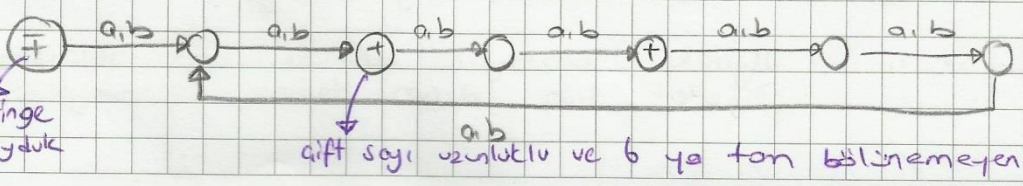


$(a+b)^*$

$$\left[(aa+bb+(ab+ba)(aa+bb)^*(ba+ab)\right]^*$$

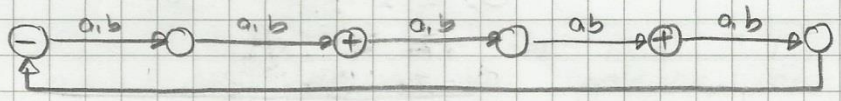
ÖRN

" λ dahil çift sayı uzunluklu ve uzunluğu b sayısına tam bölünmeyen tüm kelimelerin dili için FA çizin.



* Jeffrey Ullman	Youtube videoları bak.

λ 'sız çözüm için ;



25.02.2012 / 13:00

KLEENE'S THEOREM

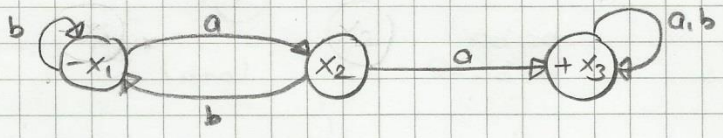
3 bölümden oluşur.

Part III

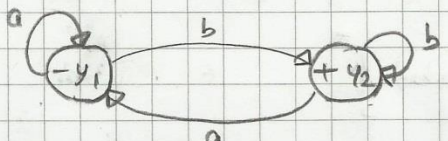
Rule 2: If there is an FA called FA_1 that accepts the language defined by the regular expression r_1 and there is an FA called FA_2 that accepts the language defined by the regular expression r_2, then there is an FA we shall call FA_3 that accepts the language defined by the regular expression $(r_1 + r_2)$.

$FA_3 = FA_1 + FA_2$ → kuralının ispatını yapacaz

ÖRN :



FA_1



FA_2

$FA_3 = FA_1 + FA_2$ → oluşturacağımız FA_3 hem FA_1 in hem de FA_2 nin kabul ettiği stringleri kabul edecek.

$-\xi_1 = x_1$ or y_1 diyerek başlangıç durumu seçtiğimiz (FA_3 için) Diğerleri için durum tablosu oluşturuyoruz.

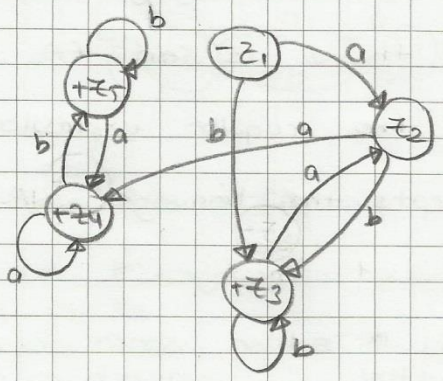
* $FA_3 = FA_1 + FA_2 = FA_2 + FA_1$

$-z_1 = x_1 \text{ or } y_1$

	a	b
$-z_1$	z_2	z_3
z_2	z_4	z_3
$+z_3$	z_2	z_3
$+z_4$	z_4	z_7
$+z_5$	z_4	z_7

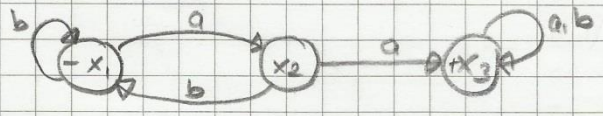
$-z_1 = x_1 \text{ or } y_1$
$z_2 = x_2 \text{ or } y_1$
$+z_3 = x_1 \text{ or } y_2$
$+z_4 = x_3 \text{ or } y_1$
$+z_7 = x_3 \text{ or } y_2$

Tüm durumlar bittikten sonra tablo tamamlanır. Daha sonra bu tablaya göre FA_3 çizilir.

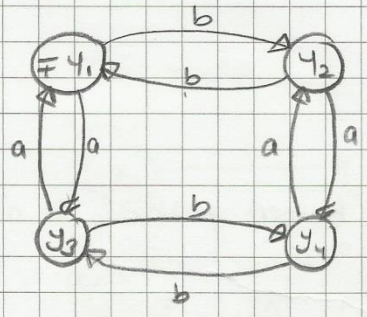


ÖRN!

FA_1

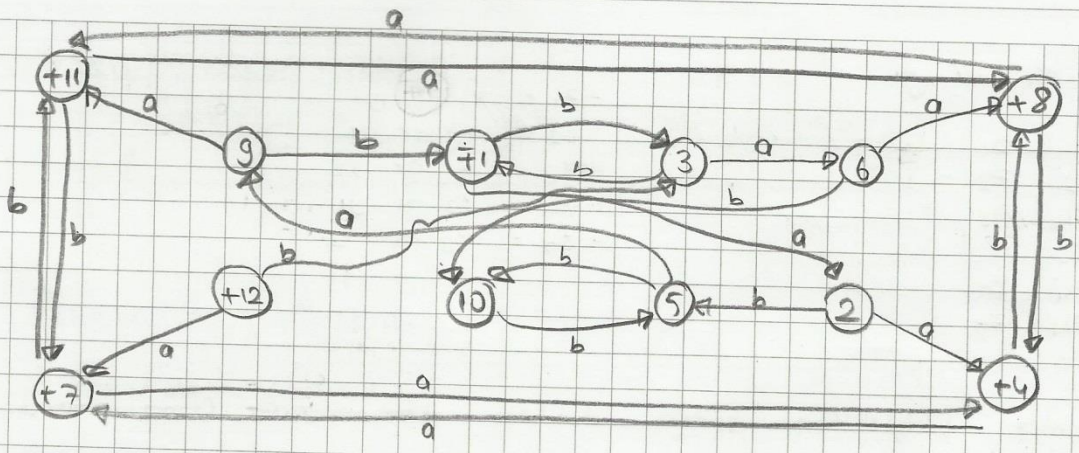


FA_2



	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

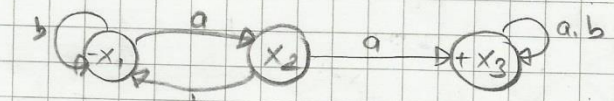
$\mp z_1 = x_1 \text{ or } y_1$
$z_2 = x_2 \text{ or } y_3$
$z_3 = x_1 \text{ or } y_2$
$+z_4 = x_3 \text{ or } y_1$
$z_5 = x_1 \text{ or } y_4$
$z_6 = x_2 \text{ or } y_4$
$+z_7 = x_3 \text{ or } y_3$
$+z_8 = x_3 \text{ or } y_2$
$z_9 = x_2 \text{ or } y_2$
$z_{10} = x_1 \text{ or } y_3$
$+z_{11} = x_3 \text{ or } y_4$
$+z_{12} = x_2 \text{ or } y_1$



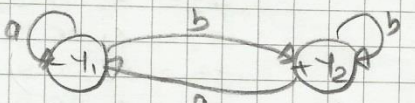
Rule 3 : If there is an FA_1 that accepts the language defined by the regular expression r_1 and there is an FA_2 that accepts the language defined by the regular expression r_2, then there is an FA_3 that accepts the language defined by the concatenation r_1r_2, the product language.

$FA_1 * FA_2 = FA_3$ → önce r_1 'in string, sonra r_2 'nin kabul ettiği " gelecek. r_2 'li string sonlandığı anda FA_3 oluşacak.

ÖRN :



* FA1 *



* FA2 *

r_1r_2 → önce r_1 in kabul ettiği string, hemen peşine r_2 nin kabul ettiği string gelecek.

$FA_3 = FA_1 * FA_2$

↓

FA_1 de (+) ya kadar gidilir. (+) gittiğinde anda FA_2 de başlangıç noktasına gidilir. FA_2 de (+) ya gittiğimizde FA_3 sonlanır.

$-z_1 = x_1$

$z_2 = x_2$

$z_3 = x_3$ or y_1 (bağlantı noktası)

	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

$z_2 \to a$: x_3 or $y_1 = z_3$

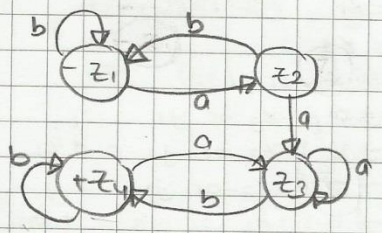
$z_2 \to b$: z_1

$z_3 \to a$: x_3 or y_1

$z_3 \to b$: x_3 or y_1 or $y_2 = +z_4$

$z_4 \to a$: x_3 or y_1

$z_4 \to b$: x_3 or y_1 or y_2



$FA_1 * FA_2$

ÖRN!

$FA_3 = FA_2 * FA_1$

	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

$-z_1 = y_1$

$z_2 = y_2$ or x_1

$z_2 \to a$: y_1 or $x_2 = z_3$

$z_2 \to b$: y_2 or x_1

$z_3 \to a$: y_1 or $x_3 = +z_4$

$z_3 \to b$: y_2 or x_1

$z_4 \to a$: y_1 or x_3

$z_4 \to b$: y_2 or x_1 or $x_3 = +z_5$

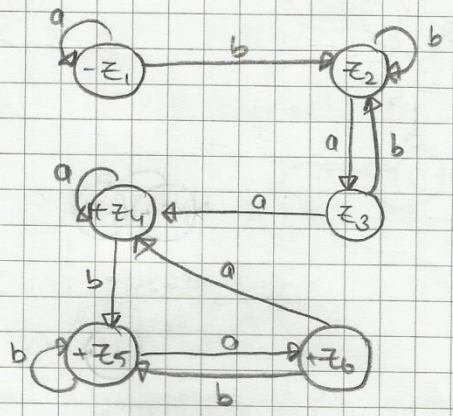
$z_5 \to a$: y_1 or x_2 or $x_3 = +z_6$

$z_5 \to b$: y_2 or x_1 or x_3

$z_6 \to a$: y_1 or x_3

$z_6 \to b$: y_2 or x_1 or x_3

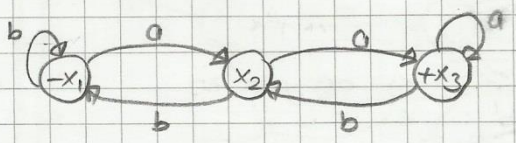
$FA_2 * FA_1$



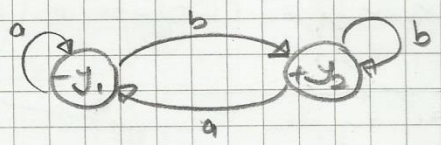
$FA_1 * FA_2 \neq FA_2 * FA_1$

ÖRN:

FA1

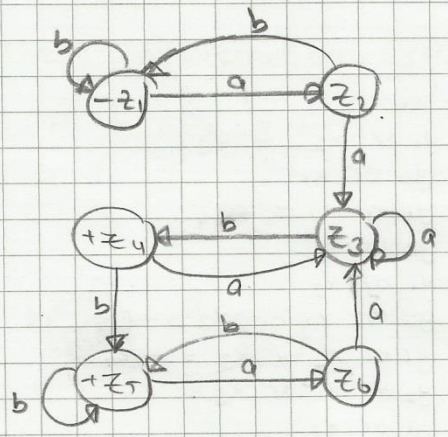


FA2



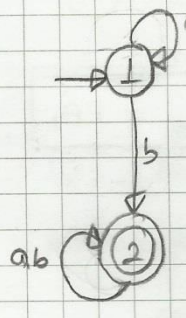
FA1 * FA2	a	b
z1	z2	z1
z2	z3	z1
z3	z3	z4
+z4	z5	z5
+z5	z6	z5
z6	z3	z5

- $z_1 = x_1$
- $z_2 = x_2$
- $z_3 = x_3$ or y_1
- $z_3 \rightarrow a$! x_3 or y_1
- $z_3 \rightarrow b$! x_2 or $y_2 = +z_4$
- $z_4 \rightarrow a$! x_3 or y_1
- $z_4 \rightarrow b$! x_1 or $y_2 = +z_5$
- $z_5 \rightarrow a$! x_2 or $y_1 = z_6$
- $z_5 \rightarrow b$! x_1 or $y_2 = z_5$
- $z_6 \rightarrow a$! x_3 or y_1
- $z_6 \rightarrow b$! x_1 or y_2

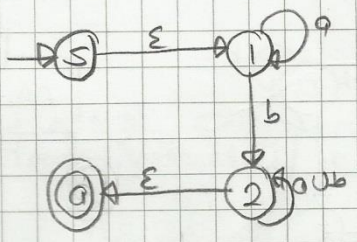


04.03.12 / 13:00

RE OLUŞTURMA



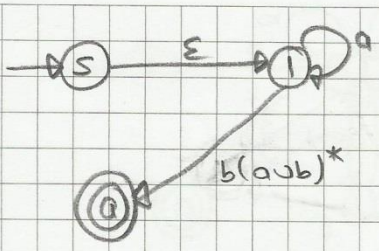
(a)



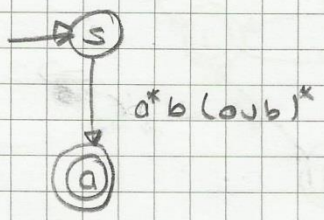
(b)

$\varepsilon \equiv \lambda$

RE sini adim adim oluşturmak için yeni başlangıç ve bitiş noktaları yazılır.



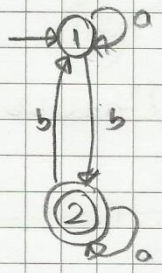
(c)



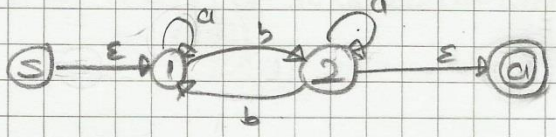
(d)

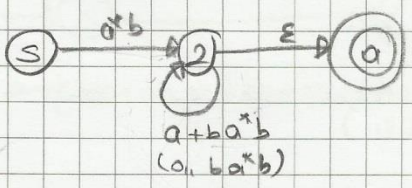
S ve a dışındakileri yok ediyoruz. Silerken belli bir kural yok. ε leri yazarak siliyoruz.

ÖRN

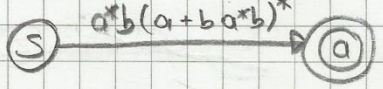


Bunun için ε sini adım adım oluşturacağız

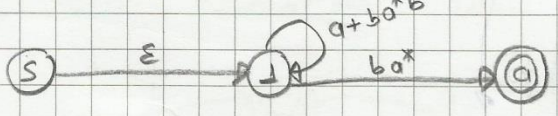


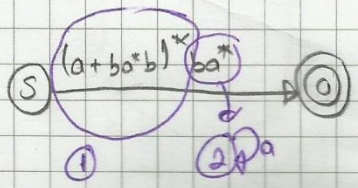


(önce 1'i sildik)



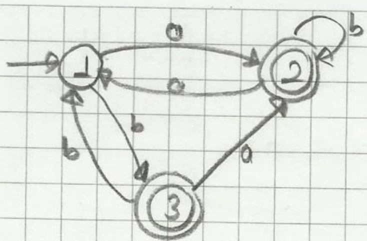
önce 2 yi silersek;



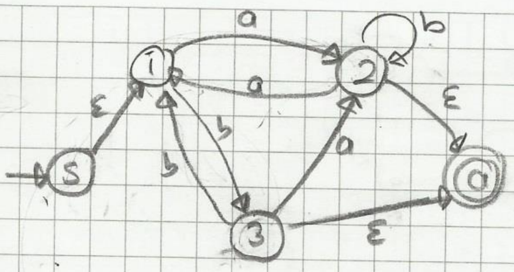


** Ara birimleri silme sırası ε leri farklı çıkarabiliyor.

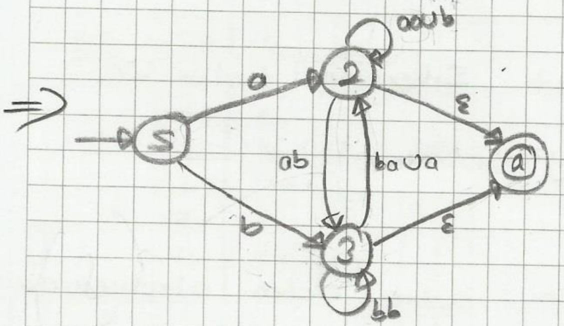
ÖRN



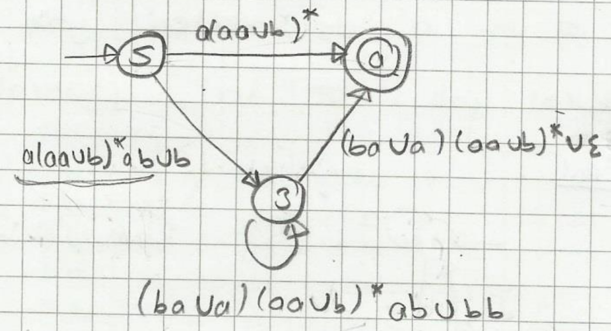
$\Rightarrow$



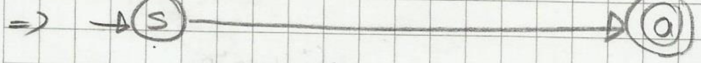
$\Rightarrow$



$\Rightarrow$

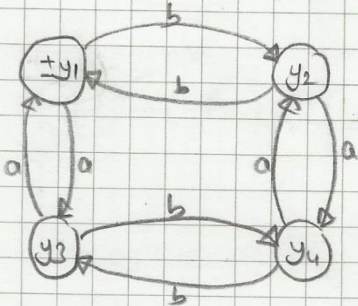


$\Rightarrow$



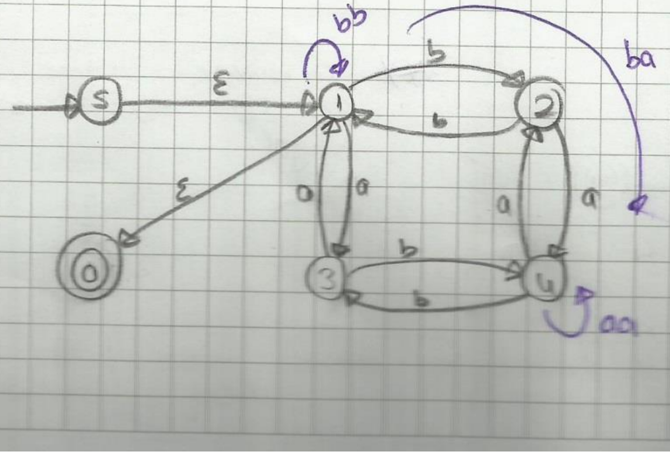
$$a(aa+b^*) + (a(aa+b)^* ab + b)\left[(ba+a)(aa+b)^* ab + bb\right]^* \left[(ba+a)(aa+b)^* + \varepsilon\right]$$

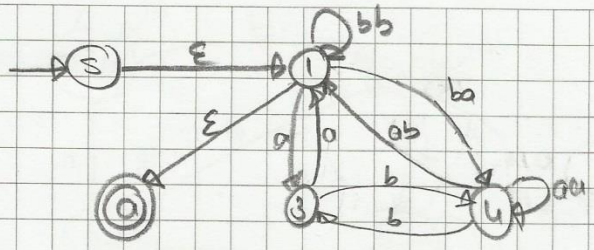
ÖRN !

