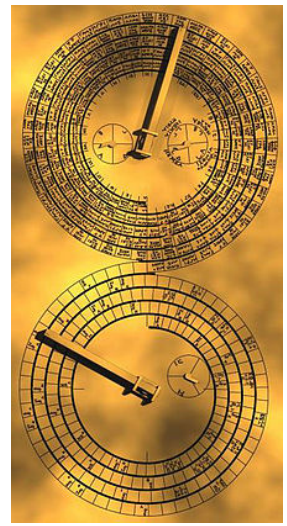
Automata and Formal Languages (1)

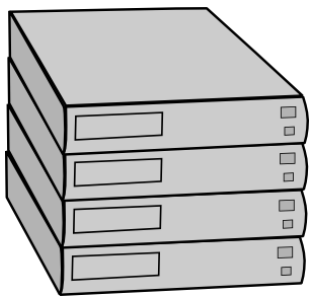


Antikythera automaton, 100 BC (Archimedes?)

Email: christian.urban at kcl.ac.uk

Office: S1.27 (1st floor Strand Building)

Slides: KEATS



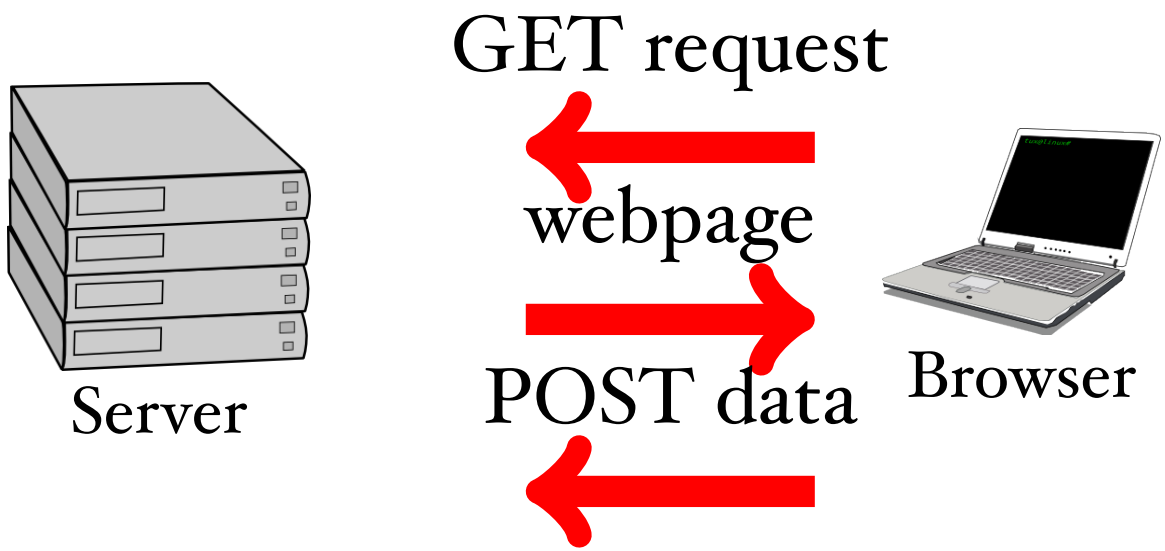
Server

GET request

webpage

POST data

Browser



- programming languages, compilers

transforming strings into structured data

Lexing

(recognising “words”)

Parsing

(recognising “sentences”)

The subject is quite old:

- Turing Machines, 1936
- first compiler for COBOL, 1957 (Grace Hopper)
- but surprisingly research papers are still published now

Grace Hopper

(she made it to David Letterman's Tonight Show,

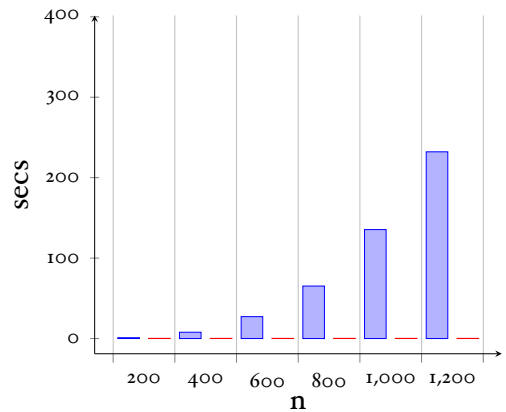
`http://www.youtube.com/watch?v=aZOxtURhfEU`)

