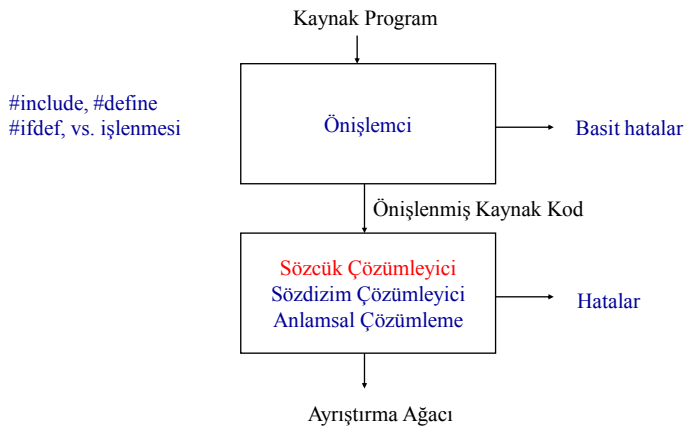
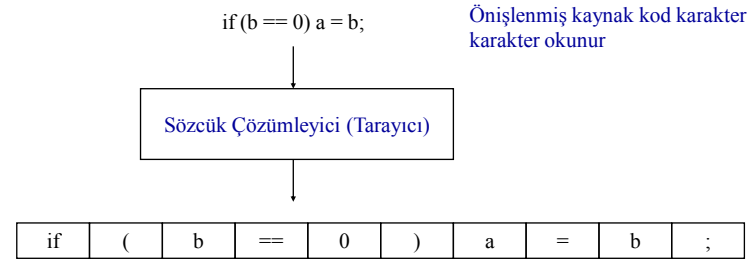
# Sözcük Çözümleyici



## Derleyicinin yapısı



# Sözcük Çözümleme İşlemi




Tarayıcı:
- Kaynak kodu sözcük dizisine dönüştürür
- Kodu gereksiz simgelerden (boşluk, açıklama, vs.) arındırır

3

# Sözcükler (token)

- Değişken adı: x k12 toplam
- Anahtar Sözcükler: if else while for break
- Tamsayılar: 2 1000 -20
- Reel sayılar: 2.0 -0.0010 .02 1e5
- Özel işaretler: + * { } ++ << < <= [ ]
- Katarlar: “x” “deniz”

4

# Tarayıcının Hazırlanması

- İki problemin çözülmesi gerekir
  - Kaynakta bulunmasına izin verilen sözcükleri tanımlamak için bir yöntem
  - Kaynakta yer alan sözcükleri tanımak için bir yöntem
- Dilin izin verdiği sözcükleri tanımlamak için "düzgün ifade" 'lerden yararlanırız
- Kaynakta yer alan sözcükleri tanımak için "sonlu otomat" 'lardan yararlanırız

5

# Katarlar ve diller

Genel tanımlar:

- Alfabe sonlu simge/karakterler kümesi -- {0,1}, ASCII
- Katar sonlu simgeler dizisi (sözcük) – 001, abc
- Uzunluk (|x|) katarı oluşturan karakter sayısı
- Boş katar "ε" uzunluğu sıfır olan katar
- Bitiştirme iki katarı birbirini izleyecek şekilde birleştirme
  - x= abc y= de → xy=abcde
  - εx=x ε=x
  - $x^i$ x katarı kendisiyle "i" kez bitiştirilir, $x^0$ → ε

6

# Katarlar ve diller (2)

- Dil belirli bir alfabeden üretilen bir katarlar kümesi
  - {ε} – boş katarı içeren küme
  - {0,1,00,01,10,11}
  - Konuşulan diller, programlama dilleri
- Bitiştirme işlemi dillere de uygulanabilir. L ve M iki ayrı dil ise, LM, L'nin içerdiği tüm katarların M'nin içerdiği tüm katarlarla bitiştirilmesinden oluşan kümedir
  - L={0,01,110} M={10,110}, LM={010,0110,01110,11010,110110}
  - $L^i$=LLL...L (i kez kendisiyle bitiştirme)

7

# Katarlar ve diller (3)

- Kılıf Operatörü (*) belirsiz sayıda bitiştirme operatörü
- $L^*$ 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

8

# Sözcüklerin tanımlanması

- Düzgün ifadeler (regular expression) kullanılır
- Bir düzgün ifade şöyle tanımlanır

  (R ve S düzgün ifadeler olmak üzere)
  - a — her karakter bir düzgün ifadedir
  - ε — boş katar bir düzgün ifadedir
  - R|S — "R veya S" bir düzgün ifadedir
  - RS — "R ve S" (bitiştirme) bir düzgün ifadedir
  - R* — R'nin kendisiyle 0 veya daha fazla bitiştirilmesiyle elde edilen bir düzgün ifadedir

9

# Dil

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

10

# 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”, “bbab”, “babbaabaab”
  - (a|b)(a|b)(a|b) ..... “aaa”, “aab”, “bab”, “abb”..

11

# Örnek Düzgün İfadeler (2)

- Düzgün İfade ..... L(R) dilinde örnek katarlar
  - rakam [0-9] ..... ... “0”, “1”, “2”, ...
  - poztamsayı = rakam$^{+}$ ..... “8”, “412”, ...
  - tamsayı = (-| ε) poztamsayı ..... “-23”, “34”, ...
  - reelsayı = tamsayı(ε |(.poztamsayı))“-1.56”, “12”, “1.056” (dikkat: bu tanım “.58” ve “45.” sözcüklerine izin vermez)
  - harf [a-z] ..... “a”, “b”, “c”,....
  - değişken_adı = harf(harf | rakam)$^{*}$..... “toplam”, “sayac”,...

