

THEORY OF COMPUTATION

Furkan Erıcı - 2017555017

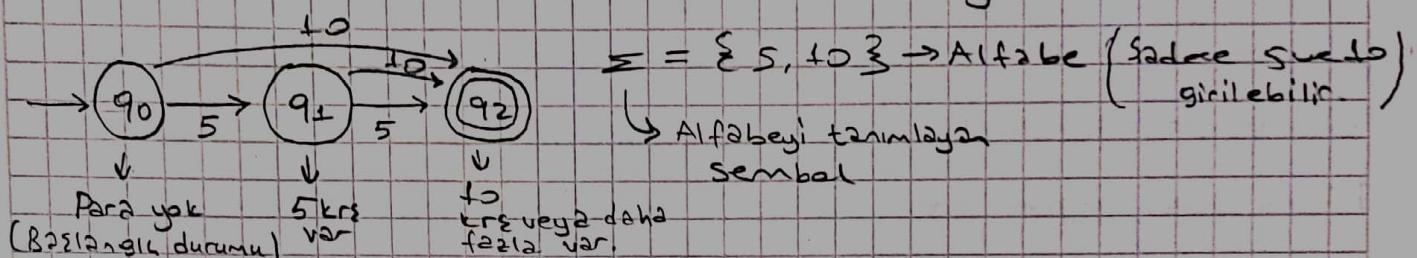
CHAPTER 2

- Finite Automata and Regular Languages -

Bir örnekle başlayalım;

Örnek: Sadece 5 ve 10 kurus kabul eden bir turnike (toll gate) olduğunu ve en az 10 kurus atılınca turnikeden döndüğü varsayılm.

O zaman bizim otomatımız şu şekilde bir yol izlemeli dir:

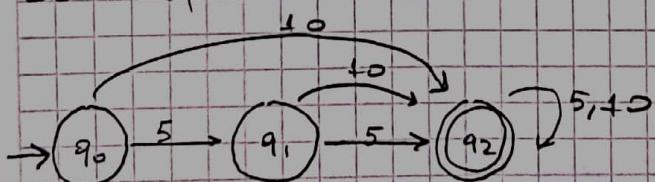


* q₀, q₁ ve q₂ durumları (state) belirtir.

* q₂ durumunun iki iki yuvarlaktan oluşmasının nedeni son durum (final state) olmasıdır. Bunun anlamı işlem bu durumda (q₂) sonlanırsa sistem bunu kabul eder.

→ Burada söyle bir sorun var, 10 kurustan daha fazla para girişini olursa sistem nasıl dövranacak?

Sistemi kullanıcının yapabileceği her kombinasyonda hazır hale getirmeliyiz. Bu durumda 10 kurusta fazla olduğunda da kapının açık kalması için tasarım şu şekilde olmalıdır.



* Burada aslında 5 ve 5 atılıncaya bize verilen alfabeye göre yazdığımız bir kelimedir. Biz koşulu sağlan kelimele rin kabul edilmesi için (10 krs ve üzeri) tasarım yapıyoruz.

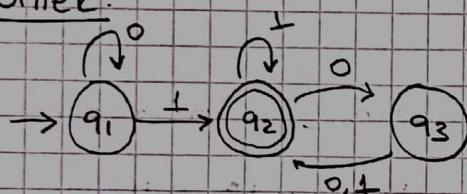
Deterministic Finite Automata (DFA):

Kararlı Sonlu Otomatlar her koşul olsa da her zaman bir durumda bulunurlar ve oluşturulan otomatlardır.

Bir C programı yazarken ilk atamayı yapmayı unuttığumuzda veya hata alırız. Bunun nedeni yazdığımız kodun deterministik olmamasıdır. Yani tutarsızlık olması ve sistemin ne yapacağını bilmemesi durumudur.

! Tüm algoritmalar deterministik olmalıdır. (belirsizlik olmaz, net)

Örnek:



$\Sigma = \{0, 1\}$ → Alfabeiniz ove 2'den oluşuyor

$\delta(q_1, 0) = q_1$ } δ bir fonksiyandır.

$\delta(q_1, +) = q_2$ } ikinci parametresi durum

: ikincisi akıbetir.

→ q_1 deiken 0 girilirse q_1 'e gider.
Bu yüzden sonuc q_1 oldu

→ 1101 kelimesi otomata tarafından kabul edilir mi? (Tüslere sırayla basılıyor)

• q_1 başlangıç durumdayken '+' e basılırsa q_2 'ye geçer.

• q_2 durumunda '+' girilirse q_2 'ye geçer.

• q_2 durumunda '0' girilirse q_3 'e geçer.

• q_3 durumunda '+' girilirse q_2 'ye geçer.

↳ Son durumda q_2 'de (kabul durumu) kaldığı için kabul edilir. ✓

* Bu otomata ne iş yapar?

Kelime en az bir tane 1 içerecek. Kelime 1 ile bitebilir.

En sağdaki '+' den sonra çift sayıda 0 ile bitebilir.

↳ Bu koşulları sağlayan her kelimeyi otomata kabul eder.

- * Bir sonlu otomayı tanımlamak için 5 parametre gereklidir. Bunlar $M = (Q, \Sigma, \delta, q, F)$ parametrelidir. Burada,
- M , otomatayı temsil ediyor.
 - 1. • Q , Durumları (State) (Yukarılıklar)
 - 2. • Σ , Alfabeyi (kullanılan işareler) $Q \xrightarrow{\text{Kullanılan işaretler}} \Sigma \rightarrow Q$
 - 3. • δ , Geçiş fonksiyonu (transition function) (Önceki sayfada ver)
 - 4. • q , Q 'nın bir elemanıdır. Başlangıç durumunu ifade eder.
 - 5. • F , Q 'nın alt kumesidir. Bitiş durumunu ifade eder. (Bitiş durumu birden fazla olabilir)

* Diagram çizmek yerine bu geçişleri tablo ile gösterebiliriz. Fakat belirtilmese, tablo metodunda, başlangıç ve bitiş durumunu bileyemeyiz.

Örnek: Bir önceki örnek için, $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, başlangıç = q_1 ve bitiş q_2 olsun.

Σ			
	0	1	
Q	q_1	q_1	q_2
	q_2	q_3	q_2
	q_3	q_2	q_2

Buradan da verilen kelimenin doğru olup olmadığı kontrolü sağlanabiliyor.

$L(M) \rightarrow$ otomatın kabul ettiği durumları ifade eder. Bu durumda bu otomatanın tanımı için:

$\rightarrow L(M) = \{w : w \text{ en az bir tane } 1 \text{ içermeli ve çift sayıda } 0 \text{ içermeli}\}$ denebilir.

* Problemi çözüzerken geçişlerden (δ) başlayın.
($\Sigma \rightarrow$ genelde soruda verilir)

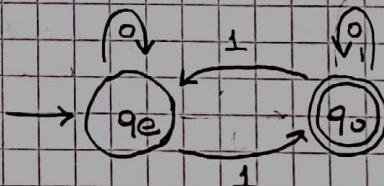
Algoritması: $w = w_1 w_2 w_3 \dots w_n$, elementları Σ ’den seçilmiş bir string olsun. Durumlar $r_0, r_1, r_2, \dots, r_n$ diye sıralanmış olsun;
 $r_0 = q_0$; Başlangıç durumunu ala.
for $i=0$ to r_{n-1} : ; Harf sayısına kadar dön
 $r_{i+1} = \delta(r_i, w_{i+1})$; Gelen harfe göre geçişleri yap.
end for ; Döngüden çıkış
if $r_n \in F$ then M accepts w . ; Eğer r durumu bitiş durumysa onyla.
if $r_n \notin F$ then M rejects w . ; Eğer r durumu bitiş değilse reddet.

Örnek-1: $A = L(M)$ olmak üzere
 (kabul x olsulu)

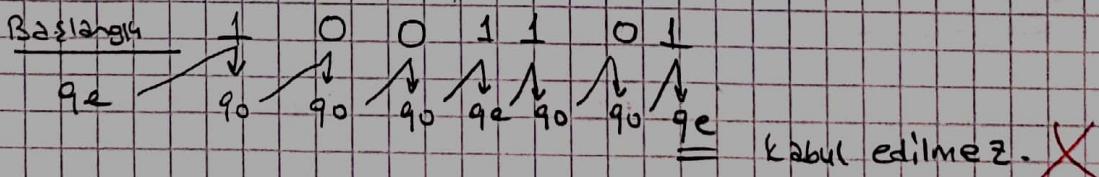
$A = \{w : w$ bir binary stringtir ve w içindeki 1'lerin sayısı tekdir}

$$Q = \{q_e, q_o\}, q_e \xrightarrow{\text{(even)}} q_o, q_o \xrightarrow{\text{(odd)}} q_e, F = q_o$$

	0	1
q_e	q_e	q_o
q_o	q_o	q_e



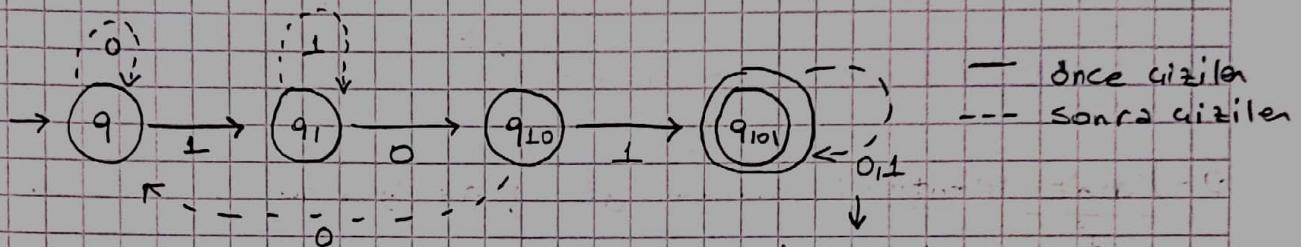
Örneğin; 1001101 kabul edilir mi?



Örnek-2:

$A = \{ w : w \text{ bir binary string} \text{dir}, w \text{ içerisinde } 101 \text{ alt-stringini içermeli}\}$

* Bu soru şeklinde direkt diagram çizilmek istenirse, öncelikle bize en istedigi durumu çizip daha sonra diğer kesimleri çizmeliyiz.



	0	1
q	q	q ₁
q ₁	q ₁₀	q ₁
q ₁₀	q	q ₁₀₁
q ₁₀₁	q ₁₀₁	q ₁₀₁

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

$$q = q$$

$$F = q_{101}$$

$$Q = \{q, q_1, q_{10}, q_{101}\}$$

f = diagram ve tabloda görünüyor. ($\delta(q, 1) = q_1$)

$$M = \text{automat}$$

L(M) = kabul durumu. (Bu örnekteki A)

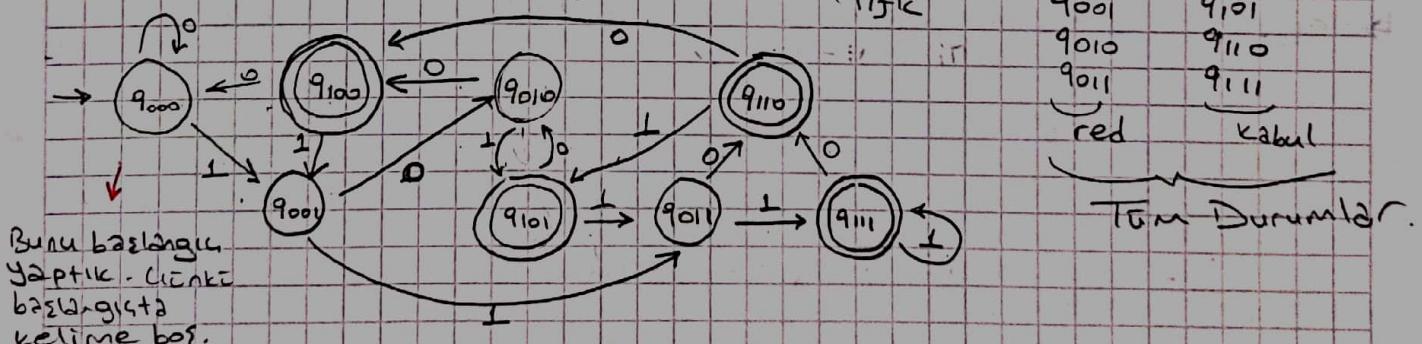
* Zaten bitirme koşuluna geldiyse k string 101 içeriyor demektir. Bünden sonra ne girildiginin önemi yok.

Örnek-3:

$A = \{w \in \{0, 1\}^*: w \text{ 'nin sağdan üçüncü pozisyonu } 1 \text{ olmalı}\}$

→ Bu soruda sağdaki 3 harfi tutmamız gerekiyor. Bu yüzden

$2^3 = 8$ tane duruma ihtiyacımız var. (q_{ijk})



q ₀₀₀	q ₁₀₀
q ₀₀₁	q ₁₀₁
q ₀₁₀	q ₁₁₀
q ₀₁₁	q ₁₁₁

red kabul Tüm Durumlar.

Regular Operations :

Regular Languages kapsamında kullanılan işlemleri temsil ediyor.

3 temel işlem vardır: (A ve B aynı alfabeye sahip diller)

1-) union: Kelimeleri sırayetli olarak kullanma
(Birleşme)

$$A \cup B = \{w : w \in A \text{ or } w \in B\}$$

2-) concatenation: Kelimeyi yan yana yazma.
(Birlestirme)

$$AB = \{ww' : w \in A \text{ and } w' \in B\}$$

Härflerin klavyeden basınca yan yana gelmesi gibi.

Alfabenin härflerinden birini seçme işi union oluyor.

Aynı durumda kalma işi de star oluyor. (\oplus örneğin),
 1^* diye gösterilir.

3-) star: Tetrarlamayı temsil ediyor.

$$A^* = \{u_1 u_2 \dots u_k : k \geq 0 \text{ and } u_i \in A \text{ for all } i = 1, 2, 3, \dots, k\}$$

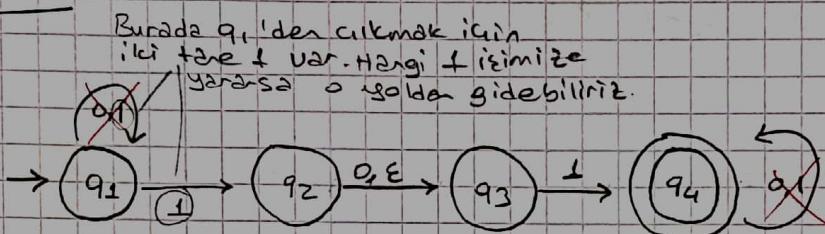
↳ Regular Expression yaparken daha detaylı anlatılacak.

Nondeterministic Finite Automata (NFA):

insan gözüyle tasarımlı temsil eder. DFA kadar detaylı değildir.

DFA'ya göre daha kolay tasarlanabilir.

Örnek - L:



- ilgilenmediğimiz kısımlar

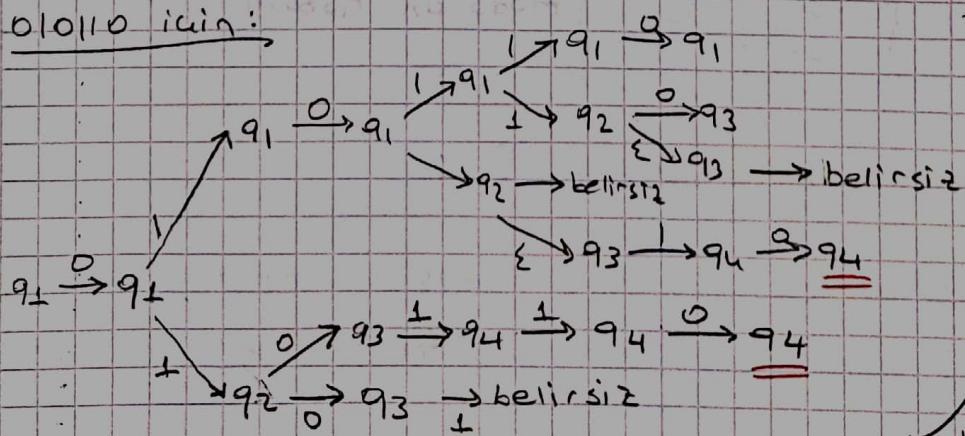
Asıl yerler
ilgilediğimiz kısımda

DFA ile arasındaki 3 temel fark:

- Bir durumda birden fazla aynı outgoing (çikış) olması. (Örneğin q_1 durumunda iki tane λ var)
- ϵ durumu (NULL). (Bunun anlamı, q_2 den herhangi bir tara basılmadan q_3 'e geçilebileceğidir)
- Bir durum için tüm outgoing'ların yazılması. (q_3 durumunda sadece 1 outgoing'i var)

→ içerisinde $\lambda\lambda\lambda$ substringini veya $\epsilon\epsilon\epsilon$ substringini bulunduran her kelimeyi kabul eder.
(ϵ de dolayı)

010110 için:

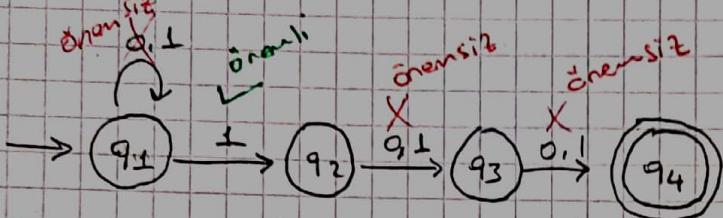


Burası deterministik bir sistemdir.
NFA'yi biz görmüze bakıp anlayabiliyoruz.
Makinenin anlaması için deterministik olmalı.

→ Bu NFA için 010110 kelimesinin tüm kombinasyonları araştırıldı. En az bir tane Final durumu (q_4) varsa kelime kabul edilir demektir. (İki tane var burada)

Örnek-2: A bir dil olsun

$$A = \{ w \in \{0,1\}^*: w \text{ 'nin sondan üçüncü harfi } 1 \text{ olsun} \}$$

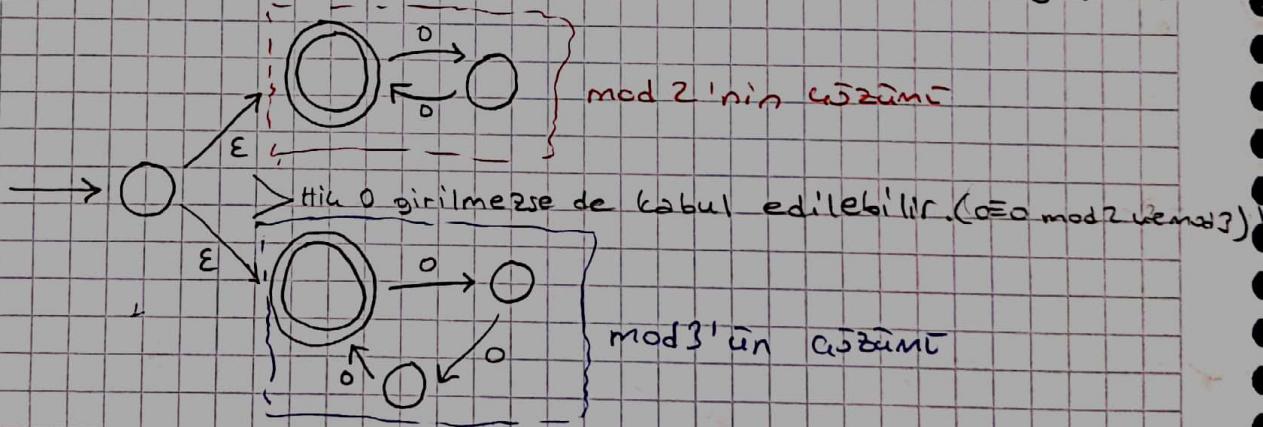


DFA'da 8 durum vardı.

