Regüler İfadeler ve Regüler Diller

Hafta 3

- Dilin izin verdiği sözcükleri tanımlamak için kullanılan araçlardan biri Regüler ifadelerdir.

Genel tanımlar:

- **Alfabe** sonlu simge/karakterler kümesi -- {a,b}, ASCII
- **Katar** sonlu simgeler dizisi (sözcük) – 001, bc, baba, while
- **Uzunluk** (|x|) katarı oluşturan simge sayısı
- **Boş katar "Λ"** uzunluğu sıfır olan katar
- **Bitiştirme** iki katarı birbirini izleyecek şekilde birleştirme
  - x= abc y= de → xy=abcde
  - Λx=x Λ=x
  - x^n x katarı kendisiyle "n" kez bitiştirilir, x^0 → Λ

- Bitiştirme (Concatenation):
  - $L_1L_2 = \{ s_1s_2 \mid s_1 \in L_1 \text{ and } s_2 \in L_2 \}$
- Birleşim (Union)
  - $L_1 \cup L_2 = \{ s \mid s \in L_1 \text{ or } s \in L_2 \}$
- Üs (Exponentiation):
  - $L^0 = \{\varepsilon\}$ $L^1 = L$ $L^2 = LL$
- Yıldız Kapanma (Kleene Closure):
  - $L^* = \bigcup_{i=0}^{\infty} L^i$
- Pozitif Kapanma (Positive Closure)
  - $L^+ = \bigcup_{i=1}^{\infty} L^i$

- **Dil sonlu sayıda simgeden oluşan** bir alfabeden üretilen katarlar kümesidir.
  - {Λ} – boş katarı içeren küme
  - {0,1,00,01,10,11}
  - Doğal diller, programlama dilleri
- **Bitiştirme işlemi:** L ve M iki ayrı dil olmak üzere, LM dilinin elemanları, L'nin içerdiği tüm katarların M'nin içerdiği tüm katarlarla bitiştirilmesinden oluşur.
  - L={0,01,110} M={10,110},
    LM={010,0110,01110,11010,110110}

- L_1 = {a,b,c,d} L_2 = {1,2}

- L_1L_2 = {a1,a2,b1,b2,c1,c2,d1,d2}

- $L_1 \cup L_2$ = {a,b,c,d,1,2}

- L_1^3 = üç uzunluklu bütün katarlar kümesi (a,b,c,d üzerinde tanımlı}
- L_1^* = a,b,c,d simgeleriyle oluşturulanbütün katarlar (Λ (empty string) dahil)
- L_1^+ = boş katarı içermez.

- Kleene Yıldızı Operatörü (*) İstenilen sayıda (belirsiz sayıda) bitiştirme için kullanılır.

- L^* L dilinin kendisiyle belirsiz sayıda bitiştirilme işlemi
  - D= 0,1,2...,9 D^* → rakamlardan oluşan tüm rakamlar, 0 dahil
  - L={aa} L^* → çift sayıda "a" karakterinden oluşan tüm katarlar

  - L^0={}, L^1={aa}, L^2={aaaa}...
- L^* "ε" da içerir. "ε" dışlamak için LL^* yazılmalıdır

- $L^+ = LL^*$ en az bir veya daha fazla sayıda bitiştirme işlemi

- Bir regüler ifade şöyle tanımlanır

  (R ve S regüler ifadeler olmak üzere)

  - a — Alfabenin her simgesi bir regüler ifadedir.
  - ε — boş katar bir regüler ifadedir
  - R+S — "R veya S" bir regüler ifadedir
  - RS — "R ve S" (bitiştirme) bir düzgün ifadedir
  - R* — 0 veya daha fazla R'nin bitiştirilmesiyle elde edilen bir regüler ifadedir.

Dil

- Bir R regüler ifadesi, L(R) nin ifade ettiği karakter katarlarını (sözcükler) tanımlar.
- L(R) = R'nin tanımladığı dil
  - L(abc) = { abc }
  - L(onay|red) = {evet|hayır}
  - L(1(0|1)*) = 1 ile başlayan tüm ikili sayılar
- Her sözcük bir düzgün ifade kullanarak tanımlanabilir

Örnek Düzgün İfadeler

- Düzgün İfade ..... L(R) dilinde örnek katarlar
  - a ..... “a”
  - ab ..... “ab”
  - a|b ..... “a”, “b”
  - (ab)* ..... “”, “ab”, “abab”, ...
  - (a| ε)b ..... “ab”, “b”
  - (aa|ab|ba|bb)* ..... “aa”, “bbbb”, “abbbaababb”
  - (a|b)(a|b)(a|b) .... “bbb”, “aab”, “bab”, “abb”..

