ACDH Digital Prosopography Summer School
Vienna 2020-07-06

Introduction to regular expressions

Gioele Barabucci

NTNU | Norwegian University of Science and Technology

Regular expressions

1. Uses and formalism
2. Syntax and concepts
3. Hands-on session
4. Beyond searching
5. Advanced topics

1. Uses and formalism

2. Syntax and concepts
3. Hands-on session
4. Beyond searching
5. Advanced topics

Example of regex

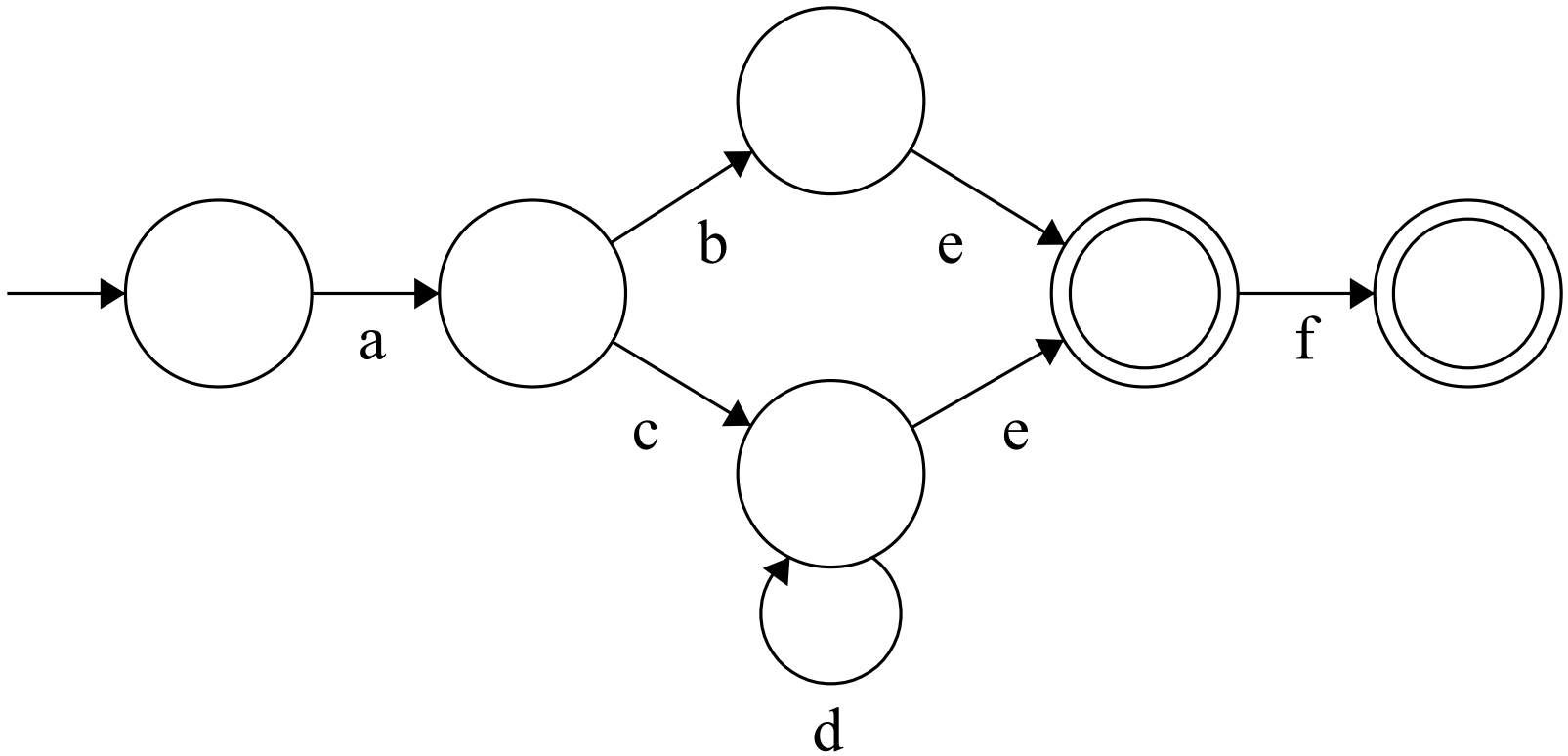
`/[A-Z][a-z]+/`

Guido, i' vorrei che tu e Lapo ed io
fossimo presi per incantamento
e messi in un vasel, ch'ad ogni vento
per mare andasse al voler vostro e mio;

