Classi di modelli

- I linguaggi possono essere rappresentati mediante
  - Insiemi
  - Pattern
  - Espressioni regolari
  - Modelli operazionali
    - Automi
    - Reti di Petri
    - Diagrammi di stato
  - Modelli generativi
    - Grammatiche
  - Modelli dichiarativi
    - Logica
  - Trasduttori

Pattern

- Un sistema di pattern è una tripla <A, V, p> dove
  - A è un alfabeto
  - V è un insieme di variabili tale che $A \cap V = \emptyset$
  - p è una stringa su $A \cup V$ detta pattern
- Il linguaggio generato dal sistema di pattern consiste di tutte le stringhe su A ottenute da p sostituendo ogni variabile in p con una stringa su A
- Esempio: $<\{0,1\}, \{v_1,v_2\},v_1v_10v_2>$
  - Stringhe che iniziano con 0 ($v_1=\varepsilon$)
  - Stringhe che iniziano con una stringa su A ripetuta due volte, seguita da uno 0 e da qualunque stringa (inclusa ε)

Espressioni regolari (RE): sintassi e semantica

- Definizione di RE su un alfabeto A:
  - Ø è una RE (che denota il linguaggio Ø)
  - ε è una a RE (che denota il linguaggio {ε})
  - Ogni simbolo di A è una RE (che denota {a}, a∈A)
  - Siano r e s due RE, allora:
    - (r.s) è una RE (che denota la concatenazione dei linguaggi denotati da r e s)
      - Per semplicità, il punto è spesso omesso
    - (r | s) è una RE (che denota l'unione dei linguaggi denotati da r e s)
      - Talvolta indicata r + s o r ∪ s
    - (r)* è una RE (che denota il più piccolo soprainsieme del linguaggio denotato da r che contiene ε e chiuso rispetto a .)
  - Nient'altro è una RE

Esempio

- ((0.(0|1)*)|((0|1)*.0)) è un'espressione regolare sull'alfabeto {0,1}
  - Stringhe che iniziano con 0
  - Stringhe che finiscono con 0

RE e pattern

- Le espressioni regolari seguono la stessa idea dei sistemi di pattern
  - Ma hanno diverso potere espressivo
- Espressioni regolari ≠ Sistemi di pattern
  - $\{xx \mid x \in \{0,1\}^*\}$ non è un linguaggio regolare
    - Sistema di pattern corrispondente: $\langle \{0,1\}, \{x\}, xx \rangle$
  - Il linguaggio denotato da 0*1* non è un linguaggio esprimibile mediante un sistema di pattern

RE e grammatiche regolari

- Le RE corrispondono esattamente ai linguaggi regolari (stesso potere di RG e FSA)
- Facile da mostrare:
  - Ogni linguaggio denotato da una RE è regolare:
    - I casi base sono tutti linguaggi regolari
    - I linguaggi regolari sono chiusi rispetto a concatenazione, *, unione
  - Data una RG G, trovare una RE r tale che L(G)=L(r)

Dimostrazione informale…

- Si consideri la grammatica

  $S \rightarrow aA$, $A \rightarrow bB$, $A \rightarrow cB$, $B \rightarrow aA$, $B \rightarrow \varepsilon$, $S \rightarrow aC$,

  $C \rightarrow cC$, $C \rightarrow \varepsilon$
- Alcune derivazioni (informali):
  - A partire da $S \rightarrow aA$:
    - $a(b|c) \Rightarrow a(b|c)a(b|c) \Rightarrow \ldots \Rightarrow a(b|c)(a(b|c))^*$
    - Scritto anche come $(a(b|c))^+$
  - A partire da $S \rightarrow aC$:
    - $a \Rightarrow ac \Rightarrow acc \Rightarrow accc \Rightarrow \ldots \Rightarrow a(c)^*$
- In generale: $(a(b|c)(a(b|c))^* \mid a(c)^*)$

RE in pratica

- Le RE sono molto comuni nella pratica:
  - Analizzatori lessicali (es. lex)
  - Funzioni avanzate di ricerca/sostituzione nei text editor e in strumenti di sistema (emacs, vi, grep, awk...)
  - Linguaggi di scripting (Perl, Python, Ruby, ...)
- C’è uno standard POSIX (API standard per unix/linux) anche per le RE
- Sintatticamente, le RE in "pratica" sono un po’ diverse da quanto mostrato prima...

RE POSIX

- Caratteri meta: ( ) . [ ] ^ \ $ * + ? | { }
- Attenzione: il . è usato per indicare qualunque carattere, non per concatenare!
- [α] denota un singolo carattere ∈α (es. [abc] denota {a,b,c}. Si può anche scrivere [a-z] per indicare qualunque lettera minuscola)
- [^α]: negazione: qualunque simbolo non in α (es. [^a-z] è qualunque carattere che non sia una lettera minuscola)

RE POSIX (cont.)