{a,b,c}'de tanımlı ve içindeki her 'a'dan önce VE her 'b'den sonra en az bir tane 'c' içeren katarları ifade eden Rİ

$$(ca+bc+c+bca)^*$$

{a,b,c}'de tanımlı ve içindeki b'lerin sayısı ile c'lerin sayısı toplamı 3 olan katarları ifade eden Rİ

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

{a,b,c}'de tanımlı ve içinde 'aa' altkatarı bulunmayan katarları ifade eden Rİ

$$(ab+ac+b+c)^*(a+\Lambda)$$

{a,b}'de tanımlı ve içinde 'aaa' altkatarı bulunmayan katarları ifade eden Rİ

$$(b+ab+aab)^*(a+aa+\Lambda)$$

{a,b,c}'de tanımlı ve içinde 'a'ların sayısı 2'nin katlarıdır.

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

- Regüler İfade ..... L(R) dilinde örnek katarlar
  - rakam [0-9] .... “0”, “1”, “2”, ...
  - poztamsayı = rakam^{+} ..... “8”, “412”, 0, ?..
  - (1-9)(0-9)*
  - tamsayı = (-| ε) poztamsayı ..... “-23”, “34”, ...
  - reelsayı = tamsayı(ε |(.poztamsayı))“-1.65”, “24”, “1.085” (bu tanım “.58” ve “45.” sözcüklerine izin verir mi?)
  - harf [a-z] ..... “a”, “b”, “c”,....
  - değişken_adı = harf(harf | rakam)*..... “Ortalama”, “sayı”,...

- Öncelik
  - * en yüksek
  - concatenation sonra
  - | en düşük
- $ab^*|c$ şöyle değerlendirilir $(a(b)^*)|(c)$
  - Σ = {0,1} alfabesinde;
  - 0|1 => {0,1}
  - (0|1)(0|1) => {00,01,10,11}
  - 0^* => {ε,0,00,000,0000,....}
  - $(0|1)^*$ => Boş katarda dahil olmak üzere 0 ve 1'li bütün katarlar.

Regular İfadeler

- Regüler ifadeleri programlama dillerinin token'larını tanımlamak için kullanırız.
- Bir regüler ifade yukarıda tanımlanan kurallarla basit regüler ifadelerin birleştirilmesiyle elde edilir.
- Her regüler ifade bir dil tanımlar.
- Bir rügüler ifade ile tanımlanan dil regüler bir kümedir.

Σ Üzerinde tanımlı Regüer ifadeler.

Reg. İfade	Dil
ε	$\{\varepsilon\}$
$a \in \Sigma$	$\{a\}$
$(r_1) \mid (r_2)$	$L(r_1) \cup L(r_2)$
$(r_1)\,(r_2)$	$L(r_1)\,L(r_2)$
$(r)^*$	$(L(r))^*$
(r)	$L(r)$
$(r)^+ = (r)(r)^*$	