A için NFA diğramı böyle çizilebilir.

* Zaten problem kelimenin en sağ ile ilgili olduğinden q_4 için 0 ve 1 geçisi yapmıyoruz. Yaparsak sonraki harf değişeceğinden asümüle ulaşamayız. Bunu yapmadığımız için geçerli bir durumdayken (örneğin, 100101) sonunda bir harf daha gelirse kelime bastan araştırılır. Aslında bu her tespit için geçerlidir. (bastan araştırma olayı)

Örnek-3: NFA alfabetesi $\{0\}$ olarak tanımlanır. (Yalnızca 0 var)



$$A = \{ 0^k : k \equiv 0 \pmod{2} \text{ or } k \equiv 0 \pmod{3} \}$$

* union işlemini göstermek için hazırlanan bir örnektir.

mod 2 ve mod 3 işlemlerinin ϵ yardımı ile birleştirilmesi gösterilmiştir.

** NFA göstergimleri bir DFA'yi kapsar. (Her DFA bir NFA'dır)



— II. Hafta SON —

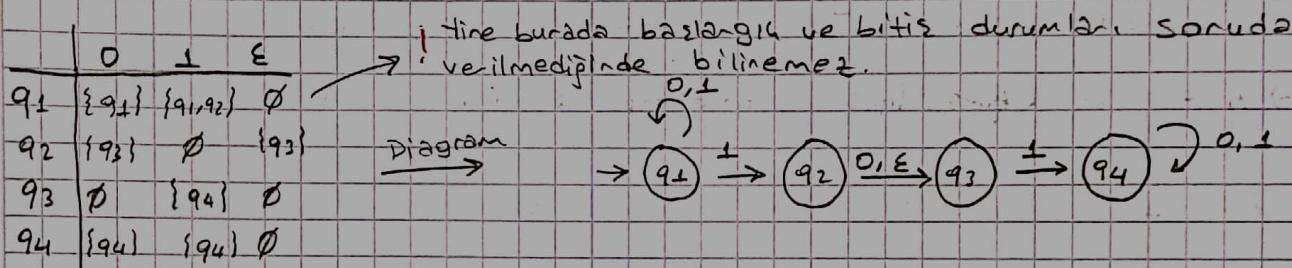
! NFA'ları tanımlamak için de, DFA'da olduğu gibi, 5 parametre gerekir.

$$M = (Q, \Sigma, \delta, q_0, F)$$

Durumlar Alfabe Geçiş Başlangıç Bitiş
 Fonksiyonu Durumu Durumu

* NFA'ların geçiş tablolarını bir örnekle anlatacak olursak.

$$Q = \{q_1, q_2, q_3, q_4\}, \Sigma = \{0, 1\}, q_0 = q_1, F = \{q_4\} \text{ olsun}$$



Burada DFA tablosunda farklı olarak ϵ geçisine yer verilmiştir.

- DFA VE NFA DÖNÜŞÜMLERİ -

NFA'dan DFA'ya dönüşüm yaparken, NFA'nın tuple'lerini $(Q, \Sigma, \delta, q_0, F)$ kullanarak DFA'nın tuple'lerini elde etmeye çalışıyoruz. Bunun için 4 adım var:

1-) $Q' = P(Q)$, $|Q'| = 2^{|Q|}$

→ DFA'nın state sayısı, NFA'nın state sayısının 2^n 'i kadardır

→ ($n = \text{NFA'nın state sayısı}$)

2-) $q'_0 = q_0$ (Başlangıç stateleri aynı)

3-) F' 'yi bulmak için NFA'nın final statelerine bakılır. Final statelerini içeren tüm alt kümeleler F' 'ye dahildir.

$$F' = \{ R \in Q' : R \cap F \neq \emptyset \}$$

4-) Geçişler,

$$\delta'(r, a) = \bigcup_{\substack{\text{DFA} \\ r \in R}} \delta(r, a)$$

NFA
Örnek üzerinde
daha iyi anlaşılır.

→ ϵ geçişlerin dizişinden fazla da state eklememiz gerekebilir.

$$q'_0 = C_\epsilon(q_0) = C_\epsilon(\{q_0\})$$

! Tüm bu işlemlerden sonra DFA'nın state sayısı çok fazla olabilir. Eğer gerekirse state varsa indirmeye yapılarak bu durum düzeltilebilir.

Örnek: Geçiş tablosu verilen NFA'yi DFA'ya çevir.

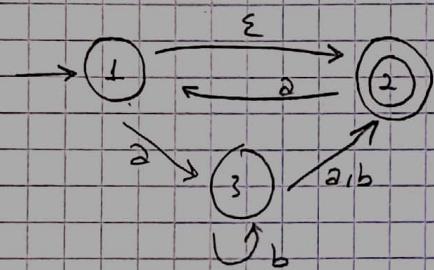
$$\text{NFA } N = (\mathbb{Q}, \Sigma, \delta, q_1, F) \Rightarrow \mathbb{Q} = \{1, 2, 3\}, \Sigma = \{a, b\}, q_1 = 1, F = \{2\}$$

	a	b	ϵ
1	{1}	\emptyset	{2}
2	{1}	\emptyset	\emptyset
3	{2}	{2}	\emptyset

Burada başlangıç durumundan (1) 2'ye keyfi geçiş olduğunu program 2'de başlayabilir. Bu yüzden.

$$C_{\epsilon}(q_1) = \{1, 2\} \Rightarrow q'_1 = \{1, 2\}$$

NFA geçiş diagramı:



ϵ 1'siz geçiş tablosu:

	a	b	ϵ den dolayı
{1}	{3}	\emptyset	
{2}	{1, 2}	\emptyset	
{3}	{2}	{2, 3}	
\emptyset	\emptyset	\emptyset	
{1, 2}	{1, 2, 3}	\emptyset	
{1, 3}	{2, 3}	{2, 3}	
{2, 3}	{1, 2}	{2, 3}	
{1, 2, 3}	{1, 2, 3}	{1, 2, 3}	

Bunları bulmak için union işlemi yapılır.
örn {1, 2} gibi bulmak için
1 ve 2 union (birleşim)
islemihe sokular.

DFA üretmek için:

- $Q' = P(\mathbb{Q}) \Rightarrow 2^3 = 8$ state olacak.

$$Q' = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

$$q'_1 = C_{\epsilon}(\{q_1\}) = C_{\epsilon}(\{1\}) = \{1, 2\}$$

Başlangıç durumunda ϵ varsa bu durum direkt alınır.

$$F' = \{\{2\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$$

icinde 2 geçen her eleman.

Geçişler için

$$\delta'(r, a) = \bigcup_{r \in R} C_{\epsilon}(\delta(r, a)) \quad \text{formulu uygulanırsa.}$$

$$\delta' =$$

$$\delta'(\emptyset, a) = \emptyset$$

$$\delta'(\{1\}, a) = \{3\}$$

$$\delta'(\{2\}, a) = \{1, 2\}$$

$$\delta'(\{3\}, a) = \{2\}$$

$$\delta'(\{1, 2\}, a) = \{1, 2, 3\}$$

$$\delta'(\{1, 3\}, a) = \{2, 3\}$$

$$\delta'(\{2, 3\}, a) = \{1, 2\}$$

$$\delta'(\{1, 2, 3\}, a) = \{1, 2, 3\}$$

$$\delta'(\emptyset, b) = \emptyset$$

$$\delta'(\{1\}, b) = \emptyset$$

$$\delta'(\{2\}, b) = \emptyset$$

$$\delta'(\{3\}, b) = \{2, 3\}$$

$$\delta'(\{1, 2\}, b) = \emptyset$$

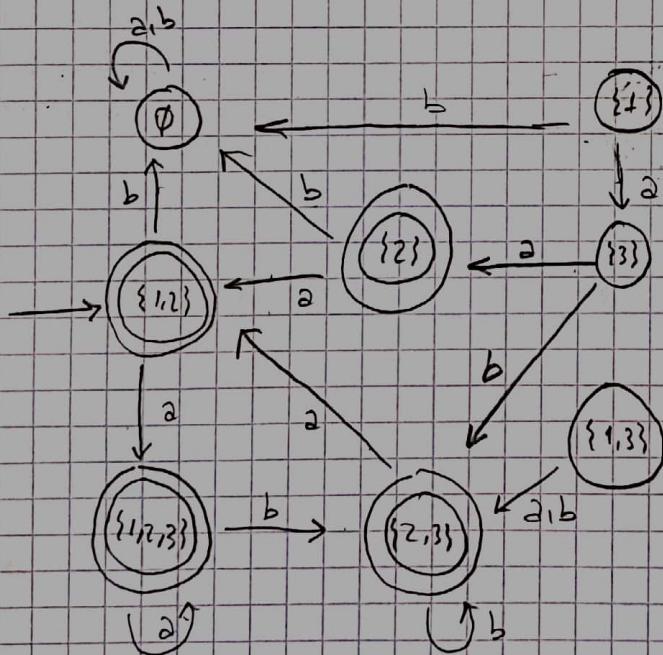
$$\delta'(\{1, 3\}, b) = \{2, 3\}$$

$$\delta'(\{2, 3\}, b) = \{2, 3\}$$

$$\delta'(\{1, 2, 3\}, b) = \{2, 3\}$$

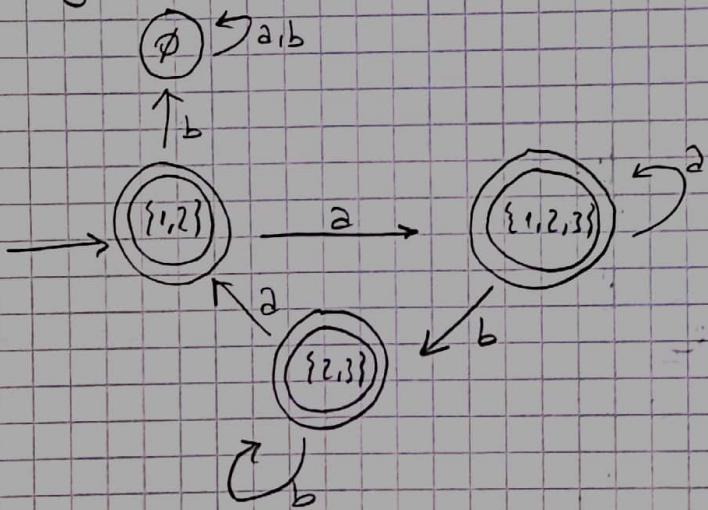
a

b



! Görüldüğü gibi bazı durumlar gereksizdir. Buna, \emptyset indirgene işlemde tabii tutmamız gerekir.

- İndirgemedi ulaşılamayan state silinir.
- ↳ $\{1\}$ ve $\{1,3\}$ durumlarına gelis yolu yoktur.
- ↳ $\{3\}$ durumuna 1 gelis vardır o da $\{1\}$ durumundadır.
 $\{1\}$ durumu silinirse $\{3\}$ durumuna da ulaşılabilir. Zaten
 $\{1\}$ durumuna ulaşamadığı için siliniyordu
- ↳ $\{2\}$ durumuna da yalnızca $\{3\}$ te gelis var.
- * Basit bir göz teraması ile de hangi durumlara ulaşamadığını
bulunabilir.
- * İndirgeme işleminden sonra DFA'nın son hali:



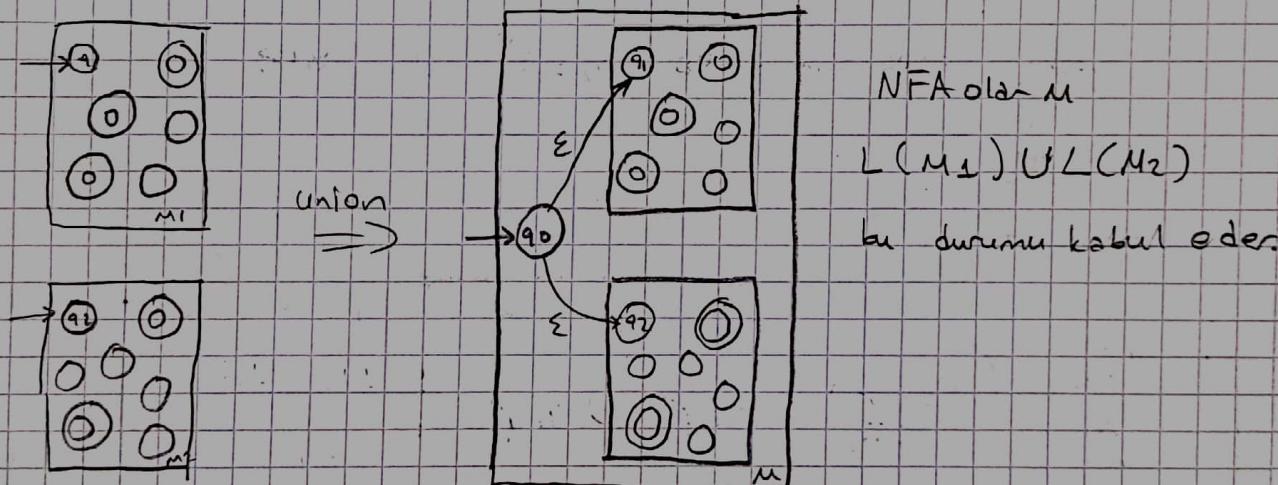
→ NFA , DFA'ya dönüştürülüyor !

Düzenli İşlemler: (Closure under the regular operations)

- A_1 ve A_2 aynı alfabeeye sahip diller olsun. Bu durumda $A_1 \cup A_2$ (union) işlemi sonucu kapalıdır.

Bir işlemin kapalı olması demek o kümeye tenezine uygulandığında sonucun yine o kümeden olması demektir. Örneğin; doğal sayılar kümelerinde toplama işlemi kapalıdır. Çünkü doğal sayılar toplandığında sonuç, yine bir doğal sayıdır. Ama bölme işlemi açıktır. Çünkü doğal sayılarında bölme işlemi sonucu, rasyonel bir sayıdır.

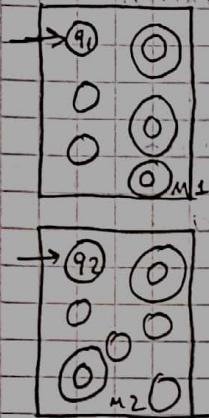
Örnek Gösterim:



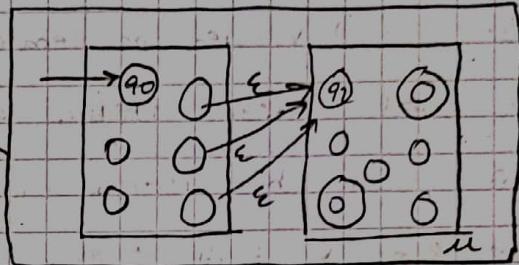
- A_1 ve A_2 aynı alfabeeye sahip düzenli diller olsun. Bu durumda $A_1 A_2$ (concatenation) işlemi sonucu kapalıdır.

İki problemi ardışık sıralanda yazma işlemidir. Sıra önemlidir.

Örnek gösterim:



concatenation
⇒

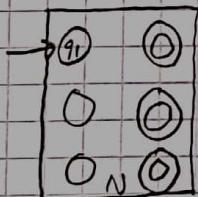


NFA olan M $L(M_1)L(M_2)$ durumunu kabul eder.

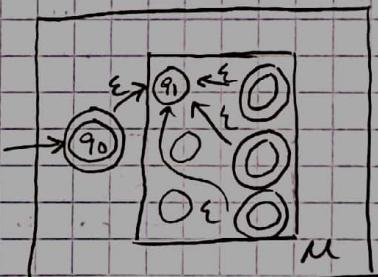
! M_1 'in final durumları reject yapılmış. M_2 'nin start durumuna ϵ ile bağlanıyor.

- A düzenli dili y idz (star) işlemi için kapalıdır. Yani A^* 'da aynı alfabeyle sahip düzenli bir dildir.

Örnek gösterim:



(star)
⇒



NFA olan M -

$(L(N))^*$

durumunu kabul eder.

- Kabul durumlarından başlangıç durumuna ϵ ile bağlantı, yapılrak rekürsif bir yapı elde edilir.

- * Tersini alma (complement) ve kesim (intersection) işlemlerine de kapalıdır. düzenli diller.

- DÜZENLİ İFADELER (REGULAR EXPRESSIONS) (RegEx) —

Düzenli işlemler kısmında gördüğümüz işlemlerde (union, concatenation, star) verilen problemleri görüşeceğiz.

Örneğin bir metin girme bölümüne girilen bir string'in sınırlanılması (yani her isteneni yazmayı engellemek) gibi bir işlemdede RegEx kullanılıyor.

Örnek: $(0 \cup 1) 0^*$

Böyle bir ifade su anlama gelir.

1. 0 veya 1 ile başlayacak.

2. ikinci sembol 0 olacak.

3. Sonda istenilen sayıda 1 olacak. (hic olmayaabılır)

{00, 001, 0011, --, 10, 101, 10111 -- }

Örnek: { $w : w$ iki tane 0 içermeli}

$1^* 0 1^* 0 1^*$

Örnek: { $w : w$ en az iki tane 0 içermeli}

$(0 \cup 1)^* 0 (0 \cup 1)^* 0 (0 \cup 1)^*$

Örnek: { $w : 1011$ alt stringini içermeli}

$(0 \cup 1)^* 1 0 1 1 (0 \cup 1)^*$

Örnek: $\{w: w \text{ nin uzunluğu çift olmalıdır}\}$

$$((0\cup 1)(0\cup 1))^*$$

Örnek: $\{w: w \text{ nin uzunluğu tek olmalı}\}$

$$(0\cup 1)((0\cup 1)(0\cup 1))^*$$

$\underbrace{\quad}_{\text{gift i garanti eder}}$

Örnek: $\{1001, 0\}$

b_1 veya b_4

100101.