[...]

E monna Vanna e monna Lagia poi
con quella ch'è sul numer de le trenta
con noi ponesse il buono incantatore:

Regex = small program



Drawn with *Finite State Machine designer* http://madebyevan.com/fsm/

1. Uses and formalism

2. Syntax and concepts

3. Hands-on session

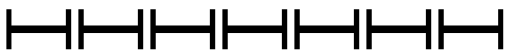
4. Beyond searching

5. Advanced topics

How regexes are written and evaluated

```
for start_idx in 0 .. len(string) {
        if (string[start_idx] == “l” &&
            string[start_idx+1] == “y” &&
            string[start_idx+2] == “r” &&
            …
            string[start_idx+6] == “ “) {
                record_match(
                    start: start_idx,
                    end: start_idx+6
                )
        }
}
```

/lyrics /



How regexes are written and evaluated

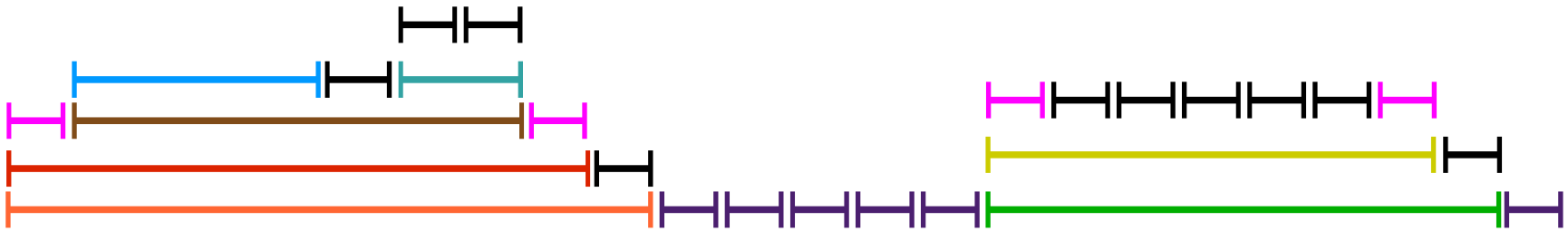
The lyrics go like “Na na na na na na na na na na, hey Jude Jude Jude Jude!”

✔ ✘

Matches = {
  [4..10] “lyrics”
}

The common syntax

```
/([Nn]a ?)+, hey( Jude)+!/
```

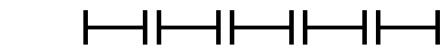


The lyrics go like “Na na na na na na na na na na, hey Jude Jude Jude Jude!”

Quantification

Identity (“a” = “a”)

/color/



color ✔

colour ✗

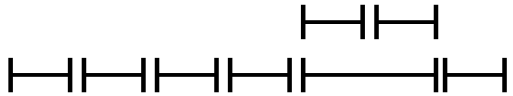
colur ✗

colouur ✗

A sequence of atoms matches if the text contains that exact sequence.

Optionality (? = zero or one)

/colou?r/

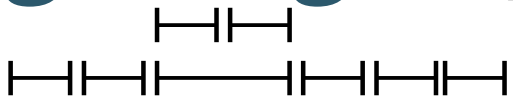


color ✔
colour ✔
colur ✗
colouur ✗

The **?** metacharacter states “this piece is matched if the wrapped atom appears in this position zero or one times”.
In other words: “the wrapped atom is optional”.