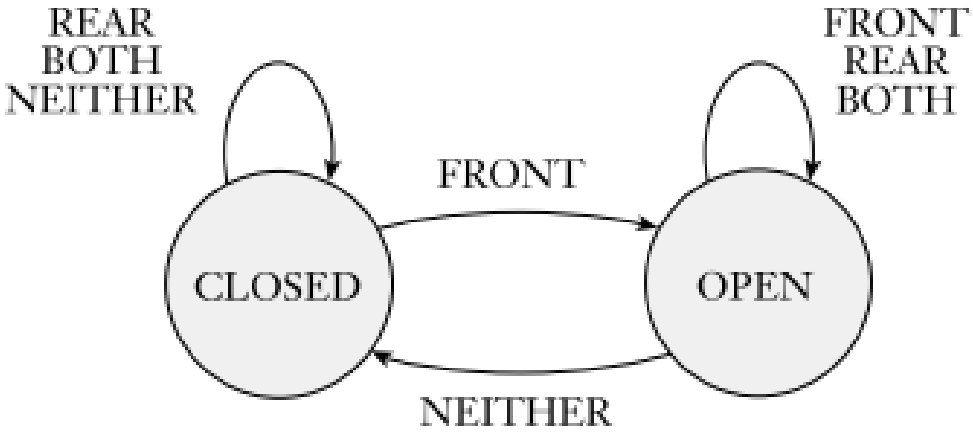
- $(r)? = (r) \mid \varepsilon$

- Örnek: Belirleyiciler (Identifiers), PASCAL
- letter $\rightarrow$ A | B | ... | Z | a | b | ... | z
  digit $\rightarrow$ 0 | 1 | ... | 9
  id $\rightarrow$ letter (letter | digit) *

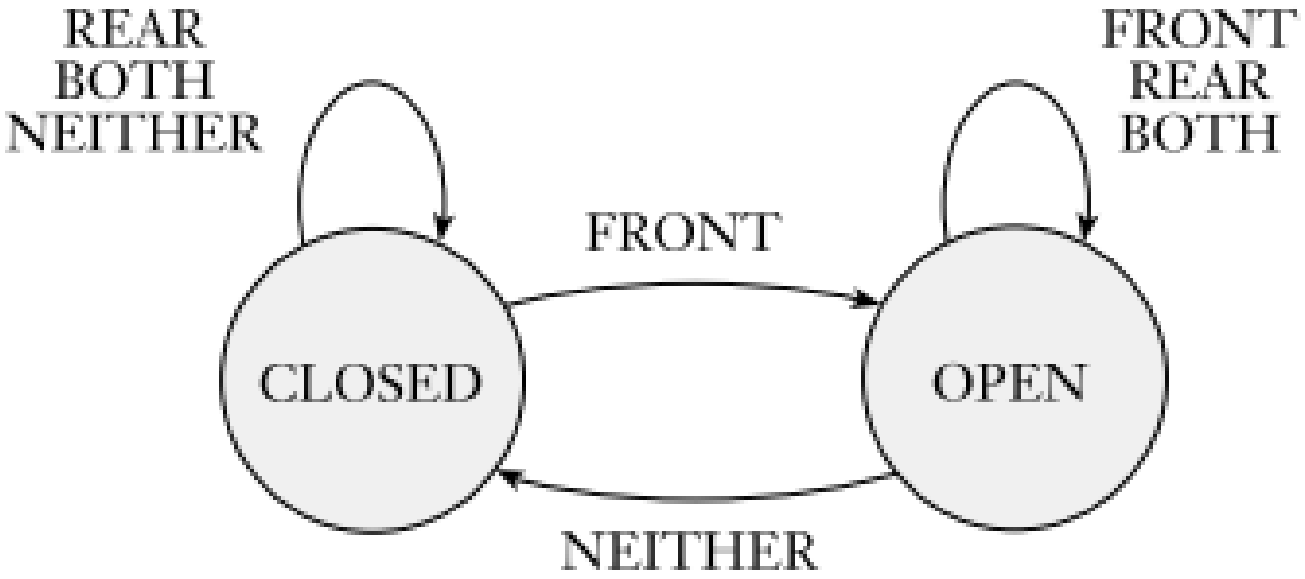
- Örnek :İşaretsiz tamsayı (Unsigned numbers):Pascal
  digit $\rightarrow$ 0 | 1 | ... | 9
  digits $\rightarrow$ digit $^+$
  opt-fraction $\rightarrow$ (. digits) ?
  opt-exponent $\rightarrow$ (E (+|-)? digits) ?
  unsigned-num $\rightarrow$ digits opt-fraction opt-exponent

DFA Soyut ve Somut Tanım

Otomatik kapının önünde, kapı aralığından geçmek üzere olan bir kişinin varlığını tespit etmek için bir sensör bulunur. Kontrolörün kapıyı kişinin geçmesine yetecek kadar uzun süre açık tutabilmesi ve ayrıca kapı açılırken arkasında duran birine çarpmaması için başka bir sensör kapı aralığının arkasına yerleştirilmiştir. Bu konfigürasyon aşağıdaki şekilde gösterilmektedir.