Örnek: $\{w: w \text{ nin ilk ve son sembolü aynı olsun}\}$

$$0(0\cup 1)^*0\cup 1(0\cup 1)^*1\cup 0\cup 1$$

! Kelime tek harfli olursa kabul edilir. Çünkü baştaki ve sondaki aynı.

Cözüm 1: 0

Cözüm 2: 1

Cözüm 3: 0 0

Cözüm 4: 1 1

$$\underline{0(0\cup 1)^*0} \cup \underline{1(0\cup 1)^*1} \cup \underline{0\cup 1}$$

1 2 3 4

Tüm çözümler union işleni ile birbirine bağlıdır.

HAFTA III. SON

= REGEX VE REGULAR EXPRESSION EŞİTLİKLERİ =

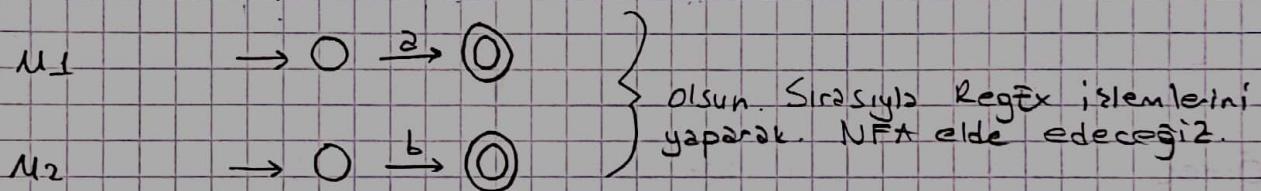
NFA, DFA ve Regex düzenli dili arayaları birbirine denktir ve bunlar birbirine dönüştürülebilir. Bunun için 3 yöntem vardır. + taneini geçen hafta gördük. ($NFA \rightarrow DFA$).

* Her düzenli dili açıklayan bir düzenli ifade (Regex) vardır.

RegEx'in Düzenli Dile Çevrilmesi: ($RegEx \rightarrow NFA$)

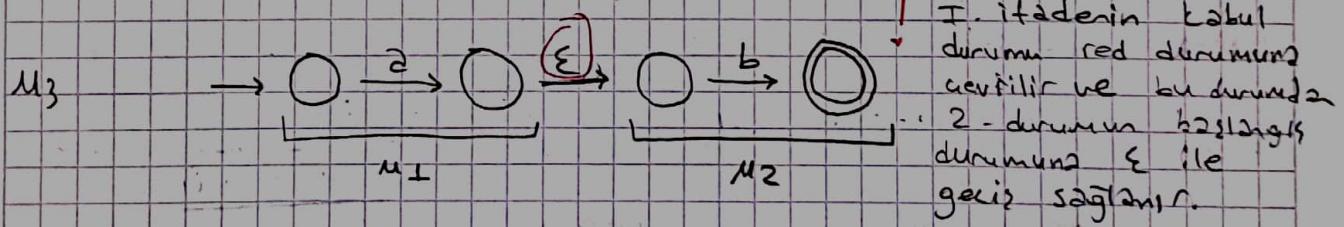
Bu da 2. yöntemdir. Bir Regex'i NFA'ya dönüştürmenizi sağlar.

Örnek: $(ab|a)^*$ RegExini NFA'ya dönüştüreceğiz.



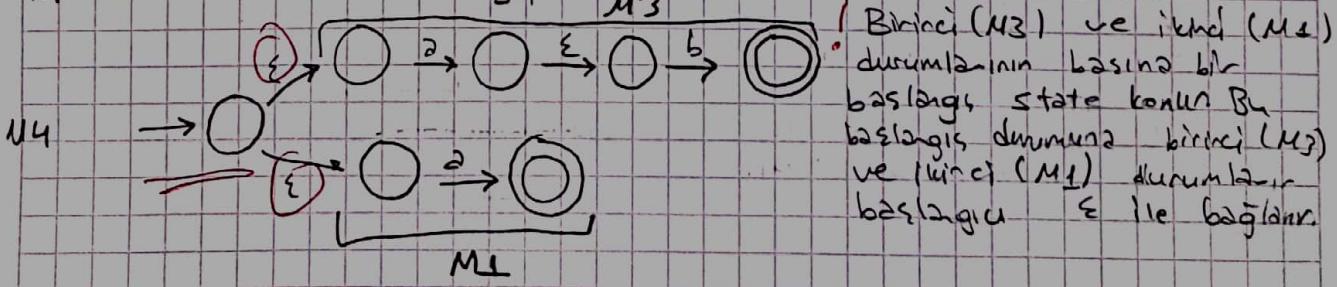
• Concat. (yanyana yazma) işlemi yaparak ab ifadesi elde edeceğiz.

NFA'da concatenation işlemi yaparken :

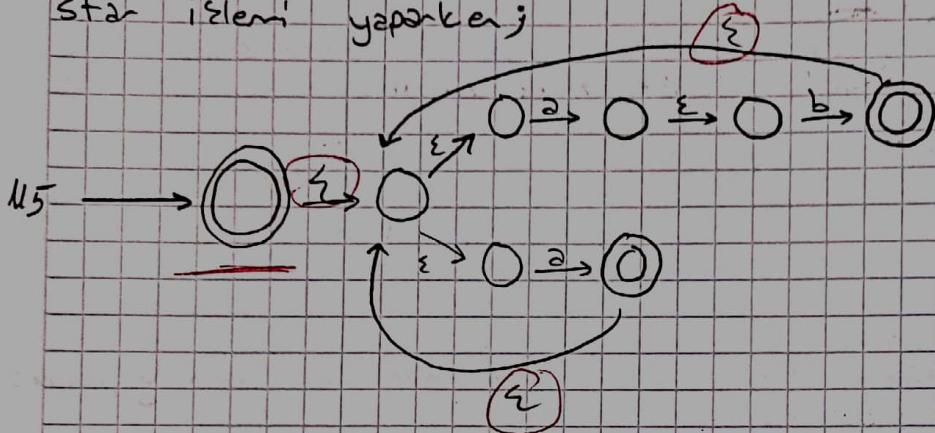


• Union (bileşme) işlemi yaparak (ab|a) ifadesini elde edeceğiz.

NFA'da union işlemi yaparken :



star işlemi yaparak $(abua)^*$ elde edeceğiz. NFA'da
star işlemi yaparken;



Başa bir tane bitiş durumu eklenerek yeni başlangıç durumu olarak belirlenir. Ardından önceki tüm final statelerden, önceki başlangıç durumunu ϵ gelisi sağlanır. Yeni başlangıç durumunda eskisine de ϵ geçisi olur.

Tüm bu işlemler bittiğinde RegEx NFA'ya dönümsel olun. Burdan sonra istenirse NFA \rightarrow DFA dönüsümü yapılarak DFA elde edilebilir.

DFA'yi RegEx'e Çevirme:

Bu da bahsedilen 3. yöntemdir.

B ve C iki bilinen dil olsun. ($\epsilon \notin B$). Bu iki dil üzerinde bilinmeyecek L dilini modelllemek için şu formül kullanılır:

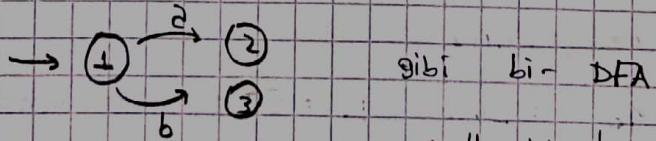
$$L = B^* C$$

Bilinmeyecek dil kendiri
 B üzerinde tetraf ederek

C ile birlikte.

b^* gibi bir şey olacak
 b^* te olabilir $b^* \epsilon$ olarak düşünürek. ($(\epsilon \in C)$ olabilir)

Örneğin:



gibi bir DFA

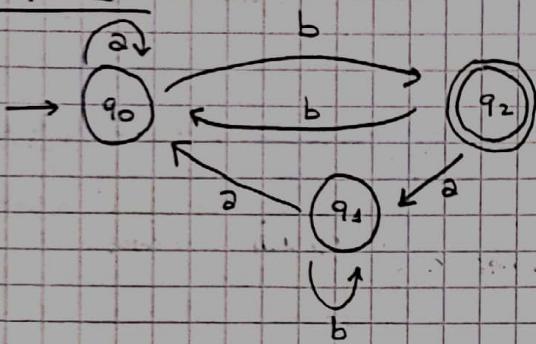
\rightarrow Her bir durumu, içindeki hanflerle kodlayacağız.

$$L_1 = a L_2 \cup b L_3$$

a ile L_2 'ye b ile L_3 'e pides

\rightarrow State final ise soldaki tanının yanına union ile ϵ ekleyiz.

"Örnek:



DFA'sı verilmiş olsun. Bunu Regülörlüce yapın.

→ Öncelikle genel sayfadaki kodlamaları tüm durumlar için yapalım:

$$L_{q_0} = a \cdot L_{q_0} \cup b \cdot L_{q_2}$$

$$L_{q_1} = a \cdot L_{q_0} \cup b \cdot L_{q_2}$$

$$L_{q_2} = a \cdot L_{q_2} \cup b \cdot L_{q_0} \cup \epsilon$$

|
final state
o buğn iyi'n
kaydet

} Hedefimiz initial (başlangıç) durumunun sağ tarafını indirgeyerek (Yani $a \cdot L_{q_0}$ kılacak şekilde düzenlemek)

$L = BLUC$ formatında yazmak

* İlk satırda (L_{q_0}), $L = BLUC$ formatı için L_{q_0} var aynı zamanda bilinmeyen bir L_{q_2} var. Bu bilinmeyen'i yok etmemiz gerekiyor. Bunu yapmak için üçüncü satırındaki (L_{q_2}) eşitliğini, L_{q_2} yerine ikinci satırda yazabiliyoruz.

$$\underline{L_{q_0}} = \underline{a \cdot L_{q_0}} \cup \underline{b \cdot (\epsilon \cup a \cdot L_{q_1} \cup b \cdot L_{q_0})} \Rightarrow (a \cup b) \cdot L_{q_0} \cup b \cdot L_{q_1} \cup b \cdot L_{q_0}$$

Sadece düzeneştirilebilir.

Bu kez de L_{q_1} bilinmiyor. Hine aynı tabitiği yaparak ikinci satırındaki, (L_{q_1}) eşitliğinin sağ tarafını bilinmeyecek yerine yazıyoruz.

Öncesinde L_{q_1} 'i düzenleyebiliyoruz.

$$L_{q_1} = \underline{b \cdot L_{q_1}} \cup \underline{a \cdot L_{q_0}} \quad \text{bunu } L = B^* C \text{ formunda da yazabilirim.}$$

$$L_{q_1} = L = B^* C = \boxed{b^* a \cdot L_{q_0}}$$

Bunu yukarıdaki form ile yazıyoruz

$$L_{q_0} = (aUb\bar{b}) \cdot L_{q_0} \cup b \cdot L_{q_0} \cup b$$

bu denkler sadece
 L_{q_0} 'a bağlı hale geldi.

\swarrow (L_{q_0} parantezine aldı)

$$= \underbrace{(aUb\bar{b} Ub\bar{b}^* a)}_B \cdot L_{q_0} \cup \underbrace{b}_C \rightarrow \text{Bunu } B^* C \text{ formatında yazabiliyoruz.}$$

\hookrightarrow (L 'yi yok et B 'yi yıldızla
B ve C'yi concat yap.)

$$L_{q_0} = (aUb\bar{b} Ub\bar{b}^* a)^* b$$

bu da bizim RegEx'imizezdir.

$$(aUb\bar{b} Ub\bar{b}^* a)^* b$$

DFA \rightarrow RegEx Dönüşüm.

- PUMPING LEMMA -

Bir problemin regular (düzeli) olmadığını ispatlamak iin, o probleme uygulanan bir yöntemdir. Örneğin; bir dil probleminin DFA, NFA, RegEx'ini bulamıysak o zaman bu yöntemi uygularız ve o problemin regular olmadığını ispatlamaya çalışırız.

*Önek vermek gereklse:

$$\{0^n 1^n : n \geq 0\}$$

Bu probleme 0'ların ve 1'lerin sayısının eşit olması isteniyor. (önce 0'lar sonra 1'ler gelecek şekilde). Biz biliyoruz ki regular dil seviyesindeki arayolların belleği yok. (Bellegi olan problemler regular degildir)

Eğer burada $0^2 1^2$, $0^4 1^4$ gibi bir şey yazsaydık bu düzeli dil problemi olurdu çünkü sayılar belli. Ancak bize verilen problemden problem " n " sayısında bağlıdır. Bu durumda n sayısı sonsuzda yaklastığında kullanılan durum sayısı da sonsuzda yaklaşırs. Dolayısıyla deterministik bir çözüm elde edemeyebiliriz.

Bu örneğin 0^{100} ile açıklanabileceğini düşünülebilir. Fakat $*$ işlemi sayıların eşit olmasını garanti etmez.

Pumping Lemma'nın Uygulanması :

A bir düzenli dil olsun. pumping length (p) sayısı belirlenir. $p \geq 1$ olmalı ve A'nın içindeki bir string (s) için $|s| \geq p$ olmalı. $s = xyz$ olmalı.

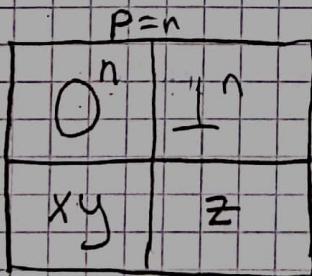
→ Kelimeyi 3 parçaya bölyoruz. (xyz diye). Her bir parça bir substringdir. Bir de o substringin özelliklerini denetleyeceğiz.

* Ortada kalan parça (y) bizim için en kritik parçadır.

1-) $y \neq \epsilon$ (yani $|y| > 1$) (ortadaki parça boş olamaz)

2-) $|xy| \leq p$ (Bastırıktaki parça (x veya y) bir arada düşünürken \rightarrow uzunluğu, pumping length (p) uzunluklarından \geq veya eşit olabilir. (tekrar eden en küçük sayı) \rightarrow $p=n$ olur.)

↳ 2. kurallar dolayı xy ikilisi, 0'ların hepsini almak zorunda degidir. ($|xy| \leq p$ olduğundan ve 0'ların sayısı n olduğundan). Bazi 0'lar z 'ye bırakılabilir. Ama z , hiçbir 1'i y 'ye bırakamaz. Çünkü öyle yaparsa 2. kural sağlanmış olur.



$$\left. \begin{array}{l} x=0 \\ y=0^{p-1} \\ x+p \end{array} \right\}$$

• gibi bir tanımlama yapılabılır. x ve y 'lerin sayısı toplamda p ye eşit olduğunda sorun yok. $\rightarrow xyz$ yan yana yazılıncaya p tane 0 ve p tane 1 olmalı.

* ilicik 2 kural sağlandığı anda 3. kural uygulanmaya başlayabiliyoruz.

3-) $i > 0$ iain, $xy^iz \in A$ olmali. (xyz parçalanmasını kullanarak y 'yi sile. Buradaki sisirme ilki türündür: hem y 'nin sayısını artırmak için kullanılabilir, hem de y 'nin sayısını azaltmak için kullanılabilir. i sayısının sıfırdan başlayabileceğini göz önünde bulundurulduğunda y^0 bir anda NULL string haline dönüse ve yazılmaz.)

$$xy^0z = \underline{0}^P \quad (\text{y dikkate alınmadı}) \quad \begin{array}{l} \rightarrow \text{Bunu diktten sonra} \\ \text{başka } xyz \text{ parçalanmaları} \\ \text{üzerinde durulur.} \end{array}$$

$\downarrow \quad \downarrow$

$$0^0 0^{P-1} \quad \downarrow \quad \downarrow \quad \begin{array}{l} \rightarrow \text{Bu sonu } 0^n 1^n \text{ 'i sağlamaz.} \end{array}$$

Bu yöntemde bir $i=0$ bir de $i=2$ ihtimalerine bakılır.

* Başka parçalama var mı diye bak. + ve 2. kurallı göre tüm parçalama ihtimallerini düşünüp, 3. kuralı uyguladığımızda yine de bir çözüm bulamıyorsak o zaman bu problem regular degildir.

Bir çözüm bile bılsak regulardir.

Bir diğer parçalama (1. ve 2. kurallı göre) :

$$\begin{aligned} x &= 0 \\ y &= 0 \\ z &= 0^{P-2} 1^P \end{aligned} \quad \left. \begin{array}{l} \text{• 1. kural sağlanır. } (y \neq \epsilon) \\ \text{• } |xy| = 2 \leq P \text{ sağlanır (2. kural)} \\ \text{• 3. kuralı } i=0 \text{ için deysek -} \end{array} \right\}$$

$$xy^0z = \underline{0}^P \underline{0}^{P-2} 1^P = 0^{P-1} 1^P \notin A \quad \begin{array}{l} \text{olduğundan şartı} \\ \text{sağlamaz.} \\ (0^n 1^n) \end{array}$$

Bir diğer parçalama i

$$\begin{aligned} x &= \epsilon \\ y &= 0^P \\ z &= 1^P \end{aligned} \quad \left. \begin{array}{l} \text{1. ve 2. kural sağlanır.} \\ \text{i=0 için ; } xy^0z = 1^P \text{ olur. } \notin A \text{ olduğundan şartı} \\ \text{sağlamaz.} \\ (0^n 1^n) \end{array} \right\}$$

* Her parçalama (mantıksal olarak) denedik ama pumping testini geçen bir parçalama bulamadık. Bu yüzden " $0^n 1^n$ " regular degildir.

Örnek: $A = \{w \in \{0,1\}^*: 0'ların ve 1'lerin sayısı eşit olmalı\}$

(0^n1^n gibi blok olma şartı konulur)

Bu problem;

$$= \{0^n1^n, (01)^*, \dots\} \text{ gibi elemanları kapsar.}$$

\downarrow \downarrow
regular regular
değil

* Hem regular olan hem de regular olmayan bir dili kapsayan bir problem kesinlikle regular değildir. Regular olması için ierulediği tüm stringlerin regular olması gereklidir.

Örnek: $A = \{ww : w \in \{0,1\}^*\}$

Bu durumda problemi çözme bilmek gerekliliğinden sikintı olacaktır.

w

Örnek: $A = \{0^m1^n : m > n > 0\}$

Sıfırların ve birlerin sayısı farklıdır. Bu durumda tekrar eden en küçük değişken pumping lemma olur. ($p=n$)

Örnek: $A = \{w \in \{0,1\}^* : \dots\}$ (7. örnek kitaptaki)

* DFA'sı çizilebilen bir problem regulardır.

* Kitaptaki örnekleri incele. (Sayfa 69 - 76)

HAFTA IV. SON

CHAPTER 3

- Context-Free Languages -

Bir örnekle başlayacak olursak:

$$\begin{array}{l} S \xrightarrow{1} A\beta \\ A \xrightarrow{2} a \\ A \xrightarrow{3} aA \\ B \xrightarrow{4} b \\ B \xrightarrow{5} bB \end{array}$$

} S, A, B \rightarrow değişkenler, a, b \rightarrow terminaller, Bir kelimenin üretilip üretilmeyeceğini su 3 adımla anlarız

1-) Üretim \rightarrow S başlangıcından başlanarak üretilmeli.

