Regular Expressions and Pattern Matching

Philipp Reinecke

ReineckeP@cardiff.ac.uk

Mid-Module Feedback

https://forms.office.com/Pages/ResponsePage.aspx?id=MEu3vWiVVki9vwZ1l3j8vOXDWRZEM-JCmeKQSO6KdXpUNVZHS0ZVTVNXWlE4VFZZNVNBVTQ0SkJCNy4u

Goals

- Pattern matching: Check a sequence of tokens for presence of some pattern
- Uses:
  - Finding strings
  - Search and replace
  - Splitting strings
- Goals of the lecture:
  - Understand basics of pattern matching with regular expressions
  - Understand use of regular expressions in Perl

Why regular expressions?

- Text processing: Find all first words of a sentence

“It is the nature of an hypothesis, when once a man has conceived it, that it assimilates every thing to itself, as proper nourishment; and, from the first moment of your begetting it, it generally grows the stronger by every thing you see, hear, read, or understand. This is of great use.”

Laurence Sterne, the Life and Opinions of Tristram Shandy, Gentleman

```
$i = 0;
while ($i <= $#s) {
  if ($s[$i] == ‘.’ || $i == 0) {
     $i++;
     while ($s[$i++] == ‘ ‘) {}
     $w = ‘‘;
     while ($s[$i] != ‘ ‘
              && $s[$i] != ‘.’
              && $s[$i= != ‘,’
              && $s[$i= != ‘;’) {
         $w .= $s[$i];
         $i++;
     }
         print “$w\n”;]
       } else {
         $i++;
     }
}
```

```
while ($s =~ /(\.|^)\s*(\w+)/g) {
   print “$2\n”;
}
```

Why regular expressions?

- Search and replace: Put an HTML newline after each sentence

> “It is the nature of an hypothesis, when once a man has conceived it, that it assimilates every thing to itself, as proper nourishment; and, from the first moment of your begetting it, it generally grows the stronger by every thing you see, hear, read, or understand. This is of great use.”
>
> *Laurence Sterne, the Life and Opinions of Tristram Shandy, Gentleman*

```
$i = 0;
$r = “”;
while ($i <= $#s) {
  $r .= $s[$i];
  if ($s[$i] == ‘.’
      || $s[$i] == ‘!’
      || $s[$i] == ‘?’) {
     $r .= “<br>\n”;
  }
  $i++;
}
print “$r”;
```

```
$s =~ s/([.!?])/\1\<br\>\n/g;
print “$s”;
```

Why regular expressions?

- Split into words:

  “It is the nature of an hypothesis, when once a man has conceived it, that it assimilates every thing to itself, as proper nourishment; and, from the first moment of your begetting it, it generally grows the stronger by every thing you see, hear, read, or understand. This is of great use.”

  Laurence Sterne, the Life and Opinions of Tristram Shandy, Gentleman

```
$i = 0;
while ($i <= $#s) {
  while ($s[$i] == ‘ ’    || $s[$i] == ‘,’
         || $s[$i] == ‘;’ || $s[$i] == ‘–’
         || $s[$i] == ‘”’ || $s[$i] == ‘!’
         || $s[$i] == ‘?’) {
     $i++;
  }
  $w = “”;
  until ($s[$i] == ‘ ’    || $s[$i] == ‘,’
         || $s[$i] == ‘;’ || $s[$i] == ‘–’
         || $s[$i] == ‘”’ || $s[$i] == ‘!’
         || $s[$i] == ‘?’) {
     $w .= $s[$i]
     $i++;
  }
  print “$w\n”;
}
```

```
while ($s =~ /(\w+)/g) {
   print “$1\n”;
}
```

What is a regular expression?

- A load of incomprehensible gibberish
- Compact expression for complex program
- Formally:

  "A sequence of characters that define a search pattern."

  https://en.Wikipedia.org/wiki/Regular_expression
- An encoding of a DFA or NFA
  - Deterministic Finite Automaton
  - Nondeterministic Finite Automaton
- Note: Regular expression engines in practice are typically more powerful than DFA/NFA

We will try to change that bit.

