Regular Expressions

สุธี สุดประเสริฐ

ใช้ข้อมูลบางส่วนจาก: Steve Renals – Regular Expressions
http://www.inf.ed.ac.uk/teaching/courses/icl/lectures/2006/regexp-lec.pdf

จุดมุ่งหมาย

- ให้นิสิตมีความพื้นฐานในเรื่องนิพจน์ปกติ (regular expressions) และสามารถนำไปประยุกต์ในการใช้งานได้
- สามารถใช้ Python ในการเขียน RE ได้

การประยุกต์ใช้งาน

- ค้นหาสายอักขระ (string) ที่อยู่ในรูปแบบที่ต้องการ
- แล้วจะหาทำไม?
  - ตรวจสอบการเติมข้อมูลในช่องว่าง เช่น อีเมล์
  - ตัวกรองคำ (คำหยาบ คำต้องห้าม)
  - แปลสายอักขระในรูปแบบหนึ่งให้อยู่ในอีกรูปแบบหนึ่ง เช่น ***cat***/***N*** → <word pos="***N***">***cat***</word>
  - อื่นๆ อีกมากมาย

แนวคิดพื้นฐาน

- RE เป็นการสร้างภาษาที่ใช้สำหรับแสดงรูปแบบหนึ่งๆ ที่ต้องการ
- ในที่นี้ภาษาคือเซตของสายอักขระ (string)
- ดังนั้น RE ใช้ในการกำหนดเซตของสายอักขระ
- สายอักขระ คือ ตัวหนังสือ ตัวเลข เครื่องหมาย ช่องว่าง เป็นต้น

ตัวอย่างเบื้องต้น

	รูปแบบ	ผลลัพธ์
cancatenation	abc	abc
disjunction	a | b (a | bb) d	a, b ad, bbd
kleen closure	a*	ε ,a, aa, aaa, ...
positive closure	a+	a, aa, aaa, ...

Formal Definition

Say that *R* is a regular expression if R is

1. a for some a in the alphabet ,
2. ε, (the empty string)
3. $\emptyset$, (the language that doesn't contain any strings)
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expression.
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expression.
6. (R_1^*), where R_1 is a regular expression.

Formal Definition

Is this a circular definition?

- Do we use the notion of regular expression in terms of itself ?
- No, because R_1 and R_2 always are smaller than R.

A definition of this type is called an ***inductive definition***.

Formal Definition

- For convenience, we let R^+ be shorthand for RR^*. So $R^+ \cup \varepsilon = R^*$.
- And we let R^k be shorthand for the concatenation of k R's with each other.
- We write L(R) to be the language of R.

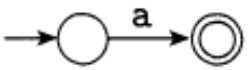
Finite State Automata

Regular expressions and finite automaton are equivalent in their descriptive power.

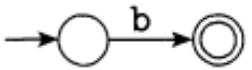
- Any regular expression can be converted into a finite automaton that recognizes the language it describes, and vice versa.
- **Regular languages** are those that can be recognized by FSA or characterized by RE.