2-) Kuralları kafamızı göre düzenliyoruz. (Soldaki variableler yeine sağdaki karşılıklarını yazarak)

3-) Tüm variableler terminal olana kadar Adım 2'yi tekrarla.

- gramer.
• (Mali kisimları yazmayı pek yok!)

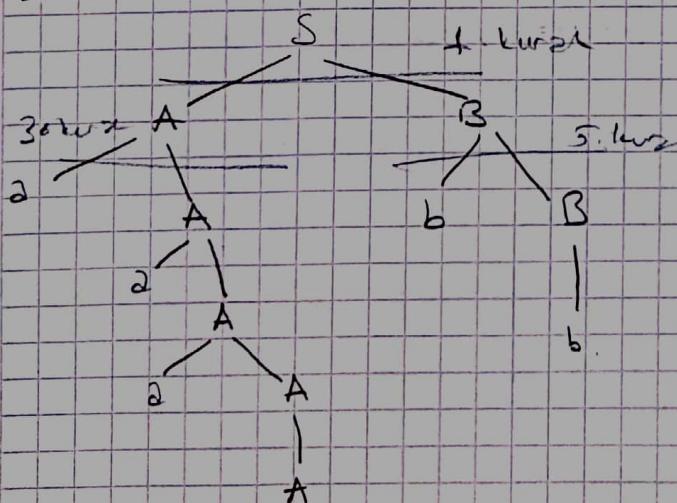
Örnek; aaaabb stringi bu gramer ile üretilir mi?

Yukarıdaki 3 adımı kullanarak bu grmerden verilen stringin üretildigini gördük.

$$\begin{array}{l} S \xrightarrow{1} A\beta \\ \xrightarrow{2} aA\beta \\ \xrightarrow{2} aA\beta B \\ \xrightarrow{2} aaA\beta B \\ \xrightarrow{2} aaaA\beta B \\ \xrightarrow{2} aaaab\beta B \\ \xrightarrow{4} aaaabb \end{array}$$

* Kuraldaki ok (\rightarrow)
Üretimdeki ok (\Rightarrow)

Parse Tree:



Bunu uzeren üretimi daha anlaşılır hale getirebiliriz.

Örneki: $\{a^m b^n : m \geq 1, n \geq 1\}$ (Yukarıdaki örnek)

a ve b birbirinden bağımsız sayıda üretilir. Bu aslında recursion'dır. 3. adım recursion kismı, 2. ve 4. adım durdurma kismıdır.

RegEx: a^m bⁿ (Burada en az bir faire a ve b'yi garanti etti) $(n \geq 1, m \geq 1)$ (Düşündür)

* Bir context-free grammar için 4 parametreye ihtiyacımız var

$G = (V, \Sigma, R, S)$. Bunlarla ilgili kurallar ve tanımlar:

- $V \rightarrow$ değişkenleri (non-terminal, büyük harf) temsil eder.
- $\Sigma \rightarrow$ terminaller (küçük harf) temsil eder.
- $V \cap \Sigma = \emptyset$ (Yani bir sembol aynı anda değişken ve terminal olmamalı)
- $S \in V$ olup; S , başlangıç değişkenidir.
- R kümesi geçişleri ifade eder. (Önceki sayfadaki örneğimizde $R = \{S \rightarrow AB, A \rightarrow a, A \rightarrow aA, B \rightarrow b, B \rightarrow bB\}$)

Örnek: (iç içe parantezler)

$G = (V, \Sigma, S, R)$, $V = \{S\}$, $\Sigma = \{a, b\}$, $R = \{S \rightarrow \epsilon, S \rightarrow aSb, S \rightarrow SS\}$

olsun.

R 'deki üç genisi yazacak olursak:

$$S \rightarrow \epsilon \quad | \quad aSb \quad | \quad SS$$

$\xrightarrow{\text{! or kullanabiliriz.}}$

* Bu örnekte iç içe parantez mantığı kurulmuştur. $a \rightarrow ($
 $b \rightarrow)$ temsil eder. Ortadaki ' aSb ' her açılan parantez için kapanmayı garanti eder.

$$\begin{aligned} S &\Rightarrow SS \\ &\Rightarrow aSbS \\ &\Rightarrow aSbSS \\ &\Rightarrow aSSbSS \\ &\Rightarrow aabSbSS \\ &\Rightarrow aababSbSS \\ &\Rightarrow aabbabSS \\ &\Rightarrow aabbabbS \\ &\Rightarrow aabbabbS \\ &\Rightarrow aabbabbabS \\ &\Rightarrow aabbabbab \end{aligned} \quad \begin{aligned} &() \\ &() \\ &() \\ &(()) \\ &(()) \\ &(()) \\ &(())() \\ &(())() \\ &(())()() \\ &(())()() \end{aligned}$$

* Geçen haftalı $L_1 = \{0^n 1^n : n \geq 0\}$ regular değilidi.

Burada bu dilin context-free olduğunu göreceğiz.

$S_1 \rightarrow \epsilon \mid 0S_1 1$ yaparak aslında $0^n 1^n$ i saglamış oluruz.
Lütfen her 0 üretiminde 1 de ürettilir.
↓
sonlandırma kuralları

← Eğer bunun tersini yapıyor olsaydık yani $L_2 = \{1^n 0^n : n \geq 0\}$

$S_2 \rightarrow \epsilon \mid 1S_2 0$

- $L = L_1 \cup L_2$ olsun. Yani. $L = \{0^n 1^n : n \geq 0\} \cup \{1^n 0^n : n \geq 0\}$ olsun.

Kurallar:

$S \rightarrow S_1 \mid S_2$

iki kuralı union yaparak birlestirdik

$S_1 \rightarrow \epsilon \mid 0S_1 1$

$S_2 \rightarrow \epsilon \mid 1S_2 0$

A context-free grammar for the complement of a nonregular language

Non-regular bir dilin tümleyenini aldigımızda context-free olabilir.

L bir nonregular dil olsun ve bu dil $L = \{0^n 1^n : n \geq 0\}$ kurallarından olsun.

\bar{L} 'nin (L 'nin tersi) bir context-free language olduğunu göstermeye

yalıracagız. \bar{L} ile kastedilen şey tüm binary stringler içinde

\bar{L} 'yi ükarmat yani $\bar{L} = (\text{0U1})^*$ - $0^n 1^n$

↳ bunun içinde 0 ve 1'lerin blok setlerinde
olarası ve olmazları mevcut.

w string olsun

1-) $w = 0^m 1^n$, $0 \leq m < n$

2-) $w = 0^m 1^n$, $0 \leq n < m$

3-) $w = 10$ içermeli.

} Bu üç kurallın birleşimi bize

\bar{L} 'yi verir.

↳ w 'nin tanımı demek $000..111..$ şeklindeki blok
yapısının bozulması demektir. L 'de 0 ve 1'ler blok şeklinde bulunduğunda
burası tercih olur \bar{L} , blok şeklinde olmazları da içermeli.

Simdi bu üçün birleştirerek bir kural yazacağız. (context-free için)

→ daha fazla | işaretini (örn 01111...)

$$S_1 \rightarrow \perp | S_1 \perp | 0S_1 \perp \quad (w = 0^m 1^n : 0 \leq m < n \text{ i sağlar})$$

↳ Bu kural 1'lerin sayısının fazla olacağını garanti eder. (S_1 'in durdurma kriteri + olduğu için)

↳ 0'lar ve 1'ler blok şeklinde din

$$S_2 \rightarrow 0 | S_2 0 | 0S_2 \perp \quad (w = 0^m 1^n : 0 \leq n < m \text{ i sağlar})$$

↳ Yukarıdakini aynı (sadece 0'ın fazlasını garanti eder)

$$S_3 \rightarrow X \perp 0 X \quad (\perp 0 \text{ substringi için})$$

X yazıları yerinde herhangi bir şey olabilir. Bu durumda

$$X \rightarrow \varepsilon | 0X | \perp X$$

Simdi bu kuralları birleştirelim:

$$S \rightarrow S_1 \perp | S_2 \perp | S_3 \perp \quad \rightarrow \text{bunları kullanarak birleştirildik.}$$

$$S_1 \rightarrow \perp | S_1 \perp | 0S_1 \perp$$

$$S_2 \rightarrow 0 | 0S_2 | 0S_2 \perp$$

$$S_3 \rightarrow X \perp 0 X$$

$$X \rightarrow \varepsilon | 0X | \perp X$$

Bu şekilde context-free gramer elde ettik.

Örnek: $L = \{a^n b^m c^{n+m} : n \geq 0, m \geq 0\}$. Bu da bir hafızalı problemidir. Dolayısıyla regular değildir.

* Her a ürettiğimizde bir c, her b ürettiğimizde bir c üretmeliyiz.

$S \rightarrow \Sigma | A$
 $A \rightarrow \Sigma | aAc | B$
 $B \rightarrow \Sigma | bBc$

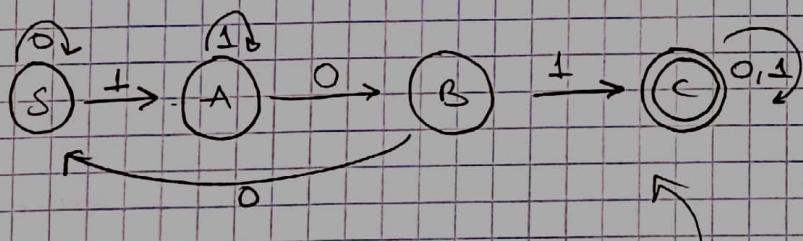
A işi bittiğten sonra B ye geçilir ve
 geri A'ya dönmemez. Bunun sebebi:
 blok mantığını bozmamak. Çünkü b'den
 aaaa... bbbb... ccccccc... şeklinde istenmiş.
 Hepside ϵ olasının nedeni n ve m'nin 0
 olabilmesidir.

Regular diller context-free'dır:

Kitabın sonundaki kapsamda da görüldüğü gibi Context-free, regular dilleri kapsar.

Örnek: $L = \{w \in \{0,1\}^*: w, 101 \text{ substring'i içermeyi}\}$

L düzeldidir. DFA'sını oluştursek



* Buna göre su turul yazılabilir.

$S \rightarrow 0S | 1A$
 $A \rightarrow 0B | 1A$
 $B \rightarrow 0S | 1C$
 $C \rightarrow 0C | 1C | \epsilon$

$\underbrace{\hspace{10em}}$
 010011011 bu dilde kabul edilen bir string'tir.
 Bunu söyle türeriz.
 $S \Rightarrow 0S$
 $\Rightarrow 01A$
 $\Rightarrow 010B$
 $\Rightarrow 0100S$
 $\Rightarrow 01001A$
 $\Rightarrow 010011A$
 $\Rightarrow 0100110B$
 $\Rightarrow 01001101C$
 $\Rightarrow 010011011C$
 $\Rightarrow 010011011$

- Chomsky Normal Form -

Context-free sorularını üzerindeken, bilgisayara bu işi öğretmek istedığımızda yani bir türerim sorusu sorduguunda bilgisayarın bunu yapması için biraz daha sistematik bir çözüme ihtiyaci var. (Context-free'de istediğimiz kuralı seçtiğimiz için sonsuz kadar dönmeye ihtiyacım var.)

Chomsky normal formunun 3 kuralı vardır:

1-) Değişkenler iki değişken dönüştürbilir.

$$A \rightarrow BC \quad (A, B, C, \text{değişken}, B \neq S, C \neq S)$$

* B ve C'nin start variable olmamasının nedeni, start variable azla sağ tarafta olamaz.

2-) Değişkenler tek bir terminal'e dönüştürbilir.

$$A \rightarrow a \quad (A \rightarrow \text{değişken}, a \rightarrow \text{terminal})$$

3-) $S \rightarrow \epsilon$ sevgəsi olmali.

* Bazı bir değişkenin sağında ϵ bulunamaz.

→ n uzunluğundaki bir kelime için 2^{n-1} uzunkulu kombinasyonla, derse. Eğer ulaşamazsa, der ki bu kelime bu gramle üretilmez (Dereme sayısı, belli oldupundan sonsuz gitmez)

* Şimdi context-free gramer olarak verilen bir gramer:

Chomsky normal forma dönüştürme adımlarını örnek üzerinde göreceli (5 adımlı dönüştüm)

Örnek : (context-free \rightarrow Chomsky)

$$A \rightarrow BAB | B | \epsilon$$

$$B \rightarrow 00 | \epsilon$$

Verilmiş olsun.

Adım 1 : Başlangıç değişkenini sağ taraftan yok et!

Bunun için yeni bir start variable tanımlanır ve yeni start variable eskiine atanır.

$$S \rightarrow A$$

$$A \rightarrow BAB | B | \epsilon$$

$$B \rightarrow 00 | \epsilon$$

Adım 2 : Başlangıç değişkeninde (start variable) olmayan tüm ϵ 'ller yok etti.

(Başlangıç değişkenin önceki başlangıç ϵ 'sına ulaşabiliir bu yüzden önceki ϵ 'sini sil ve başlangıçta ϵ 'yi)

* ϵ 'ler silindikten sonra, ϵ sayesinde olan geçişleri de değişkenlere ekle.

$$S \rightarrow A | \epsilon$$

$$A \rightarrow BAB | B$$

$$B \rightarrow 00 | \epsilon$$

Bu adımdan sonra 2. kuralda ϵ geçisinden dolayı oluşan geçişler de bu kurala eklenmelii.

| 2. kuralın sağında A göründüğünde ϵ yazılmalı.
| Çünkü 2. kuralda ϵ varken BB elde edebiliriz.
| bunu kaybetmemek için böyle yaptıktı.

$$S \rightarrow A | \epsilon$$

$$A \rightarrow BAB | B | BB$$

$$B \rightarrow 00 | \epsilon$$

Bundan sonradan 3. kuralda ϵ silinmeli

| BAB 'nın ilk B 'si ϵ ye giderse AB kalır.

| BAB 'nın ikinci B 'si ϵ ye gidersse A kalır.

| BAB 'nın üçüncü B 'si ϵ ye giderse BA kalır.

Burda da 2. kurala ekliyoruz.

Son durum :

$$S \rightarrow A | \epsilon$$

$$A \rightarrow BAB | B | BB | AB | BA | A$$

$$B \rightarrow 00$$

* Yani ϵ 'ler kaldırınca olacak tüm durumlar hesabe katılır ve yazılır.

Adım 3: Tek başına bulunan non-terminalları sil!

* Nihayi, silinen deşirklerin kuralları nasıl etkilediği hesabına katılmalı.

$S \rightarrow A \Sigma$ kurallında A'yı silersek 2. kurallı şeuma ihtiyacını ortadan kaldırıyoruz. Dolayısıyla yok edilen durumu düzeltmek için 2. kurallın sağını olduğum gibi S'ye taşıyoruz.

$S \rightarrow \epsilon / BAB / B / BB / AB / BA$
 $A \rightarrow BAB / B / BB / AB / BA$
 $B \rightarrow oo$

Bu islemlerden sonra geriye B tek başına kalmış. Bunu da silince B kurallını sildiğimiz gese yazıyoruz.
Yani budurunda:

$S \rightarrow \epsilon / (BAB) / BB / AB / BA / oo$

$A \rightarrow BAB / BB / AB / BA / oo$

$B \rightarrow oo$

Adım 4: İki deşirkeden daha uzun olan yerlei parçalıyoruz

$S \rightarrow BA \beta$ için:

BA₁

$\beta \rightarrow AB$ olabilir.

Aynı ayri alıyoruz (Normalde tek bir BA₁ kuralı yazabiliriiz.) (Kitaptaki gösterimi bu şekilde yoksuz bir farklılık)

$A \rightarrow BAB$ için

BA₂

$\beta \rightarrow AB$ olabilir.

$S \rightarrow \epsilon / BB / AB / BA / oo / BA \beta$

$A \rightarrow BB / AB / BA / oo / BA \beta$

$B \rightarrow oo$

$A_1 \rightarrow AB$

$A_2 \rightarrow AB$

Adım 5 : Cıktı terminaleri parçalı.

1. durum

S' deki 00 iin :

$A_3 \rightarrow 0$ deriz ve $A_3 A_3$ yazızız

2. durum

A' deki 00 iin :

$A_4 \rightarrow 0$ deriz ve $A_4 A_4$ yazızız

3. durum

B' deki 00 iin :

$A_5 \rightarrow 0$ deriz ve $A_5 A_5$ yazızız

Burada da yine hepsi ortak yapabiliriz ama kitapta böyle yapıyor

En son durumda :

$S \rightarrow S | BB | AB | BA | BA_1 | A_2 A_3$

$A \rightarrow BB | AB | BA | BA_2 | A_4 A_4$

$B \rightarrow A_5 A_5$

$A_1 \rightarrow AB$

$A_2 \rightarrow AB$

$A_3 \rightarrow 0$

$A_4 \rightarrow 0$

$A_5 \rightarrow 0$

X Bu 5 adımı kullanarak context-free \rightarrow Chomsk dörisini yaptık

HAFTA IV. SON

- Pushdown Automata - (Stack (küllənlilik))

* Hücrelərə ayrılmış bir bant vardır. Her hücre Σ (bant alfabesi) içinde bulunan bir simbol sakları. Burada özəl bir simbol vardır. \square ($\# \rightarrow$ həcənin göstərdiği) boşluq simboludur. ve bu simbol bant alfabesi (Σ) içinde bulunmaz. $\#$ (veya \square) simbolunun anlamı; bu hücre boştur.

* Bant sağa hareket edebilir veya olduğu yerde kalabilir. Ama sola doğru hareket edeməz. Bant kafası hücreləndeki simbollerini (tape head) təzarə.

* Γ ile göstərilən bir stack alfabesi vardır. Bu alfabe özəl bir simbol içərir '\$'. Bu simbol (\$) stack'in boş olduğunu göstərir.

* Stack head, stack'in en üstünü okur ve üsttəki eleməni stackten çıxır (pop). Sonrasında stack'in üstünə Γ 'dan simbol ekler (push).

* Durum denetimi tüm durumları kontrol eder. Q bir tane özəl state (state kontrol) (Q) içərir bunu başlangıç durumu (q) denir.

→ Pushdown Automatın için girdi Σ^* içindeki stringləndir. Bu string (PDA) Γ (sigma)

PDA'nın bantına yerleştirilir.

