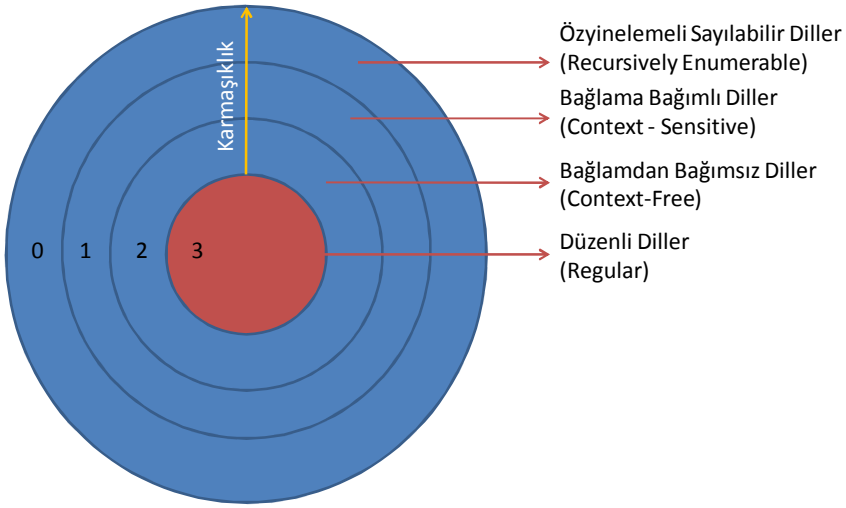
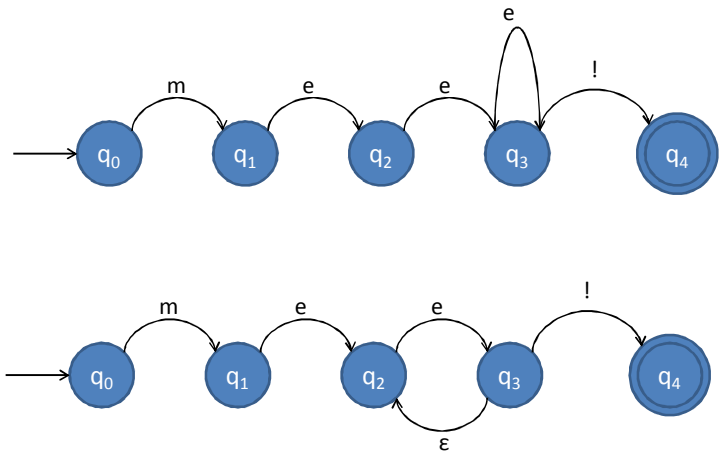
Düzenli Diller ve İfadeler

Doç.Dr.Banu Diri

Chomsky Hiyerarşisi

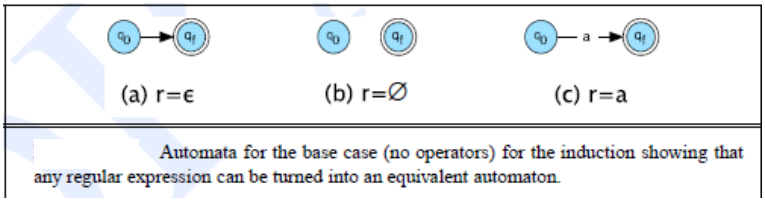


Sonlu Durum Otomatları (FSAs)