Basics: Alphabets, Strings, and Languages

- Alphabet:
  - Finite set Σ
  - Elements: Symbols
  - Examples:
    - Σ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
    - Σ = {a, b, …, x, y, z}
- String:
  - Concatenation of symbols, with finite length
  - Null string: ε
  - Examples:
    - 42
    - hello
  - Σ*: Set of all strings over Σ with finite length
- Language:
  - Any subset $L \subseteq \Sigma^*$
  - Examples:
    - {42, 1, 34, 7}
    - {hello, test, regex, perl}

Basics: Formal definitions

Regular expressions

Given an alphabet Σ,

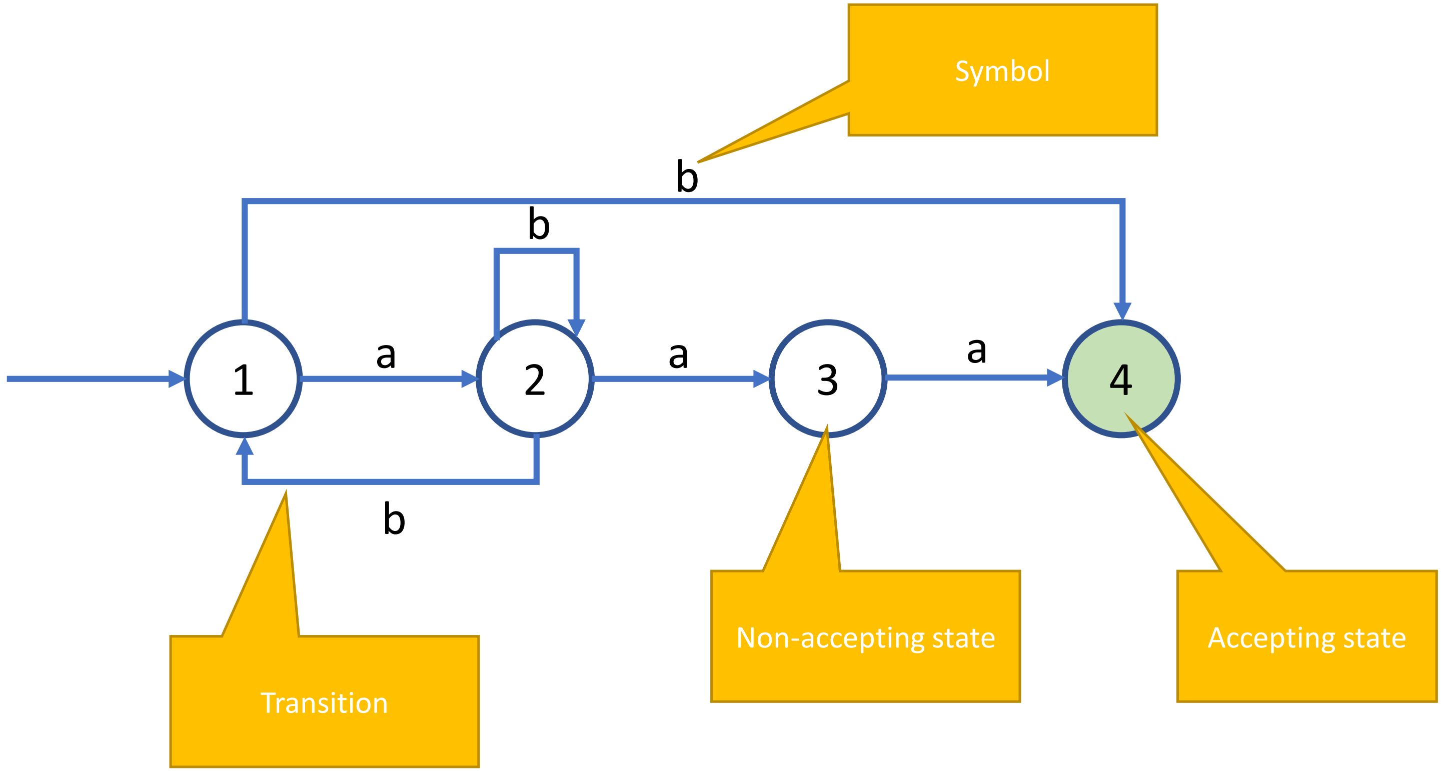
- Every symbol $a \in \Sigma$ is a regular expression
- ε is a regular expression
- If r, s are regular expressions, then
  - $(r|s)$ is a regular expression
    - Meaning: “r OR s”
  - rs is a regular expression
    - Meaning: “r followed by s”
  - $r *$ is a regular expression
    - Meaning: “zero or more repetitions of r”
- Any expression built by finitely many applications of these rules is a regular expression