* Başlangıç durumunda;

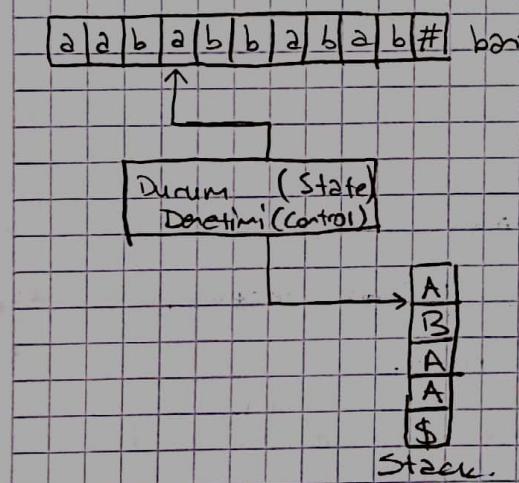
→ Bant kafası stringin en solundakı simbolün üstündədir.

→ Stackte sadəcə \$ işarəti vardır.

→ PDA q başlangıç durumundadır.

→ Herhangi bir adımda PDA zayle dauranır :

- PDA τ diye bir durumda olsun. ' a ' simbolu Σ 'nın bir (bant alf.) elemanı olsun ve bu simbol bant kafası tarafından okunsun. Aynı zamanda ' A ' simbolu, Γ 'nın bir elemanı olsun ve stack'in (stack alf.) en üstünde bulunsun.
 - Yani mevcut τ durumuna bağlı olarak, bant simbolu \Rightarrow_2 ,



2-) PDT, durumu Q 'nın elementi olar r' durumu ile değiştirir. ($r' = r$ olabilir)

b-) Bant kafası ya bir hücre sağa kayar
ya da olduğu yerde kalır

c-) Stackin en üstündeki A simbolü Γ^* lya ait olur bir 'w' stringi ile değiştirilir.

i) Eger $w = \epsilon$ ise sadere A stacken
populär. \downarrow \rightarrow (primär)

ii) Eger $w = B_1 B_2 \dots B_k$, $k > 1$ ve
 $B_1 B_2 \dots B_k \in T$ ise T yerine w
stringi yazıllır ve B_k stack'in
en üstü olur.

* Deterministic PDA iin 5 deger gerekir. $M = (\Sigma, \Gamma, Q, \delta, q_0)$

- Σ , bant alfabelidir. # (blank symbol), bant alfabelisinde bulunmaz.
 - Γ , stak alfabelidir. Bu alfabe \$ özel sembolünü içenir.
 - Q , durumlar.
 - q , Q 'nın elemanıdır ve başlangıç durumunu temsil eder.
 - f genis fonksiyonudur ve şu şekilde dir:

~~✓~~ Sol tarafında 3 harf vardır.
Birinci → durumu
ikinci → bantları gelen kelimeyi
üçüncü → stäckter gelen kelimeyi
ifade et.

Ömeli: $q_2 A \rightarrow q_2 R AAA$ very $q_2 b A \rightarrow q_2 NO$ fast

$r \in Q$, $a \in \Sigma \cup \{\#\}$, $A \in \Gamma$ ve $r' \in Q$, $\sigma \in \{L, N\}$, $w \in \Gamma^*$

$$\delta(r, a, A) = (r', \sigma, w)$$

* Geçiş fonksiyonunun özetli hali: Bunun anlamı:

PDA r durumundaysa, bant kafası a sembolünü okursa ve stack'in üstünde A sembolü varsa,

r' durumuna gel, σ 'nın durumuna göre yerinde kal veya bir sağa gel bantta. ($N \rightarrow$ kal, $R \rightarrow$ bir hücre sağa git) ve stack'in üstündeki A 'yi sil ve w stringini stack'e yaz.

$$\boxed{r a A \rightarrow r' \sigma w}$$

* PDA'ya girilen input'un kabul edilmesi için:

→ PDA, bu kelime üzerinde sonlanmalı. Yani kelimenin harfleri banttan tek tek okunduktan sonra stack boş olmalı.

→ Sonuncu harfin bulunduğu hücreden sonraki hücrede $\#$ olmalı.

! PDA sonlanmazsa, sonsuz döngü oluşturur.

*~~Kısaca~~, kelime sonlandığında stack boşsa kelime kabul edilir.

Önek: (iki içe parantezler) (deterministic PDA)

'(' sayısı ')' sayısına eşit olmalı. (Kelime sonlandığında)

(a) (b)

sol parantezin (kaytane '2' olduğunu tutuyor)

$M = (\Sigma, \Gamma, Q, \delta, q_0)$, $\Sigma = \{a, b\}$, $\Gamma = \{\$\}, Q = \{q\}$

Geçiş fonksiyonunu şu şekilde dir:

q_0 durumunda $\$$ gelirse stack'e $\$$ yaz.

- $q_0 \$ \rightarrow q_R \$\$$ \rightarrow stack'in üstüne $\$$ pushlanır. (parantez açma geldiği için)
- $q_2 S \rightarrow q_R SS$ \rightarrow stack'in üstüne S pushlanır.
- $q_2 S \rightarrow q_R E$ \rightarrow stack'in üstündeki elemen pop edilir. (önün parantez kapama geldi, biz kesişigini olmayan sollar tutuyoruz)
- $q_2 \$ \rightarrow q_N E$ \rightarrow b'lerin sayısı, Σ 'ların sayılarından fazla olma durumu. Bu durumda kelime reddedilir. (stack boşken parantez kapama gelir)
- $q \# \$ \rightarrow q_N E$ \rightarrow kelime sonlandı ve stack boş, kelime kabul edilir.
- $q \# S \rightarrow q_N S$ \rightarrow Σ 'ların sayısı, b'lerden fazla. Boşluk gelmesine (kelime sonlamasına) rağmen stack'te elemen var. Bu durum reddedilir.

* Kaç tane adım yazmamız gerektiğini söyle bulunuz:

Q 'nın elemen sayısı $*(\Sigma$ 'nın elemen sayısı) \rightarrow Γ 'nın elemen sayısı $\hookrightarrow \#$ 'den dolayı,

\rightarrow Bu örnek için $Q \rightarrow 1$ elemaklı, $\Sigma = 2$ elemaklı $\Rightarrow 3$, $\Gamma = 2$ elemaklı
birde $\#$ var

$$1 * 3 * 2 = 6 \text{ adım}$$

* Sol tarafta olası tüm adımları yazıp ne olacağını sağ tarafta yazıyoruz.

Örnek: $\{0^n 1^n : n > 0\}$ (deterministik):

- Her 0 okundugunda stack'e S pushlanacak ve q_0 durumunda kılacak.
- İlk 1 okundugunda q_1 durumundan geçilir. Bu durumda (q_1 da):
 - Okunur her 1 iin stacken 1 tane S silinir ve q_1 durumunda kalmaya devam ederiz.
 - Eğer 0 okunursa sonsuz dangleye girer ve PDA sonlanır.

$$M = (\Sigma, \Gamma, Q, \delta, q_0), \quad \Sigma = \{0, 1\}, \quad \Gamma = \{\$, \$\$}, \quad Q = \{q_0, q_1\}, \quad q_0 \rightarrow \text{baslangic}.$$

δ genis fonksiyonu su şekilde dir:

* $(|\Sigma|+1) * |\Gamma| + |Q| = (2+1) * 2 + 2 = 12$ adim vardır.

$$q_0 0 \$ \rightarrow q_0 R \$\$$$

Stack'e S ekle

$$q_0 0 \$ \rightarrow q_0 R S\$$$

Stack'e S ekle

$$q_0 1 \$ \rightarrow q_0 N \$$$

ilk girilen deger 1 olursa : sonsuz dengi.

$$q_0 1 \$ \rightarrow q_1 R \epsilon$$

Karsilashilar ilk 1.

$$q_0 \# \$ \rightarrow q_0 N \epsilon$$

(Kelimenin sonu gelmir ve stack bos (kabul)
(O uzunlukta kelime) n=0)

$$q_0 \# \$ \rightarrow q_0 N S$$

Kelime sadece 0 larda olusur : sonsuz dengi

$$q_1 0 \$ \rightarrow q_1 N \$$$

En az bir tane + girildikten sonra 0 gelmir : sonsuz dengi

$$q_1 0 \$ \rightarrow q_1 N S$$

En az bir tane + girildikten sonra 0 gelmir : sonsuz dengi

$$q_1 1 \$ \rightarrow q_1 N \$$$

1lerin sayisi 0'dan fazla : sonsuz dengi

$$q_1 1 \$ \rightarrow q_1 R \epsilon$$

Stack'ten bir tane S pop edilir

$$q_1 \# \$ \rightarrow q_1 N \epsilon$$

Bosluk geldiginde stack'te bosmus ; kabul
(n>0)

$$q_1 \# \$ \rightarrow q_1 N S$$

Bosluk geldiginde Stack'te hala element var
yani 0'ların sayısı dahi 1'den fazla : sonsuz dengi.

Örnek : Ortasında 'b' olan kelimeler (non-deterministic)

$$L = \{Vbw : V \in \{a, b\}^*, w \in \{a, b\}^*, |V| = |w|\}$$

* iki durum vardır, $q \rightarrow$ ortadaki simbol okundu, $q' \rightarrow$ ortadaki okunmadı.

* Eğer q durumundayken b gelirse :

- 1. ihtimal : Henüz kelimenin basina harfler giriliyor olabilir

- 2. ihtimal : Bu 'b' kelimenin tam ortası olabilir.

* Kelime q' durumuna gectikten sonra hep burada kalır geri
 q durumundan dönmeyez.

$$\Sigma = \{\Sigma, T, Q, S, q\}, \Sigma = \{a, b\}, T = \{\$, \$\}, Q = \{q, q'\}$$

ve q başlangıç durumu olsun. S söyledir.

\rightarrow Deterministik olmadığı için adım sayısı hesaplanamaz. Fazla da olabilir ve de olabilir. (Genelde fazla adım okuyor)

$3 \times 2 \times 2 = 12$ adım olmalydi. Ama 14 adım var.

$$q_2\$ \rightarrow qR\$S$$

Kelimenin ortası okunmadan λ gelirse stacke S pushla.

$$q_2S \rightarrow qRS\$$$

Kelimenin ortası gelmeden λ gelirse stacke S pushla.

$$q_2\$ \rightarrow q'R\$$$

Ortaya ulaştı. (Tek 'b' harfi olan kelime) (tele uzantular)

$$q_2\$ \rightarrow qRS\$$$

Henüz kelimenin başına bir şeyler ekleniyor. ' S ' pushla stacke

$$q_2S \rightarrow q'RS$$

Ortaya ulaştı.

$$q_2S \rightarrow qRSS$$

Henüz ortaya ulaşmadı. Stacke S pushla.

$$q\#\$ \rightarrow qNS\$$$

Sonsuz döngü. (Kelimenin ortası gelmeden kelime sonlandı ve stack boş)

$$q\#S \rightarrow qNS$$

Sonsuz döngü. (stackte eleman var ama kelimenin sonu geldi)

$$q'a\$ \rightarrow q'Ne$$

Sonra bir fakat reddedilir çünkü ortadaki eleman okunmuştur ve stack boş olmasına rağmen kelime sonlanmamıştır.

$$q'aS \rightarrow q'RG$$

Bir S 'yi pop et.

$$q'b\$ \rightarrow q'Ne$$

Sonra bir ama kabul edilmez. Bu sebeple dolaylı yine.

$$q'bS \rightarrow q'RG$$

Bir S 'yi pop et.

$$q'\#\$ \rightarrow q'Ne$$

Kabul ediliir.

$$q'\#S \rightarrow q'NS$$

Sonsuza döngü. Kelime sonlanmasına rağmen stackte eleman var.

— HAFTA VI. SON —

Equivalence of Pushdown Automata and Context-Free Grammars

$$CFG \equiv NPA \supset PDA$$

Nondeterministic PDA, CFG'ye dönüştürülebilir.

- Bu konuda Chomsky Formülünde yazılmış context-free grammar kurallarıyla (CFG) PDA arasındaki dencliliği sağlayarak dönüşümü öğreneceğiz. Bu dönüşüm 3 kuralla özetleyebiliriz.

1 - Chomsky'de genel olan $A \rightarrow BC$, $A, B, C \in V$ olan kural, PDA'da $q_A A \xrightarrow{\cdot} q_N C B$ 'ye dönüşür. (tüm $a \in \Sigma$ için) Bunun enlemi \Rightarrow nonterminalin tanesi (variable).

\equiv (signal) atı her elemen için bu PDA kurallını yaz.

2 - Chomsky'de genel olan $A \rightarrow a$ kuralı ($A \in V, a \in \Sigma$), PDA'da.

$q_A A \xrightarrow{\cdot} q_R \epsilon$ 'a' dönüşür.

! Bu dönüşümde tek bir kural vardır o da q başlangıç durumu.
 \Rightarrow start variable (S' de diyebiliriz)

3 - Chomsky'de genel olan $\$ \rightarrow \epsilon$ kuralı PDA'da.

$q \# \$ \xrightarrow{\cdot} q_N \epsilon$ (kabul durumu)

"Örnek! (Palindrom)

$S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$ (Bu CFG'dir) bunu önceki Chomsky'e dönüştirmeliyiz

CFG \rightarrow Chomsky

- Öncelikle,

$A \rightarrow a$

$B \rightarrow b$ kuralları yazılıarak adımlar su sıtında düzenlenir.

$S \rightarrow ASA \mid BSB \mid a \mid b \mid \epsilon$

$A \rightarrow a$

$B \rightarrow b$

\Rightarrow (Bu ikisi kurala uygun olduğundan değiştirmedik.)

- Sonrasında, bir değişken içi değişken gitmeliydi bunu da düzeltmeliyiz. Bunun için,

$$S \rightarrow CA | DB | a | b | \epsilon$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$C \rightarrow AS$$

$$D \rightarrow BS$$

- Göruldüğü gibi adımların sağ kısımlarında hala başlangıç değişkeni var. Bunun olmaması gerekiyor Chomsky için. Bu sorunu düzeltmek için, başka bir başlangıç değişkeni atiyoruz.

$$\$ \rightarrow CA | DB | a | b | \epsilon$$

$$S \rightarrow CA | DB | a | b | \epsilon$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$C \rightarrow AS$$

$$D \rightarrow BS$$

- Farklı başlangıç atadığımızda öncelik başlangıç sağ tarafını olduğu gibi alabiliyoruz.

Bir başka kurala göre sadece başlangıç değişkenin sağında ' ϵ ' olabilir ama burada S'in sağında da var. S'üki ' ϵ 'yi silmek için sağında S'ın sağundan her yere ' ϵ ' koymak test yaparız:

$$\$ \rightarrow CA | DB | a | b | \epsilon$$

$$S \rightarrow CA | DB | a | b$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$C \rightarrow AS | A\epsilon$$

$$D \rightarrow BS | B\epsilon$$

* Burada CFG \rightarrow Chomsky dönüşümünü tamamladık.

Chomsky \rightarrow PDA *

* 3. kural $\$ \rightarrow \epsilon$ dönüşümü için genelidir.

Örnek bir kural uygulaması

$$D \rightarrow BS | b \text{ için}$$

I. kuralı kullanarak $D \rightarrow BS'$ 'yi

! Bunu ters yazmamızın nedeni stack'e ters 2 tip okurken düzgün okumak için.

$$\begin{aligned} q_0 D &\xrightarrow{} q_1 NSB && \text{ifadesine} \\ q_1 D &\xrightarrow{} q_2 NSB && \text{dönüşürebiliriz.} \end{aligned}$$

II. kuralı kullanarak $D \rightarrow b^l$ 'yi

$$q_0 D \xrightarrow{} q_1 R \epsilon \quad \text{ifadesine} \quad \text{dönüşürebiliriz.}$$

* Tam bu dönüşümleri uygularsak bu problem için PDA'sının admalarını buluruz.

The Pumping Lemma for Context-free Languages

Düzenli dil (regular language) kapsamında anlatılan pumping lemma da olduğu gibi CFG kapsamında da bir pumping lemma vardır.

Regülardan 3 parçaya ayrılmış string, burada 5 parça.

• L bir context-free language olsun. $p \geq 1$ olacak şekilde bir pumping length belirlesin. L dili içindeki her s stringi ($|s| \geq p$)

su şekilde beş parçaya ayrılabilir: $s = uvxyz$

1 - $|vy| \geq 1$ (v ve y aynı anda boş olamaz)

2 - $|vxy| \leq p$

3 - $uv^ixy^iz \notin L$, tüm $i \geq 0$ için.

↳ Bu bölünme sonucu oluşan kelime L dilinin içindeyse bu bölünme doğrudur ve bu problem CFG'dir denir.

↳ Tüm bölünme metodlarını denemeye rağmen parçalanan kelimelerde yoksa bună CFG degildir deniz.

* Bir problemin context-free dil ailesine ait olup olmadığını kanıtlamak için bir kuradır.

Proof of the Pumping Lemma (Pumping Lemma'nın Kanıtı)

Örnek:

$A = \{a^n b^n c^n : n \geq 0\}$ (Bu CFG degildir bunu ispatlayacağımız)

- Genel olarak pumping length (p) için tekrar eden en küçük değişken ifadesini p olarak kullanılması tercih edilir. Dolayısıyla $|P|=n$ tercih edilir.
- Bunun sonucunda kelimenin (s) uzunluğun $3p$ kadar olur ki $3p \geq p$
- Daha sonra bu kelimeyi parçalayacağız ve üç kuralı test edeceğiz.

$$|V_y| \geq 1$$

$|V_{xy}| \leq p \Rightarrow$ Sıslanacak olan harfler.

$\forall i \geq 0$ için, olmalıdır.

- Uçuncu kuralı düşünerek olursak, v harfi a 'ya karşılık geliyor ve y harfi boş olsun diyalim. Bu durumda b ve c 'lerin sayısı \geq kadar arttıramaz. Başka bir denemede, v harfi b 'ye y harfi c 'ye karşılık geliyor diyalim. Bu durumda c ve b 'lerin sayısı eşit olur ama a 'nın sayısını garanti etmemeyiz. Kısacası bu problemin context-free dil ailesine ait olmadığını anlıyoruz.

Case 1: vxy substringi c 'yi hiç içermesin. (Yani $vxy = ab^n$ kısmından oluşur)

- Bu durumda c sayısı, a harfinda kaldığı için, pump edilme sırasında a ve b 'ye göre $a \geq b$ veya çok olabilir.

Case 2: vxy substringi a 'yı hiç içermesin.

- Case 1 ile aynı durum.

Case 3: vxy substringi en az $1 + n$ tane a ve en az $1 + n$ tane c içerir.

Yani $\boxed{ab^pc}$

$v=a$, $x=b^{p-1}$, $y=bc$ olabilir. Bu parçalanma ile üretilen kelimeler a, b, c sayılarının eşit olmasını garanti eder. Fakat $i=2$ durumunda.

$v^i a^{n+i} b^{n-i} bcb^{n-1}$ olur ve b ve c için blok yapısı bozulur bu nedenle $a^n b^n c^n$ problemine çözümz. Aynı zamanda uzunluk $p+1$ olmasından 2. kuralı uymaz.