RE vs. FSA

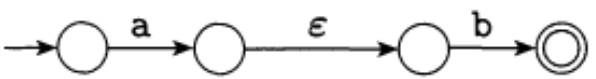
a



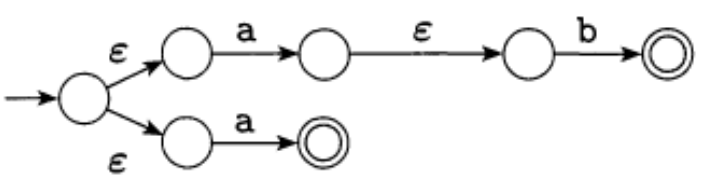
b



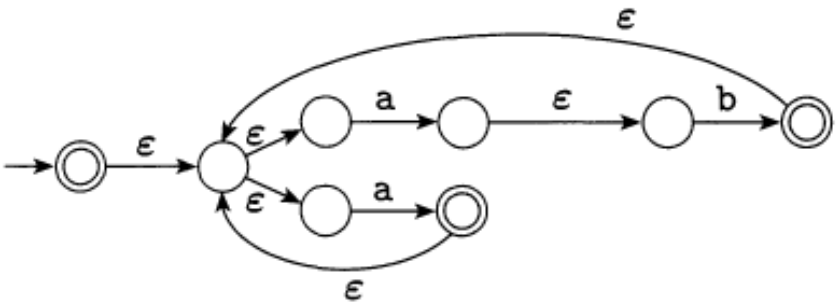
ab



ab ∪ a



(ab ∪ a)*



ชนิดของ RE

Literals

- อักขระปกติทุกตัวเป็น RE ซึ่งแทนตัวมันเอง

Metacharacters

- อักขระพิเศษซึ่งใช้แทนเซตของสายอักขระ เช่น a*, a+, เป็นต้น

Metacharacters

Metacharacters ในภาษาหลักๆ จะใช้สัญลักษณ์เหมือนกันเช่น

- Disjunction: |
- Wild card: . หมายถึง อักขระทุกชนิด
- Optionality: ? เช่น a? = a | ε
- Quantification: * และ +
- Choice: [abcde] อักขระ a b c d และ e
- Range: [a-z] อักขระ a ถึง z
- Negation: [^Mm] อักขระทุกตัว ยกเว้น M และ m

Special Backslash Sequence

Stand escape sequences

\t: tab, \n: newline

Abbreviatory forms (ตัวย่อ)

\d: digit [0-9] \D: non-digit

\s: whitespace [\t\n] \S: non-whitespace

\w: alphanumeric [a-zA-Z0-9] \W: non-alphanumeric

\\ ใช้แทนอักขร \ (backslash)

\. ใช้แทนอักขระ . (period)

Anchors

หรือเรียกว่า zero-width characters

Anchors ใช้ในการหา "ตำแหน่ง" ในสายอักขระ

- ^: จับคู่ตำแหน่งเริ่มต้นของบรรทัด
- $: จับคู่ตำแหน่งสุดท้ายของบรรทัด
- \b: จับคู่ตำแหน่งเริ่มต้นหรือสุดท้ายของคำ

Raw string

- การใช้ Regex ใน Python มักนิยมใช้ raw string ในการเขียนรูปแบบจาก Regex
  - `r'Hello World\n'`
  - `'Hello World\\n'`
- ใน `raw string` ตัวอักษรที่เขียนลงไปจะไม่มีการเปลี่ยนเป็นอักษรพิเศษ

Multiple-line string

- '''...''' หรือ """..""" ใช้ในการแสดงสตริงแบบหลายบรรทัด

```
'''
Hi, Jane

   I'm fine and how are you?

Best,
John
'''
```

```
"\nHi, Jane\n\n   I'm fine and how are you?\n\nBest,\nJohn\n"
```

Unicode string

- `u"..."` ใช้ในการแสดงสตริงในรูปแบบ unicode

```
>>> 'การ'
'\xe0\xb8\x81\xe0\xb8\xb2\xe0\xb8\xa3'
>>> u'การ'
u'\u0e01\u0e32\u0e23'
>>> len('การ')
9
>>> len(u'การ')
3
```

Example

- Wildcard

```
>>> from nltk.util import re_show
>>> s = '''If you find it useful,
... please consider donating to NLTK via PayPal.'''
>>> re_show('...',s)
{If }{you}{ fi}{nd }{it }{use}{ful},
{ple}{ase}{ co}{nsi}{der}{ do}{nat}{ing}{ to}{ NL}{TK }{via}{ Pa}{yPa}l.
```

Example

- Wildcard

```
>>> from nltk.util import re_show
>>> s = '''If you find it useful,
... please consider donating to NLTK via PayPal.'''
>>> re_show('...',s)
{If }{you}{ fi}{nd }{it }{use}{ful},
{ple}{ase}{ co}{nsi}{der}{ do}{nat}{ing}{ to}{ NL}{TK }{via}{ Pa}{yPa}l.
>>> re_show('.a.',s)
If you find it useful,
pl{eas}e consider do{nat}ing to NLTK v{ia }{Pay}{Pal}.
```

Example

- Wildcard with quantifiers

```
>>> re_show('c.*r',s)
If you find it useful,
please {consider} donating to NLTK via PayPal.
```

Example

- Wildcard with quantifiers

```
>>> re_show('c.*r',s)
If you find it useful,
please {consider} donating to NLTK via PayPal.
>>> re_show(' t.*o ',s)
If you find it useful,
please consider donating{ to }NLTK via PayPal.
```