This Course

- the ultimate goal is to implement a small compiler (a really small one for the JVM)

Let's start with:

- a web-crawler
- an email harvester
- a web-scraper



A Web-Crawler

1. given an URL, read the corresponding webpage
2. extract all links from it
3. call the web-crawler again for all these links

A Web-Crawler

1. given an URL, read the corresponding webpage
2. if not possible print, out a problem
3. if possible, extract all links from it
4. call the web-crawler again for all these links

A Web-Crawler

1. given an URL, read the corresponding webpage
2. if not possible print, out a problem
3. if possible, extract all links from it
4. call the web-crawler again for all these links

(we need a bound for the number of recursive calls)
(the purpose is to check all links on my own webpage)

Scala

a simple Scala function for reading webpages

```
1  import io.Source
2
3  def get_page(url: String) : String = {
4    Source.fromURL(url).take(10000).mkString
```

Scala

a simple Scala function for reading webpages

```
1  import io.Source
2
3  def get_page(url: String) : String = {
4    Source.fromURL(url).take(10000).mkString

get_page("""http://www.inf.kcl.ac.uk/staff/urbanc/""")
```

Scala

a simple Scala function for reading webpages

```
1  import io.Source
2
3  def get_page(url: String) : String = {
4    Source.fromURL(url).take(10000).mkString
```

```
get_page("""http://www.inf.kcl.ac.uk/staff/urbanc/""")
```

slightly more complicated for handling errors properly:

```
1  def get_page(url: String) : String =
2    Try(Source.fromURL(url).take(10000).mkString) getOrEl
3      { println(s"  Problem with: $url"); ""}
```

Why Scala?


twitter


LinkedIn


theguardian


Morgan Stanley


CREDIT SUISSE


EDF
ENERGY


Novell


foursquare


HSBC

...

Why Scala?

twitter

LinkedIn

theguardian

Morgan Stanley

CREDIT SUISSE

EDF ENERGY

Novell

foursquare

HSBC

...

Why Scala?

twitter

LinkedIn

theguardian

Morgan Stanley

CREDIT SUISSE

EDF ENERGY

Novell

foursquare

HSBC

...

5 yrs {
- 2013: 1%
- 2014: 3%
- 2015: 9%
- 2016: 27%
- 2017: 81%
- 2018: 243% 😁

Why Scala?

twitter

LinkedIn

theguardian

Morgan Stanley

CREDIT SUISSE

EDF ENERGY

Novell

foursquare

HSBC

...

5 yrs {
- 2013: 1%
- 2014: 3%
- 2015: 9%
- 2016: 27%
- 2017: 81%
- 2018: 243% 😁

in London today: 1 Scala job for every 30 Java jobs;
Scala programmers seem to get up to 20% better salary

Why Scala?