Örnek: (tersi)

• $A = \{ww^R : w \in \{a,b\}^*\}$

olsun. Aslında bu palindrom demektir. Örneğin $w=abb$ olsun
o zaman $w'=bba$ olur bunları yan yana yazarsak $abbba$ olur.
Bu A dili context-free deriz.

$$S \rightarrow aSa / bSb / \epsilon$$

• $B = \{ww : w \in \{a,b\}^*\}$ olsun.

w 'yi yan yana yaz anlamındadır, $w=abb$ ise bizer $abbabb$
yazdırmanız istenir. Bu durumda ilk basta ϵ sunu düşürebiliriz:

$$\begin{array}{l} S \rightarrow AA \\ A \rightarrow aA / bA / \epsilon \end{array} \quad \begin{array}{l} \text{Buradaki ilk } A \text{ esnek bir kelime üretir. İkinci} \\ \text{ } A \text{ da bir kelime üretir. Bir önceki } \xrightarrow{\text{B}} \text{ ettiğini} \\ \text{hafızada tutamadığı için birinci ve ikinci } A, \\ \text{farklı bir kelime üretebilir. Bunun için } B \text{ di} \\ \text{context-free değıldır.} \end{array}$$

Dilin context-free olmadığını ispatlamak için pumping lemma kullanılır.

HWB'ının konusu burası. JFLAP üzerinde karşılığı var bu konunun.

Ortak örneklerden birini yapmanız isteniyor.

(Context-free pumping lemma seviyegini seq JFLAP'te)

————— HAFTA VII. SON ———

$\forall i \geq 1$

————— HAFTA VIII. SON ———

CHAPTER 4

- Turing Machines and the Church-Turing Thesis -

Bugünkü bilgisayarlar, karar verilebilirlik (decidability) kapsamında deðerlendirilirse bilgisayarların Turing makinesi seviyesinde olduğunu söyleyebiliriz.

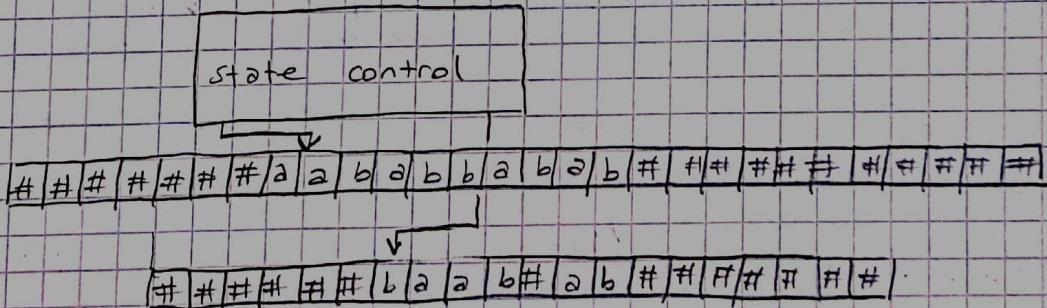
Gerçek bilgisayarlar ile çözülen her problem Turing makinesi ile de çözülebilir.

Turing Machine :

↳ (Church-Turing Tezinin özeti)

Bu zamana kadar gördüğümüz diğer yaklaşımlardan (DFA, NFA, RegEx, CFL, PDA ...) daha akıllı, programlanabilir ve dinamik bir yapısı vardır.

* En az 1 tane teyp'e sahip olmak zorunda. Teyp, hücreler bâlinmûstür ve uzunluğu sonsuz olarak kabul edilir. Her hücrede bir sembol bulunur. Bu sembol teyp alfabetesine aittir. (Γ). Teyp alfabesi boşluk sembolü (\square) veya (#) içerebilir. Bunun anlamı 'o hücre boştur.'



- 2 teypli Turing Makinesi gösterimi -

* Her teypte, teyp kafası vardır. Bu kafa her harekette bir hücreye gidebilir ve hareket teyp boyunca devam edebilir. Sağ'a, sola hareket edebilir veya olduğu yerde kalabilen

* State control, Q durum kümесini kontrol eden. Q kümесinde 3 tane özel durum vardır, başlangıç, kabul, ret.

* Deterministik Turing Makineleri 7 degerden olusur:

$$M = \{\Sigma, \Gamma, Q, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}\}$$

- Σ , giris alfabetesidir. # (blank symbol), bant alfabetesinde bulunmaz
- Γ , teyp alfabetesidir. # (blank symbol) icerir. $\Sigma \subseteq \Gamma$ (Σ , Γ 'nin alt kumesidir)
- Q , durumları ifade eder.
- q_0 , Q 'nın bir elemanıdır ve baslangic durumu olarak adlandırılır.
- q_{accept} , kabul durumudur.
- q_{reject} , ret durumudur.
- δ , gecis fonksiyonudur ($\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, N\}^k$) ($k = \text{teyp sayisi}$)
↳ Birden fazla teyp varsa; teyp sayısı kadar harf üzerine teyp sayısı kadar harf yazılır ve teyp sayısı kadar hareket yapılır.

* Baslangic durumunda, kullanıcının girdiği string ilk teyp'te bulunur. Teyp rafası kelimenin en sonuna yerleştirilir. Diğer $k-1$ kadar teyp boştur.

(Yani blank symbol icerir). Turing makinesi q_0 baslangic durumundadır.

* Hesaplama ve Sonlanma, Turing makinesi kabul veya ret durumuğa gelince hesaplama sond erer. (hesaplamanın sonlanmama durumu da vardır)

* Kabul, w kelimesi sonlandığında kabul durumunda bulunuyorsa Turing makinesi, kelime kabul edilir.

Ret durumu veya sonsuz dongü versə kelime reddedilir.
(sonlanmama)

* Turing makinesi, regular language ve context free language'i kapsar.

Önekler:

1-) Bir Teypli Palindrom :

* Teyp kafası en soldaki simbolü okur, bunu siler ve hatırlar. Daha sonra en sağdaki simbol ile hali hâzırda silinenin en sol simbolü karşılaştırır.

- Eğer aynıysa, teyp kafası yeni en soldaki simbole gider ve işlemeye devam eder.
- Değilse, Turing makinesi ret durumuna geçer ve hesaplamayı sonlandır.
- Teyp boşalırsa kabul durumuna geçilir.

Bu problem için sunları kullanacağız:

$$\Sigma = \{a, b\}, \Gamma = \{a, b, \#\}, \text{ Rümesi şu şekilde dir:}$$

q₀ : başlangıç durumu, teyp kafası en soldaki simbolün üzerindedir.

q_a : soldan okunan simbol a'dır

} teyp kafası sağa gider bundan sonra:

q_b : soldan okunan simbol b'dır

q̄a : en sağdaki simbol a ise bunu siler }

} burada sol ile eşit mi

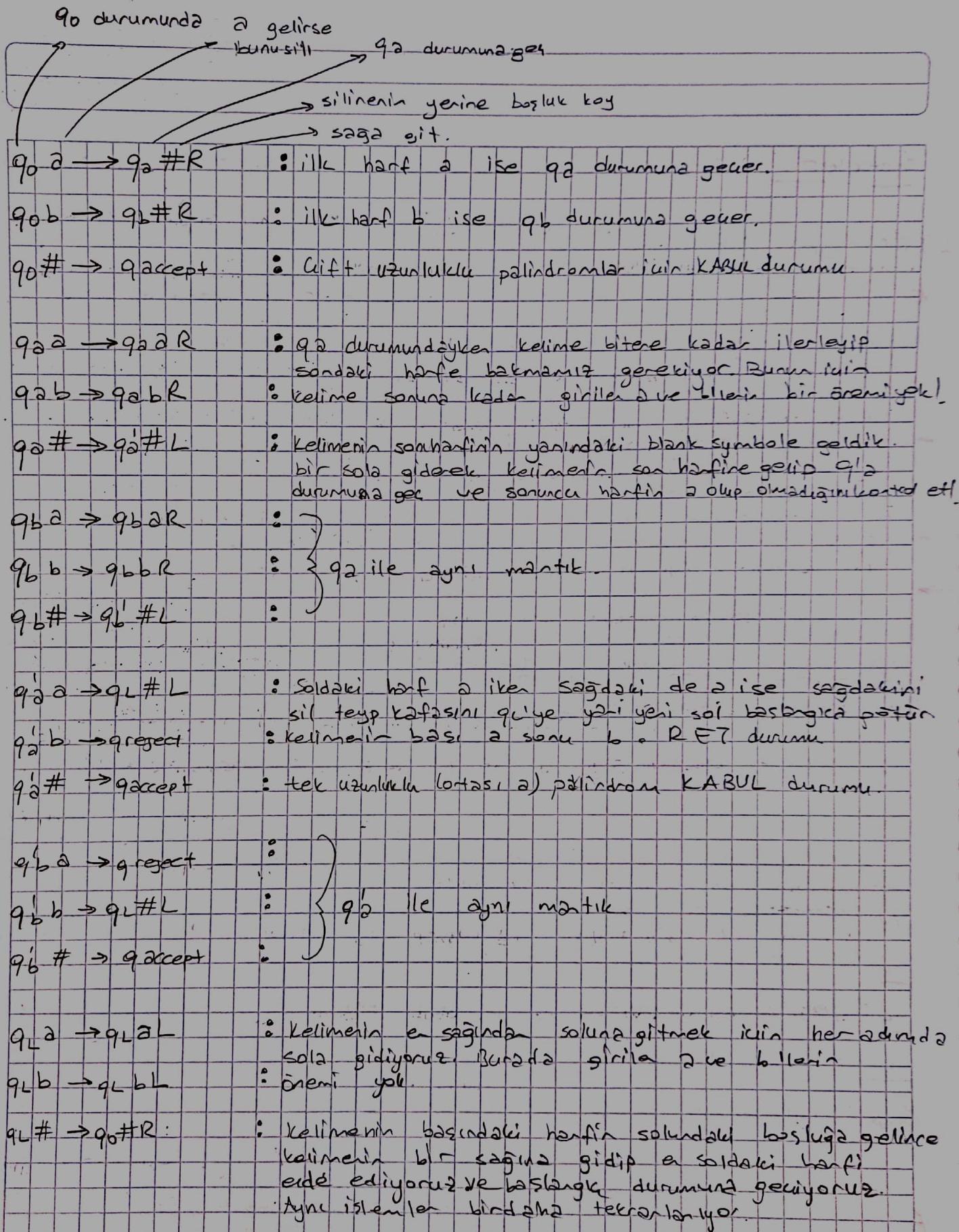
q̄b : en sağdaki simbol b ise bunu siler. } testi yapılır.

q_l : test sonucu pozitifse, teyp kafası yeni en sola gider.

q_{accept} : kabul durumu

q_{reject} : ret durumu.

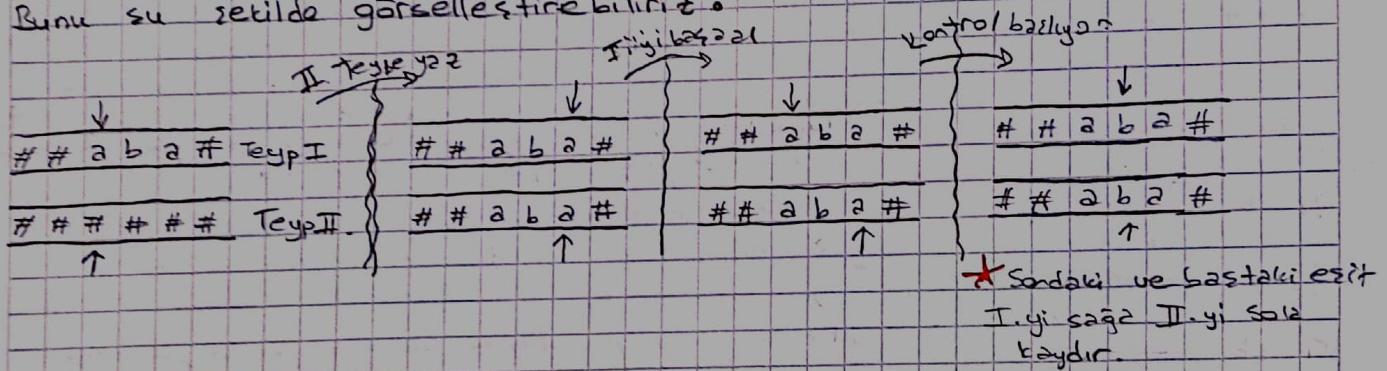
* Geçici fonksiyon δ diğer sayfada gösterilmisti



2-) İki Teypli Palindrom :

* ilk durumda, kelime birinci teyp'te yazılıdır ve ikinci teyp boştur. ve okuma-yazma kafası kelimenin ilk harfini gösterir. Öncelikle birinci teypten okunan her harfi ikinci teybe yazıyoruz. Kelimenin sonu geldiğinde iki okuma-yazma kafası da kelimenin sonunda bulunuyor. Birinci teybin okuma-yazma kafasını en sola getiyoruz. (ikinci teybin okuma-yazma kafası en sağda (kelimenin sonunda) kalsı.) Bu adımlardan sonra bir soldan bir sağdan kelimenin sonuna kadar okuyarak her harfin esit olup olmadığını kontrol edilir.

Bunu su şekilde görselleştirebiliriz :



Bu problem için sunları kullanacağız :

$\Sigma = \{a, b\}$, $\Gamma = \{a, b, \#\}$, Q kümesi su şekilde dir :

q₀ : Başlangıç durumu. w stringini II. teybe kopyalar.

q₁ : iki okuma-yazma kafası da sağdadır. I. teybintini en sola get

q₂ : Kontrol aşaması. I. teyp adım adım sağa, II. teyp sola doğru gider.

q_{accept} : Kabul durumu

q_{reject} : Ret durumu.

* δ geçiş fonksiyonu diğer sayfada.

I. teypten II. teypten

q_0 durumunda $\alpha \xrightarrow{\quad} \#$ gelirse q_0 durumuna gel. α $\xrightarrow{\quad}$ I. teype II. teype

α yaz ve ikisini de sağa hareket ettir.

$q_0 \alpha \# \longrightarrow q_0 \alpha \alpha RR$: Birinci teypten okunan değer α ise, birinci ve ikinci teyplere α yaz ve ilerle.
$q_0 b \# \longrightarrow q_0 b b RR$: Birinci teypten okunan değer b ise, birinci ve ikinci teyplere b yaz ve ilerle.
$q_0 \# \# \longrightarrow q_1 \# \# LL$: Kelimenin sonuna gelindiğinde. (II. teypde yazma işi bitti)

$q_1 \alpha \alpha \longrightarrow q_1 \alpha \alpha LN$:	q ₁ durumuna ilk gecildiğinde her iki okuma yazma kafesi dır. Birinci teybin okuma yazma kafasını en sola almak için sadece birinci teybin kini sola doğru hareket ettiriyoruz. II. teyp sabit. Herhangi bir işlem yapmadığımız için okunma değerleri tekrar yerine yazıyoruz.
$q_1 \alpha b \longrightarrow q_1 \alpha b LN$:	
$q_1 b \alpha \longrightarrow q_1 b \alpha LN$:	
$q_1 b b \longrightarrow q_1 b b LN$:	

$q_1 \# \alpha \longrightarrow q_2 \# \alpha RN$:	Birinci teyp kelimenin en solundaki kelimenin bir solundaki boşluğunca gelmiş. Bunu bir sağa kaydırarak kelimenin başına alıyoruz. Artık tüm sistem kontrol için hazır.
$q_1 \# b \longrightarrow q_2 \# b RN$:	
$q_1 \# \# \longrightarrow q_{\text{accept}}$:	

$q_2 \alpha \alpha \longrightarrow q_2 \alpha \alpha RL$:	En soldan ve en sağdan okunmlar α ise I. teybi sağa II. yi sola kaydır ve devam et.
$q_2 \alpha b \longrightarrow q_{\text{reject}}$:	

$q_2 b \alpha \longrightarrow q_{\text{reject}}$:
$q_2 b b \longrightarrow q_2 b b RL$:

$q_2 b b \longrightarrow q_2 b b RL$:	En soldan ve en sağdan okunmlar b ise I. teybi sağa II. yi sola kaydır ve devam et.
$q_2 \# \# \longrightarrow q_{\text{accept}}$:	

★ Ayni problemi tek teygli hizimiz ile karşılaştırılınca, teyip sayısı arttı fakat durum sayısı azaldı, geçiş fonksiyonundaki adım sayısı azaldı, algoritmus kolaylaştı.

3-) $\Sigma^n b^n c^n$ (Extra Teyp Alfabesi karakteri ile) :

* $\{\Sigma^n b^n c^n, n \geq 0\}$.

Bu problemede teyp alfabetesine bir karakter ekliyoruz. Bu nöf 'd' olsun.

Bir 'a' bir 'b' bir 'c' silindiği zaman bunların yerine d koymuyoruz.

Bunu yapma amacımız ; harflerin silinmesi halinde kelimenin basındalcı blank symbolün kaybedilir. Bu önlemek istenmiştir.

Bu problem için euniti kullanacağız:

$\Sigma = \{a, b, c\}$, $\Gamma = \{a, b, c, d, \#\}$, Q kumesi su sekilde dir:

q_a : Baslangıç durumudur. En solda 'a' yi arastırır.