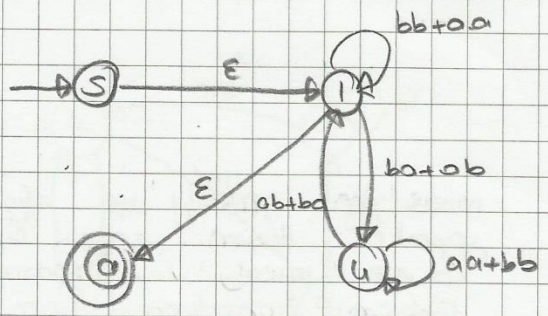


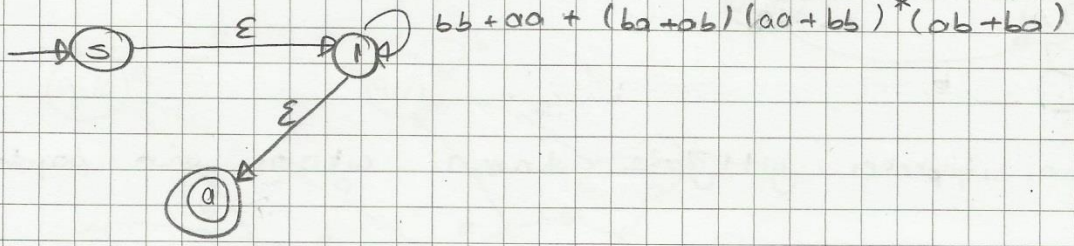
RE'sini bul.

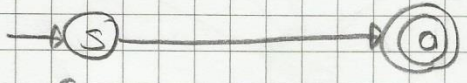
(2 veya 3 önce silinirse öbürüne kolay gidilir)











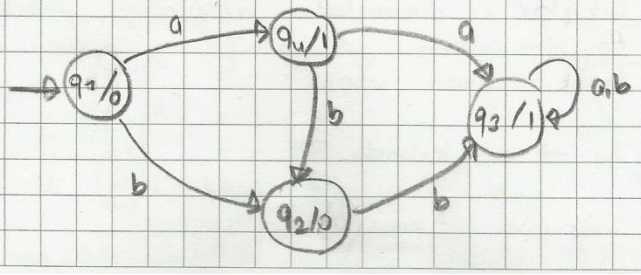
$$\left[bb+aa+(ba+ab)(aa+bb)^*(ab+ba)\right]^*$$

Finite Automata with Output

İki türlü durum diyagramı çizilir. mealy ve moore türü.

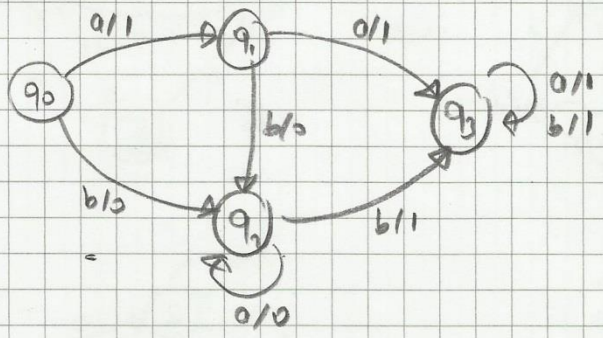
Moore türü geçişlerde,

mealy türü durumlarda çıkış üretir

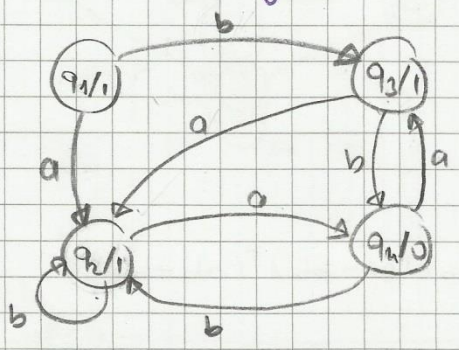


#Moore türü diyagram#

Mealy türü diyagram:

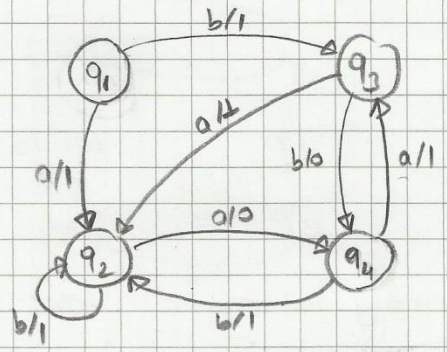


Moore → Mealy Dönüşümü

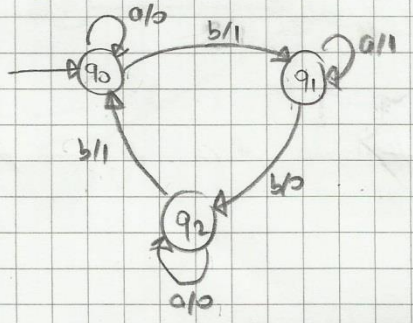


Moore'dan Mealy'ye dönüşüm yaparken durum sayısı aynen korunur. Mealy'den Moore'ye dönüşüm yaparken durum sayısı artabilir.

Dönüşüm yaparken gittiğimiz durumun çıkışını onun paydasına yazarız.

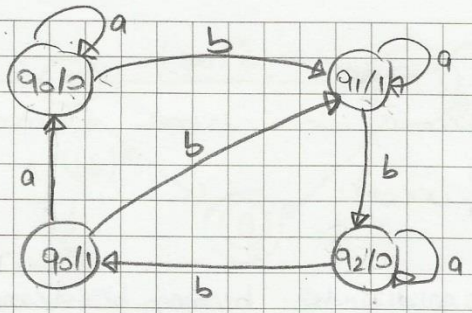


Mealy → Moore Dönüşümü

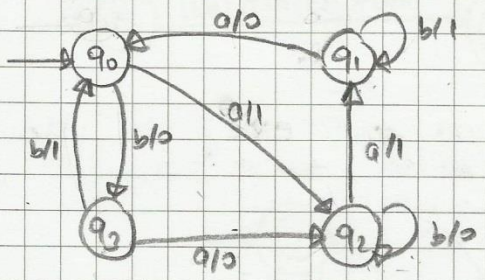


Çıkışlar, durumların içerisine yazılacak. Hangi çıkışın yazılacağı geçişlere bakılarak karar verilir.

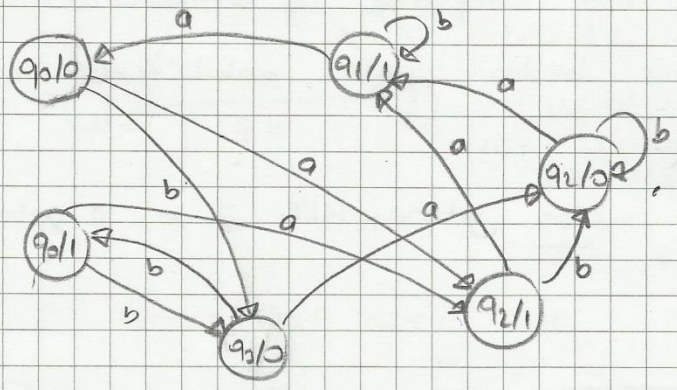
q_0 → a çıkışında 0
b " 1 old. için iki farklı durumla temsil edilir.



ÖRN



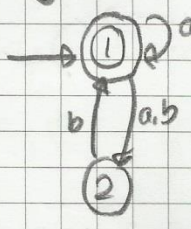
Dönüşümünü yap.



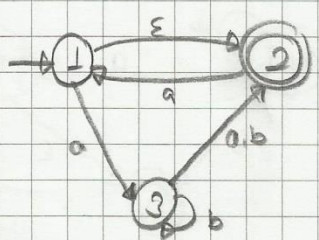
11.03.2013 / 13:00

EQUAVALANCE OF NFA AND DFA

Nondeterministik FA → herhangi bir durumdan çıkan geçişler birden fazla duruma gidiyorsa olur. Determinist ikte ise tek bir duruma gider.



(a) nondeterministik

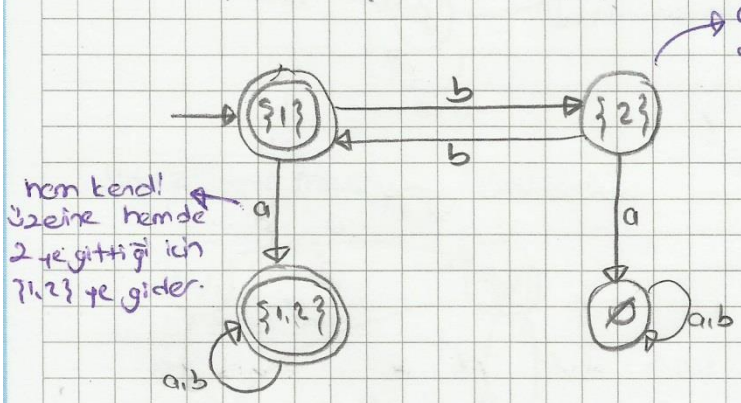


(b) deterministik

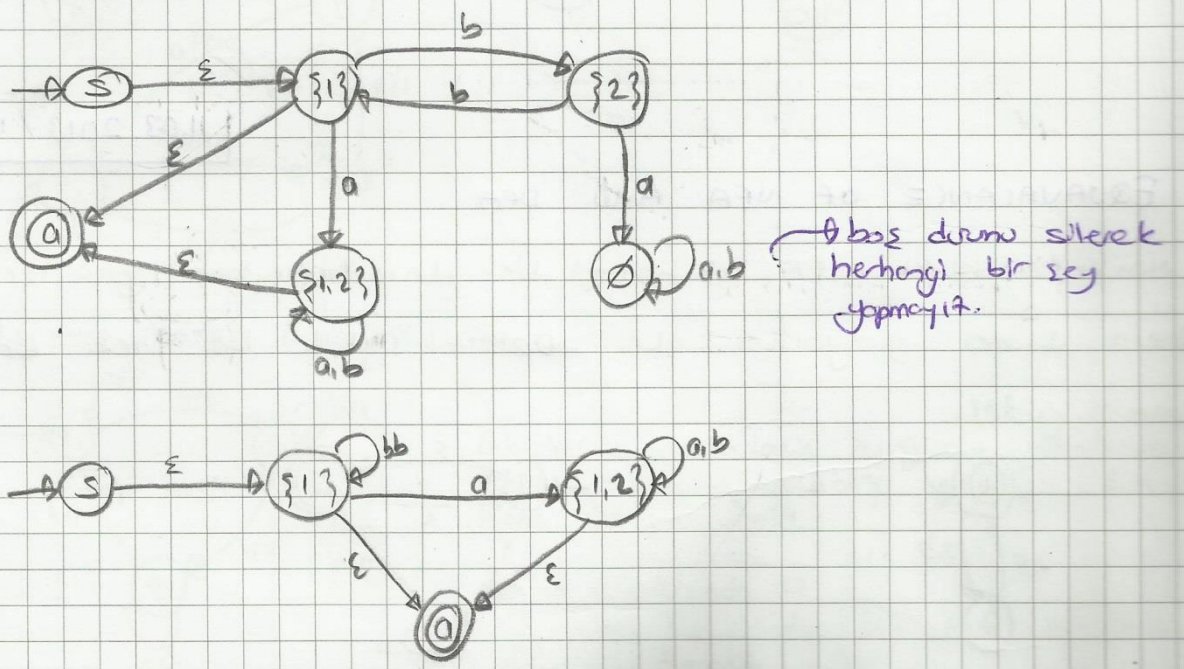
NFA dan DFA ya geçiş:

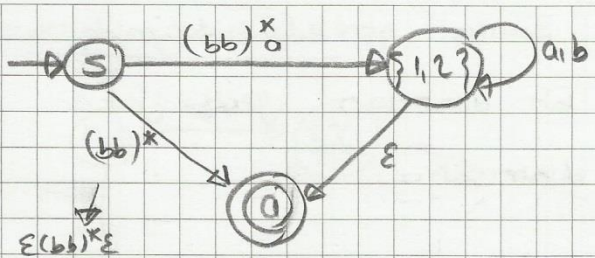
alt küme sayısına bakılır. (a) durumu için küme elemanları {1,2} dir. 2^n tane alt küme vardır.

{1}, {2}, {1,2}, { }

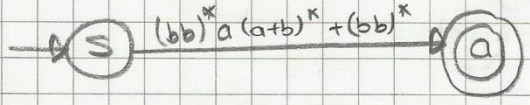


- deterministik çıkarılırken alt kümelerde 1 bulunanların hepsine (+) konulur. Bitiş durumu birden fazla olabilir Ama başlangıç durumu 1 tane olacak
- NFA de bazı durumlar çıkışa gitmeyebilir. 2 de a gelmemesi gibi.

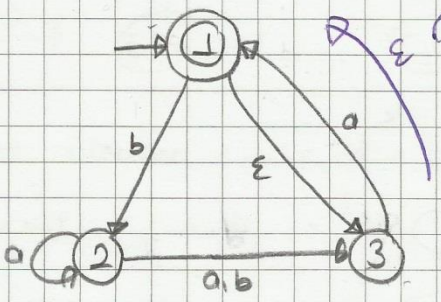




$\varepsilon(bb)^*\varepsilon$ şeklinde yazılması da aynı anlamdadır.



ÖRN

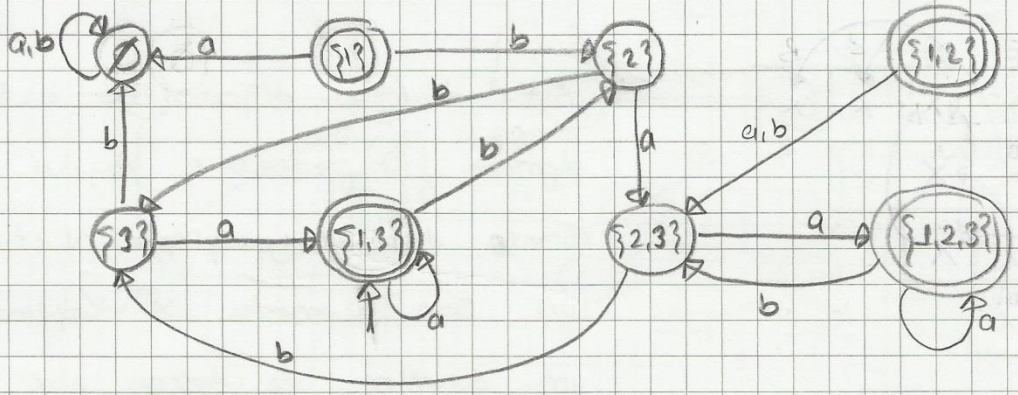


→ böyle bir geçiş olmadığı için 3 tek başına bitiş olmaz

NFA dan DFA ya geçiş = ?