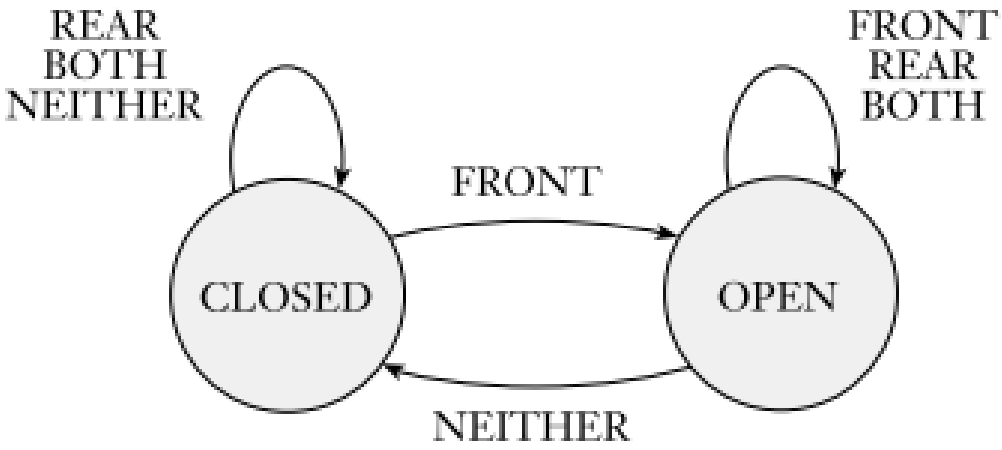


Kontrolör iki durumdan birindedir: kapının ilgili durumunu temsil eden «OPEN» veya «CLOSED». Aşağıdaki şekillerde gösterildiği gibi, dört olası giriş koşulu vardır: «FRONT" (bir kişinin kapı aralığının önündeki sensör üzerinde durduğu anlamına gelir), «REAR» (bir kişinin arka tarafta sensör üzerinde durduğu anlamına gelir) , “BOTH” (insanların her iki yüzeyde de durduğu anlamına gelir) ve “NEİTHER” (hiç kimsenin her iki yüzeyde de durmadığı anlamına gelir).



input signal

state	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN



input signal

state	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

Denetleyici aldığı girdiye bağlı olarak durumdan duruma hareket eder. "KAPALI" durumdayken ve «NEITHER" veya «REAR" girişini alırken, «CLOSED" durumda kalır. Ayrıca, «BOTH" girişi alınırsa «CLOSED" kalır çünkü kapının açılması birinin arka tarafa çarpma riski taşır. Ancak «FRONT" girişi gelirse «OPEN" duruma geçer. «OPEN" durumunda, «FRONT", «REAR" veya «BOTH" girişi alınırsa «OPEN" durumda kalır. «NEİTHER" girişi geldiğinde «CLOSED" durumuna döner.

Örnek: FRONT, REAR, NEITHER, FRONT, BOTH, NEITHER, REAR, NEITHER

CLOSED (start)

	input signal			
state	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

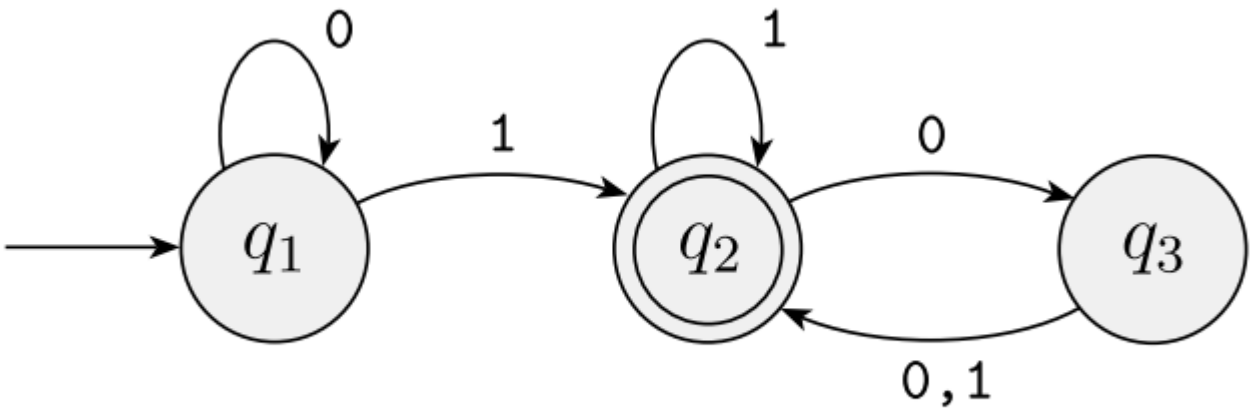
OPEN, OPEN, CLOSED, OPEN, OPEN, CLOSED, CLOSED, CLOSED.