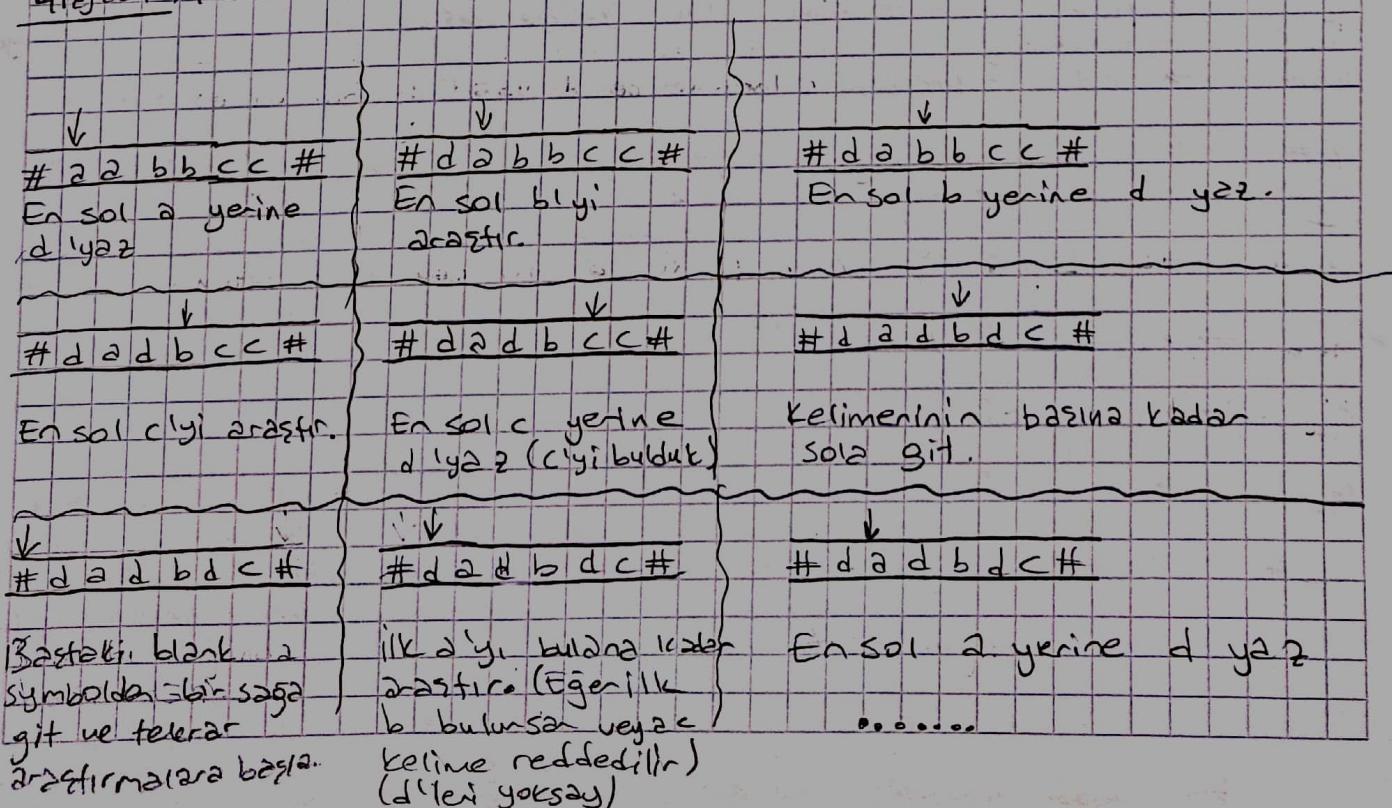
q_b : En soldaki 'a' yeine d yerlestikten sonra buraya gelinir. Bu durum en sol 'b' yi arar.

q_c : En sol 'a' yeine d koymulca ve en sol 'b' yeine d koymulca buraya gelinir. Bu durum en solda 'c' yi arastırır.

q_L : a, b, ve c'lerin en solu bulunduktan sonra tekrar okuma yapmak için okuma yazma kafasını en sola götürür.

qaccept: kabul durumu.

qreject: Ret durumu.



δ geçiş fonksiyonu şu şekilde dir :

- $q_a \bar{a} \rightarrow q_b d R$: En soldaki \bar{a} 'yi bulduktan sonra durumuna geçip b'leri aratır.
- $q_a b \rightarrow q_{\text{reject}}$: Kelimenin en başında a ' yok. Kelime b ile başlıyor.
- $q_a c \rightarrow q_{\text{reject}}$: Kelime c ile başlıyor.
- $q_a d \rightarrow q_a d R$: Kelimenin en solundaki a (veya a' ler) silinip yerine d yazılmış. İlk a 'yi arastırmaya devam et. d 'leri görme.
- $q_a \# \rightarrow q_{\text{accept}}$: Hiybir kelime girilmemisse ya da teypteki tüm harfler d olmussa kabul edilir.
- $q_b \bar{a} \rightarrow q_b \bar{a} R$: \bar{a} 'yi arastıryoruz fakat hala a bloğundan çıkışmadık.
- $q_b b \rightarrow q_c d R$: \bar{a} bloğundan sonraki ilk b 'yi bulduk.
- $q_b c \rightarrow q_{\text{reject}}$: b 'ler yok kelime \bar{a} ve c 'lerden oluşuyor. Ret.
- $q_b d \rightarrow q_b d R$: Önceden b 'ler yerine d yazılmış. Yeni en sol b 'yi bulmak için d 'leri görmezden gelip sağa git.
- $q_b \# \rightarrow q_{\text{reject}}$: q_b 'deyken kelime sonlanmış. Yani hiç b yok. Ret.
- $q_c \bar{a} \rightarrow q_{\text{reject}}$: Kelimenin blok yapısı bozuk ($a\bar{a}bb\bar{a}\dots$) gibi. Ret.
- $q_c b \rightarrow q_c b R$: İlk c 'yi arastıryoruz fakat hala b bloğundan çıkışmadık.
- $q_c c \rightarrow q_d d L$: En sol c 'yi bulduk kelime basına dönmek için q'ya gel.
- $q_c d \rightarrow q_c d R$: Önceden c 'ler yerine d yazılmış. Bu nedenle es geç.
- $q_c \# \rightarrow q_{\text{reject}}$: q_c 'deyken kelime sonlanmış. Yani hiç c yok.
- $q_L \bar{a} \rightarrow q_L a L$: $\left. \begin{array}{l} \text{hiybir harfi görme, kelimenin en soluna} \\ \text{ulaşmaya çalış.} \end{array} \right\}$
- $q_L b \rightarrow q_L b L$:
- $q_L c \rightarrow q_L c L$:
- $q_L d \rightarrow q_L d L$:
- $q_L \# \rightarrow q_{\text{accept}} R$: Kelimenin en solundaki harfin bir solundaki blank okudu. bir sağa giderken kelimenin basınıg et.

4- $a^nb^nC^n$ (Ekstra harf olmadan) :

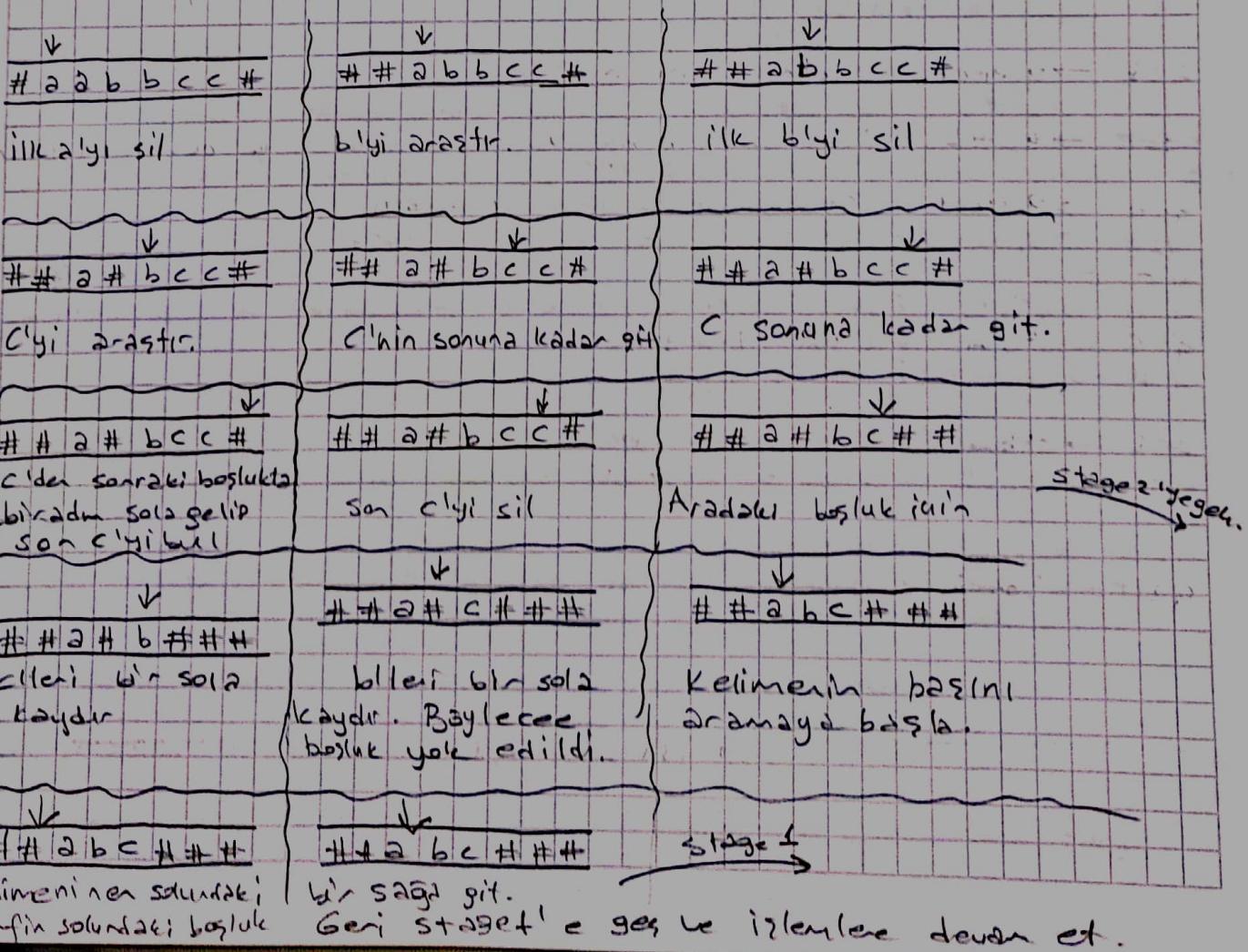
★ Ekstra teyp harfi olmadan daha uzun adımlar ve daha fazla durum kullanarak yapacagız. Bunu yaparken 2 stage kullanacagız.

Stage 1:

En soldaki a'yi silip b'ler arastir. En soldaki b'leri silip c'leri arastir. En sağdaki c'yi sil. Bunu yaparken blok kontroleini yap. Tüm bu islemler yapıldıktan sonra a ve bc ler arasında bir tane boşluk olacak. Bu boşluğun kapatmak için Stage 2'ye gec.

Stage 2:

Sıkıştırma istemi yapacagımız stage. b'lerin birer sola kaydırarak silme sırasında oluşan boşluk yok edilmesi olur.



$q_0 a \rightarrow q_0 \# R$: a bloğunun en sondakı a'yı sil.
$q_0 b \rightarrow q_{reject}$: b ile başlama durumu. RET.
$q_0 c \rightarrow q_{reject}$: c ile başlama durumu. RET.
$q_0 \# \rightarrow q_{accept}$: ta hıfız harf girilmedi ya da aranınca olduğu için hepsi silindi tek tek kabul.
$q_1 a \rightarrow q_1 a R$: l'lerin arastırıyonum ama hala a bloğundayım
$q_1 b \rightarrow q_1 \# R$: ilk b'yi bulup sildik.
$q_1 c \rightarrow q_{reject}$: b arastırırken c bulduk. RET.
$q_1 \# \rightarrow q_{reject}$: b arastırırken kelime bitti RET.
$q_2 a \rightarrow q_{reject}$: c arastırırken a bulduk blok yapısı bozukken RET.
$q_2 b \rightarrow q_2 b R$: c arastırıyonuz fakat hala b bloğundayız.
$q_2 c \rightarrow q_{reject}$: ilk c'yi bulduk. Son c'yi bulmak için q'ye git.
$q_2 \# \rightarrow q_{reject}$: c ararken kelime sonlandırsa. RET.
$q_3 a \rightarrow q_{reject}$: sondaki c'yi ararken a bulduk. blok yapısı bozuk.
$q_3 b \rightarrow q_{reject}$: sondaki c'yi ararken b bulduk. blok yapısı bozuk.
$q_3 c \rightarrow q_3 C R$: c bloğundayız hala ve sei c'yi arıyorumuz
$q_3 \# \rightarrow q'_3 \# L$: san c'ken sonrası boşluğun bulduk bir sola git.
$q'_3 C \rightarrow q'_1 \# L$: son c'yi sil ve stage 1'ye git.

* Olusamayacak durumlar yazılmasının (örn: $q_1 a$ gibi. Blok yapısının korundugunu stage 1'de sağla此文)

$q_1 b \rightarrow q_{reject}$: c'lerin sayısı ≥ 2.
$q_1 c \rightarrow q'_1 \# L$: En sağdaki c'yi silip kaydirmak için hafızada tutuyoruz.
$q'_1 \# \rightarrow q'_1 \# L$: Birer tane a bc veya en sağ c'yi okunduktan sonra boşluğunu gormes ols.
$q'_1 a \rightarrow q_{reject}$: Eğer fazladan a varsa reddedilir.
$q'_1 \# \rightarrow q_{accept}$: Eğer birer tane a b c varsa kabul edilir.
$q^c b \rightarrow q^b c L$: ilk gördüğü b'yeine c yaz. Bu kez de hafızada kayıtlı.
$q^c c \rightarrow q^c c L$: c'lerin pas geç
$q^c \# \rightarrow q_{reject}$: b'lerin sayısı ≥ 2.
$q^b b \rightarrow q^b b L$: b'lerin pas geç.
$q^b \# \rightarrow q^b b L$: a ve b arasındaki boşluğun b'yi yaz
$q_2 a \rightarrow q_2 a L$: kelimenin en başına dönmek için en sola kay
$q_2 \# \rightarrow q_0 \# R$: kelimenin başını buluncaya q0 yani stage 1'e git.

Church - Turing Thesis :

Genel olarak, bir problemin çözümünü düşünürebiliyorsak bu problemi herhangi bir programlama dilinde kodlayabiliriz.

Bu tez bir problemin yapılabilirliğini (bilgisayarda çözülebilirliği) test eder.

CHAPTER 5

- Decidable and Undecidable Languages -

Bir problemin çözümünün sonsuz zamanda çözülemez olması "undecidable", çok uzun sürede bir sonuca ulaşmamızda "decidable" denir.

Decidability

Eğer bir problem, bir algoritma tarafından çözüleyorsa ; algoritma verilen doğru inputlarda kabul, yanlış inputlarda reti üretmeli ve bu işlemleri de garanti ediyorsa bu probleme 'decidable' denir.

DFA:

$$\text{ADF} = \{ \langle M, w \rangle : M \text{ DFA'sı } w \text{ stringini kabul eder} \}$$

\downarrow
algoritma
 \rightarrow input
(DFA)

DFA'lar decidable sınıfında girer. Çünkü w stringi algoritmanın yapısını uygunsa kabul edilir, uygun değilse reddedilir ve bu işlemler DFA tarafından garanti edilir. Kelime ya kabul edilir ya da reddedilir. (İkinci bir seçenek yok). Kelime ne kadar uzun olursa olsun her türlü sonlanır.

* Aslında bundaki kavram 'Turing-decidable' dir. Yani bir eşiği Turing'de kodlaşımında durup durmamasına bekliyoruz.

NFA: NFA problemleri de decidableldir. Çünkü NFA'lar.

DFA'lar dönüşebilir ve tüm DFA'lar decidableldir.

* ANFA \rightarrow ADFA \rightarrow Compiler for ADFA

CFG: CFG'lerin sonsuz döngüye girme ihtimali vardır. Sonsuz döngüye girme ihtiyalının olması undecidable olduğunu anlatır. Fakat CFG undecidable degildir. Çünkü her CFG bir şekilde Chomsky formuna dönüşebilir. CFG'ler insan çözümü için uygun, makine çözümü için uygun degildir. Ama Chomsky Normal Formu makine için uygundur ve CFG, Chomsky'e dönüşügorsa demekti CFG'ler de makineler tarafından çözülebilir. Bu nedenle CFG'ler decidable dir deriz. ($CFG \equiv \text{Chomsky}$) (Chomsky durumları garanti eder)

* ACFG \rightarrow AChomsky \rightarrow Compiler for Chomsky

TM: TÜM'ler, günümüzde kullandığınız derleyicileri temsil eden örneğin bir C kodu yazarken bilecek veya bilmeyenek bir sonsuz döngü oluşturabiliyoruz. Bu sebeple TM'ler undecidable dir deriz.

Derleyicilerde TM seviyesinde bir dil ile yazıldı. Derleyicinin iline yazdığımız kodlar da TM seviyesinde bir dildir. Dolayısıyla TM seviyesindeki bir derleyici, TM seviyesindeki bir programın decidable olup olmadığını anlayabiliyor ve kapasitede değildir.

Denetlenme için bir üst seviyeye ihtiyaç vardır. TM, CFG'yi, DFA'yı, NFA'yı yönetebilir/denetleyebilir.

TM'nin decidable olmadığını ispatı :

TM decidable olsun,

Decidable olduğuna göre TM'yi denetleyen bir 'H' derleyici si olsun.

Derleyici iinine yazdığımız kod ' M ', input w olsun.

- Eğer $\langle M, w \rangle \in A_{TM}$ ise (w , M tarafından kabul edilirse) H , M 'nin, w 'yi kabul edeceğini karar verip accept state'de durur.
- Eğer $\langle M, w \rangle \notin A_{TM}$ ise (M , w girilince accept te durmaz,) H (ya rejectte durur yad sonlanmaz) derleyicisi, sonsuz döngüye girmeden, M programının w üzerinde durmayaçağın belirler.
- Sonuç olarak H derleyicisi her türlü sonlanır.
 - (Yukarıdaki • işaretleri varsayımlıdır)
 - H derleyicisinin tersi olan bir D derleyicisi yapalım. Yani rejectte duruyorsa veya sonsuz döngüye giriyorsa kabul etsin, accept te duruyorsa reddetsin. (H 'nin tam tersi) ($\neg H \equiv D$)
 - M programına input olarak kendisi verilirse yani $\langle M, \langle M \rangle \rangle$ kabul edilirse ($H \rightarrow \text{accept}$, $D \rightarrow \text{reject}$) sonucunu verecektir. M programı kendi kodlarını reddederse veya sonsuz döngüye girerse ($H \rightarrow \text{reject}$, $D \rightarrow \text{accept}$).
 - * ($Bu H$ ve D $\langle M \rangle$ 'yi kabul veya reddediyor) (Burada H aslın M oluyor) (yani derleyici!)
 - Yukarıdaki kodları kullanarak, $M=D$ yapabiliyor. Sonunda her ikisi de bir program. Böyle bir durumda $D, \langle D \rangle$ 'yi kabul ederse, $D \langle D \rangle$ 'yi reddeder. $D, \langle D \rangle$ 'yi redderse veya sonsuz döngü olusa, $D \langle D \rangle$ 'yi kabul eder.

Halting Problem:

TM seviyesinde, durabilen tüm programlar Halting Problem dâhilindedir.

TM seviyesindeki bir problemin her zaman durmaya garanti edemediği ile ilgili bir ispattır.

$$H(P, w) \begin{cases} \text{true} & \text{if } P(w) \text{ sonlanırsa} \\ \text{false} & \text{if } P(w) \text{ sonlanmazsa} \end{cases}$$

(Halt)

Algorithm Q ($\langle P \rangle$):

```

while H( $P, \langle P \rangle$ ) = true
    do gray-iü
endwhile
  
```

* Q derleyicisine parametre olarak P programının kodlarını verdiğ.

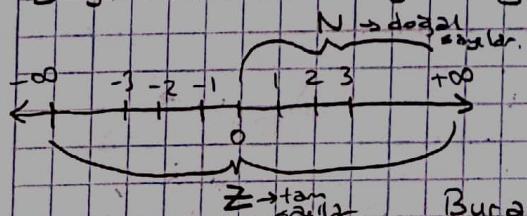
* P programı kendi kodlarını çağırıyor ve eğer sonlanırsa Halt problemi true olacağını dengüye girilen $\langle P \rangle$ degişmediği için birazda bir sonsuz döngü olusur.