* Başlangıç durumu ε dan dolayı 3 de olabilir. Bitiş durumu da içinde 1 ya da 1, 3 olanlar olabilir. Tek başına 3 için (+) kaynayan çünkü ters yönde geçiş yok.

$2^3 = 8$ tane alt küme var.

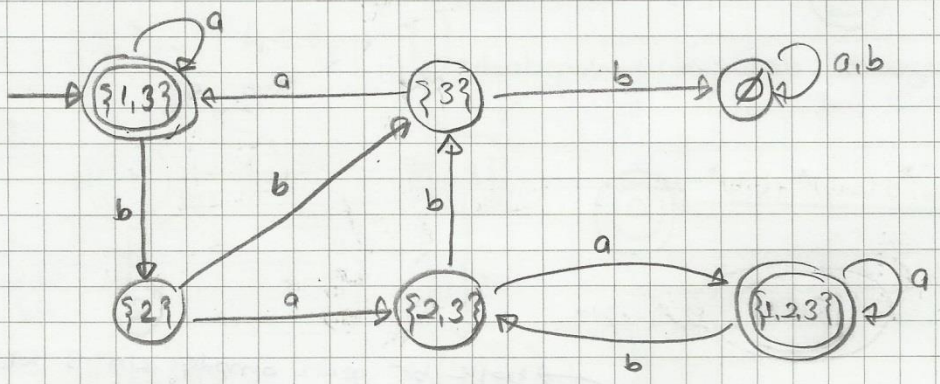


* 3 nolu durumda a geldiğinde 1'e gidiyor. Ama ε ile kendi üzerine de gelebilir.

Deterministik FA oluşturulduktan sonra gereksiz durumlar atılırlar.

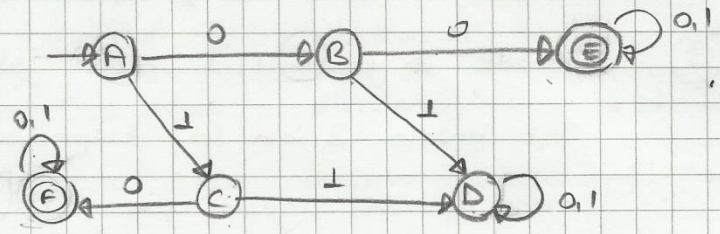
Gereksiz durum: Kendisine başka bir durumdan geçiş yok.

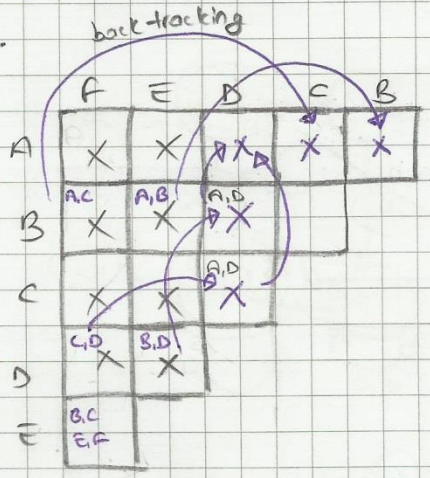
{1,2} ve {1} gereksiz durumdur.



MINIMIZING FA

Amaç, eşdeğer durumları birleştirmek

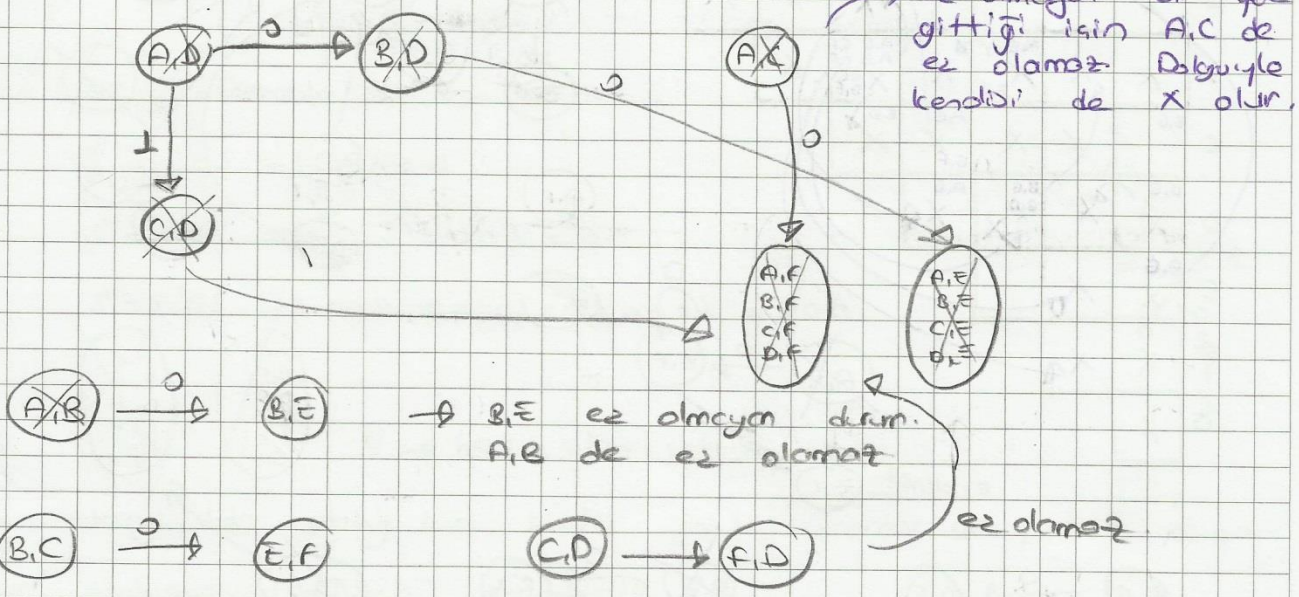




Bitiş durumları diğerleri ile eşdeğer olamaz. O yüzden x kaplar.

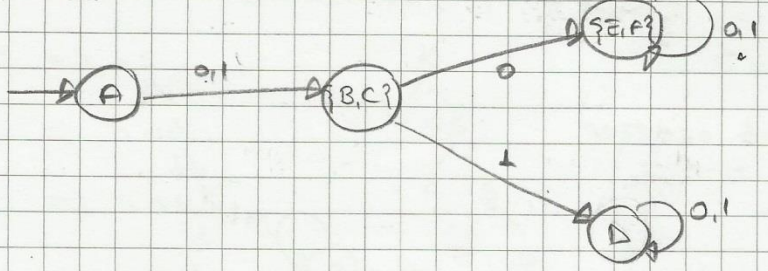
Sonra dependency graph oluşturulur. Son aşamada x kaplamayan durumlar eş değer olur

Dependency Graph

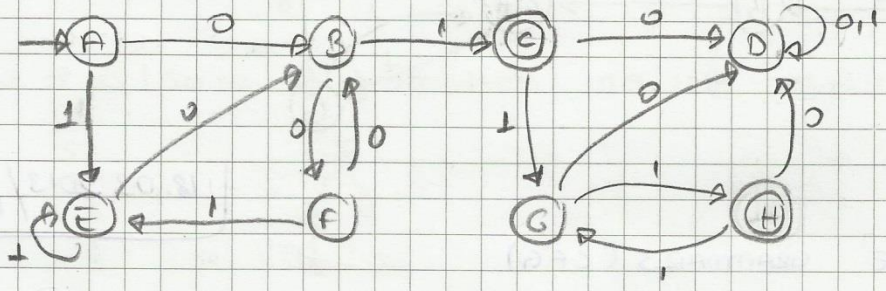


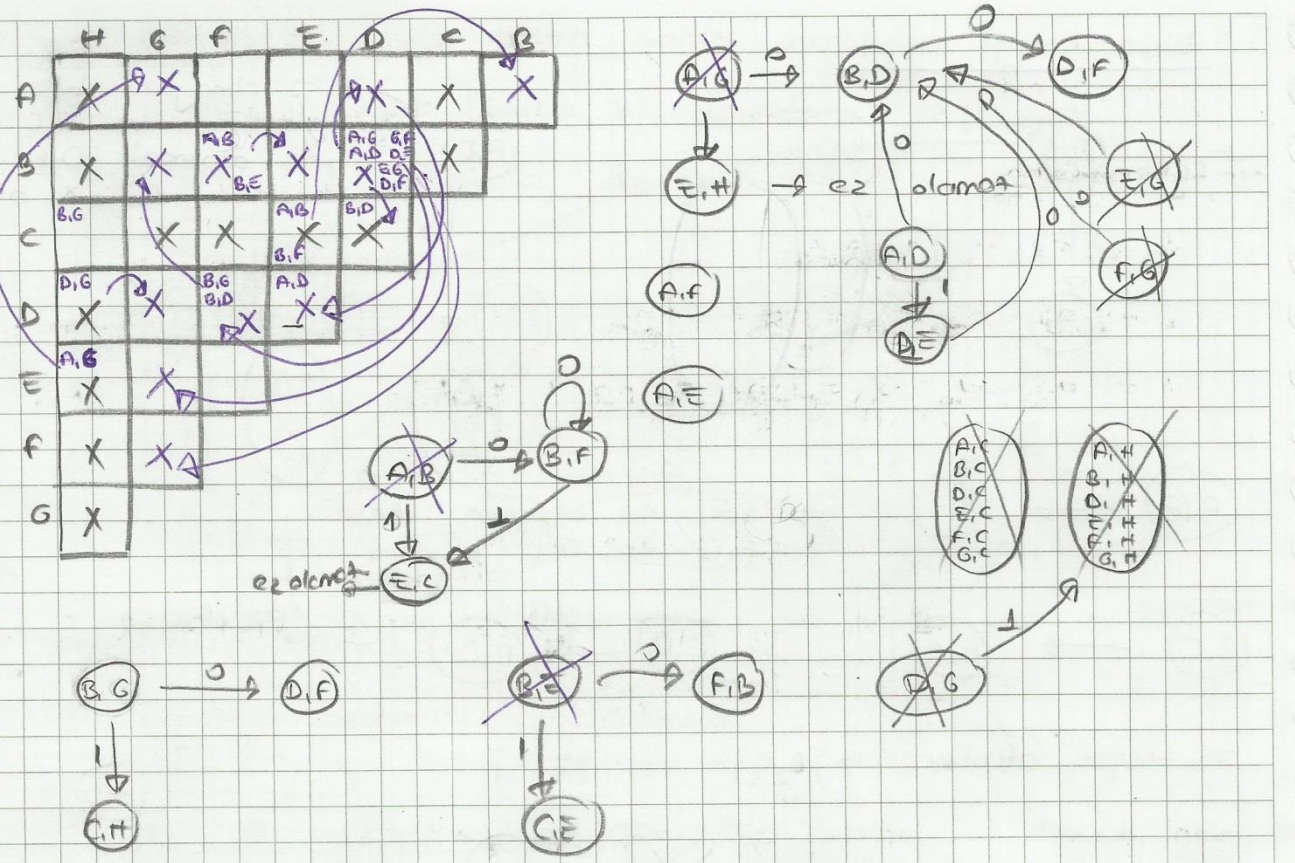
Eşdeğer olanlar {B,C} ve {E,F}

son durum!

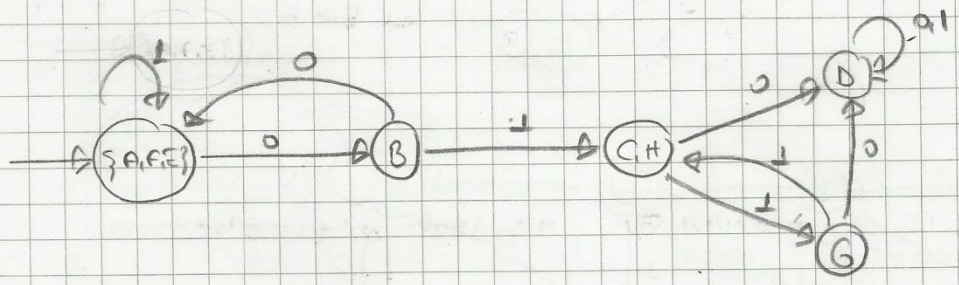


Ödev





{A,F,E}, {C,H} → eşdeğer



18.03.2013 / 13:00

CONTEXT - FREE GRAMMARS (CFG)

CFG oluştururken boş semboller kullanılacak.

terminal (sonlanan) : {a, b, λ}

nonterminal (sonlanmayan) = {S, A, B, X, Y, Z}

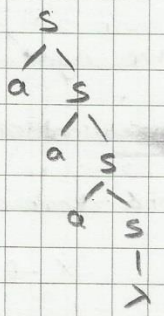
(start)

$S \to aS$

$S \to \lambda$

her biri production.

$S \to aS \mid \lambda$ (CFG gösterimi)



$S \Rightarrow aS \Rightarrow a\lambda = a$

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

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

⋮

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

* Herhangi bir string için birden fazla gramer çizebiliyorsak o gramer ambiguous (belirsiz) gramerdir.

ÖRN $S \to aS \mid bS \mid (a \mid b \mid \lambda)$ → terminaldir. Recursive çağrıyı sonlandırır.

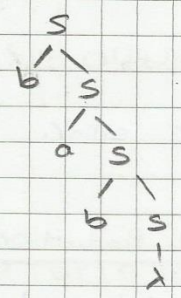
bab nin belirsiz olduğunu gösterelim.

$S \Rightarrow bS$

$\Rightarrow baS$

$\Rightarrow babS$

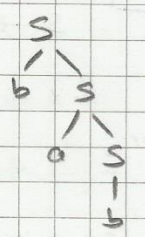
$\Rightarrow bab\lambda$



$S \Rightarrow bS$

$\Rightarrow baS$

$\Rightarrow bab$



ÖRN

$S \to aS \mid Sa \mid a$ gramerinde aa için oluşturalım.

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

CFG → λ yok etme → Chomsky Normal Form (CNF) → CYK Parsing

λ yok Etme Algoritması

1) Delete all λ productions

2) for each production X → something with at least one nullable nonterminal on the right hand side (RHS) do the following each possible nonempty subset of nullable nonterminals on the RHS: Generate X → something where the new RHS is the same old RHS expect with entire current subset of nullable nonterminals removed.

ex X → Y (X → Y → λ → X dolaylı olarak null olabilir)

Y → a | λ (Y doğrudan null olabilir)

ex S → aSa | a ⇒ S nullable olamaz.

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

Örn |

S → a | Xb | aYa

nullables : X, Y

X → Y | X | λ → ilk etapta λ lar silinir.

Y → X | a

Original Production	new Production
S → Xb	S → b } mevcut gramere eklenir.
S → aYa	S → aa } mevcut gramere eklenir.
X → Y	nothing
X → X	nothing
Y → X	nothing

S → a | Xb | aYa | b | aa
X → Y | X
Y → X | a

} new productionlar eklenerek, gramerin son hali elde edilir.

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 : S, x, z, w

Original production

$S \to x$
$S \to xy$
$S \to z$
$x \to z$
$y \to wa$
$z \to wx$ → subset oldu için ayrı ayrı silinir.
$z \to az$
$z \to zb$
$w \to xyz$
$w \to bxa$

(iki nonterminal olma durumu.)

New production

nothing
$S \to y$
nothing
nothing
$y \to a$
$z \to x$, $z \to w$ (ikisi aynı anda nullable olmayabilir)
$z \to a$
$z \to b$
$w \to yz$, $w \to xy$, $w \to y$
$w \to ba$

$S \to x \mid xy \mid z \mid y$

$x \to z$

$y \to wa \mid a$

$z \to wx \mid az \mid zb \mid x \mid w \mid a \mid b$

$w \to xyz \mid bxa \mid yz \mid xy \mid y \mid ba$

CNF (Chomsky Normal Form)

Each of production has the one of two forms:

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

2) Nonterminal → one terminal ($S \to a$, $A \to a$ gibi)

Bu iki formdan birini sağlamalı. CNF ye çevirmeden λ'lar yok edilmeli. Aynı zamanda gramerin sağ tarafında büyük küçük harf yanyana olmaz.

* Nonterminaller 2, terminaller 1 tane olmak zorundadır.

Örn: $S \to aXX$

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

=> $X \to a$ den dolayı CNF ye dönüştürüre 2. B durumda diğerleri CNF ye uydurulur.

$S \to aXX$ ⟶ $S \to AR_1$

$X \to AS \mid BS \mid a$ $R_1 \to XX$

$A \to a$ $X \to AS \mid BS \mid a$

$B \to b$ $A \to a$

$B \to b$

Örn:

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

$S \to AR_3$ or $S \to AR_1$

$R_3 \to XR_2$ $R_1 \to XR_2$

$R_2 \to XR_1$ $R_2 \to XR_3$

$R_1 \to BA$ $R_3 \to BA$

*

$S \to a \mid XB \mid AY$

$X \to Y \mid a \mid \lambda$

$Y \to XA \mid a$ → Unit old. için bunları da yok etmemiz gerekir. Tek başına nonterminal olamaz

CYK Parsing Algoritması

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

$S \to BS \mid AX \mid b$

$X \to BX \mid AS \mid a$

$A \to a$

$B \to b$

bu gramerin abbab stringini kabul edip etmediğine bakalım.

Stringin boyu kadar tablo oluşturulur.

	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 bureye gelirse, gramer stringi kabul etmiştir demek.

