Applicazione: le Espressioni Regolari

Matteo Pradella, 2004

RE: sintassi (e semantica)

- La stringa vuota è una RE (denota $\{\varepsilon\}$)
- Ogni simbolo dell'alfabeto terminale Σ è una RE (denota $\{a\}$, $a \in \Sigma$)
- Siano r e s due RE, allora:
  - (r.s) è una RE (denota r concatenato s)
  - (r | s) è una RE (denota r unione s)
  - (r)* è una RE (denota il + piccolo soprainsieme di r contenente ε e chiuso rispetto a .)
- niente altro è una RE

RE e grammatiche regolari

- Le RE denotano esattamente i linguaggi regolari (stessa potenza GR, AF)
- Dimostrazione abbastanza facile:
  - prima parte banale: regolari chiusi rispetto concat, *, unione
  - dato GR G, trovare RE r t.c. L(G)=L(r)

Senza entrare nei dettagli, consideriamo la grammatica ristretta:

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

$S \rightarrow aC$, $C \rightarrow cC$, $C \rightarrow \varepsilon$

proviamo a generare...

• Da S scegliamo A:

a(b|c)

a(b|c)a(b|c)

a(b|c)a(b|c)a(b|c)

...

ergo: a(b|c)(a(b|c))* -- si puo` scrivere anche come $(a(b|c))^+$

• Proviamo con C:

a, ac, acc, accc...

L'espressione regolare corrispondente e`:

E = (a(b|c)(a(b|c))* | a(c)*)

In pratica

- RE molto utilizzate in Informatica:
  - analisi lessicale linguaggi artificiali (es. lex)
  - cerca&rimpiazza sofisticati in vari editor e utilità di sistema (emacs, vi, grep, awk...)
  - linguaggi di scripting, es. Perl, Python, Ruby, Guile/Scheme...
- Esiste uno standard POSIX (standard api per unix/linux) anche per RE
- Sintatticamente le RE "pratiche" sono un po' diverse da quanto visto prima...

POSIX RE

- Metacaratteri: ( ) . [ ] ^ \ $ * + ? | { }
- Attenzione: . viene utilizzato per indicare un carattere qualsiasi, non per concatenare!
- [α] denota un singolo carattere $\in \alpha$ (es. [abc] = {a,b,c}. Si può anche dire [a-z], che indica una qualunque lettera minuscola)
- [^α]: negazione: un qualunque simbolo $\notin \alpha$ (es. [^a-z] è un qualsiasi carattere che non sia una lettera minuscola)

- ^ e $ denotano ε risp. all'inizio e fine di una riga di testo
- *,+,| sono i soliti, come pure le parentesi tonde
- \ serve come "escape" (es. \$ denota il carattere $)

Qualche esempio:

>> 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 \{ .

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

SYNOPOSIS

DESCRIPTION

DIAGNOSTICS

BUGS

>> 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.

Ancora qualche operatore...

$\alpha?$ -- α e` opzionale

$\alpha\{n\}$ -- α^n

$\alpha\{n,m\}$ -- $\alpha^n \cup \alpha^{n+1} \cup \alpha^{n+2} \cup .. \cup \alpha^m$

Esempi

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

grep [-[[AB]]<num>] [-[CEFGLSVbchilnqsvwx?]] [-[ef]] <expr> [<files...>]
available functionality using either syntax. In other implementations,

Un po’ di pratica: traslazione sottotitoli

- traslazione di orari per sottotitoli in formato .srt in Python

- esempio di sottotitolo:

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

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

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

re.VERBOSE:
Ignora gli spazi all'interno della RE, pero` ci serve \s (spazi di ogni genere = [\t\n\r\f\v])

?P<nome> serve a dare un nome ad un blocco tra parentesi (per poterlo poi richiamare)

```
def gettime(retime, s) :
    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'))]
```

Esempio d'uso:

```
trasla_srt("ilMioFilm.srt", [1,25,37,00])
```

Funzioni ausiliarie:

```
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 prcifra(n,ncifre) :
        h = str(n)
        while len(h) < ncifre : h = '0'+h
        return h
    return prcifra(t[0],2)+':'+prcifra(t[1],2)+':' \
           +prcifra(t[2],2)+','+prcifra(t[3],3)
```