Repetition (+ = one or more)

/goo+gle/



gogle ✗

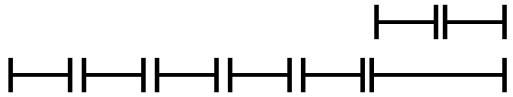
google ✔

gooogle ✔

The **+** metacharacter states “this piece is matched if the wrapped atom appears in this position one or more times”.
In other words: “the wrapped atom can be repeated”.

Optional repetition (* = zero or more)

/hurrah*/



hurra ✔

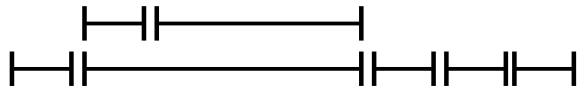
hurrah ✔

hurrahhh ✔

The * metacharacter states “this piece is matched if the wrapped atom appears in this position zero or more times”.
In other words: “the wrapped atom can be omitted or repeated”.

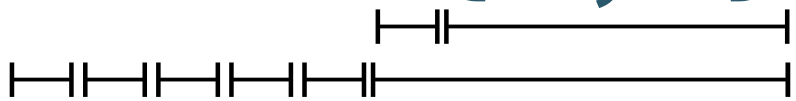
Arbitrary quantifiers ({n,m} = n to m times)

`/go{3}gle/`



google	✗
gooogle	✓
goooogle	✗

`/hurrah{2,6}/`



hurrah	✗
Hurrahhh	✓
hurrahhhh	✓

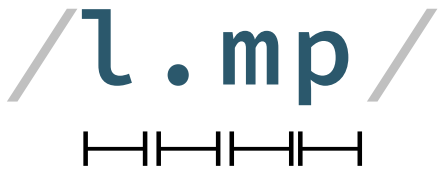
The **{n,m}** quantifier states “this piece is matched if the wrapped atom appears in this position at least *n* and at most *m* times”.
The **{n}** quantifier is a shortcut for **{n,n}**.

Quantification (summary)

- `a` matches “a”
- `a?` matches zero or one “a” (“” or “a”)
- `a+` matches one or more “a” (“a”, “aa”, “aaa”, …)
- `a*` matches zero or more “a” (“”, “a”, “aa”, …)
- `a{n}` matches exactly n “a”
- `a{n,m}` matches between n and m “a”

Wildcard, sets and ranges

Wildcard (. = any character)



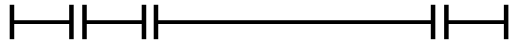

```
lmp     ✗
lamp    ✓
lump    ✓
liimp   ✗
```

The `.` metacharacter is an atom that matches any character*.

** with a few exceptions that we can ignore for the moment*

Set ([abc] = any of a, b or c)

`/gr[ae]y/`



`gry` ✗

`gray` ✓

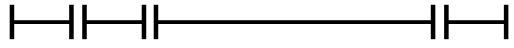
`grey` ✓

`groy` ✗

The bracket expression **`[abc]`** is an atom that matches any single character from the list of enclosed characters.

Negated set
([^abc] = anything except a, b or c)

/gr[^ae]y/



gry ✔

gray ✗

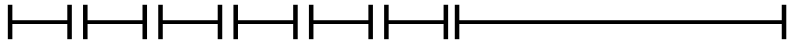
grey ✗

groy ✔

The bracket expression **[^abc]** is an atom that matches any single character from the list of enclosed characters.

Ranges ([a-f] = any between a and f)

`/codex [G-Z]/`



`codex A` ✗

`codex G` ✓

`codex K` ✓

`codex Å` ✗

The bracket expression **`[a-f]`** is an atom that matches any single character in the range of characters from **`a`** to **`f`**.