Matching

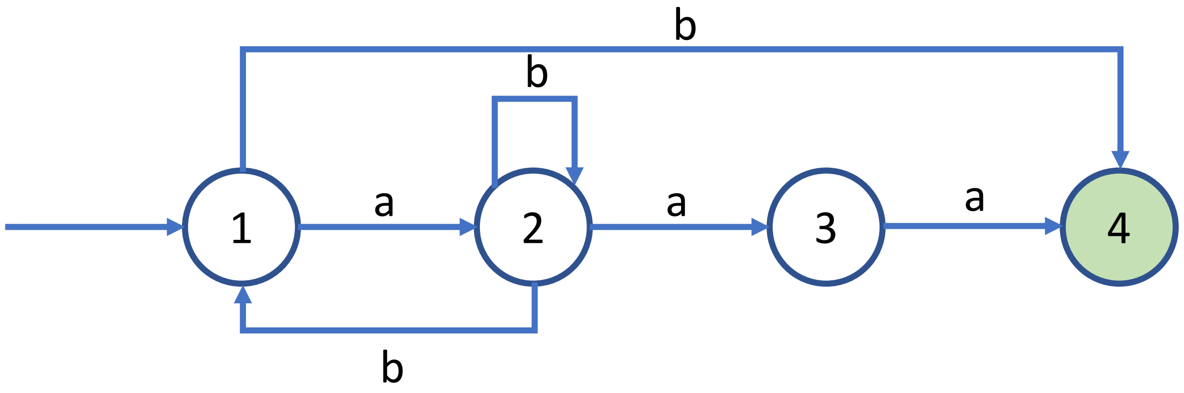
A string u matches an expression v

- if $u = v$
- if $v = (r|s)$ and ($u = r\ OR\ u = s$)
- if $v = rs$ and
  - $u = u_1 u_2$
  - and u_1 matches r and u_2 matches s
- if $v = r *$ and
  - $u = \epsilon$
  - or $u = u_1 u_2 u_3$ … and all u_i match r

Basics: Nondeterministic Finite Automaton



Basics: Matching with an NFA



An NFA matches a string u if

- it can produce u by a sequence of steps ending in an accepting state
  or, equivalently,
- we can consume all symbols of u in order by traversing the matching transitions and end up in an accepting state

Examples: Does this NFA match these strings?