Bu denetleyici, yalnızca tek bitlik belleğe sahip olan ve denetleyicinin iki durumdan hangisinde bulunduğunu (0/1) kaydedebilen bir bilgisayardır.

Diğer yaygın cihazlarda biraz daha büyük hafızalı denetleyiciler bulunur. Bir asansör kontrol cihazında, bir durum asansörün bulunduğu katı temsil edebilir ve girişler butonlardan alınan sinyaller olabilir. Bu bilgisayarın bu bilgiyi takip edebilmesi için birkaç bit'e ihtiyacı olabilir.

Bulaşık makineleri ve elektronik termostatlar gibi çeşitli ev aletlerinin kontrolörlerinin yanı sıra dijital saat ve hesap makinelerinin parçaları da sınırlı hafızaya sahip bilgisayarlara örnektir.

Bu tür cihazların tasarımı, sonlu otomatların metodolojisinin ve terminolojisinin akılda tutulmasını gerektirir.



M_1 finite automaton

strings 0, 10, 101000.

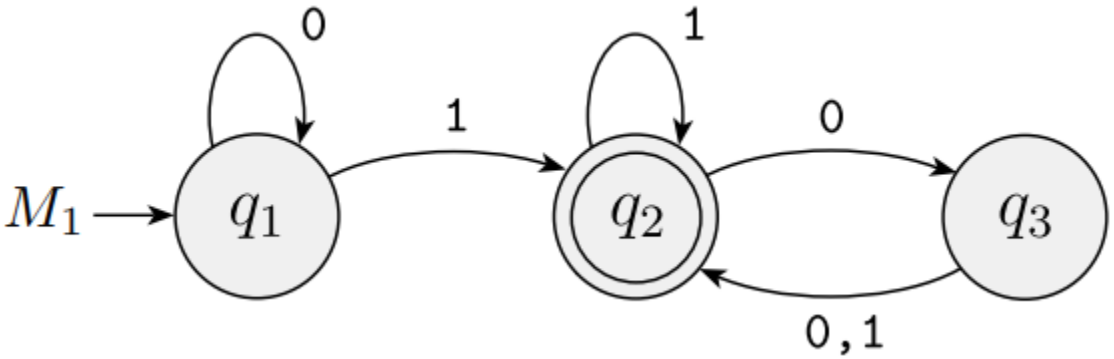
input string 1101 ***accept***

reject

1. Start in state q_1.
2. Read 1, follow transition from q_1 to q_2.
3. Read 1, follow transition from q_2 to q_2.
4. Read 0, follow transition from q_2 to q_3.
5. Read 1, follow transition from q_3 to q_2.
6. *Accept*

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



$M_1 = (Q, \Sigma, \delta, q_1, F)$

1. $Q = \{q_1, q_2, q_3\}$,

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

3. δ

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

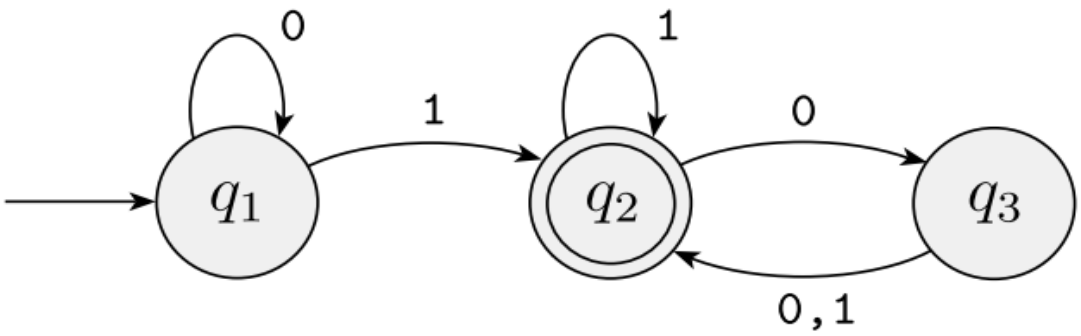
4. q_1 = start state **5.** $F = \{q_2\}$.