Let’s combine stuff together

/gray/

gry ✗

gray ✓

grey ✗

Gray ✗

Fray ✗

Greys ✗

/gr[ae]y/

gry ✗

gray ✓

grey ✓

Gray ✗

Fray ✗

Greys ✗

/[Gg]r[ae]y/

gry ✗

gray ✓

grey ✓

Gray ✓

Fray ✗

Greys ✗

/[FfGg]r[ae]y/

gry ✗

gray ✓

grey ✓

Gray ✓

Fray ✓

Greys ✗

/[FfGg]r[ae]ys?/

gry ✗

gray ✓

grey ✓

Gray ✓

Fray ✓

Greys ✓

Group and branches

/[Gg]ray’s/

gray ✔

Gray ✔

Gray’s ✔

Gray’ ✗

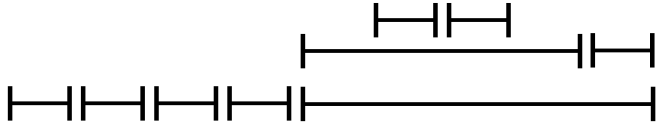
Grays ✗

/[Gg]ray’?s?/

gray ✔
Gray ✔
Gray’s ✔
Gray’ ✔ (✘)
Grays ✔ (✘)

Groups ((abc) = abc as a single atom)

`/Gray('s)?/`



`Gray` ✔

`Gray's` ✔

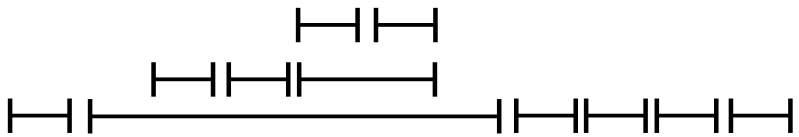
`Gray'` ✗

`Grays` ✗

The group **(abc)** is treated as a single atom and matches the enclosed subsequence of pieces **abc**.

Branches ((aa|bb|cc) = aa or bb or cc)

/M(ü|ue)ller/



Muller ✗

Müller ✓

Mueller ✓

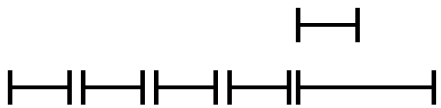
Mller ✗

The branch **(aa|bb|cc)** is treated as a single atom (it is a group) and matches if any of the enclosed subsequences of pieces **aa**, **bb** or **cc** matches.

Escaping

Escaping (\X = interpret X as normal char)

/ready?/

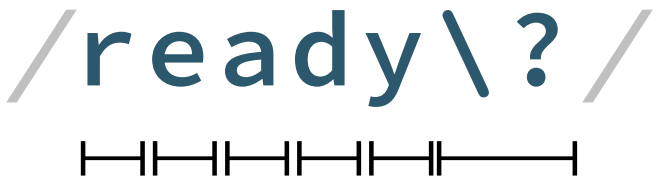


ready? ✗

read ✔

ready ✔

Escaping (\X = interpret X as normal char)



`ready?` ✔

`read` ✘

`ready` ✘

The metacharacter **** forces the following metacharacter to be interpreted as a normal character.*

** or the other way around, in certain cases. It's complicated...*

1. Uses and formalism
2. Syntax and concepts
3. **Hands-on session**
4. Beyond searching
5. Advanced topics

Summary

`a` = the character “a”

`a?` = “a”, 0 or 1 times,

`a+` = “a”, 1 or more times

`a*` = “a”, 0 or more times

`a{n,m}` = “a”, n to m times

`.` = any character

`[abc]` = either “a”, “b” or “c”

`[^abc]` = any character except “a”, “b” or “c”

`[a-g]` = any character between “a” and “g”

`(aa)` = treat the sequences “aa” as a single atom

`(aa|bb|cc)` = any sequence that matches “aa”, “bb” or “cc”

