Regular Expressions

Object Oriented Programming

http://softeng.polito.it/courses/09CBI

Version 2.0.0 - May 2018

Regular Expression

- Represent a simple and efficient way to describe a sequence of characters
- They can be used to:
  - generate a conforming sequence of chars
  - recognize a sequence of chars as conforming with the RE
- The ability to recognize a valid sequence is fundamental in text processing.

Regular Expressions

- Represent a simple and efficient way to describe sets of character strings
- Operators allow representing:

characters	`c`
sets of characters	`[abc]` o `[a-c]`
optionality	*exp* `?`
repetition (0 o more)	*exp* `*`
repetition (1 o more)	*exp* `+`
alternatives	*exp1* `|` *exp2*
concatenation	*exp1exp2*
grouping	`(` *exp* `)`

Examples of RE

- Positive integer number
  - `[0-9]+`
- Positive integer number w/o leading 0
  - `[1-9][0-9]*`
- Integer number positive or negative
  - `[+-]?[0-9]+`
- Floating point number
  - `[+-]?(([0-9]+\.[0-9]*)|`
    `([0-9]*\.[0-9]+))`

Regular expressions

- RE can be used to check whether an input string correspond to a given set
- RE describes a sequence of characters and use a set of operators:
  - `" \ [ ] ^ - ? . * + | ( ) $ / { } % < >`
- Letters and numbers in the input text are described by themselves
  - `val1` represents the sequence `'v'` `'a'` `'l'` `'1'` in the input text

Character set

- Character sets are described using `[]`:
  - `[0123456789]` represents any integer number
- In a set, the symbol `-` indicates a range of characters:
  - `[0-9]` represents any numeric character
- To include `-` in the set, it must be first or last char:
  - `[-+0-9]` represents a number in the input text.
- When a set begins with `^`, the characters are excluded:
  - `[^0-9]` represents any non numeric character
- The set of all characters except new line can be described by a dot: `.`

Special characters

- The new-line is represented by `\n`
- Any white space is described by `\s`
- Any digit is described by `\d`,
  - i.e. `[0-9]`
- Any word char is described by `\w`,
  - i.e. `[A-Za-z0-9_]`
- The beginning of text is `^`
- The end of text is `$`

SoftEng http://softeng.polito.it

Optional and alternative

- The operator **?** makes the preceding expression optional:
  - **ab?c** represents both ac and abc.
- The operator **|** represents an alternative between two expressions:
  - **ab|cd** represnts both the sequence ab and the sequence cd.
- The round parentheses, **(** and **)**, allow expressing a grouping to define the priorities among operators
  - **(ab|cd+)?ef** represents such sequences as ef, abef, cdddef, etc.