Example

- Wildcard with quantifiers

```
>>> re_show('c.*r',s)
If you find it useful,
please {consider} donating to NLTK via PayPal.
>>> re_show(' t.*o ',s)
If you find it useful,
please consider donating{ to }NLTK via PayPal.
>>> re_show(' t.+o ',s)
If you find it useful,
please consider donating to NLTK via PayPal.
```

Example

- Non-greedy quantifiers

```
>>> re_show('I.*f',s)
{If you find it usef}ul,
please consider donating to NLTK via PayPal.
```

Example

- Non-greedy quantifiers

```
>>> re_show('I.*f',s)
{If you find it usef}ul,
please consider donating to NLTK via PayPal.
>>> re_show('I.*?f',s)
{If} you find it useful,
please consider donating to NLTK via PayPal.
```

Example

- Non-greedy quantifiers

```
>>> re_show('v.+a',s)
If you find it useful,
please consider donating to NLTK {via PayPa}l.
```

Example

- Non-greedy quantifiers

```
>>> re_show('v.+a',s)
If you find it useful,
please consider donating to NLTK {via PayPa}l.
>>> re_show('v.+?a',s)
If you find it useful,
please consider donating to NLTK {via} PayPal.
```

Example

- Disjunction

```
>>> re_show('you|it',s)
If {you} find {it} useful,
please consider donating to NLTK via PayPal.
```

Example

- Disjunction

```
>>> re_show('you|it',s)
If {you} find {it} useful,
please consider donating to NLTK via PayPal.
>>> re_show('(i|I).',s)
{If} you f{in}d {it} useful,
please cons{id}er donat{in}g to NLTK v{ia} PayPal.
```

Example

- Zero-width Characters

```
>>> re_show('p|P',s)
If you find it useful,
{p}lease consider donating to NLTK via {P}ay{P}al.
```

Example

- Zero-width Characters

```
>>> re_show('p|P',s)
If you find it useful,
{p}lease consider donating to NLTK via {P}ay{P}al.
>>> re_show('^(p|P)',s)
If you find it useful,
{p}lease consider donating to NLTK via PayPal.
```

Example

- Zero-width Characters

```
>>> re_show('p|P',s)
If you find it useful,
{p}lease consider donating to NLTK via {P}ay{P}al.
>>> re_show('^(p|P)',s)
If you find it usefu{l},
{p}lease consider donating to NLTK via PayPal.
>>> re_show(r'\b\w+\b',s)
{If} {you} {find} {it} {useful},
{please} {consider} {donating} {to} {NLTK} {via} {PayPal}.
```

Example

- Escaping special characters

```
>>> re_show('.',s)
{I}{f}{ }{y}{o}{u}{ }{f}{i}{n}{d}{ }{i}{t}{ }{u}{s}{e}{f}{u}{l}{,}...
```

Example

- Escaping special characters

```
>>> re_show('.',s)
{I}{f}{ }{y}{o}{u}{ }{f}{i}{n}{d}{ }{i}{t}{ }{u}{s}{e}{f}{u}{l}{,}...
>>> re_show('\.',s)
If you find it useful,
please consider donating to NLTK via PayPal{.}
```

Example

- Metacharacters and negated ranges

```
>>> re_show('\w',s)
{I}{f} {y}{o}{u} {f}{i}{n}{d} {i}{t} {u}{s}{e}{f}{u}{l},...
```

Example

- Metacharacters and negated ranges

```
>>> re_show('\w',s)
{I}{f} {y}{o}{u} {f}{i}{n}{d} {i}{t} {u}{s}{e}{f}{u}{l},...
>>> re_show('[^a-z\s]',s)
{I}f you find it useful{,}
please consider donating to {N}{L}{T}{K} via {P}ay{P}al{.}
```

Example

- Metacharacters and negated ranges

```
>>> re_show('\w',s)
{I}{f} {y}{o}{u} {f}{i}{n}{d} {i}{t} {u}{s}{e}{f}{u}{l},...
>>> re_show('[^a-z\s]',s)
{I}f you find it useful{,}
please consider donating to {N}{L}{T}{K} via {P}ay{P}al{.}
>>> re_show('[^\w]',s)
If{ }you{ }find{ }it{ }useful{,}{
}please{ }consider{ }donating{ }to{ }NLTK{ }via{ }PayPal{.}
```