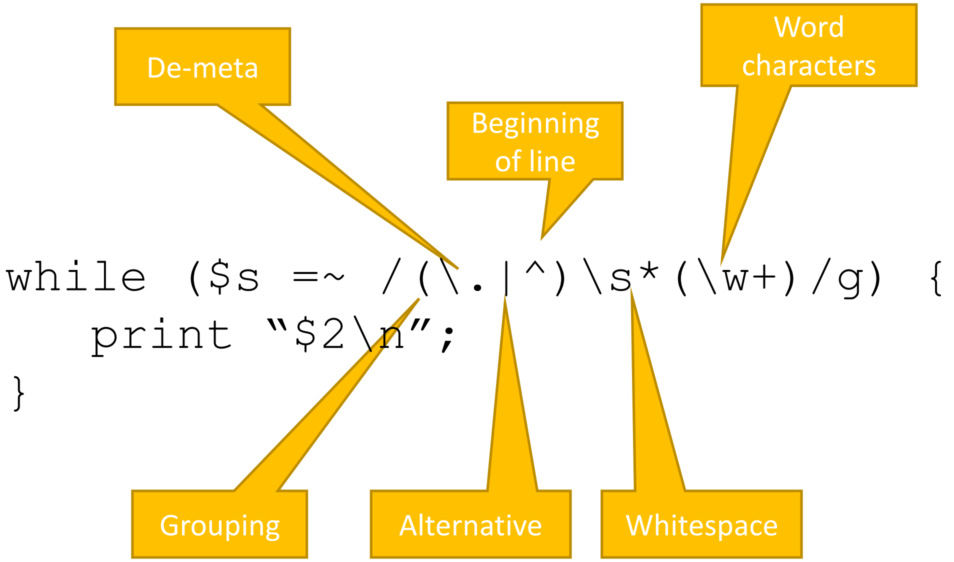
• aaa	Yes
• abbbbbbaa	Yes
• abababaa	Yes
• a	No
• baaa	No
• aaaa	No

Regular expressions in Perl

- Regular expressions define an NFA
  - actually, Perl's regex engine is a bit more powerful
- Regular expressions tightly integrated
  - Matching: `$s =~ m/EXPRESSION/OPTIONS`
    or `$s =~ /EXPRESSION/OPTIONS`
  - Substitution: `$s =~ s/SEARCH/REPLACE/OPTIONS`
  - Transliteration: `$s =~ tr/SEARCH/REPLACE/OPTIONS`
- Splitting strings:
  - `@a=split /PATTERN/, STRING`

Regular expressions: Metacharacters

- Usually mean something
  - a|b – match a or b
  - (…) – grouping:
    - Match group
    - Capture
  - […] – character class:
    - Match one from set
    - Match except set: [^SET]
  - ^, $ -- Match beginning/end of string, respectively
  - . – match one character
  - \ -- de-meta next meta character, or turn non-meta into meta-character
- Important metasymbols:
  - \s – whitespace, \S – non-whitespace
  - \w – word character, \W – non-word character
  - \d – digit, \D – non-digit



```
while ($s =~ /(\.|^)\s*(\w+)/g) {
    print “$2\n”;
}
```