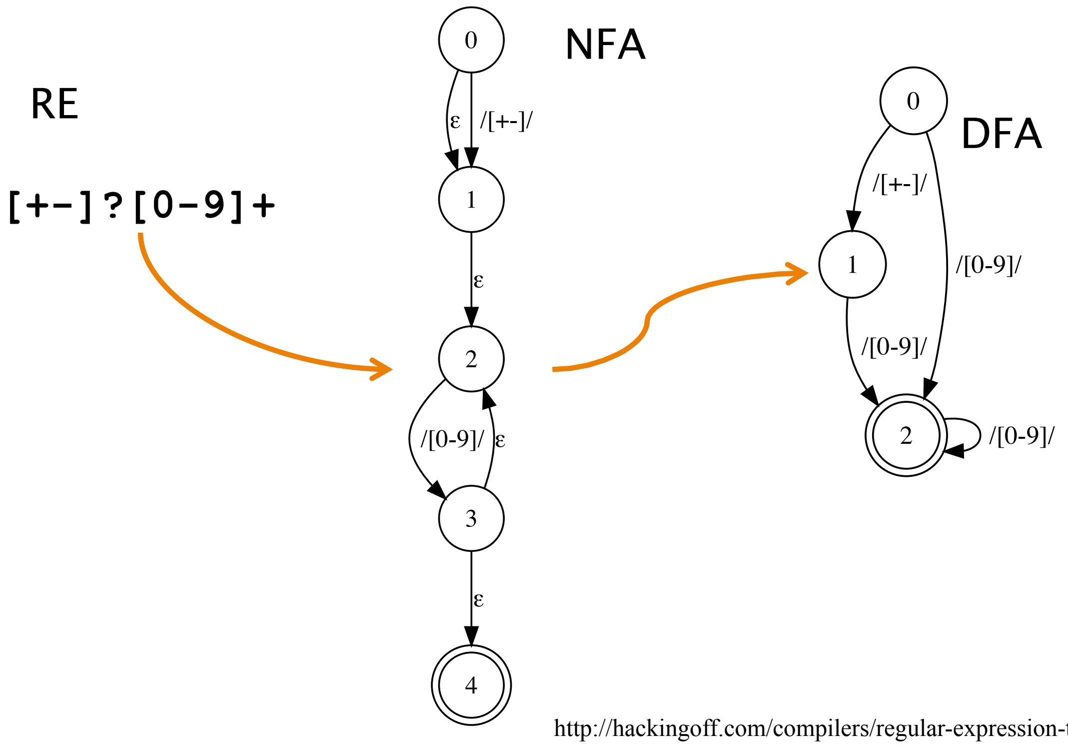
Repetitions

- The operator **+** makes the preceding expression can be repeated 1 or more times:
  - **ab+c** represents sequences starting by a, ending in c, and containing at least one b.
- The operator ***** indicates the preceding expression can be repeated 0 or more times:
  - **ab*c** represents sequences starting by a, ending in c, and containing any number of b.
- The operator **{l,h}** matches from *l* to *h* repetitions of the preceding expression

Recognizer

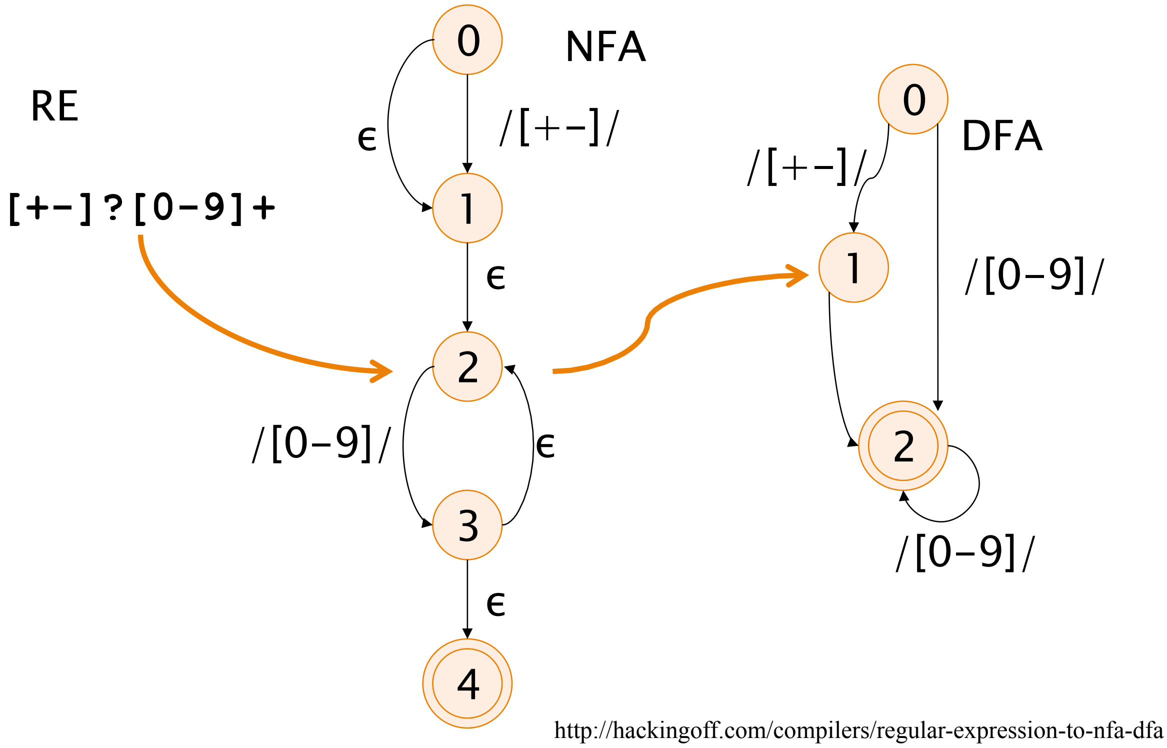
- An RE can be transformed into NFA (Non-deterministic Finite-state Automata)
  - Using the Algorithm Thompson-McNaughton-Yamada
- Then an NFA can be transformed into a DFA (Deterministic F-s A)
- A DFA can be encoded into a table that defines the rules *executed* by a state machine to recognize a sequence of characters