twitter

LinkedIn

theguardian

Morgan

CREDIT S

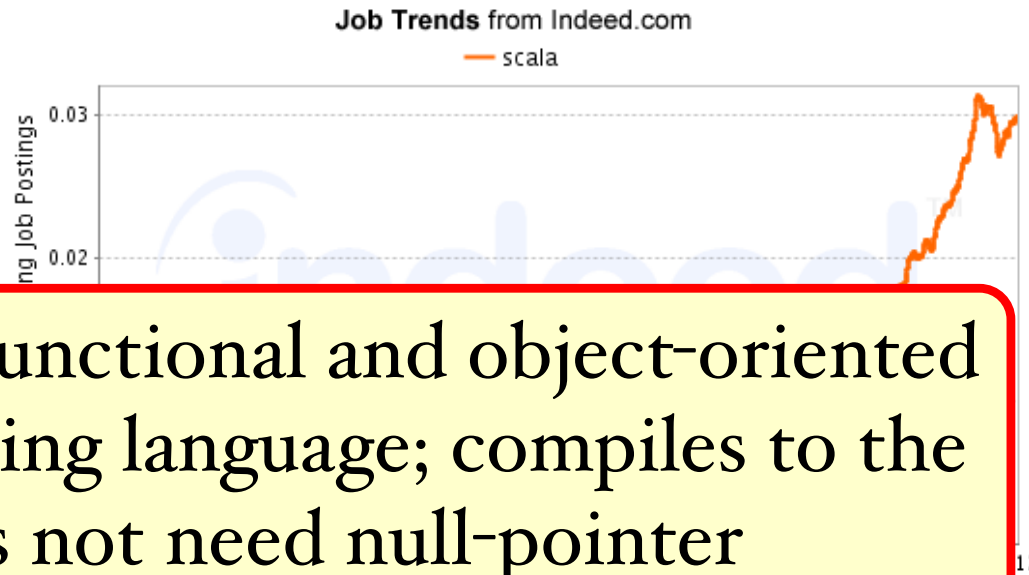
EDF ENERGY

Nove

foursquare

HSBC

...



Scala is a functional and object-oriented programming language; compiles to the JVM; does not need null-pointer exceptions; a course on Coursera

http://www.scala-lang.org

2016: 27%
2017: 81%
2018: 243% 😁

in London today: 1 Scala job for every 30 Java jobs; Scala programmers seem to get up to 20% better salary

A Regular Expression

- ... is a pattern or template for specifying strings

```
"https?://[^"]*"
```

matches for example

```
"http://www.foobar.com"
"https://www.tls.org"
```

A Regular Expression

- ... is a pattern or template for specifying strings

```
"""""https?://[^"]*""""".r
```

matches for example

```
"http://www.foobar.com"
"https://www.tls.org"
```

`rexp.findAllIn(string)`

returns a list of all (sub)strings that match the regular expression

`rexp.findFirstIn(string)`

returns either `None` if no (sub)string matches or `Some(s)` with the first (sub)string

```
1  val http_pattern = """\"https?://[^\"]*\"""".r
2
3  def unquote(s: String) = s.drop(1).dropRight(1)
4
5  def get_all_URLs(page: String) : Set[String] = {
6    http_pattern.findAllIn(page).map(unquote).toSet
7  }
8
9  def crawl(url: String, n: Int) : Unit = {
10   if (n == 0) ()
11   else {
12     println(s"Visiting: $n $url")
13     for (u <- get_all_URLs(get_page(url)))
14       crawl(u, n - 1)
15   }
16 }

crawl(some_start_URL, 2)
```

a version that only "crawls" links in my domain:

```
1  val my_urls = """urbanc""".r
2
3  def crawl(url: String, n: Int) : Unit = {
4    if (n == 0) ()
5    else if (my_urls.findFirstIn(url) == None) ()
6    else {
7      println(s"Visiting: $n $url")
8      for (u <- get_all_URLs(get_page(url)))
9        crawl(u, n - 1)
10   }
11 }
```

a little email "harvester":

```
1  val http_pattern = """\"https?://[^\"]*\"""".r
2  val my_urls = """urbanc""".r
3  val email_pattern =
4    """([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})""".r
5
6  def crawl(url: String, n: Int) : Unit = {
7    if (n == 0) ()
8    else {
9      println(s"Visiting: $n $url")
10     val page = get_page(url)
11     println(email_pattern.findAllIn(page).mkString("\n"
12     for (u <- get_all_URLs(page))
13       crawl(u, n - 1)
14   }
15 }
```

http://net.tutsplus.com/tutorials/other/8-regular-expressions-you-should-know/

Regular Expressions

Their inductive definition:

r	$::=$	$\varnothing$	null
	$\mid$	ϵ	empty string / "" / []
	$\mid$	c	character
	$\mid$	$r_1 \cdot r_2$	sequence
	$\mid$	$r_1 + r_2$	alternative / choice
	$\mid$	r^*	star (zero or more)

Regular Expressions

In Scala:

```
1  abstract class Rexp
2
3  case object NULL extends Rexp
4  case object EMPTY extends Rexp
5  case class CHAR(c: Char) extends Rexp
6  case class ALT(r1: Rexp, r2: Rexp) extends Rexp
7  case class SEQ(r1: Rexp, r2: Rexp) extends Rexp
8  case class STAR(r: Rexp) extends Rexp
```