Eğer A kümesi, M makinesinin kabul ettiği tüm katarların kümesi ise, A kümesinin M makinesinin dili olduğunu söyleriz ve L(M) = A yazarız. M makinesinin A dilini tanıdığını veya M'nin A dilini kabul ettiğini söyleriz.

"kabul-accept" terimi katarları kabul eden makineler ve dilleri kabul eden makinelerden bahsettiğimizde farklı anlamlara sahip olduğundan, karışıklığı önlemek amacıyla diller için "tanıma-recognise" terimini tercih ediyoruz.

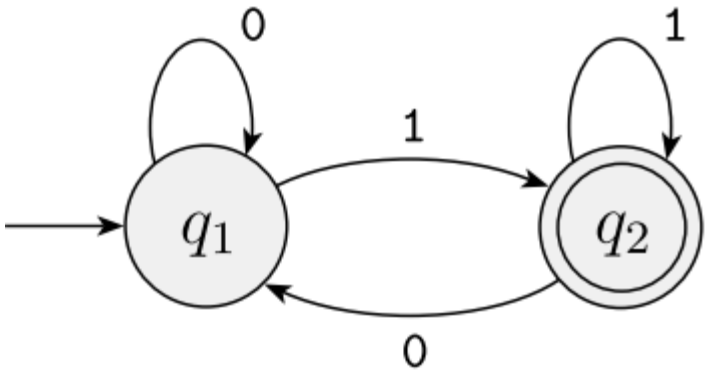
Bir makine birçok katarı kabul edebilir, ancak yalnızca bir dili tanır. Makine hiçbir katarı kabul etmiyorsa, Bu durumda yine bir dili (boş dili (Ø)) tanır.

Örnek1:L(M1)=A = {w| w en az bir 1 içerir ve son 1'den sonra çift sayıda 0 bulunur}. O halde L(M1) = A veya eşdeğer olarak M1, A'yı tanır.