Recognizer example



http://hackingoff.com/compilers/regular-expression-to-nfa-dfa

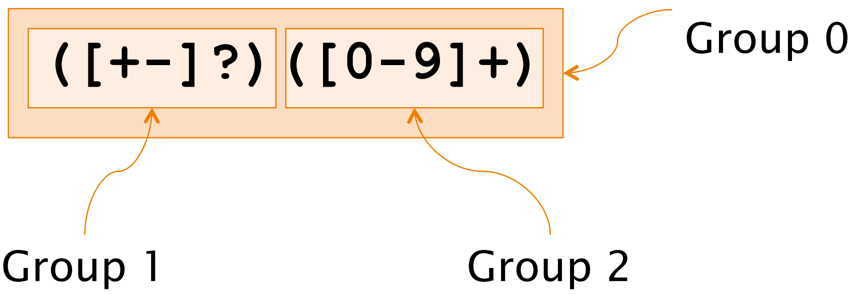
Recognizer example



http://hackingoff.com/compilers/regular-expression-to-nfa-dfa

Capture groups

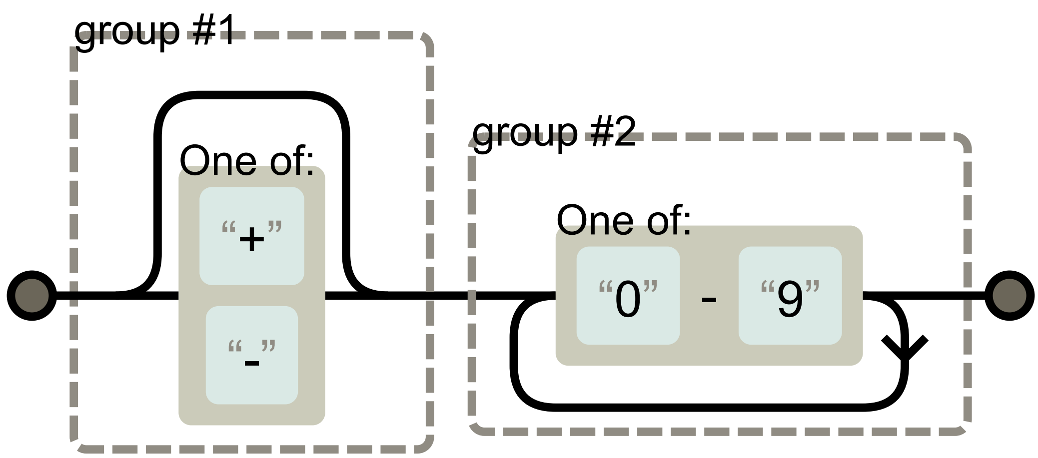
- Every pair of matching parentheses defines a capture group
  - Group 0 is the whole matched string



  - Non capturing group: **(?:E)**

Railroad diagram

`([+-]?)([0-9]+)`



Generated with: http://regexper.com

Context

- Look-behind
  - **(?<=E)** means that **E** must precede the following RE, though E is not part of the recognized RE
  - **(?<!E)** means **E** must not precede
- Look-ahead
  - **(?=E)** means that **E** must follow the preceding RE, though E is not part of the recognized RE
  - **(?!E)** means that **E** must not follow

REGEXP IN JAVA

RegExp in Java

- Package
  - `java.util.regex`
- **`Pattern`** represents the automata:

```
Pattern p=Pattern.compile("[+-]?[0-9]+");
```

- **`Matcher`** represents the recognizer

```
Matcher m = p.matcher("-4560");
boolean b = m.matches();
```

Matcher

- Three recognition modes
  - `matches()`
    - Attemp matching the whole string
  - `lookingAt()`
    - Attempt a partial matching starting from beginning
  - `find()`
    - Attempt matching any substring
- Recognized string:
  - `group()`

Capture groups

```
([+-]?)([0-9]+)
```

```
m = p.matcher("-4560");
if(m.matches()){
  for(int i=0; i<=m.groupCount(); ++i){
    System.out.println("Group "+i+" : '"
                        + m.group(i) + "'");
  }
}
```

```
Group 0 : '-4560'
Group 1 : '-'
Group 2 : '4560'
```

Example: CSV with groups

```
(,|^|\n|\r|\r\n)
[ \t]*
(?:([^\",\n\r]*)
|"((?:[^"]*|"")*)")
[ \t]*
```

Group 1 : preceding delimiter

Group 2 : normal cell

Group 3 : delimited cell

- When translating to a string in the code pay attention to special characters:
  - Backslash: \
  - Quotes: "