De-meta

Beginning of line

Word characters

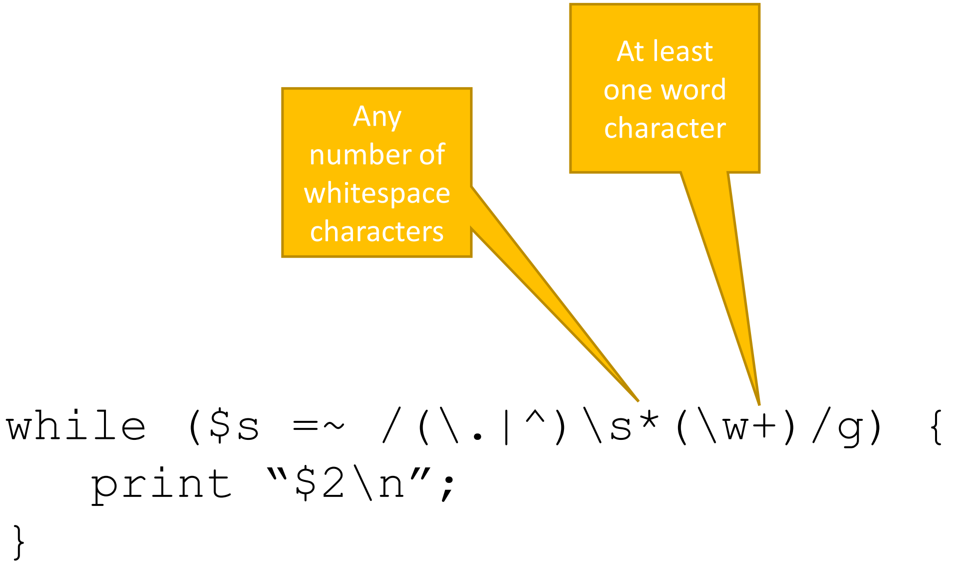
Grouping

Alternative

Whitespace

Regular expressions: Quantifiers

- Apply to preceding expression
- Maximal matching:
  - * -- match 0 or more times
  - + -- match 1 or more times
  - ? – match 0 or 1 times
  - {COUNT} – match exactly COUNT times
  - {MIN, } – match at least MIN times
  - {MIN, MAX} – match at least MIN but at most MAX times
- Minimal matching: Use ? after quantifier from above



Any number of whitespace characters

At least one word character

```
while ($s =~ /(\.|^)\s*(\w+)/g) {
    print “$2\n”;
}
```

Capturing and Clustering

- Capturing: Extract matching substring for later use
- Syntax:
  - Capturing: Use parentheses around wanted substring
  - Use (outside regex): Use $i for the i-th substring
  - Use (within regex): Use \i for ith substring
- Examples:

  -
    ```
    while ($s =~ /(\w+)/g) {
         print “$1\n”;
       }
    ```

  -
    ```
    $s =~ s/([.!?])/\1\<br\>\n/g;
    ```
- Clustering:
  - Capturing without extraction
  - Syntax: (?: PATTERN)

Substitution

- Replaces parts of string matching pattern with other strings
- Syntax:
  - $s =~ s/PATTERN/REPLACEMENT/;
- Pattern: Any regular expression
- Replacement:
  - String
  - Can use previously captured parts: \1, \2, etc.
- Example:

```
s/([.!?])/\1\<br\>\n/g;
```

Important options

- Regex operators can have options:
    - /PATTERN/OPTIONS
    - /SEARCH/REPLACE/OPTIONS
- Options modify matching behavior
- Important options:
    - /i – ignore case
    - /x – ignore whitespace and allow comments in pattern
    - /o – Compile pattern only once
    - /g – Global match

Regular expressions in Java

- Use similar syntax
- Less tightly integrated than in Perl
- Use:
  - Compile String containing pattern:

    ```
    String pattern = “\d+”;
    Pattern p = Pattern.compile(pattern);
    ```

  - Build Matcher:

    ```
    Matcher m = p.matcher(s);
    ```

  - Does the string match?

    ```
    if (m.matches()) …
    ```

  - Get all occurrences:

    ```
    while (m.find()) {
      String ss = s.substring(m.start(), m.end());
    }
    ```

  - Replace all occurrences:
    ```
    m.replaceAll()
    ```

Regular expressions in Python

- Use similar syntax
- Also less tightly integrated than in Perl
- Use:
  - Compile String containing pattern:

    ```
    import re
    pattern = ‘\d+’
    p = re.compile(pattern);
    ```

  - Does the string match?

    ```
    p.match(s)
    ```

Regular expressions elsewhere

- grep
- awk
- sed
- tr
- find
- Shell
- Often slightly different syntax and functionality → check man pages
- Useful trick: Run Perl on command-line

```
perl –e ‘while (<>) { print if /regex/; }’
```

Conclusion

- Regular expressions are a powerful tool
- Basic theory: Deterministic and Nondeterministic Finite Automata (DFA/NFA)
- Tightly integrated into Perl
- Supported by many languages, e.g. Java, Python, etc.
- Use regular expressions wisely
  - They can be confusing
  - They can be inefficient
  - It is easy to get them wrong

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Source: www.xkcd.com, Creative Commons Attribution NonCommercial 2.5 License