X,A / S,B } XS, XB, (AS) → X'te var. AB

S,B / S,B } SS, SB, (BS) → S BB

S,B / X,A } SX, SA, (BX) → X BA

II. sütun

a,b'ler hangi productionlarda karşısımıza çıkıyorsa ilk sütuna onları yazarız.

X,A	X	
	S	
S,B		

3. kolon 1. satır

X,A / S } XS, (AS) → X

soldan sağa 1. kutu ile o kutunun (3. sütun 1. satır) sol alt çaprazındaki ilk kutu çaprazlanır.

X / S,B } XS, XB

Sonra soldan sağa 2. kutu ile o kutunun sol alt çaprazındaki 2. kutu çaprazlanır.

3. kolon 2. satır

S,B / X } SX, (BX) → X

S / X,A } SX, SA

3. kolon 3. satır

S,B / X } SX, (BX) → X

X / S,B } XS, XB

Aynı işlemler tekrarlanır.

4. kolon 1. satır

X,A / X } XX, (AX) → S

XX

X / XA } XX, XA

2. satırda soldan sağa ilk kutuyla, o kutunun sol alt çaprazındaki ilk kutu çaprazlanır.
soldan sağa 2. kutu ile sol alt çaprazdaki 2. kutu çaprazlanır.

X / X } XX

4. kolon 2. satır

S,B / X } SX, (BX) → X

SX

X / SB } XS, XB

5. kolon

XA / X } XX, (AX) → S

XX

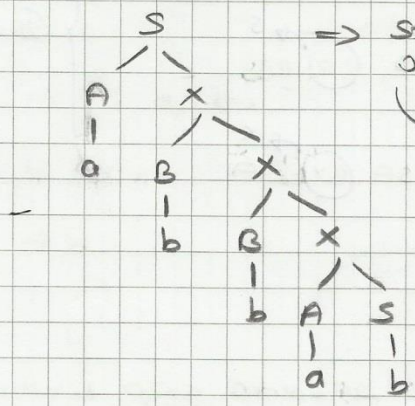
XX

S / S,B } SS, SB

* Ambiguas gramer nasıl oluşur?

Parse ağacı nasıl oluşturulacak?

Terstен gidiyoruz (Çaprazlamalar sonucu oluşan S ve X'ler bulacağız.)



=> Son kutuyu S'yi yazarken AX kullanmıştık. O yüzden AX'i S'nin altına yazdık.

bunu yazınca 4. kolan ile işimiz bitti.

4. kolana baktığımızda; elimizde A ve X var. A'yı a'dan elde ettik. Ağaçta A'nın altına a yazarız. X'e ise (4. kolan 2 satır) SB ile X'in çaprazlamasından elde edilir. (SB ile X'in çaprazlaması sonucu BX olan X elde edilir) Bu durumda X'in altına BX yazarız.

3. kolanda ikeni;

B'yi b'den elde ettiğimiz için b yazarız. X'i ise (3. kolan 3. satır) SB ile X'in çaprazlamasından BX olduğu için X'in altına BX yazdık.

2. kolanda ikeni

B yerine b yazdık (X'i, XA - SB çaprazlamasında elde ettik, bu çaprazlamadaki AS bize X'i verdi). X'in altına AS yazdık ağaçta.

1. kolan

A'nın altına a yazdık. S'in altına b yazdık

25.03.2013 / 13:00

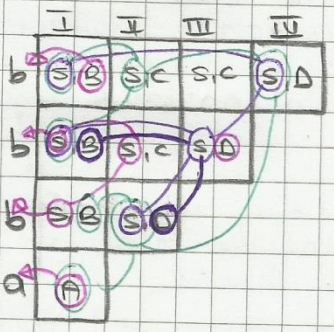
Örn:

$S \to SS \mid AC \mid BD \mid AB \mid BA \mid b$
$C \to SB$
$D \to SA$
$A \to a$
$B \to b$

a) bbba kelimesi için parse ağaçlarını çiziniz.

b) Aynı tabloyu kullanarak bbb ve bba için de parse ağacı çiz.

İpucu: Toplam 8 farklı parse ağacı vardır.



II

S,B / S,B) SS, SB, BS, BB

S,B / S,B) SS, SB, BS, BB

S,B / A) SA, BA

III

S,B / S,C) SS, SC, BS, BC

S,C / S,B) SS, SB, CS, CB

S,B / S,D) SS, SD, BS, BD

S,C / A) SA, CA

IV

S,B / S,D) SS, SD, BS, BD

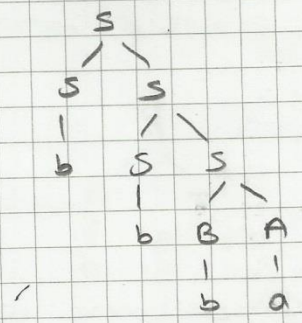
S,C / S,D) SS, SD, CS, CD

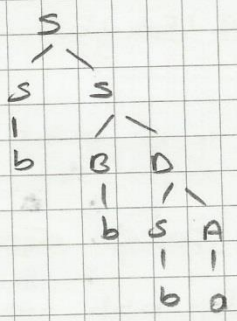
S,C / A) SA, CA

Son sütunda S olduğu için stringi kabul eder. Yanında başka birşeyler de olabilir.

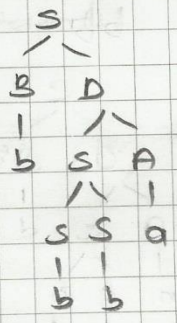
Parse ağacını çizerken son sütunda sadece S olmalı. Parse ağaçlarında root'ta her zaman S vardır.

Başlarken 3 tane S alternatifimiz var.

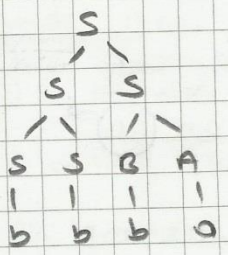




1. alternatif için oluşan parse ağaçları.



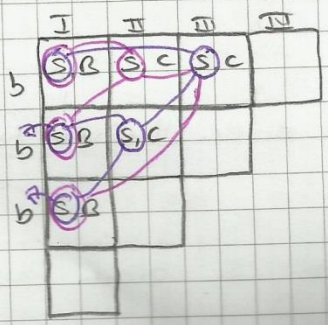
2. alternatif için oluşan parse ağacı

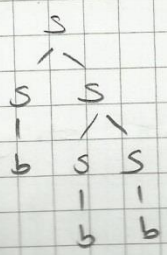


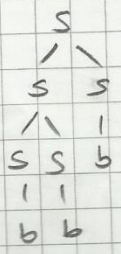
3. alternatif için oluşan parse ağacı.

b) AH stringle için mevcut ağacı kullanabiliriz

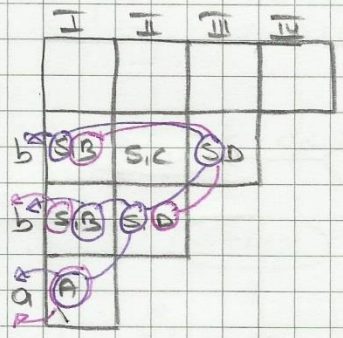
bbb için

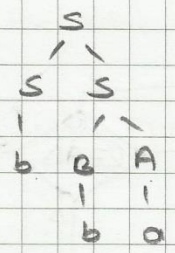


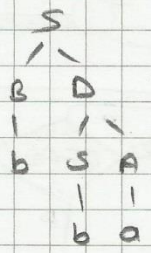




bba için



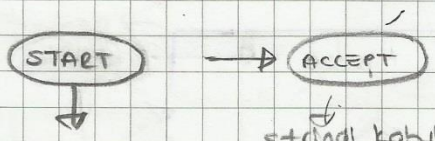




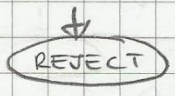
PUSH DOWN AUTOMATA (PDA)

Dil tanıtılı yapacağız. Tanıtı ederken PDA yı kullanacağız. Akış diyagramlarına benzerdir.

PDA'nın alt bileşenleri:

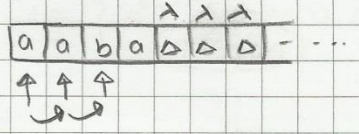


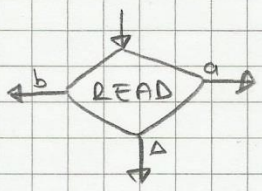
stringi kabul etmesi



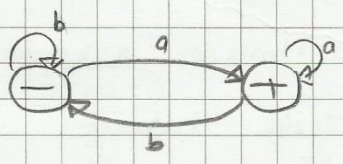
stringi kabul etmeme durumu

·TAPE

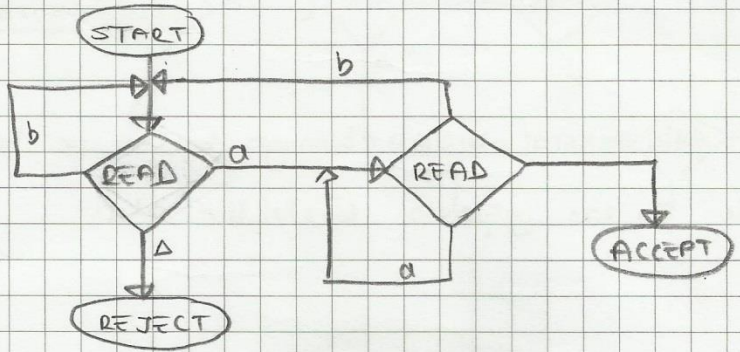




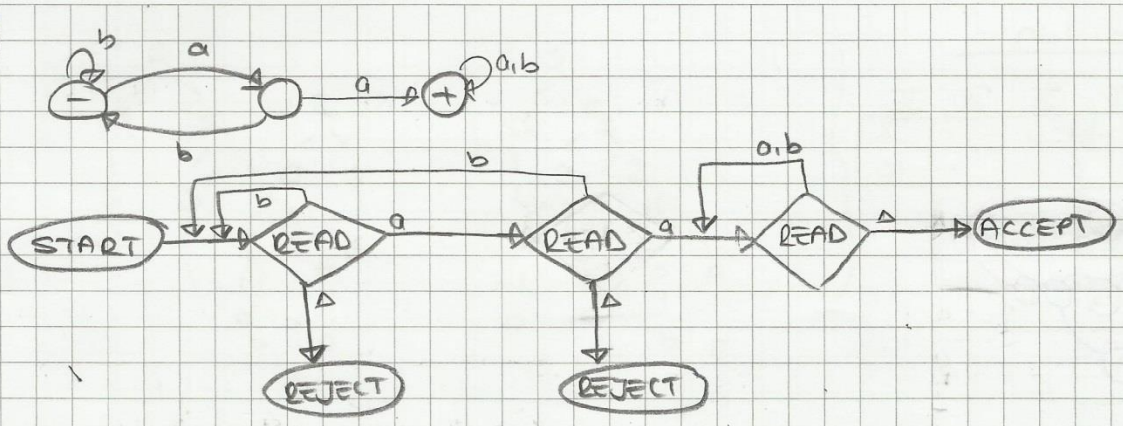
ÖRN



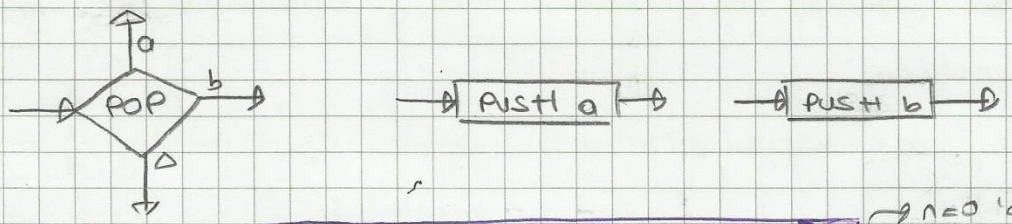
PDA formunu çıkaralım.



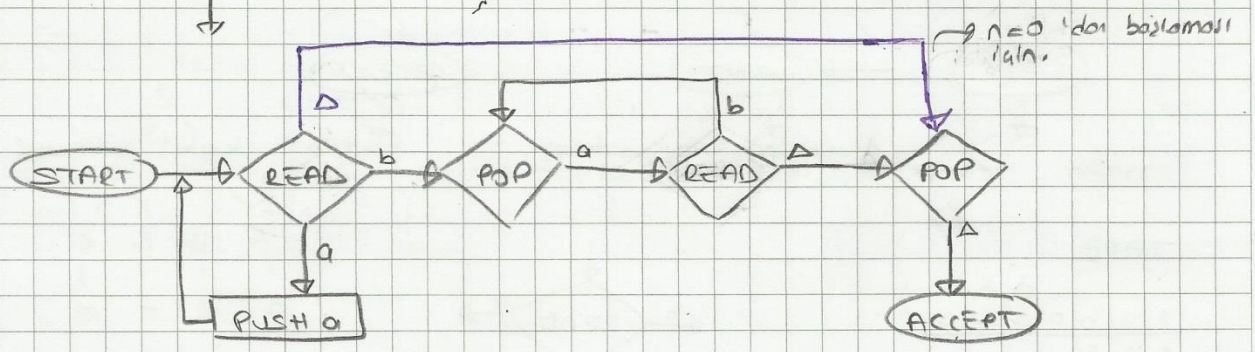
ÖRN



* Kullanacağımız PDA'larda stack ekleyeceğiz.



*

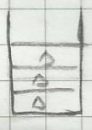


Tepe den önce a'ları okur ve b gelene kadar stacke iter. b gelince, stackten a çeker. Kaç tane a itmişse o kadar b okuyor. Tepede b bittiğinde, stackte a'da kalmaz.

ÖRN 1

a	a	a	b	b	b	Δ	Δ	--

Önce a'lar okunur, yığına itilir. Sonra b'ler okunmaya başlar ve her b de yığından bir a silinir. En son yığında boşluklar olur.

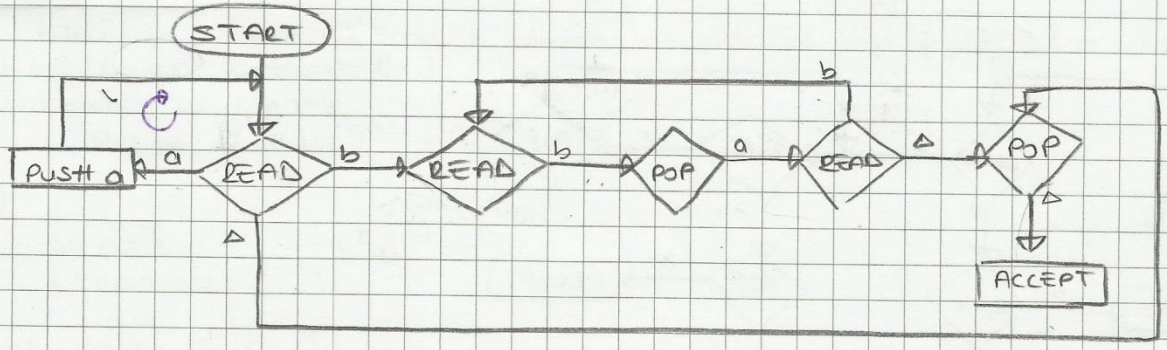


L= language

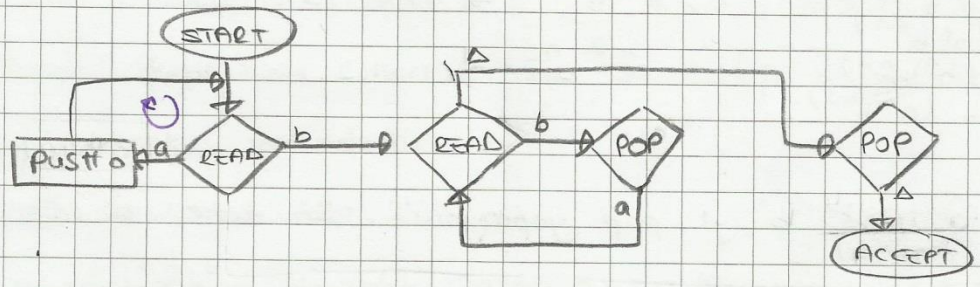
$L = \{ a^n b^n,\ n = 1,2,3,\ldots \}$ → n'in 0'dan başlamasını istersek, READ'i boşluktan başlatabiliriz.
$n = 0,1,2,3\ldots$

ÖRN:

$L = \{ a^n b^{2n},\ n = 0,1,2,3\ldots \}$ için PDA şeklini çiz



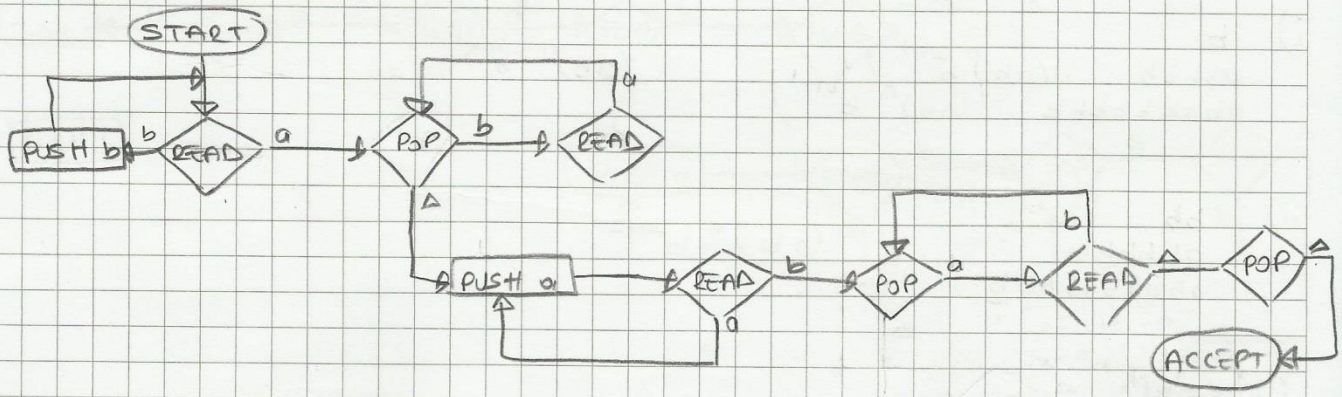
ÖRN:



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

ÖRN:

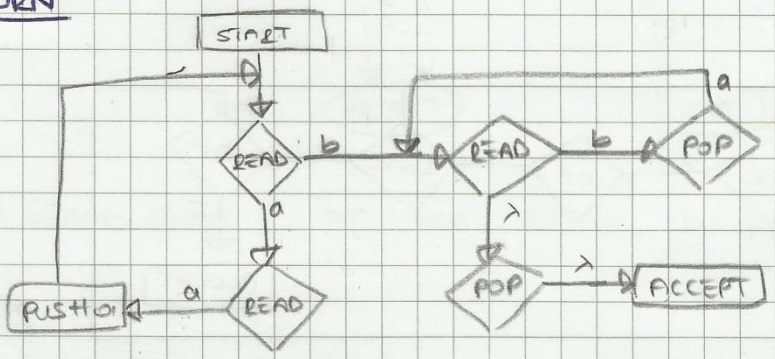
$L = \{ b^i a^j b^k,\ j = i+k,\ i = 1,2,3,\ldots,\ k = 1,2,3\ldots \}$



b'ler geldikçe yığına ittik. Sonra a'lar gelince b leri yığından çektik. Sonra kalan a'ları yığına ittik ve b'leri okuyunca a'ları çektik.