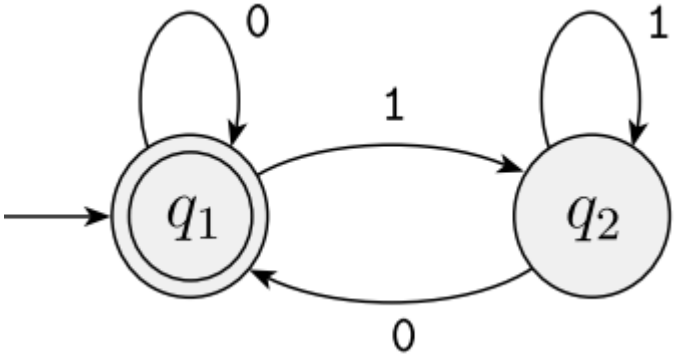
M_1 finite automaton

Örnek 2:L(M2) = {w| w katarı bir 1 ile biter1}

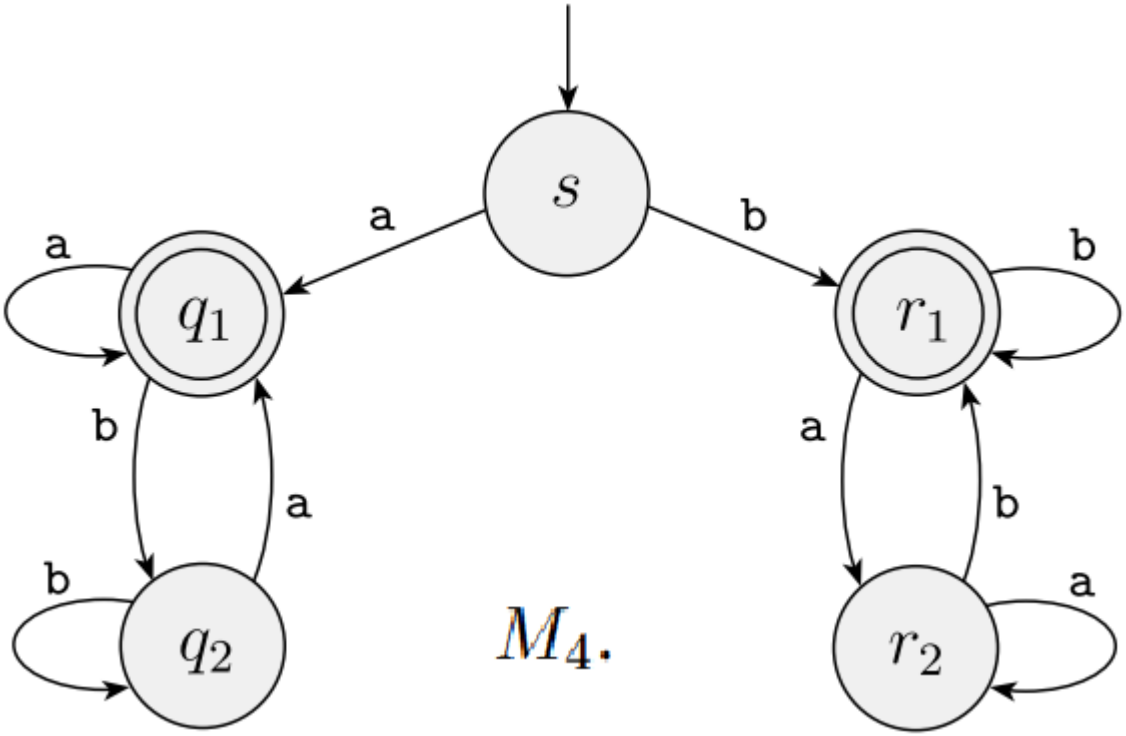


	0	1
q_1	q_1	q_2
q_2	q_1	q_2.