The Meaning of a Regular Expression

$$
\begin{array}{rcl}
L(\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
L(\epsilon) & \stackrel{\text{def}}{=} & \{""\} \\
L(c) & \stackrel{\text{def}}{=} & \{"c"\} \\
L(r_1 + r_2) & \stackrel{\text{def}}{=} & L(r_1) \cup L(r_2) \\
L(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \{\, s_1 @ s_2 \mid s_1 \in L(r_1) \wedge s_2 \in L(r_2) \,\} \\
L(r^*) & \stackrel{\text{def}}{=} &
\end{array}
$$

The Meaning of a Regular Expression

$$
\begin{array}{rcl}
L(\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
L(\epsilon) & \stackrel{\text{def}}{=} & \{""\} \\
L(c) & \stackrel{\text{def}}{=} & \{"c"\} \\
L(r_1 + r_2) & \stackrel{\text{def}}{=} & L(r_1) \cup L(r_2) \\
L(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \{\, s_1 @ s_2 \mid s_1 \in L(r_1) \wedge s_2 \in L(r_2) \,\} \\
L(r^*) & \stackrel{\text{def}}{=} & \\
\end{array}
$$

$$
\begin{array}{rcl}
L(r)^0 & \stackrel{\text{def}}{=} & \{""\} \\
L(r)^{n+1} & \stackrel{\text{def}}{=} & L(r) @ L(r)^n
\end{array}
$$

The Meaning of a Regular Expression

$$
\begin{array}{rcl}
L(\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
L(\epsilon) & \stackrel{\text{def}}{=} & \{""\} \\
L(c) & \stackrel{\text{def}}{=} & \{"c"\} \\
L(r_1 + r_2) & \stackrel{\text{def}}{=} & L(r_1) \cup L(r_2) \\
L(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \{\, s_1 @ s_2 \mid s_1 \in L(r_1) \wedge s_2 \in L(r_2) \,\} \\
L(r^*) & \stackrel{\text{def}}{=} &
\end{array}
$$

$$
\begin{array}{rcl}
L(r)^\circ & \stackrel{\text{def}}{=} & \{""\} \\
L(r)^{n+1} & \stackrel{\text{def}}{=} & L(r) @ L(r)^n
\end{array}
$$

(append on sets)

$$\{\, s_1 @ s_2 \mid s_1 \in L(r) \wedge s_2 \in L(r)^n \,\}$$

The Meaning of a Regular Expression

$$
\begin{array}{rcl}
L(\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
L(\epsilon) & \stackrel{\text{def}}{=} & \{""\} \\
L(c) & \stackrel{\text{def}}{=} & \{"c"\} \\
L(r_1 + r_2) & \stackrel{\text{def}}{=} & L(r_1) \cup L(r_2) \\
L(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \{\, s_1 @ s_2 \mid s_1 \in L(r_1) \wedge s_2 \in L(r_2) \,\} \\
L(r^*) & \stackrel{\text{def}}{=} & \bigcup_{n \geq 0} L(r)^n
\end{array}
$$

$$
\begin{array}{rcl}
L(r)^0 & \stackrel{\text{def}}{=} & \{""\} \\
L(r)^{n+1} & \stackrel{\text{def}}{=} & L(r) @ L(r)^n
\end{array}
$$

(append on sets)

$$\{\, s_1 @ s_2 \mid s_1 \in L(r) \wedge s_2 \in L(r)^n \,\}$$

The Meaning of Matching

a regular expression r matches a string s is defined as

$$s \in L(r)$$

This Course

We will have a look at:

- regular expressions / regular expression matching
- derivatives
- automata
- parsing
- grammars
- a small interpreter / compiler

Written Exam

- Accounts for 75%.
- You will understand the question "Is this relevant for the exam?" is very demotivating for the lecturer!
- Deal: Whatever is in the homework (and is not marked "optional") is relevant for the exam.

Coursework

- Accounts for 25%. Two strands. Choose **one**!

Strand 1

- four programming subtasks:
  - matcher (5%, 13.10.)
  - lexer (5%, 03.11.)
  - parser (5%, 27.11.)
  - compiler (10%, 12.12.)

Strand 2

- one task: prove the correctness of a regular expression matcher in the Isabelle theorem prover
- 25%, submission 12.12.

- Solving more than one strand will **not** give you more marks.
- The exam will contain in much, much smaller form elements from both (but will also be in lectures and HW).