- ^ e $ denotano ε rispettivamente all'inizio e alla fine di una riga di testo
- *,+,|,(,) sono come al solito
- \ funge da "escape" (per esempio, \$ denota il carattere $)

RE POSIX– Esempio 1

>> grep -E "^((Two)|(In))" grep.txt

In addition, two variant programs egrep and fgrep are available. Egrep is

Two regular expressions may be concatenated; the resulting regular

Two regular expressions may be joined by the infix operator |; the

In basic regular expressions the metacharacters ?, +, {, |, (, and) lose

In egrep the metacharacter { loses its special meaning; instead use \{ .

(tutte le righe che iniziano con “Two” o “In”)

RE POSIX– Esempio 2

>> grep -E "^[A-Z]+$" grep.txt

SYNOPSIS

DESCRIPTION

DIAGNOSTICS

BUGS

(tutte le righe che consistono di maiuscole)

RE POSIX– Esempio 3

>> grep -E "^[^a-zA-Z].+\.$" grep.txt

-S Search subdirectories.

[:print:], [:punct:], [:space:], [:upper:], and [:xdigit:].

[[:alnum:]] and \W is a synonym for [^[:alnum]].

? The preceding item is optional and matched at most once.

** The preceding item will be matched zero or more times.*

+ The preceding item will be matched one or more times.

{ n } The preceding item is matched exactly n times.

(tutte le righe che iniziano con un carattere che non è una lettera e finiscono con un punto)

Altri operatori

- α? α è opzionale
- α{n} α^n
- α{n,m} $\alpha^n \cup \alpha^{n+1} \cup \alpha^{n+2} \cup .. \cup \alpha^m$

Esempio

>> grep -E "[a-zA-Z]{15}" grep.txt

grep [-[[AB]]<num>] [-[CEFGLSVbchilnqsvwx?]] [-[ef]] <expr> [<files...>]

available functionality using either syntax. In other implementations,

(tutte le righe che contengono una stringa di almeno 15 lettere)

Caso pratico

- Traslare i tempi dei sottotitoli nei file in formato .srt
  - In Python
- Esempio:

00:00:51,444 --> 00:00:57,515

The Military Department has decided to assign more troops in order to crush the resistance.

Caso pratico

```
def srtShift (fname, shift) :
    out = fname+"_out"
    x = open(fname,'r')
    y = open(out,'w')
    for t in x :
        time = gettime(t)
        if time :
            x0 = tostr(sub(time[0],shift))
            x1 = tostr(sub(time[1],shift))
            y.write(x0+" --> "+x1+'\n')
        else :
            y.write(t)
    x.close()
    y.close()

srtShift(“myMovie.srt", [1,25,37,00])
```

Caso pratico

```
def gettime(s) :
    retime = re.compile(r""" ^ \s*
      (?P<h1> [0-9][0-9]):
      (?P<m1> [0-9][0-9]):
      (?P<s1> [0-9][0-9]),
      (?P<z1> [0-9][0-9][0-9]) \s* --> \s*
      (?P<h2> [0-9][0-9]):
      (?P<m2> [0-9][0-9]):
      (?P<s2> [0-9][0-9]),
      (?P<z2> [0-9][0-9][0-9]) \s* $
    """, re.VERBOSE)
    m = retime.match(s)
    if m :
        return [int(m.group('h1')),int(m.group('m1')), \
            int(m.group('s1')),int(m.group('z1'))], \
            [int(m.group('h2')),int(m.group('m2')), \
            int(m.group('s2')),int(m.group('z2'))]
```

re.VERBOSE ignora gli spazi nella RE, ma ci serve \s (qualunque tipo di spazio = [\t\n\r\f\v])

?P<name> è usato per dare un nome a un blocco (per potercisi riferire)

Caso pratico

```
def sub(t1,t2) :
    x = t1[:]
    if x[2]-t2[2] < 0 :
        x[2] += 60
        x[1] -= 1
    x[2] -= t2[2]
    if x[1]-t2[1] < 0:
        x[1] += 60
        x[0] -= 1
    x[1] -= t2[1]
    if x[0]-t2[0] < 0 :
        x[0] = 0
    else :
        x[0] -= t2[0]
    return x
```

```
def tostr(t) :
    def prdigit(n,ndigits) :
        h = str(n)
        while len(h) < ndigits: h = '0'+h
        return h

    return prdigit(t[0],2)+':'+prdigit(t[1],2)+':' \
           +prdigit(t[2],2)+','+prdigit(t[3],3)
```