01.04.2013 / 13:00

ÖRN



PDA nın kabul ettiği string L = {... n = ...} şeklinde bulunur.

b
aabb $(a^n b^n)$ $a^{2n} b^{n+1}$ → n = 0, 1, 2, ...
aaaabbb $(a^n b^{n-1})$
aaaaaabbbb $(a^n b^{n-2})$ $a^{2n-2} b^n$ → n = 1, 2, ...
aaaaaaaabbbbbb $(a^n b^{n-3})$

İlk okudğumuz a için b yi pop yapmıyoruz. 2n tane a için n tane a yı push yapıyor ve n tane b yi pop yapıyor. +1 ve ilk okudğumuz b yi pop yapmıyoruz.

ÖRN: 2011 - final

PDA sorusu

i) b
abbbb $(ab)^n b^{2n+1}$ $(ab)^n b^{2n+1}$
ababbbbbb $(ab)^n b^{2n+1}$

ii) bb
abbbbbb $(ab)^n b^{2n+2}$
ab ab bbbbbb
n +1 2n +1

iii) bb
abbbb $(ab)^n b^{n+2}$
abababbbbb
ababababbbbbb

ÖRN) 2012 - 2. sınav

i) ba
bbaa
bbbaaa
bbbb
bbbbaaaa

$b^n a^n$, $n>0$

ii) bba (min 2 tane b olmak zorundadır. Pop yapmak için stacke itmeli dir)
bbaa
bbbbaaa

$b^{n+1} a^n$

iii) baa
bbaaa
bbbaaaaa

$b^n a^{n+1}$

ÖRN) 2012 - final

PDA sorusu : $a^n b^m a^m b^n$, $n>0$, $m>0$

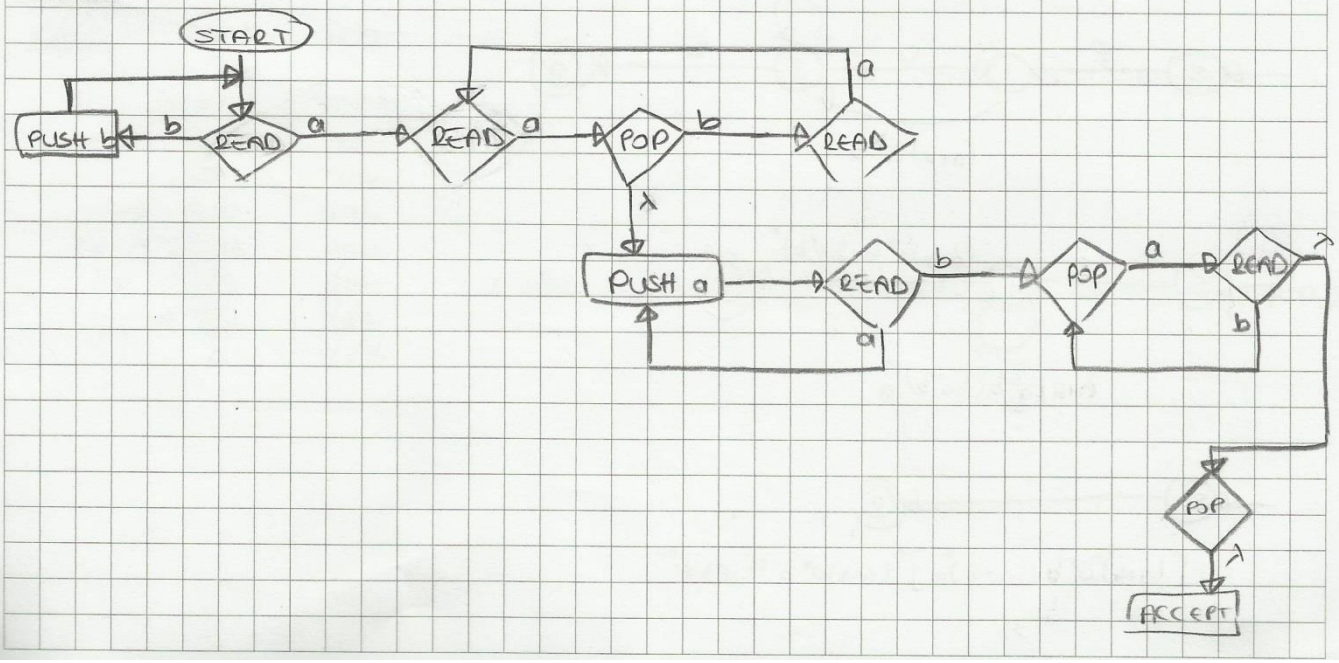
Yığının tepesinde b'ler var. 3. okuduğu a'larla 3. ittiği b'ler aynı olmak zorundadır.

ÖRN) 2012 - 1. vize

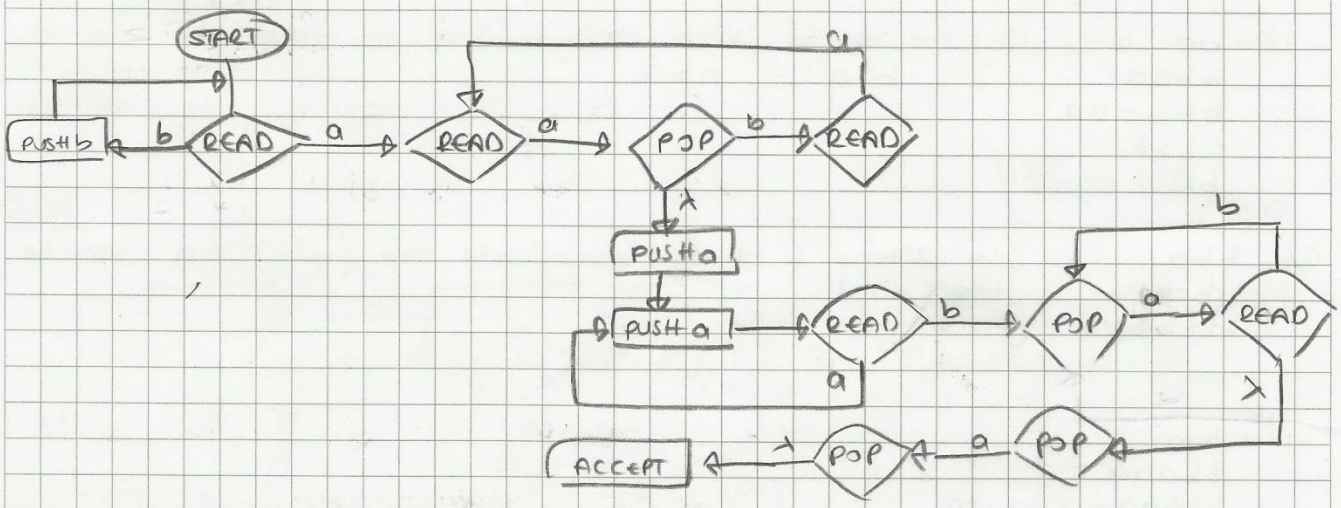
$L = \{ b^i a^j b^k$, $j = 2i+k+1$, $i>0$, $k>0 \}$ dili için PDA çiziniz.

$b^i a^i a^k a b^k$ → min string baaaab olur.

1. Çözüm



II. Çözüm

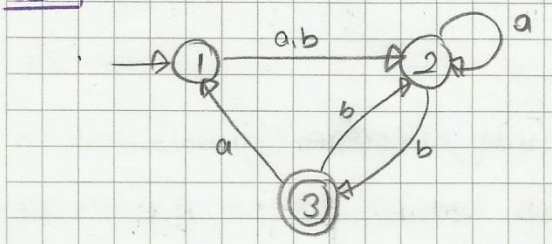


ÖRNEK 2008 - 1. vize

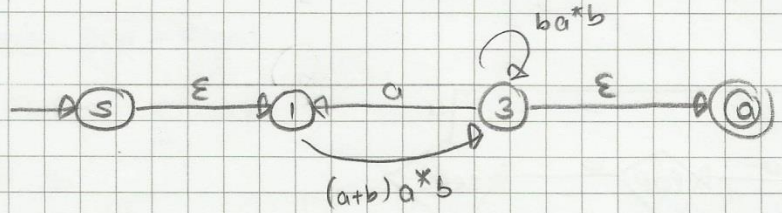
Tek sayıda harften oluşan kelimelerin dili için bir RE yazınız.

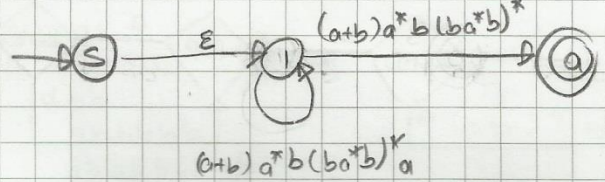
$(a+b)[(a+b)(a+b)]^*$

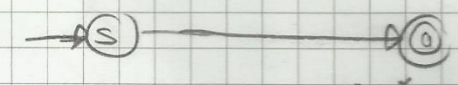
ÖRNEK



RE sini bul.







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

15.04.2013 / 13:00

CFG = PDA

Elimizdeki CFG nin eşdeğer PDA sını nasıl çizeriz?

S → SB | AB Bu gramerin eşdeğer PDA sı nasıl çizilir?

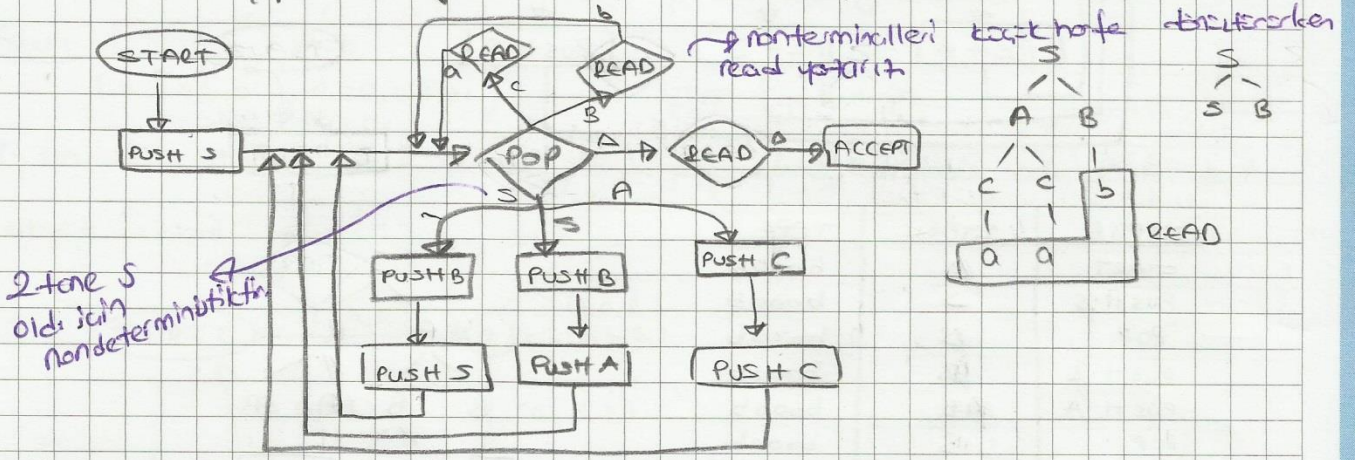
A → CC → AB oluşturabilmek için yığına önce B, sonra A itilir.

C → a

B → b,

PDA çizilirken önemlilik nondeterministik olduğuna dikkat ederiz.

Yani Pop yaparken farklı dallara dağılabiliriz.



Productionun sağ tarafında büyük harf varsa onları Push yaparız.

~~XXXX~~ aab için

	STACK	TAPE
→	Δ	aab
	S	aab
→	AB	aab
pop ←	CCB	aab
pop ←	CB	~~a~~ab
pop ←	B	~~a~~~~a~~b
	Δ	~~a~~~~a~~~~b~~

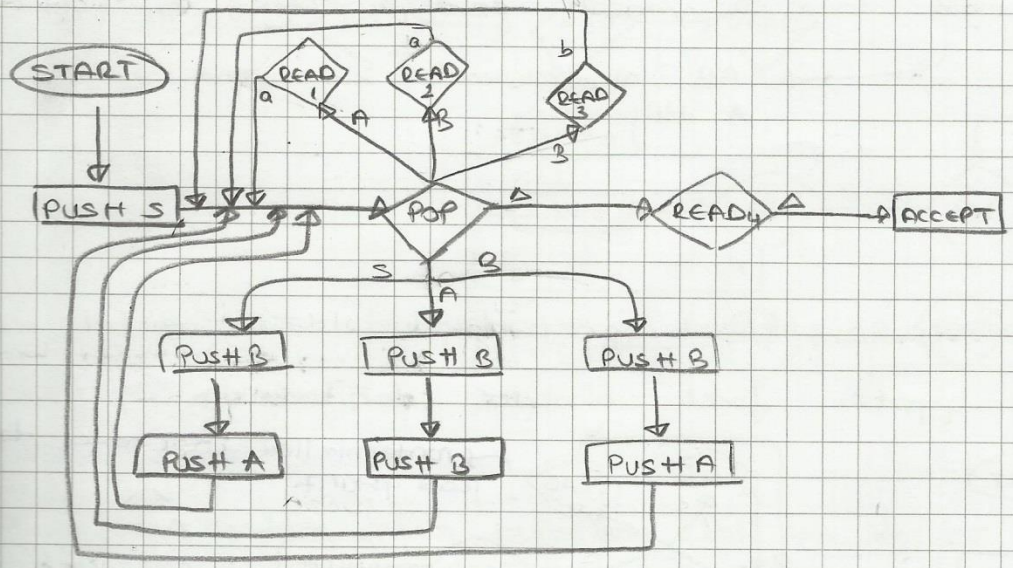
(1 tane a'yı read yaptı, onun için 1 tane C yi pop yaptı)

ÖRN:

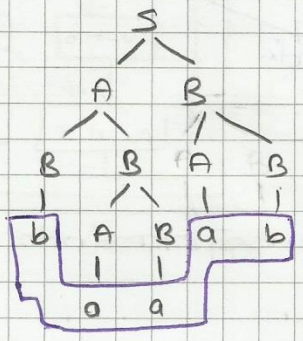
$S \rightarrow AB$

$B \rightarrow AB \mid a \mid b$

$A \rightarrow BB \mid a$

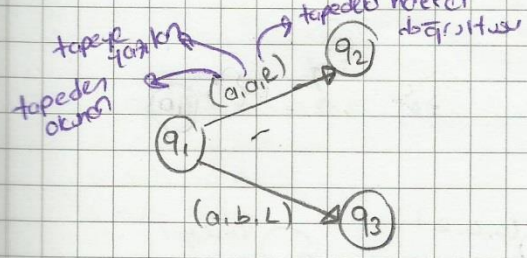


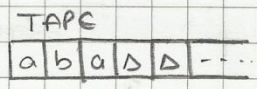
STATE	STACK	TAPE
START	Δ	baaab
PUSH S	S	baaab
POP	Δ	baaab
PUSH B	B	baaab
PUSH A	AB	baaab
POP	B	baaab
PUSH B	BB	baaab
PUSH B	BBB	baaab
POP	BB	baaab
READ3	BB	~~b~~aaab
POP	B	~~b~~aaab
PUSH B	BB	~~b~~aaab
PUSH A	ABB	~~b~~aaab
POP	BB	~~b~~aaab
READ1	BB	~~b~~~~a~~aab
POP	B	~~b~~~~a~~aab
READ2	B	~~b~~~~a~~~~a~~b
POP	Δ	~~b~~~~a~~~~a~~b
PUSH B	B	~~b~~~~a~~~~a~~b
PUSH A	AB	~~b~~~~a~~~~a~~b
POP	B	~~b~~~~a~~~~a~~b
READ1	B	~~b~~~~a~~~~a~~~~b~~
POP	Δ	~~b~~~~a~~~~a~~~~b~~
READ3	Δ	~~b~~~~a~~~~a~~~~b~~
POP	Δ	~~b~~~~a~~~~a~~~~b~~
READ4	Δ	~~b~~~~a~~~~a~~~~b~~
ACCEPT	Δ	~~b~~~~a~~~~a~~~~b~~



TURING MACHINE (TM)

FA ile PDA nın birleştirilmiş hâli gibi düşünebiliriz. Tape kullanması PDA ya benzer özelliğidir. Tape nin üzerine hem yazıp hem de okuyabiliyoruz.