Örnek 3. w boş dize ε'dur veya 0 ile biter

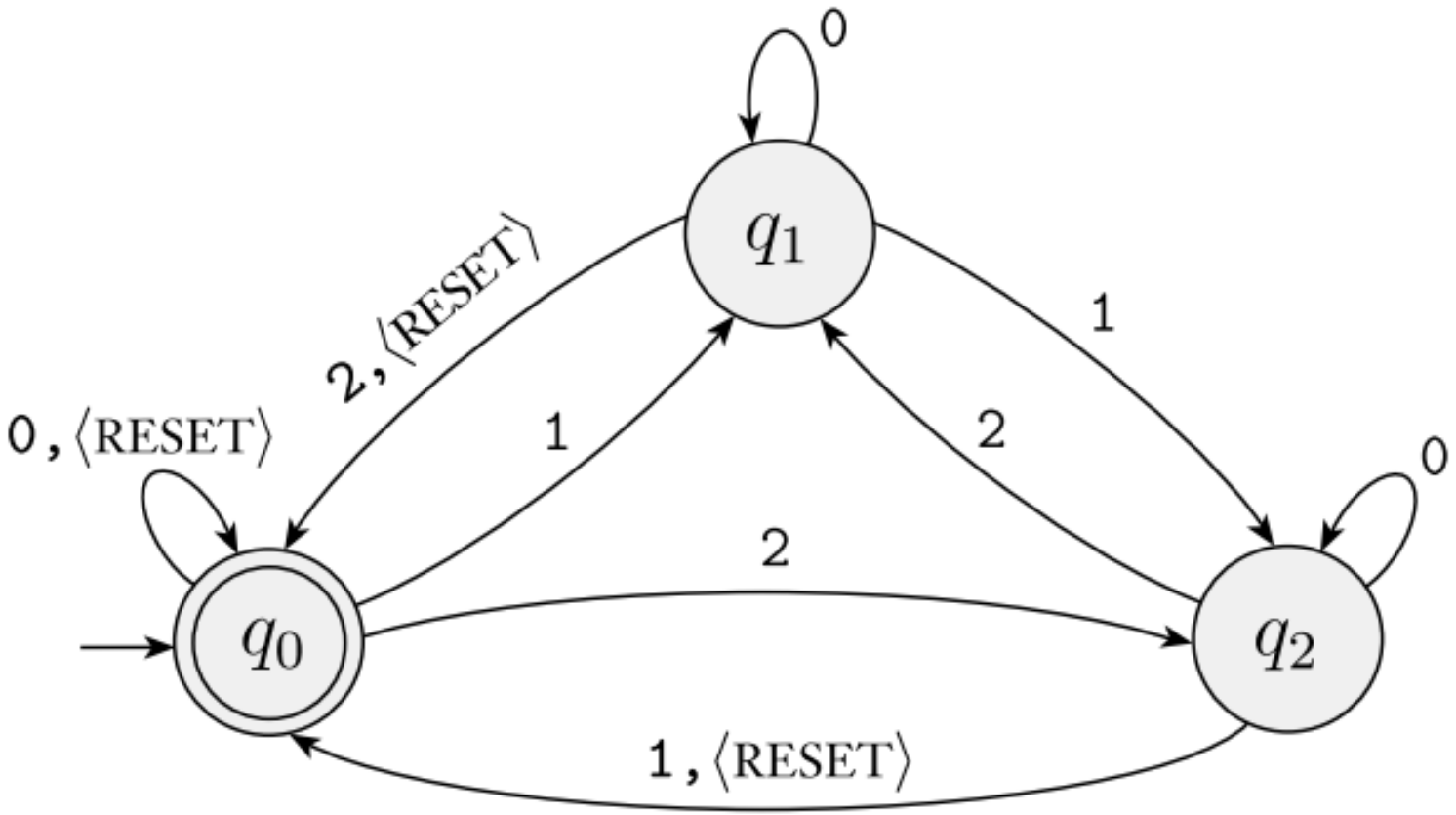


Örnek: M4 makinesi , a ile başlayan ve biten VEYA b ile başlayıp biten tüm katarları kabul eder. Yani M4, aynı sembolle başlayan ve biten dizeleri kabul eder.

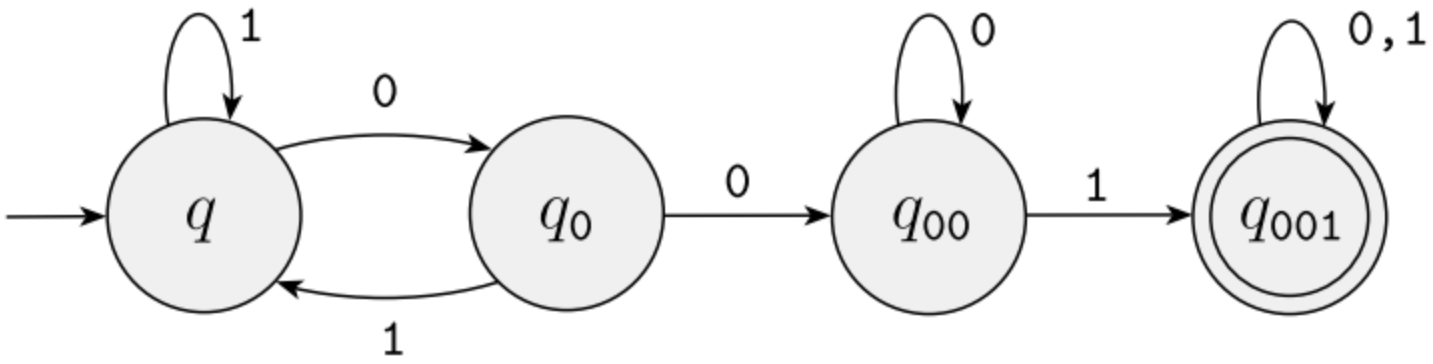


Giriş dizesindeki ilk sembol a ise sola gider ve dize a ile bitince kabul eder. Benzer şekilde, ilk sembol b ise makine sağa gider ve dize b ile bitince kabul eder.

Örnek: Makine M5, okuduğu sayısal giriş sembollerinin toplamının Mod(3)'e göre sürekli bir sayımını tutar. ⟨RESET⟩ sembolünü her aldığında, sayımı 0'a sıfırlar. Toplamın 0 veya 3'ün katı ise kabul eder. Σ = {⟨RESET⟩, 0, 1, 2}.



Bu örnek, bir alt dize olarak 001 dizesini içeren tüm dizelerin düzenli dilini tanımak için sonlu bir E2 otomatının nasıl tasarlanacağını gösterir. Örneğin, 0010, 1001, 001 ve 11111110011111'in tümü dilde yer alır, ancak 11 ve 0000 değildir.



L(M) dili{1,2,3} alfabesinde tanımlanmıştır ve ‘123’ alt katarını mutlaka içerir. Bu dili tanıyan DFA makinesini çiziniz.