Example: CSV

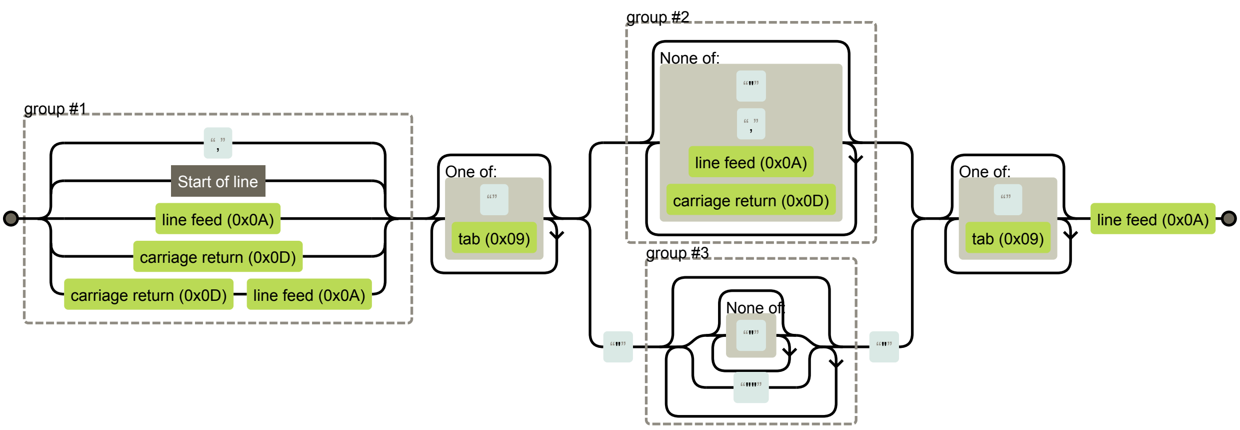
```
Pattern re  = Pattern.compile(
"(,|^|\n|\r|\r\n)" +     // G1  : prec sep
"[ \t]*" +               // -   : lead spaces
"(?:([^\",\n\r]*)" +     // G2  : normal cell
"|\"((?:[^\"]*|\"\")*)\")"+//G3: delim cell
"[ \t]*"                  // - : trail spaces
);
```

Example: CSV

```
Matcher m = re.matcher(csvContent);
while(m.find()){
  if(!m.group(1).equals(","))   // new row
     System.out.println("Row:");
  String c = m.group(2);
  if(cell==null)
    c = m.group(3).replaceAll("\"\"","\"");
  System.out.println("\tCell:" + c);
}
```


Example CSV – Context

- Railroad diagram



Generated with: http://regexper.com

Named groups

- Capture groups can be named:
  - E.g. `(?<c>[^\",]*)`
- Named groups can be accessed using `group()` method:
  - E.g. `c = m.group("c");`

SOftEng http://softeng.polito.it

Example: CSV

```
Pattern re  = Pattern.compile(
"(?<sep>,|^|\n|\r|\r\n)" +// G1 : prec sep
"[ \t]*" +                // -  : lead spaces
"(?:(?<c>[^\",\n\r]*)" +// G2 : normal cell
"|\"(?<dc>(?:[^\"]*|\"\")*)\")"+//G3: delim
"[ \t]*"                  // - : trail spaces
);
```

SOftEng http://softeng.polito.it

Example: CSV named groups

```
Matcher m = re.matcher(csvContent);
while(m.find()){
  if(!m.group("sep").equals(",")) //new row
     System.out.println("Row:");
  String c = m.group("c");
  if(cell==null)
   c=m.group("dc").replaceAll("\"\"","\"");
  System.out.println("\tCell:" + c);
}
```


Class **Scanner**

- A basic parser that can read primitive types and strings using regular expressions
- Basic usage
  - Construction from a stream, file, or string
    - E.g. `new Scanner(new File("file.txt"))`
  - Check present of ***next*** token (optional)
    - E.g. `hasNextInt()`
  - Detection of ***next*** token:
    - E.g. `nextInt()`

Scanner advanced usage

```
File file = new File("file.csv");
try(Scanner fs = new Scanner(file)){
while(true){
  String c;
  while((c=fs.findInLine(pattern))!=null){
    System.out.println(c);
  }
  if(!fs.hasNextLine()) break;
  fs.nextLine();
}}
```


Summary

- Regular expression express complex sequences of characters
- Used to recognize parts of strings
  - **`Pattern`** contains the DFA
  - **`Matcher`** implements the recognizer
- RE are used extensively
  - String: **`replaceAll()`**, **`split()`**
  - Scanner: **`findInLine()`**