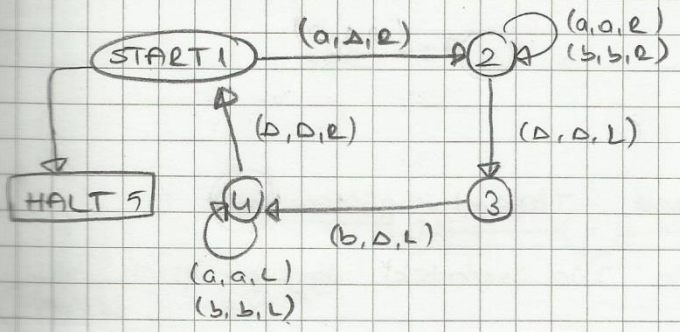


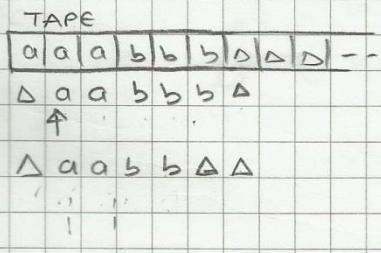
tape de ilk okunan stringten sonra sağa gitmek zorundayız. Sonrakilerde her iki yöne de gidilebilir.

ÖRN:

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

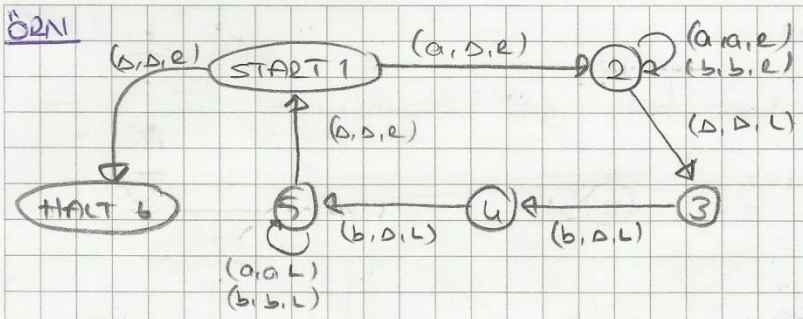
* Tm de stack yok. Ama tape'e yazabildiğimiz için PDA daki stack işlevi görür.





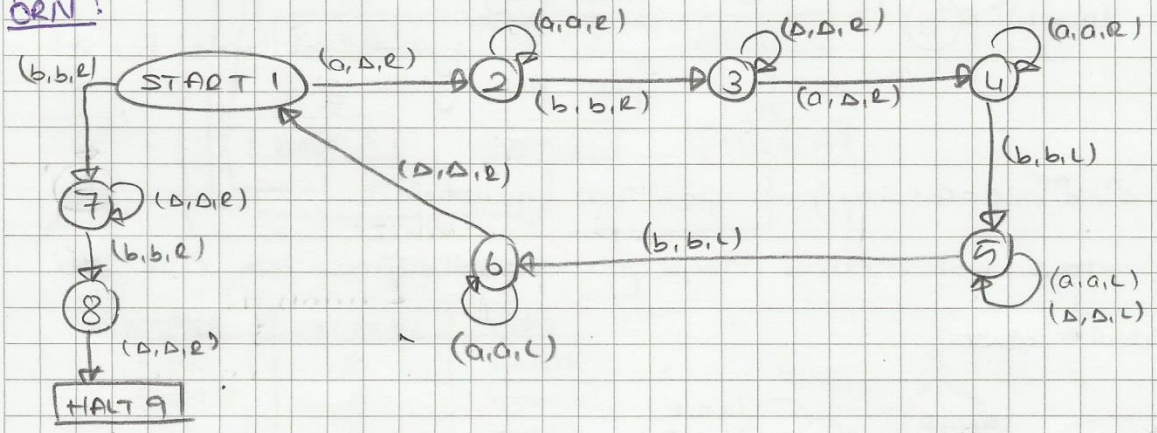
→ hepsi Δ olana kadar devam edilir ve bu aşamaya gelince işlem tamamlanır.

ÖRN



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

ÖRN:



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

aaabaaab stringi üzerinde TM yi gösterelim.

ΔaabaaabΔ → ilk boşluk için 3ün üzerindeki döngüye gerek yok.

ΔaabΔaab

ΔΔabΔΔab

ΔΔΔbΔΔΔb → a'lar bittiği için sağa gitmeyecek artık

ΔΔΔbΔΔΔbΔΔΔ

ÖRN:

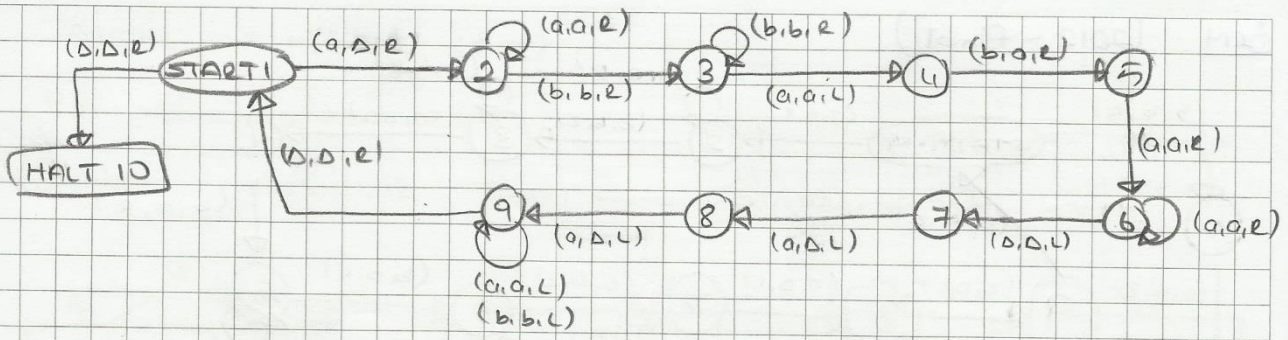
$L = \{a^n b^n a^n, \quad n = 0, 1, 2, \ldots\}$ TM yi göster.

aaabbbaaa

a gelince lefte git ve a yaz.

ΔaabbaaaΔΔΔ

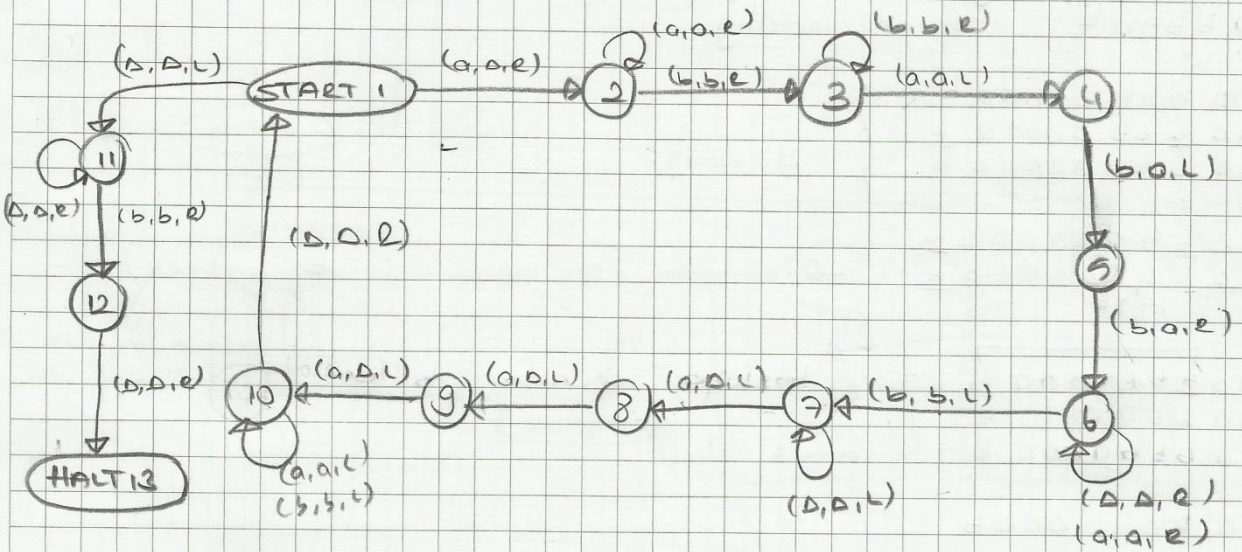
ΔΔabaΔΔΔ



ΔaabbaaΔΔΔ
ΔΔabaΔΔΔ
ΔΔΔaaΔ
ΔΔΔΔΔΔ
↑

22.04.2013

Örn: (2012 - 2. Vize)

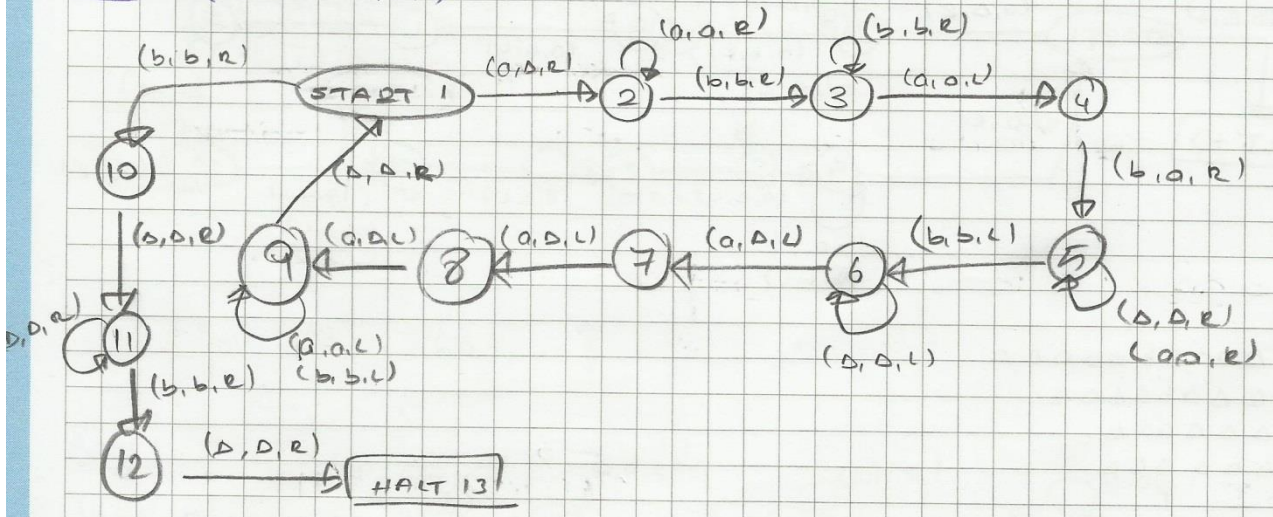


abbab => $a^n b^{2n} a^n b$
ΔaaabΔ
ΔΔΔΔbΔ
↑
ΔΔΔΔbΔ

aabbbaaab
ΔabaaaabΔ
ΔabbaΔΔΔb
ΔΔbaΔ

aabbbbaab
ΔabbbbaaΔ
ΔabbaaaΔΔ
ΔabbaΔΔΔb
ΔΔbbbΔ
ΔΔaaaΔΔΔb

ÖRN (2012 - final)



string ortadan ve sonda b kalır.

→ b' gelir.

a b b a a b → min string

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

} 1. döngü

Δ Δ b b a a a Δ Δ Δ b
Δ Δ b b Δ Δ Δ Δ Δ Δ b → olmaz. (Peş peşe 2 b gelmez)

boşluğa çevrilen b'ler.

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

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

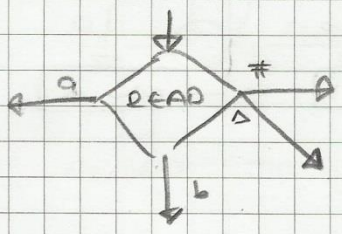
$L = \{ a^n b^{n+1} a^{2n} b \;,\; n = 1, 2, \ldots \}$

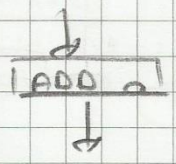
→ tape ye yazılacak olan stringleri birbirinden ayırt etmek için kullanılır.

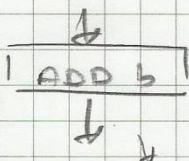
22.04.2013

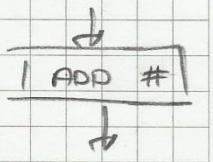
POST MACHINES (PM)

PM'de stack yok. Tape var. Tape'de stringe müdehale edebiliriz. Tape'de yazma yetkisi vardır.

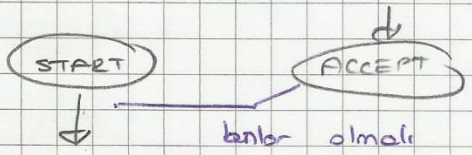




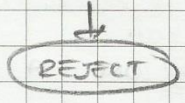




tape yazılırken add yapıyoruz



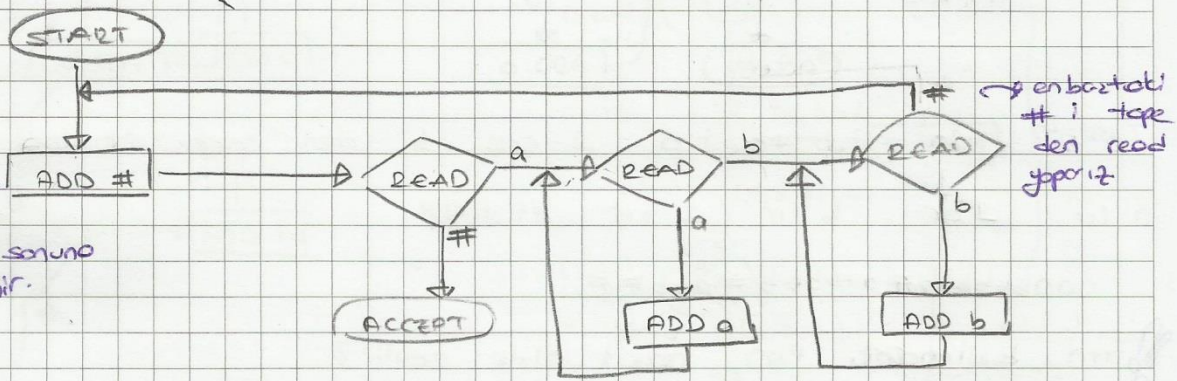
bunlar olmalı



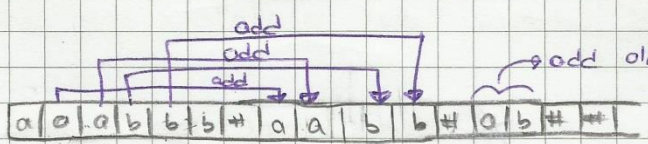
→ bu durum şart değil

Mevcut stringin peşine # eklenebilir.

Örn



2 a okunup 1 tane a add yapılmış.



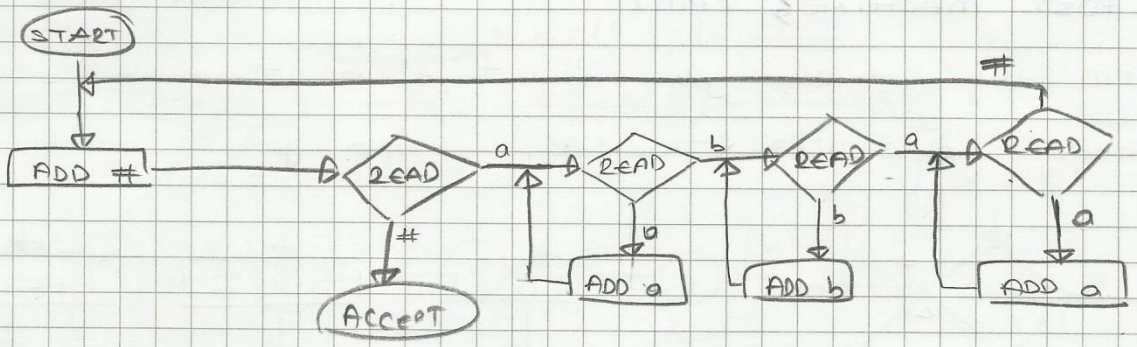
→ add olmuyor.