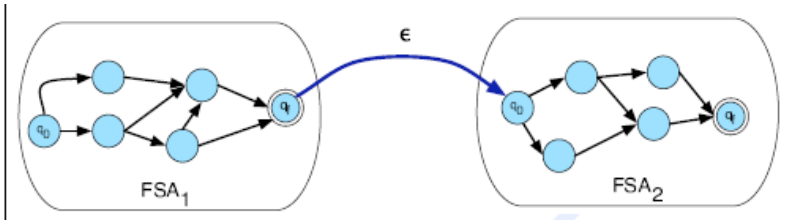


Düzenli Diller

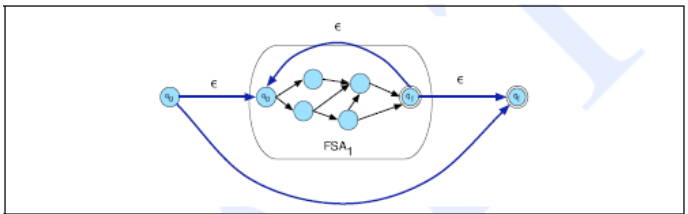
- Σ sonlu bir alfabe
- Ø boş küme
- ε, {ε} kümesini göstermek üzere;
- Σ üzerinde tanımlanabilen *düzenli dillerin* formel tanımı:
  - Ø düzenli bir dildir.
  - $\forall$ A Є Σ Uε , {a} düzenli bir dildir.
  - Eğer L_1 ve L_2 dilleri düzenli diller ise:
    a) $L_1 \cdot L_2 = \{ xy \mid x \in L_1, y \in L_2 \}$ ***ekleme işlemi (concatenation),***
    b) $L_1 \cup L_2$ ***birleşim veya kesişim (union, disjunction),***
    c) L_1^* ise ***Kleene sonlandırması (Kleene closure)*** ile tanımlanan diller de düzenli dillerdir.



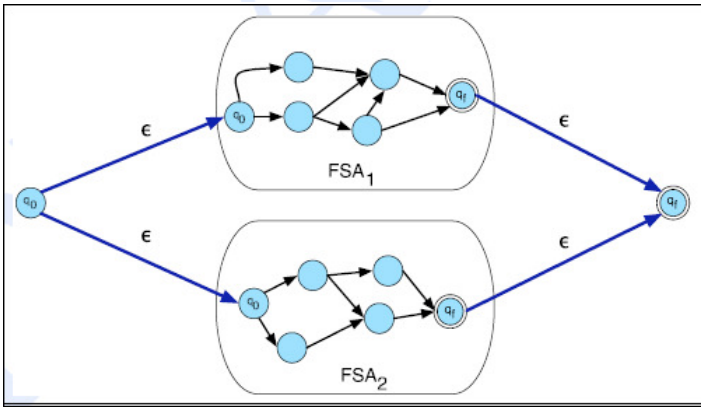
Automata for the base case (no operators) for the induction showing that any regular expression can be turned into an equivalent automaton.



The concatenation of two FSAs.



The closure (Kleene *) of an FSA.



The union (|) of two FSAs.

Uygulama Alanları

- Arama motorları
- Bilgi edinim (Information retrieval)
- Kelime işleme (Word processing)
- Derlem (corpus) içinde frekans hesaplama
- Veri doğrulama (Data validation)
- Sözdizim belirginleştirme (Syntax highlighting)
- …

Düzenli İfadeler (Regular Expressions-Regex – Regexp)

Karakter dizileri içinde belirli örüntüleri (pattern) aramayı sağlayan bir dildir.
İlk olarak 1956 yılında *Stephen C. Kleene* tarafından formel bir model olarak sunulmuştur. Herhangi bir düzenli ifade doğrudan NFSA'ya, buradan da DFSA'ya dönüştürülebilir.

Regex Nerelerde Kullanılır ?

- Nasıl yazıldığından emin olamadığımız bir kelimeyi aratırken.
  "*Scholarships*" kelimesini arayalım, ancak harflerin yerinden emin değiliz. İlk, son ve bir kaç karakteri verip arama yapabiliriz.
- Belirli kalıpları arayabiliriz.
  http:// ile başlayan ve arada noktalar içeren adresleri
- Kullanıcıdan alınan bir bilginin belli bir format yapısına uyup uymadığını kontrol edebiliriz.
  Kullanıcılardan e-posta adresi girmelerini isteyip, girilen adresin geçerli bir e-posta adresi olup olmadığını kontrol edebiliriz.(@ işareti içerecek, belli karakterlerden oluşacak, ikinci blokta en az bir nokta olacak ...)