→ Yani P , $\langle P \rangle$ üzerinde sonlanıyorsa, $Q, \langle P \rangle$ üzerinde sonlanmaz / durmaz. Burada Q 'da P 'de bir programdır. Yani P yerine Q yazabilirim. Bu durumda $Q, \langle Q \rangle$ üzerinde sonlanıyorsa, $Q, \langle Q \rangle$ üzerinde sonlanmaz / durmaz. (Gelişti)

* * Bu durumda H isminde bir fonksiyonu yazılamayacağı, ispatlanmış olun. Böylece bunu genelleserek, ikiine yazılık kodun sonanıp sonlanmayacağıni belirleyen bir derleyici yazılamaz.

Countable Sets

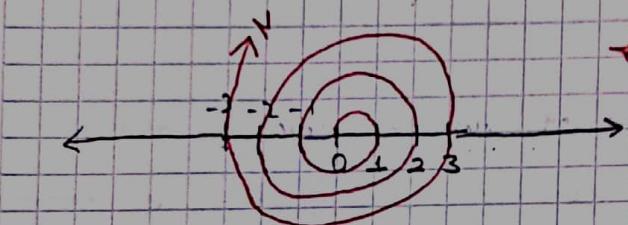
Sayı kümelerinin doğal sayılar ile sayılabilirliğini temsil eden bir terimdir.



* Sonsuz bir sayı olarak düşününce \mathbb{Z} kumesi, N 'nin yaklaşık 2 katı gibi görünüyor. Anna sonsuzuk işin içine girince öyle oluyor.

Burada N kümelerinin elementleri ile \mathbb{Z} kümelerini etiketlemeye çalışıyorum. Normalde bu mümkün değil gibi görünüyor. Fakat sonsuz kavramının esnekliğini kullanarak bunu yapabiliyoruz. ($+\infty$, $2\pi r$ ile aynı şeytedir) (\mathbb{Z} ligi 200 olarak düşünüyorum).

\mathbb{Z} yi \mathbb{N} ile numaralandırmak için su yöntemini kullanabiliriz.

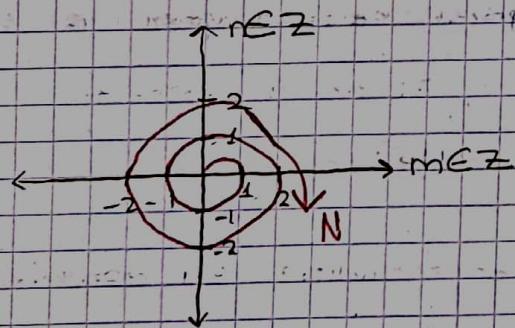


* Etiketlemeyi bir seferde bir soldan olacak şekilde tır spirál üzerinde yapayız. Bu sonuzda kaden dönebilir ve \mathbb{Z} yi kaplayabilir.

$$f(N)=z \text{ ise, } f(0)=0, f(1)=1, f(2)=-1, f(3)=2, f(4)=-2, \dots$$

- Yukarıdaki gibi bir etiketlemeyi tüm rasyonel sayıları kapsayıpcaz şekilde yapabilir miyiz? (Rasyonel sayı: İki tam sayıın birbirine bölünür)

$\mathbb{Q} = \{m/n : m \in \mathbb{Z}, n \in \mathbb{Z}, n \neq 0\}$ iki eksen kullanarak bunu temsil edebiliriz.



→ Bu durumda bir N ile无限 tane sonsuz kapsamı oluyoruz.

* Bu şekilde \mathbb{Z} ve \mathbb{Q} kümləri countable'dır deriz.

- Bu zigzag teoremini (etiketlemeyi) Reel sayılar için kullanamıyoruz.

Cantor's Diagonal Argument (Reel sayıları spiralle sayınız)

Basamak sayıları

	0	1	2	3	4	...
0.	1	3	5	4	...	
0.	0	8	9	7	...	
0.	2	1	7	0	9	...
0.	9	5	0	9	...	
0.	7	0	6	8	...	
...

* Basamak sayıları sonsuz olan sonsuz tane sayı yazılır.

* Sonsuz bir sayı gibi düşünelim olursak soldaklı setin bir kare matrisi olur. Böyle bir durumda kare matrisin köşegenini kullanabiliriz. (Yeni bir sayıının üretilemeyeceğini belirtmek istiyorsak)

* Bu tablo birazın tüm reel sayıları, 0 ile ∞ arasındaki reel sayıları bire temsil ederler. (sayamaz)



* Bu tablo birazın tüm reel sayıları, 0 ile ∞ arasındaki reel sayıları bire temsil ederler. (sayamaz)

* Köşegen üzerindeki sayıları bir artırarak ($1 \rightarrow 2, 9 \rightarrow 0$) dördüncü matriste yazılmamış bir sayı üretmeye çalışıyoruz.

0.2980.... \Rightarrow birinci sayıdan kesinlikle farklı olduğunu söyleyebiliriz. Bunu böyle devam ettirerek tablodaki hiçbir elemanla aynı olmadığını gösterebiliriz.

★ Eğer L dili decidable ise \bar{L} 'de decidable'dur.

$$L \in \Pi \leftrightarrow \bar{L} \in \Pi$$

Mantıqlı ebu: Program duruyorsa (ya kabul, ya ret) decidable'dur. L dili decidable ise duruyordur. (terminate oluyordur.). \bar{L} , $L \rightarrow \text{accept} \Rightarrow \text{reject}$, $L \rightarrow \text{reject} \Rightarrow \text{accept}$ te duruyordur. Sonuç olarak durduğu için decidable'dır.

Enumerability :

★ Decidable olan her dil enumerable'dür.

Bir problem'in accept'te duran bir parçasına veya reject'te duran parçasına verilen isimdir.

- Eğer bir dil enumeratore sahipse enumerable'dir deniz.

→ Asal sayı probleminin çözümünde sunu söyleyebiliriz: Bu problem reject'te duran kısmı (asal olmayan sayılar) enumerable dir. Çünkü sayının kendisi sonsuz kadar büyük bile olsa onun asarpalanın ararken sonsuz kadar gitmemize gerek yoktur. Herhangi bir asarpanını bulduğumuzda reject'e gidecektir. Yani bir sayıının asal olduğunu bulmak, asal olmadığını bulmaktan daha kolaydır.

Decidable ve Enumerable Arasındaki İlişki:

- * A dili decidable'dır ancak ve ancak A ve \bar{A} her ikisi de enumerable ise.
- * Problemi iki paruya bölyoruz accepte durmayı ve rejectte durmayı garanti ediyorsak decidable'dır. Eğer -ikisinden birini veya her ikisini garanti edemeysek decidable degildir.

$$A \in D \leftrightarrow (A \in E \text{ } \underset{\text{decidable}}{\cap} \text{ } A \in E)$$

$$A \in D \leftrightarrow \bar{A} \in D \quad (\text{Bunu önceli sayfada gördük})$$

$$\Rightarrow A \in D \rightarrow A \in E \quad (\text{Böyle bir şey söylemek mümkün mü?})$$

\rightarrow (A decidable ise enumerable'dır)

* Bu doğru bir ifadedir.

HAFTA XII. SON

CHAPTER 6

- Complexity Theory :-

Decidable problemlerin sınıflandırılmasında kullanılır.

P Karmasılık Sınıfı :

Polinom zamanında deterministik makinelerle çözülebilen problemlerdir.

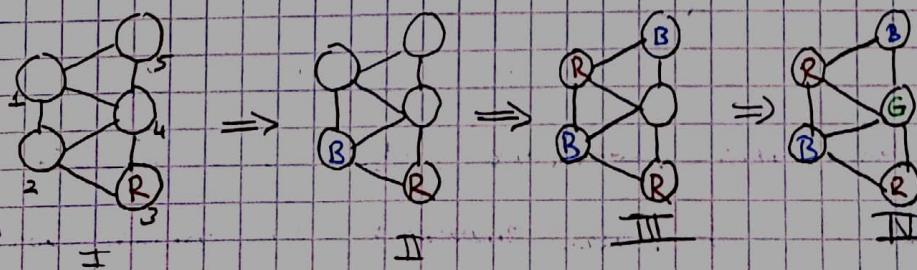
* Polinom zamanında deterministik olmayan makinelerle çözülebilen (henüz böyle bir makine yok) problemler ise NP karmasılık sınıfına girer.

* Kolay (P), Zor (NP) diye ayıralım. P problemleri, sonucunu bir insan ömrü süresi içinde görebileceğimiz, NP problemleri çok daha uzun süren problemlerdir. (Her ikisi de sonuçlanır). $x^2 \rightarrow P$, $2^x \rightarrow NP$ denebilir.

* Java programı, tek tayıplı Turing Makinesi, k-tayıplı Turing Makinesi birbirine denktir. Bir problem, Java'da polinom zamanında yazılabilirse diğerlerinde de polinom zamanında yazılabilir. Birinde P sınıfında olsa diğerlerinde de zittir.

2-coloring problemi : (k-coloring)

Renklendirme probleminde düğümler boyanır. Komşu olan düğümlerin, aynı renk olmasını dikkat edilir. Minimum sayıda renk kullanarak bu problem çözülmeye çalışılır. (2-coloring problemi: verilen düğümü, yukarıda verilen şartlara göre, iki renkle boyayıp - boyamaya çalışılmıştır.)



Gördüğün gibi bu graph'ı boyamak için en az üç renk gerekiyor.

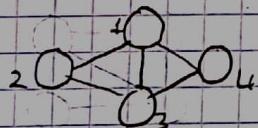
* III adımda 5. node R'de olabilirdi ama bu kez de 1. node için ayrı renk kullanılmak gerekti ve 4. renk kullanıldı.

* 2-coloring problemine graph verildiği zaman her zaman evet ya da hayırda sontanır. Dolayısıyla decidable'dır. P sınıfına mı NP sınıfına mı aittir o zaman? K-coloring probleminin çözümü kombinasyonel bir çözümdür. Genelde kombinasyonel çözümler NP sınıfıdır. Fakat 2-coloring'in dinamik programlama sayesinde polinom zamanında bir çözüm üretiliği için, 2-coloring problemi P sınıfıdır.

NP Kärmazılık Sınıfı:

K-coloring problemleri, NP sınıfına aittir idemiyet. Örneğin;

? Bu graphi 3-color için incelerse.



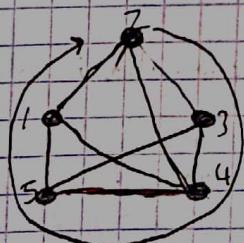
2. düğüm \rightarrow R, 4. düğüm (G veya B olabilir) G,

3. düğüm (Yalnızca B olabilir) B, 4. düğüm (Yalnızca R olabilir) R.

* Eğer bir yerde uygunuz bir renk olsaydı reküratif olarak geniye doğru gidip ibaska bir renk deneyip tekrar ilerlenirsiniz yani kombinasyon yapmanız gerekecekti.

Hamilton Cycle Problem: (HC)

Graph üzerinde cycle bulmaya çalışır. (Tüm düğümlerden geçmeye çalışıyoruz.)



* Böyle bir graph HC'yi sağlar

* Kırmızı keşf etmesi: 1 - 2 - 3 - 4 - 1 olacaktı,

aramaya devam edecektik, 1 - 5 - 3 - 4 - 1 olacaktı,

yine aramaya devam edecektik ve kombinasyonları

deneyecektik. Bu yüzden HC, NP problemidir.



The sum of subset Problem : (SOS)

$SOS = \{ \langle a_1, a_2, a_3, \dots, b \rangle \}$... a sayılarının toplamı b sayısını eder mi?

Örnek

1, 5, 9, 25, 40 } Amaç: inputlardan bazılarını veya hepsini
inputları hedef sayıya toplayarak hedef sayıya ulaşmak.
sayıları

* Bunu bulmak için deneme yapılıyor: 1. ve 2. sayıların toplamı hedef sayı mı? 1. ve 3., 1. ve 4., 2. ve 3. Kombinasyonları deniyor.

| Bu sebeple bu problemde NP problemdir

* Örneğin; $n=5$ için $\neq 0$ sayıye

$n=10$ için 1000000 sayıya sunuyorsa bu problem NP'dir.

Nprim Problem :

* Nprim problemi 2002 yılından önce NP, 2002 yılından sonra P problemi olarak sınıflandırılmıştır. ($N\text{ prim} = \text{Asal sayı olmayan sayılar}$)
(not prime)

* Sezgisel olarak, $P \subseteq NP$ diyebiliriz. (P , NP 'nin alt kumesidir.)

* NP problemlerinin bir uzunluğu (SOS problemi için bir kumesi: $\{1, 5, 9\}$ gibi)
polinom zamanda test edilebilir. Onu için $P \neq NP$ olapsa,

+ HAFTA XIII. SÖN

Non-Deterministik Algoritmalar:

Eğer bir non-deterministik Turing makinesi (veya Java programı) olsaydı NP problemi, bu makine ile polinom zamanda çözülebilirdi.

NP-complete:

NP problemleri 'zor' problemlerdir. Bu sınıf içindeki problemlerin bazıları çok zor, bazıları daha az zordur. En zor denilen problemler, NP-complete, NP-hard sınıflarına aittir.

* NP-complete sınıfında ait olan A ve B problemleri, birbirine polinom zamanda dönüştürülebilir. Yani

$$B \xrightarrow{\text{indirgeme}}_{P \text{ zamanda}} A$$

* Eğer ilerde A için polinom zamanda bir çözüm bulunursa: yani A problemi P sınıfına gelirse, dolaylı yoldan B problemi de P sınıfına gelmiş olur.

* $A \leq_p B$ (Polinom zamanda A, B'ye dönüştürülebilir) (B , en az A kadar zor bir problemdir)

Knapsack Problem: (KP) (NP problemidir)

$K := \{(w_1, w_2, \dots, w_m, k_1, k_2, \dots, k_m; N, k)\}$ Sos'un genişletilmiş halidir.

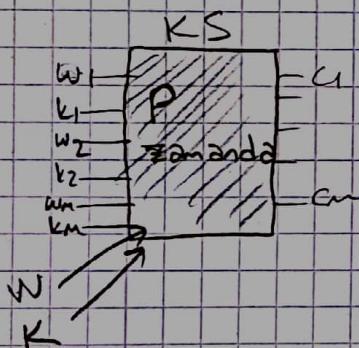
i 'inci grup sayısın toplamının 2 farklı sayıya vermesi isteniyor.

→ Buna su şekilde örnek verebiliriz: Çantaya 10 kilo ve 1500 kalori olacak şekilde yiyecek almalıyız diyelim. Böyle bir durumda hem ağırlıkları hem de kalorilerini mikrelereiz gerekiyor.

(Burada da tüm altsetleri düşünüyoruz.) (w ve k'lar, bitlikte düşün)

İndirgene

Eğer KS problemi için polinom zamanında bir çözüm bulunursa,
SOS problemi de dolaylı yoldan P sınıfına gelir.



(c_i ler kombinasyonları temsil ediyor)

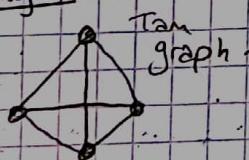
* Bunu SOS için uygularsak $w_1 \rightarrow z_1$, $k_1 \rightarrow z_1$,
 $w_2 \rightarrow z_2$, $k_2 \rightarrow z_2$..., $w \rightarrow b$, $k \rightarrow b$ yazılır.

Clique Problem: (NP problemidir)

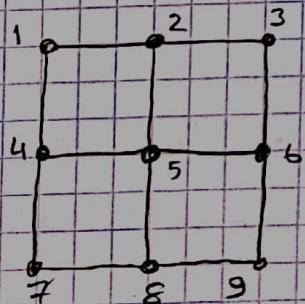
Graph üzerinde tanımlanır. Graph üzerindeki en büyük tam graphın düğüm sayısını verir. (Yani graph üzerinde arama işi yaparız)

Tam graph: Tüm elemanları birbirine bağlı olan graph.

Örneğin:



Örnek:



* K_3 ararken 1'in komşularına bakılır (2-4)

2 ve 4'ün arasında yol olmadığı

için K_3 'ü sağlamaž. Aramaya devam edilir.

Bir graphta K_3 yoksa K_4, K_5, \dots 'de yoktur.

3SAT Problem: (satisfy)

Clique probleminin karmazlığı $n!$, 3SAT'ın ki 2^n . Yani clique daha zor. Clique problemini polinom zamanında çözən bir algoritma, 3SAT'ı da çözər. Dolayısıyla 3SAT'ı, clique'ye dönüştürmeliyiz.

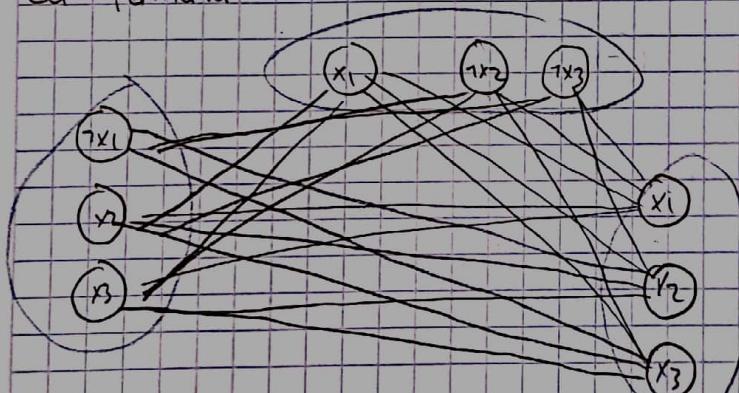
$3SAT \leq_p \text{Clique}$

kombinasyonlar (satisfying)
 $\{Q = C_1, C_2, \dots, C_k\}$

3 kısında olustuğu için 3SAT denir

$$P = (x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

Bu formülü



• Gruplar or, bağlantılar and işlevini ifade eder.

• Her eleman (tersler hariç) birbirine bağlıdır.

• Artık clique problemini bu graph'ta araştırabiliriz.

En fazla K_3 var.

* Bir de POS (product of subset) var. (SOS' ile aynı mantık
sadece çarpma yapıyor). SOS ve POS dönüşümüne yaparak
logaritma kullanabiliyoruz. (Toplama ve çarpma arasındaki genisi sayları)

$$\rightarrow \log_2 2 \cdot 2 = \log_2 2 + \log_2 2 \text{ gibi.}$$

($POS \rightarrow SOS$ 'a dönüştürülebilir
her ikisinden de 2 ve 6'ların
logunu alırsak)
($SOS \rightarrow POS$ yaparsaksa 65'de 21)

HAFTA XIV SON

HAFTA XV SON

DÖNEM SONU