$\Rightarrow a^n b^n$

3 a dan 2 a ya
3 b den 2 b ye düştü

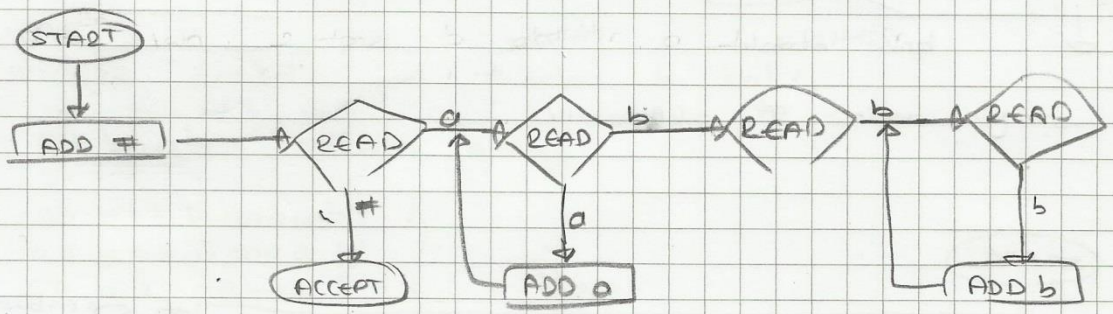
okuma-yazma kafası var. Yazma, stringin sonuna olur her zaman.

ÖRN



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

SIMULATING A PM ON A TM

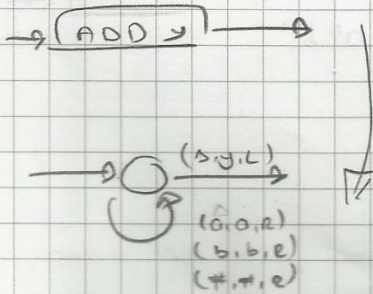


$a^n b^{2n}$ için 1 tane a 2 tane b den sonra ekleltme başlar.. a'lar 1'er ; b'ler 2'şer eksiltilir.

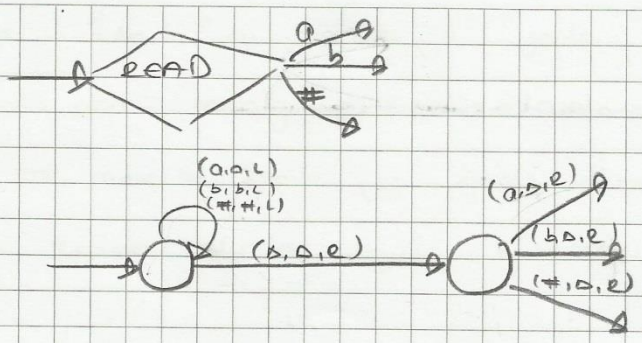
aaabbbbbb#aabbbb#abb##

TM kullanarak, PM nasıl elde edilir?

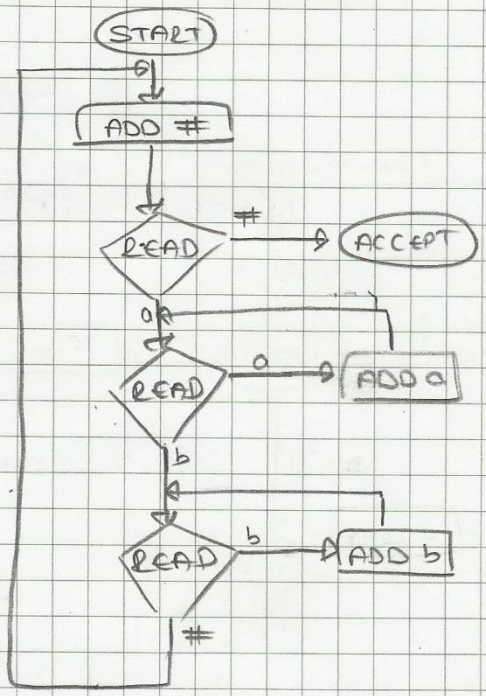
PM'de okuma baştan olur. Yazma stringin sonuna olur. TM'de okuma-yazma kafası 1 tanedir. PM'de okuma TM ile aynı, yazma ise tape nin sonuna değru dur.

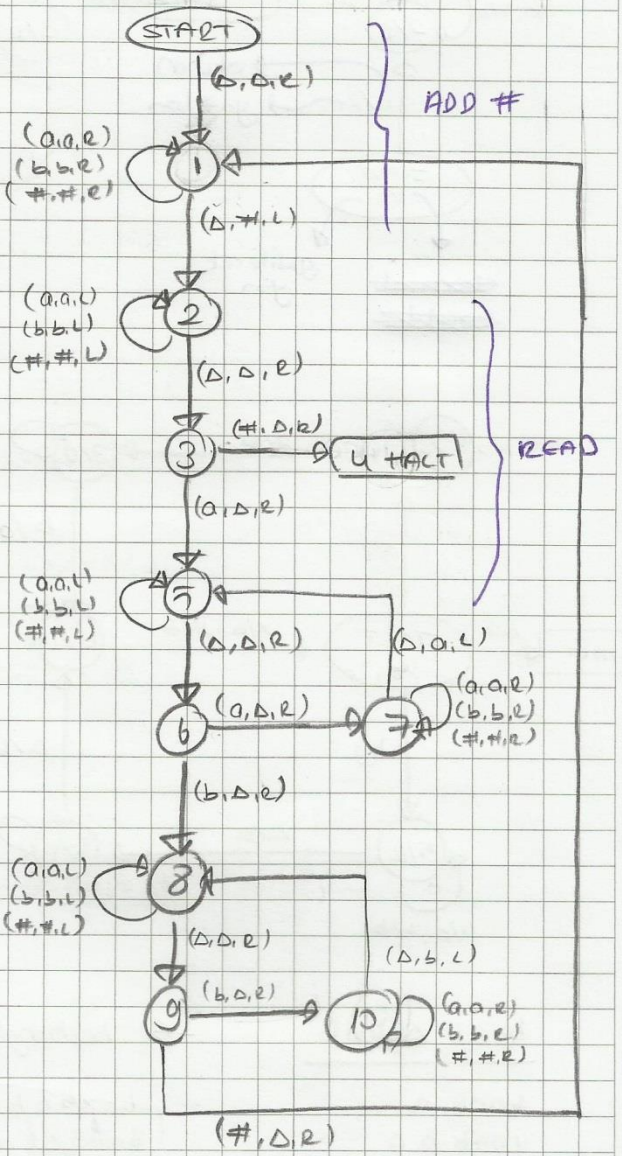


* Read işlemi için tapenin başına gitmek gerekir. Döngülerle gidebilen başında özel karakter olduğunu algılar. Tapenin başında 1 tane Δ karakteri oluşturulur.



Örn eşdeğer TM yi çiz.





Δ	a	a	a	b	b	b	Δ	Δ	Δ	Δ

Add yaparken string sonuna git.

String sonundaki boşluğu a'ya sayır.

ADD a yapılmış oldu.

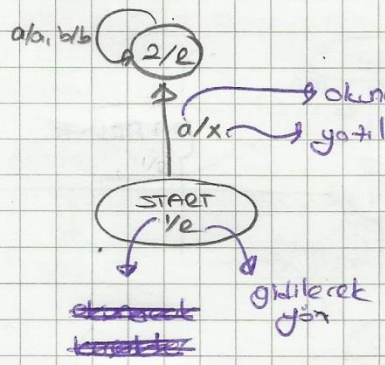
29.04.2013

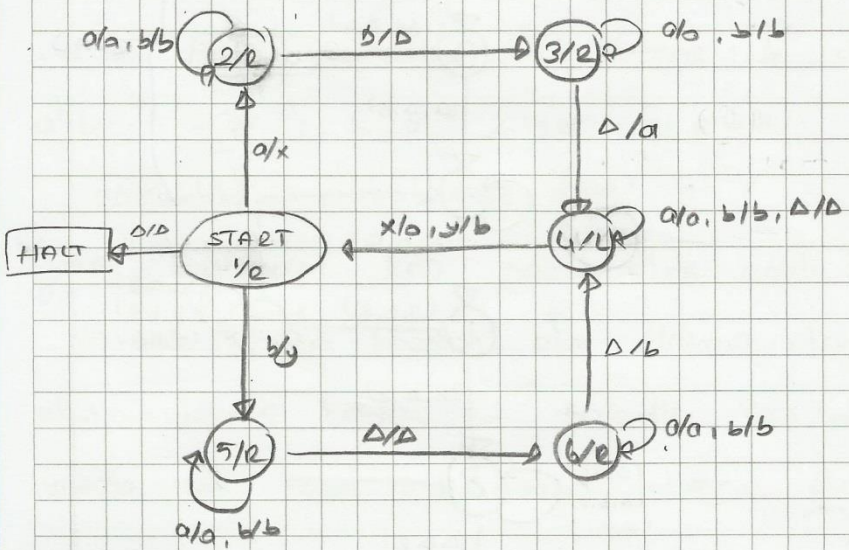
Variations on the TM

3 tane TM varyasyonu olacak.

- MOVE-IN-STATE MACHINE
- STAY-OPTION MACHINE
- KTRACK TM

MOVE IN STATE MACHINE → TM ye benzerdir.





baab Δ baab → herhangi bir string yazıp, Δ ve sonuna kopyası kopyalanır.

baab Δ
yaab Δ a
yaab Δ b
yaab Δ b
baab Δ b

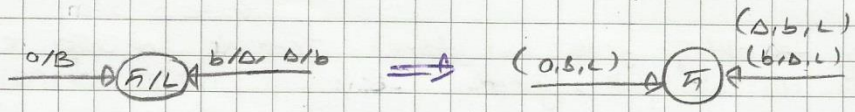
baxb Δ baa
baay Δ baab → bu şekilde stringin kopyası oluşturuldu.

bxaba Δ ba

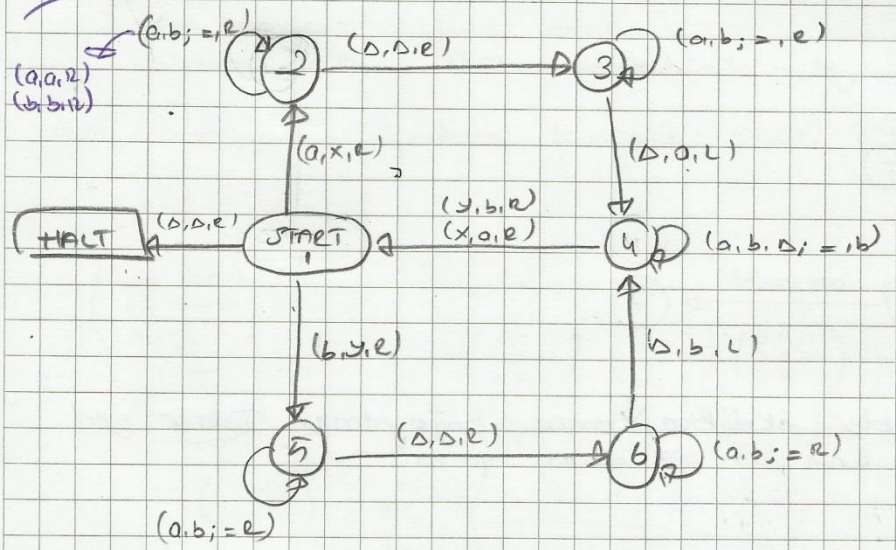
* move in state den TM ye geçiş nasıl olacak?

Moore dan mealy ye geçişe benzerdi. Durum sayısı değişmiyordu.

* TM → move in state ye geçiş, mealy → moore ye geçişe benzer. Burada durum sayısı artabilir.

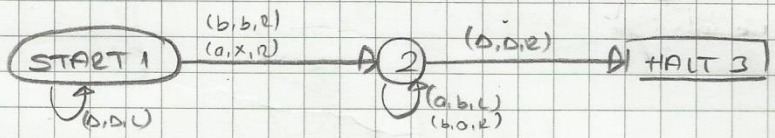


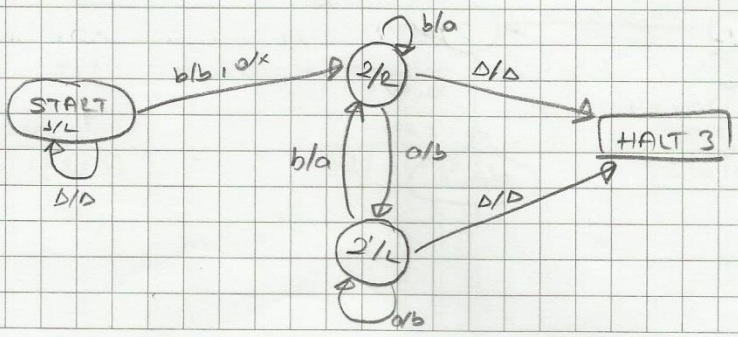
ÖRN



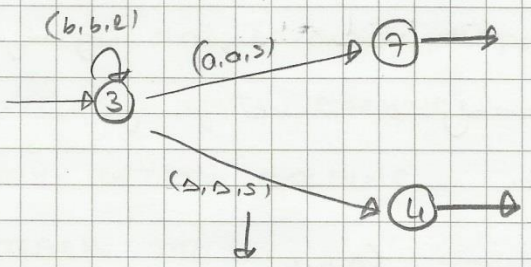
ÖRN

TM → MOVE-IN-STATE





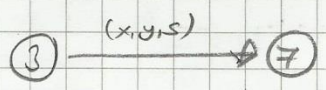
STAY - OPTION MACHINE

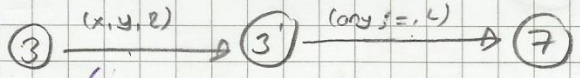


stay → olduğu yerde kal demek. Tape de ne yaptığı ne de sola bir yere gitme.

TM den stay-option a dönüşüm yapılamaz. Çünkü TM de stay option yok konusu olamaz

Stay-option dan TM ye dönüşüm

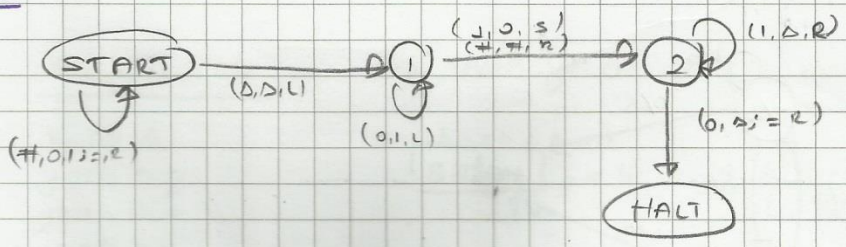


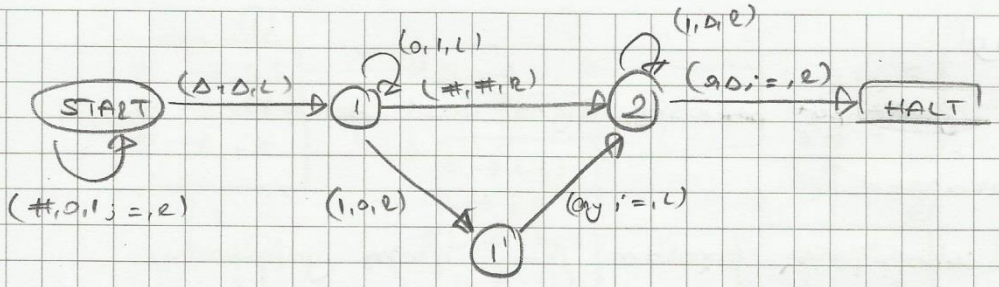


tape de bir kere okuduktan sonra sağa gitmek. Tekrar geri gelmesi lazım. Bunun için ara durum yazmış.

stay-option

Ex





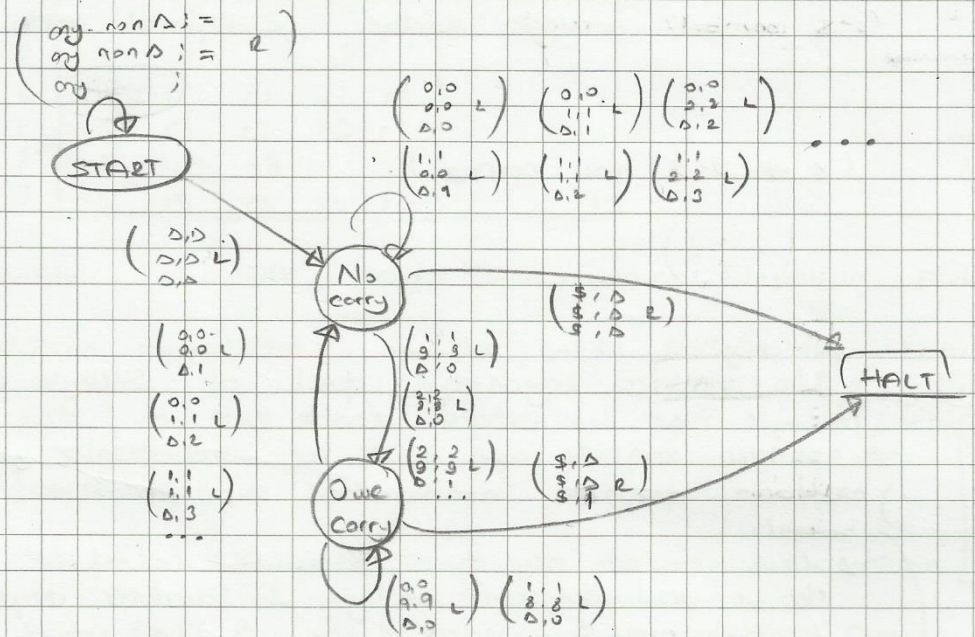
K-TRACK TM

(X, Y, 3R) → δ x oku, y yaz, 3 hücre sağa git

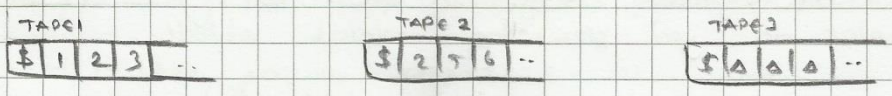
k - track

↳ tape sayısını ifade eder.
4 tape için LITM yazılır.