Kullanım Alanları

vi, emacs, kate, ... metin düzenleyicilerde
grep, sed,... konsol uygulamalarında
Perl, Python, PHP, JavaScript,... programlama dillerinde
Eclipse, Visual Studio, ... geliştirme ortamlarında

Kısaca, bir yazının içerisinden istediğimiz bir bilgiyi bulmak ve/veya değiştirmek için kullanılır.

En basit düzenli ifadeler karakterlerin sıralı biçimde dizilmesiyle oluşur.

Düzenli ifadeler büyük – küçük harfe duyarlıdır.

okula → 'Ali okula gelmedi.'
okul → 'Ali okula gelmedi.'
a → 'Ali okula gelmedi.'

Örneklerde, boşluk karakteri yerine "•" kullanılacaktır.

Doğal•Dil •İşleme

REGEX – Satır Başı ^ ve Satır Sonu $

^ işareti satır başlarını,
$ işareti satır sonlarını
gösterir.

^Banu : Satır başlarındaki "*Banu*" ları bulur.
Banu$: Satır sonlarındaki "*Banu*" ları bulur.
^Banu$: Aynı anda hem satır başında, hem de satır sonunda yer alan "Banu" ları bulur. Sadece "Banu" yazan satırları bulur.

^$: Satır başından hemen sonra gelen satır sonlarını bulur. Yani boş satırları bulur.

Örnekler
^gül → gül güler (bulur) songül (bulamaz)
gül$ → gül songül (bulur) güler (bulamaz)
^gül$ → gül (bulur) güler songül (bulamaz)

REGEX – Karakter Sınıfları

[...] yapısı, ***REGEX***'te karakter sınıfı olarak bilinir.

Örnek: "*makina*" kelimesini arayalım. Bazen "*makine*" diye de yazılabilir. Her ikisini de yakalamak istiyorsak:

makin[ae] *düzenli ifadesini* kullanmalıyız.

[ds]e[lb]i → sebil delil (bulur) → salim (bulunmaz)
[Bb]anu → Banu banu (bulur)

HTML belgesi içerisindeki tüm *Header* elementlerini bulmak istersek yazılacak *regex*; **<h[123456]>** <h1>, <h2> … <h6> tüm elementleri bulur.

[-] Köşeli parantez içerisinde iki karakter arasında **tire (-)** işareti varsa, bu iki karakter arasındaki herhangi bir karakterle eşleşme gerçekleşeceğini gösterir. Karakterlerin nasıl dizildiği, yerel ayarlarımıza göre farklılık gösterebilir.
<h[1-6]> **[0-9]** **[a-z]** **[A-z]** **[a-zA-Z]**

REGEX – Negatif Karakter Sınıfları

[...] yerine **[^...]** kullanılırsa, belirlediğiniz kriterlere **uymayanlar** listelenir.

[^5-8] regex'i 5 ve 8 arasında olmayan karakterlerle eşleşir.

Karakter sınıfının başındaki **^** sembolü, karakter sınıfının içerisindeki karakterleri **istemediğinizi** bildirir.

^ satır başı ile karıştırılmamalıdır. Kullanıldığı yere göre anlamı değişir.
Ardından *d* harfi gelmeyen *türkçe* kelimelerini arayalım.
regex : türkçe[^d]

türkçemiz güzel bir dildir
türkçeyi çok bozduk
türkçede ses uyumu
türkçe

^ karakterinin 3 farklı şekilde kullanımı (özet)

- Aralık için olumsuzlama

  [^A-Z] → büyük harf harici karakter
  [^Ss] → S veya s harici karakter
  [^\.] → nokta harici karakter

- Satır başına bağlama

  ^Asya → satır başında 'Asya' olan durum

- Normal karakter olarak kullanımı

  [e^] → e veya ^
  a^b → a^b örüntüsü

REGEX – Nokta ve ? karakteri

Nokta karakteri **(.)** herhangi bir karakterle eşleşebilir. Joker olarak düşünebiliriz.

s.cak diye bir **düzenli ifademiz** olsun, arama sonucunda neleri bulabilir/bulamayız.

sıcak → buluruz
sicak → buluruz
Sıcak → bulamayız (s büyük harf olarak yazılmış)

Yazı içerisindeki **nokta**ları bulmak istersek? **noktanın** önüne **** işaretini koyarız.