Flags

- re.I (re.IGNORECASE)
- re.M (re.MULTILINE)
  - ^ และ $ จะจับคู่ ก่อนและหลัง newline

```
>>> print re.search(r'^test$', 'test')
>>> <_sre.SRE_Match object at 0x552ca0>
>>> print re.search('^test$', 'test', flags=re.M)
>>> None

>>> print re.search(r'^test$', '\ntest\n', flags=re.M)
>>> <_sre.SRE_Match object at 0x54f170>
```

Flags

- re.S (re.DOTALL)
  - ทำให้ '.' จับคู่กับอักษรทุกตัว รวมถึง newline ด้วย

```
>>> html = '''
<div>
  Hello world
</div>'''

>>> pat = r'<div>.+</div>'
>>> print re.search(pat,html)
None

>>> print re.search(pat,html,re.S)
<_sre.SRE_Match object at 0x54f170>
```

Flags

- re.X (re.VERBOSE)
  - ใช้สำหรับทำให้รูปการเขียน regex อ่านได้ง่ายขึ้น เนื่องจาก newline จะไม่ถูกสนใจ (ถ้าไม่ถูกนำด้วย \ หรืออยู่ใน character class) และ ข้อความหลัง # ทั้งหมดจะไม่ถูกนำมาคิด

```
a = re.compile(r"""\d + # the integral part
                   \.   # the decimal point
                   \d * # some fractional digits""", re.X)
b = re.compile(r"\d+\.\d*")
```

Using REs in Python

```
>>> import re
>>> s = 'do you say hello or hullo?'
>>> helloRE = re.compile('h[eu]llo')
```

เริ่มต้นสร้าง Pattern Object ก่อน (การทำเช่นนี้ ทำให้เราสามารถนำ Pattern Object มาใช้ได้อีกครั้ง)

Pattern Object มี methods จำนวนมาก หลักๆ ที่สำคัญมีดังนี้

- findall(s): ให้คำตอบเป็น list ของรูปแบบทั้งหมดใน s ที่จับคู่ได้
- search(s): หารูปแบบใน s โดยเริ่มจากซ้ายสุด
- match(s): พยายามจับคู่ s โดยเริ่มตั้งแต่ต้นประโยค

Using REs in Python

findall(s)

```
>>> helloRE.findall(s)
['hello', 'hullo']
```

Using REs in Python

match(s) และ search(s) ให้ค่าเป็น MatchObject หรือ None

```
>>> m = helloRE.search(s)
>>> m
<_sre.SRE_Match object at 0x019E7480>
>>> m.group() # return matched substring
'hello'
>>> m.end() # index of end of target
16
>>> m = helloRE.match(s)
>>> m
None
```

Using REs in Python

เปรียบเทียบ match(s) กับ search(s)

```
>>> m = helloRE.match('hello or hullo') # True
>>> m = helloRE.search('hello or hullo') # True

>>> m = helloRE.match('hallo or hullo') # False
>>> m = helloRE.search('hallo or hullo') # True
```

Using REs in Python

groups() จะให้ผลลัพธ์จากการจับคู่ในวงเล็บ

```
>>> import re
>>> s = 'do you say hello or hullo?'
>>> reGRP = re.compile('(d.)(.*)(e..)')
>>> m = reGRP.search(s)
>>> m
<_sre.SRE_Match object at 0x019E82A0>
>>> m.groups()
('do', ' you say h', 'ell')
```

Using REs in Python

การกำหนดชื่อให้กลุ่ม (?P<name>):

```
>>> re.compile("""
    ^From:            # Anchor to start of line
    \s*               # maybe some spaces
    (?P<user>\w+)     # 'user' groups (alphanum chars)
    @
    (?P<domain>\S+)   # 'domain' groups (non-space chars)
    \s                # a space
    """, re.VERBOSE)
```

Using REs in Python

```
>>> text = 'From: fscistsu@ku.ac.th '
>>> m = FROM.search(text)
>>> print '%s is at %s' % \
    (m.group('user'),m.group('domain'))
fscistsu is at ku.ac.th
```

เอกสารเพิ่มเติม

Python Documentation:

- http://docs.python.org/library/re.html#module-re

NLTK Tutorial: Chapter 3

- http://nltk.googlecode.com/svn/trunk/doc/book/ch03.html