Her tape için kafanın konumu sabit. Bütün tapeler için kafanın yönü aynıdır.



İlk iki satır toplanacak sayılar son satır toplamdır



06.05.2013

LR(0) Parsing

LR(k): Left to right input processing, Right most derivations and k symbols ed lookahead.

LR(0) → o an bulunduğunuz karakter için işlem yapıyoruz.

LR(1) → bir sonraki karakter için işlem yapıyoruz.

Buradaki grameri CNF formunda olma zorunluluğu yok.

Herhangi bir stringin gramer kabul etme durumu en son oluşturduğumuz parsing tablosu üzerinden değerlendirilir. Tablo üzerinden accept durumuna geldiğimizde gramer, string kabul eder.

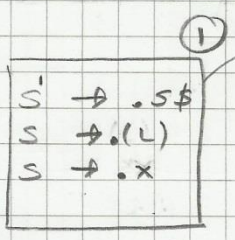
0 S' → S $ ($ sembolü stringin sonuna geldiğimizi anlamak için)

(kural) r1 S → (L)

r2 S → x ('x → 0-9 arası rakam)

r3 L → S

r4 L → L,S



→ $'a ulaşınca kaydırmaya gerek yok. Stringin sonudur.

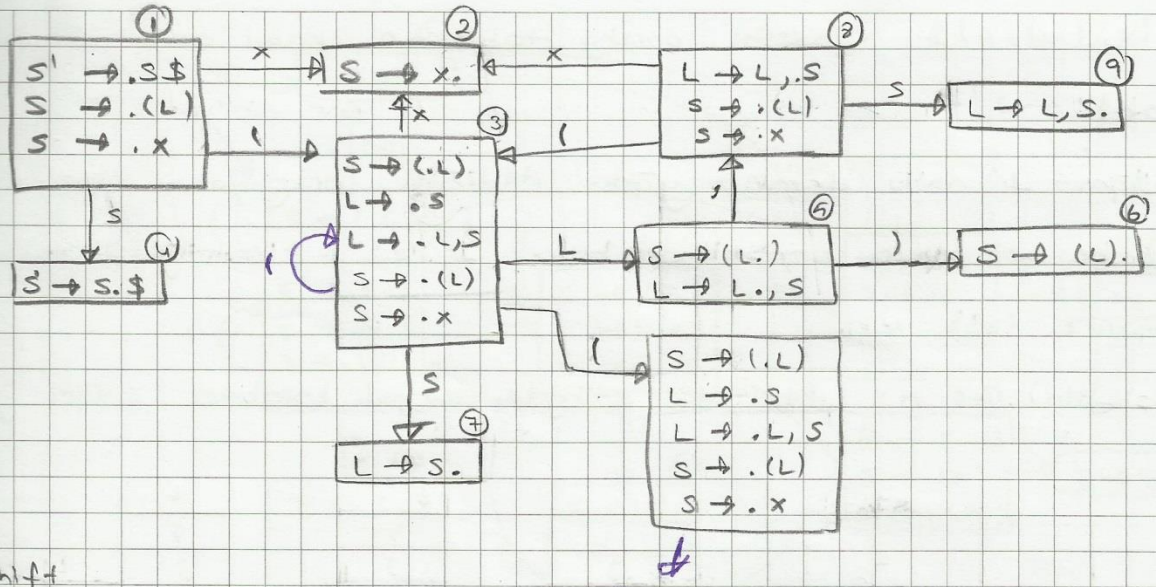
→ noktanın sağında olan durum için productionlar yazılır.

) noktanın sağında nonterminal olup olmadığına bakılır.

→ Her bir production için noktayı 1 karakter sağa kaydırıyoruz.

noktanın sağında nonterminal olan her yeni durumda onlar da yeni production olarak eklenir.

Noktayı ilerletemeyerek duruma kadar devam edilir. Daha sonra parsing tablosu oluşur. Durum sayısı, parsing tablosundaki satır sayısını verir.



s - shift
g - goto
r - reduce

③ no'lu durum ile gitmiş olduğu için kendi üzerinden geri dönmüştür.

	(	)	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		

nokta sembolü, productionın sonuna ulaşmışsa reduce yaparız.

Yığının tepesindeki durma bakılır. Stringin ilk karakterine de bakılır.

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.

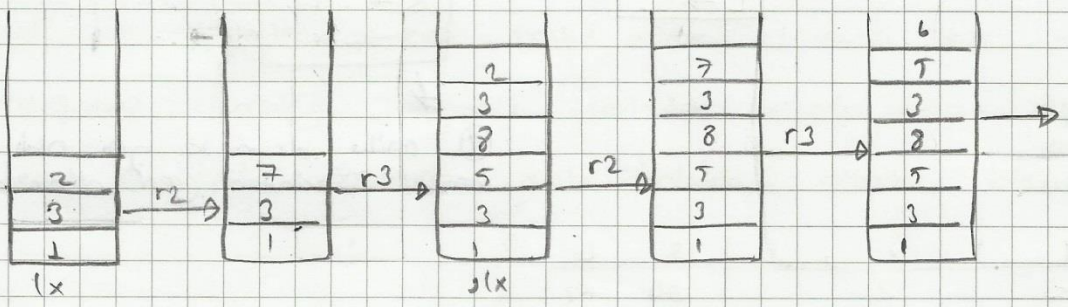
Accept : Stop parsing, report success

Error : " " " failure.

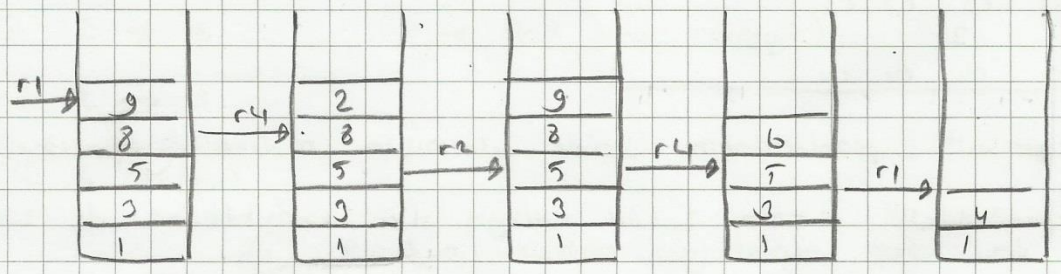
(x, (x), x) $ ifadesini, parsing tablosunun kabul edip etmediğine bak!

Yığına 1 nolu durumun yığına itildiğini varsayıyoruz. Yığının en tepesine ve input sembolüne bak. 1 ile (kesiştiği kutuya bakıyoruz. (s3)

shift(n) → n'i yığına it, stringte de 1 karakter sağa git



2 nolu durumda r2 var. rule k'daki okun sağındaki sembol kadar yığından çek.



2011 - final ÖRN

0 S' → S $

1 S → (E)

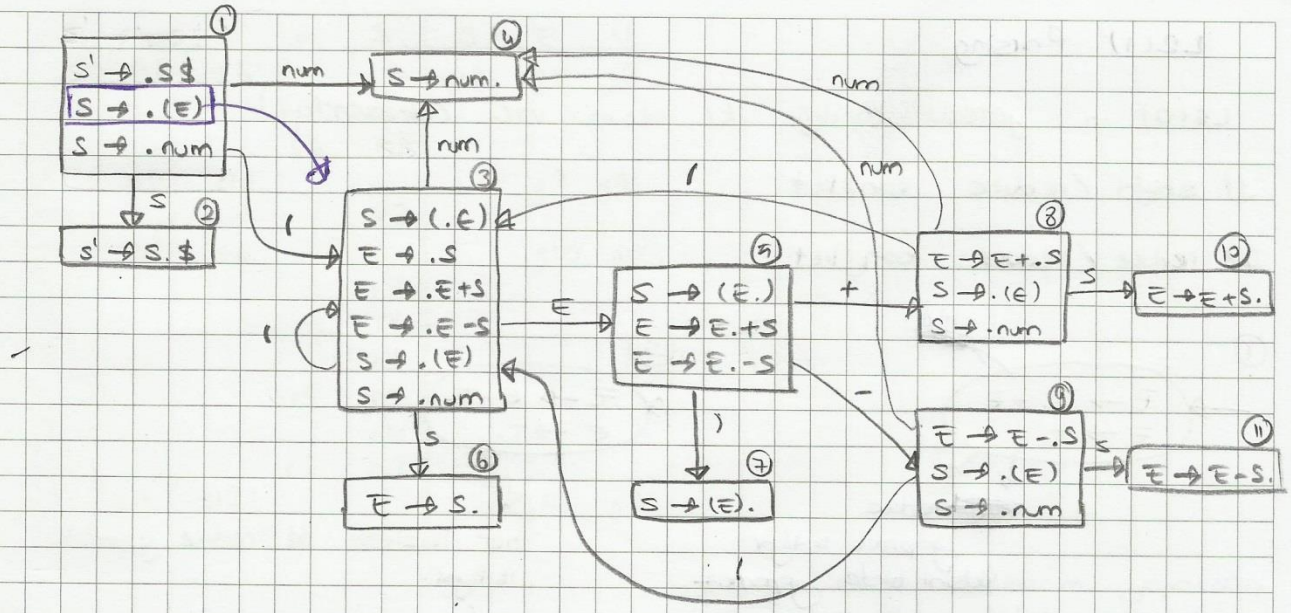
2 S → num

3 E → S

4 E → E + S

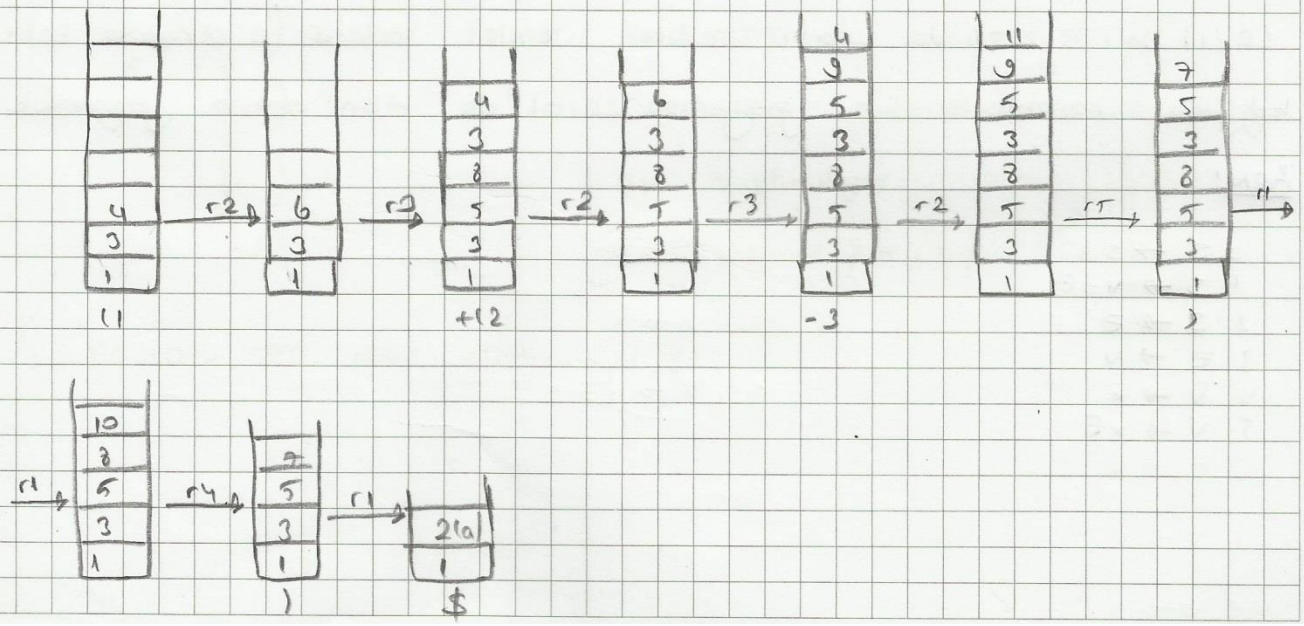
5 E → E - S

Yandaki CFG'den LR(0) yöntemine göre durum diyagramını çiziniz. Parsing tablosunu oluşturun. (1 + (2 - 3)) $ ifadesini gösterin.



Parsing tablosu

	(	)	+	-	num	$	S	E
1	s3				s4		s2	
2						a		
3	s3				s4		s6	s5
4	r2	r2	r2	r2	r2	r2	r2	r2
5		s7	s8	s9				
6	r3	r3	r3	r3	r3	r3	r3	r3
7	r1	r1	r1	r1	r1	r1	r1	r1
8	s3				s4		s10	
9	s3				s4		s11	
10	r4	r4	r4	r4	r4	r4	r4	r4
11	r5	r5	r5	r5	r5	r5	r5	r5



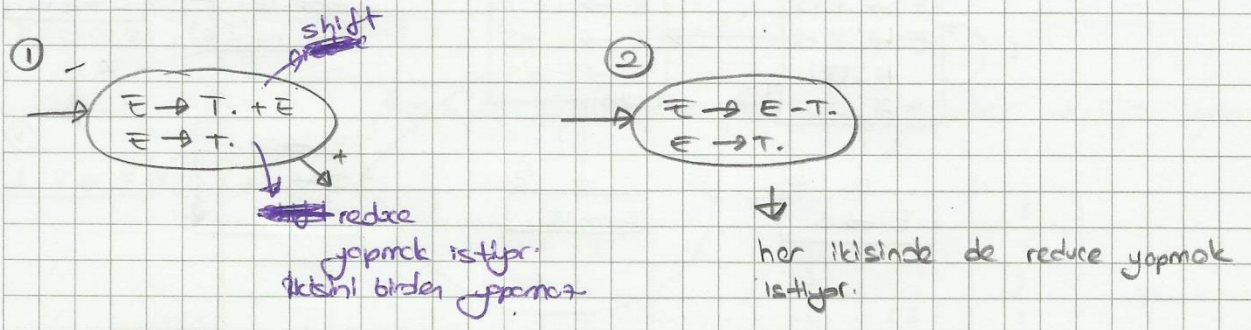
Xmoden compiler imple. in Java (s-65) | 13.05.2013

LR(1) Parsing

LR(0) in yetersiz kaldığı iki durum var. (dezavantaj)

1) shift / reduce conflict

2) reduce / reduce conflict



LR(1) bu durumlara çözüm getiriyor

LR(k) → left to right recursion
right most derivation
k look ahead symbols

LR(1) de stringten okuyacağımız bir sonraki karakteri tahmin ediyoruz.

S → V. = E	$
E → V.	$

⇒ shift / reduce conflict durumu var. Her biri ayrı olarak değerlendirilir ve yazılır.

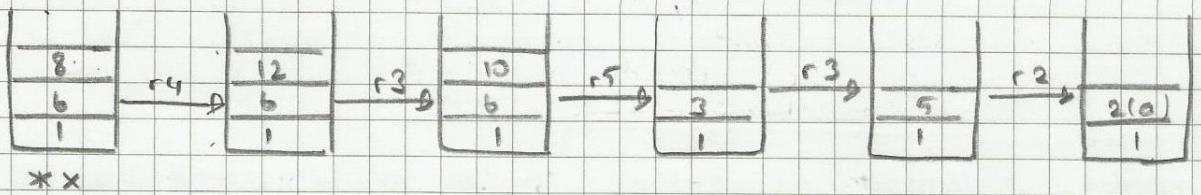
LR(1) in LR(0) dan farkı reduce işlemleri sadece o durumda bekledigimiz karakterler için yazıyoruz. LR(0)'da tüm satıra yazıyordu.

ÖRN:

0 S' → S $
1 S → V = E
2 S → E
3 E → V
4 V → x
5 V → x E

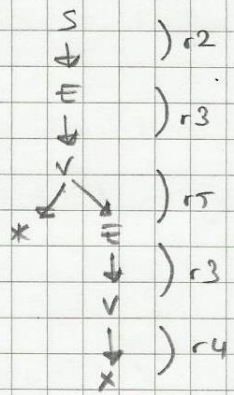
	x	*	=	$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

* x $ için;



* x

Geriye doğru giderek parse ağacını çizmek mümkün.



Lookahead sembolünü oluşturmak için 2 tane küme oluşturmamız gerekiyor.

FIRST SET ve FOLLOW SET.

FIRST SET her bir nonterminal için ayrı ayrı oluşturuluyor.

FIRST (S) = { *, x }

! FOLLOW SET nasıl oluşturuluyor ?