Örnek;
193.140.4.1 ve ardından gelen 1 karakter daha olsun.
regex: 193\.140\.4\.1.

193.140.4.13 → bulunur
193.140.4.1a → bulunur

- '?' karakteri kendinden önce gelen karakterin seçimlik olduğunu belirtir.
  evleri? → evler veya evleri
  colou?r → color veya colour

***** ve **+** (Kleene star and plus)

- '*' karakteri kendinden önce gelen karakterin 0 veya daha fazla kere ardışık olarak tekrarlandığını belirtir.
  ab*c → ac, abc, abbc, abbbc, … →abd (bulamaz)
  [0-9][0-9]* → bir veya daha fazla sayıda ardışık rakam

  YTU .* Kulüpleri → YTU Kulüpleri
  YTU Öğrenci Kulüpleri

- '+' karakteri kendinden önce gelen karakterin 1 veya daha fazla kere ardışık olarak tekrarlandığını belirtir.
  [0-9]+ → bir veya daha fazla sayıda ardışık rakam

İfade Sınırları

- \b özel karakteri, kullanıldığı yere göre, aranan ifadenin önünde veya arkasında sınırlayıcı (boşluk gibi) karakterleri sınır olarak kabul eder.
  \beli\b → önünde ve arkasında boşluk olan 'eli' ifadesini bulur

- \B karakteri sınırlandırma olmayan durumu belirtir.

REGEX – Alt İfadeler

| *veya* anlamına gelir.

Birden fazla **düzenli ifadeyi** birleştirip tek bir **düzenli ifade** oluşturmak için kullanılır.
Oluşturulan ifade içerisindeki alt ifadelerden herhangi birini karşılıyorsa sonucu listeler.

Bir yazı içerisinde geçen bütün **4. NLP Sempozyumu** ifadelerini listelemek isteyelim.
Bu ifadenin Dördüncü veya dördüncü NLP Sempozyumu diye de geçebileceğini düşünelim.

regex : (Dördüncü|dördüncü|4\.)•Sempozyumu
([Dd]ördüncü|4\.)•[Ss]empozyumu

Sayaçlar

- Herhangi bir karakterin ne miktarda tekrarlanabileceğini belirten ifadelerdir.
  - {n} → kendinden önceki karakter n defa ardışık olmalıdır
  - {n, m} → kendinden önceki karakter n ile m aralığında ardışık olmalıdır
  - {n,} → kendinden önceki karakter en az n kadar ardışık olmalıdır
- *Ateşoğlu* kelimesini bulalım. Bunun için regex ihtiyaç yoktur, herhangibir editör bize yardımcı olur. Ancak *Ateşoğlu* kelimesinden önce 4 karakter ve arkasından da 3 tane rakam geliyor olsun. Bunun için regex yapmamız gerekir.
- [a-z]{4} [0-9]{3}Ateşoğlu

İşlemlerin öncelikleri

- Yüksekten en düşük işlem önceliğine göre sıralama:

1. Parantez →()
2. Sayaçlar →* + ? {}
3. Seriler veya bağlayıcılar → evler ^Yarın gelecek$
4. Veya → | (pipe)

Özel Operatörler

\d → herhangi bir rakam
\D → rakam olmayan bir karakter
\w → alfanümerik veya boşluk karakteri
\W → alfanümerik olmayan karakter
\s → boşluk
\S → boşluk olmayan karakter
\. → nokta karakteri
* → * (asterisk) karakteri
\? → ? Karakteri
\n → newline karakteri
\t → tab karakteri