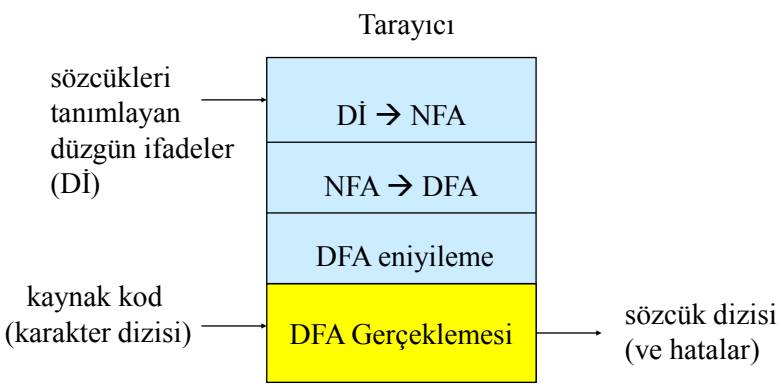
12

# Sözcük Çözümleyici Nasıl Çalışır


**"belirgin sonlu otomat"**

13

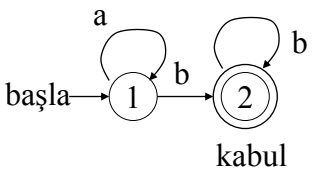
# Tarayıcı Nasıl Geliştirilir?




14

# Düzgün İfade → NFA (nondeterministic finite automata)

- Bir düzgün ifadeyi bir durum diyagramı ile göstermek mümkündür
  - Her düğüm bir durumu karşı düşer
  - Düğümler arasındaki kenarlar geçişleri gösterir ve "ε" veya karakter ile tanımlanabilir
  - Bir başlangıç ve bir veya daha fazla kabul durumu bulunur
  - Aynı giriş bilgisi için birden fazla bir sonraki durum yer alabileceği için "**belirgin olmayan sonlu otomat**" adını alır

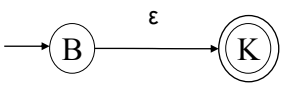



Düzgün ifade: $a^*b^+$

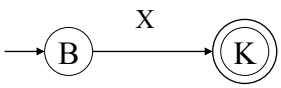
15

# Düzgün İfade → NFA (2)

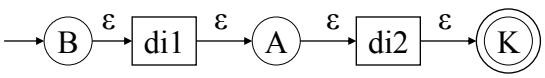
- Temel düzgün ifadeler için aşağıdaki kuralları uygula




Boş katar dönüşümü




Alfabetik simge dönüşümü

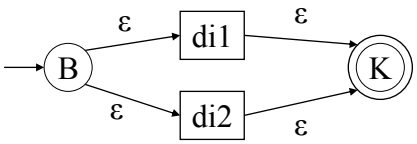



Bitiştirme (ve)
(di1.di2)

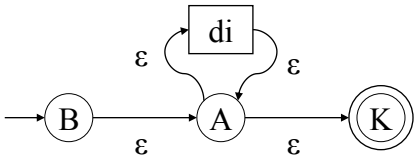
B: başlangıç durumu
K: kabul durumu di:düzgün ifade

16

# Düzgün İfade → NFA (3)



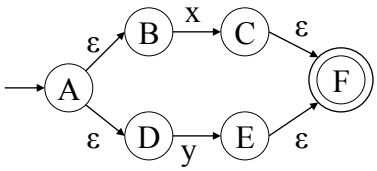

veya: (di1 | di2)



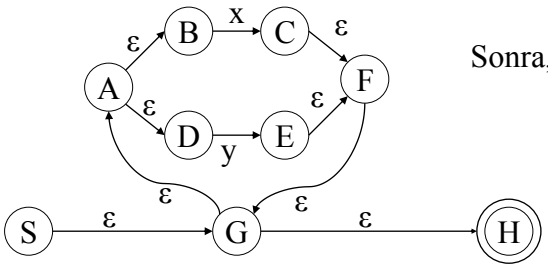

kılıf: (E*)

17

# di→NFA geçişe örnek **di: (x | y)***



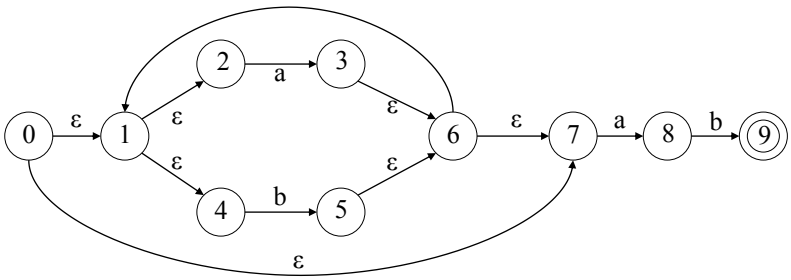

Önce (x | y) için bir NFA oluştur




Sonra, kılıf operatörünü ekle

18

# di→NFA geçişe örnek **di: (a|b)*ab**




Örnek katarlar: aab, bab, ababab, aaaab, bbab

19