`\X` = treat X as a normal character

Exercises

1. Match all variants of Gödel

   https://regex101.com/r/QeUZXG/1

Exercises

1. Match all variants of Gödel

   https://regex101.com/r/QeUZXG/1

   ```
   /G(ö|oe?)del/
   ```

Exercises

1. Match all variants of Gödel

   https://regex101.com/r/QeUZXG/1

   ```
   /G(ö|oe?)del/
   ```

2. Match all dates

   https://regex101.com/r/CoXUkV/1

Exercises

1. Match all variants of Gödel

   https://regex101.com/r/QeUZXG/1

   ```
   /G(ö|oe?)del/
   ```

2. Match all dates

   https://regex101.com/r/CoXUkV/1

   ```
   /[0-9]{1,2}\. (Mai|[0-9]{1,2}\.)( [0-9]{4})?/
   ```

Exercises

1. Match all variants of Gödel

    https://regex101.com/r/QeUZXG/1

    ```
    /G(ö|oe?)del/
    ```

2. Match all dates

    https://regex101.com/r/CoXUkV/1

    ```
    /[0-9]{1,2}\. (Mai|[0-9]{1,2}\.)( [0-9]{4})?/
    ```

3. Match all the abbreviations
    1. https://regex101.com/r/bYSwCo/1

Exercises

1. Match all variants of Gödel

   https://regex101.com/r/QeUZXG/1

   ```
   /G(ö|oe?)del/
   ```

2. Match all dates

   https://regex101.com/r/CoXUkV/1

   ```
   /[0-9]{1,2}\. (Mai|[0-9]{1,2}\.)( [0-9]{4})?/
   ```

3. Match all the abbreviations

   1. https://regex101.com/r/bYSwCo/1

      ```
      /[A-Za-zäö]+\.(?!$)/
      ```

1. Uses and formalism
2. Syntax and concepts
3. Hands-on session
4. **Beyond searching**
5. Advanced topics

Beyond searching

- Replacement
  - `rename “s/IMG_([0-9_]+).JPG/Photo \1.jpeg/”`
  - `([A-Z][a-z]+)` ⇒ `<name>\1</name>`
- Binary matching
  - `/[^0-9][0-9]{2,4}[^0-9]/ =~` ⇒ true or false
- If-line-matches-then
  - `cat records.txt | awk “/[0-9]+ [A-Z]+/ { print $2}”`

1. Uses and formalism
2. Syntax and concepts
3. Hands-on session
4. Beyond searching
5. **Advanced topics**

Not so advanced

- `^` = beginning of the string (or of the line)
- `$` = end of the string (or of the line)
- `\A` = beginning of the string
- `\Z` = end of the string
- Unicode-aware character classes
  - `[:digit:]`, `[:alpha:]`, `[:blank:]`, …
  - `\d`, `\w`, `\s`, …

More advanced

- `(?<name>abc)` = named capturing groups
- `a(?=x)` = lookaround (matches `a` if followed by `x`, but does not consider `x` matched)
- `(abc)\s\1` = backreference (matches the previously matched group)
- Greediness and non-greedy operators (`*?`)

Dialects

- POSIX ERE

  `(ab+[:blank:]c*)`
- Vim

  `\(ab\+\sc*\)`
- Java

  `(ab\+\p{Blank}c*)`
- Perl/PCRE, POSIX BRE, SRE, XML Schema, ...

Encodings

- At which level does the regex engine work?

  `/o/ =~ ö` *WHAT?*

- In Unicode, “ö” can be expressed in 4 different but equivalent normalization form, each form can be serialized in many different encodings (UTF-8, UTF-16, UCS-16, UCS-32, etc) → *CMV+P model*

Introduction to regular expressions

Thank you!

Gioele Barabucci

NTNU | Norwegian University of Science and Technology

References and useful tools

- Mastering Regular Expressions (3rd ed.), Jeffrey E.F. Friedl, O'Reilly Media, 2006
- `man 7 regex`
- https://pcre.org Perl-compatible regular expressions
- https://regex101.com/ Regular expression testing tool
- https://www.debuggex.com/ Visual regular expression tool