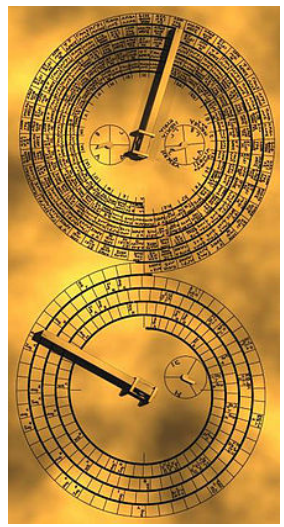
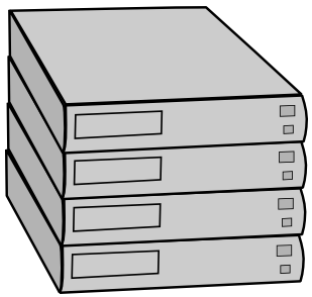
Automata and Formal Languages (1)



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 web-browser (really small one)

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 {
3      Source.fromURL(url).take(10000).mkString
4    }
5    catch {
6      case e => {
7        println("  Problem with: " + url)
8        ""
9      }
10   }
11 }
```

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("Visiting: " + n + " " + url)
13     for (u <- get_all_URLs(get_page(url))) crawl(u, n - 1)
14   }
15 }
```

```
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("Visiting: " + n + " " + url)
8      for (u <- get_all_URLs(get_page(url))) crawl(u, n - 1)
9    }
10 }
```

a little email "harvester":

```
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
```

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

Regular Expressions

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

```
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

$$
\begin{array}{rcl}
L(\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
L(\epsilon) & \stackrel{\text{def}}{=} & \{""\} \\
L(c) & \stackrel{\text{def}}{=} & \{"c"\} \\
L(r_1 + r_2) & \stackrel{\text{def}}{=} & L(r_1) \cup L(r_2) \\
L(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \{\, s_1 @ s_2 \mid s_1 \in L(r_1) \wedge s_2 \in L(r_2) \,\} \\
L(r^*) & \stackrel{\text{def}}{=} & \\
 & & \\
L(r)^0 & \stackrel{\text{def}}{=} & \{""\} \\
L(r)^{n+1} & \stackrel{\text{def}}{=} & L(r) @ L(r)^n
\end{array}
$$

The Meaning of a Regular Expression

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

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

(append on sets)

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

The Meaning of 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 a Matching

a regular expression r matches a string s is defined as

$$s \in L(r)$$

Nullability

whether a regular expression matches the empty string:

```
1  def nullable (r: Rexp) : Boolean = r match {
2    case NULL => false
3    case EMPTY => true
4    case CHAR(_) => false
5    case ALT(r1, r2) => nullable(r1) || nullable(r2)
6    case SEQ(r1, r2) => nullable(r1) && nullable(r2)
7    case STAR(_) => true
8  }
```

Derivative of a Rexp

If r matches the string $c::s$, what is a regular expression that matches s?

der c r gives the answer

The Derivative

$$
\begin{array}{lcl}
\text{der c } (\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
\text{der c } (\epsilon) & \stackrel{\text{def}}{=} & \varnothing \\
\text{der c (d)} & \stackrel{\text{def}}{=} & \text{if c} = \text{d then [] else } \varnothing \\
\text{der c } (r_1 + r_2) & \stackrel{\text{def}}{=} & (\text{der c } r_1) + (\text{der c } r_2) \\
\text{der c } (r_1 \cdot r_2) & \stackrel{\text{def}}{=} & ((\text{der c } r_1) \cdot r_2) + \\
 & & \quad (\text{if nullable } r_1 \text{ then der c } r_2 \text{ else } \varnothing) \\
\text{der c } (r^*) & \stackrel{\text{def}}{=} & (\text{der c } r) \cdot (r^*)
\end{array}
$$

The Derivative

$$
\begin{array}{lcl}
\text{der c } (\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
\text{der c } (\epsilon) & \stackrel{\text{def}}{=} & \varnothing \\
\text{der c (d)} & \stackrel{\text{def}}{=} & \text{if c} = \text{d then [] else } \varnothing \\
\text{der c } (r_1 + r_2) & \stackrel{\text{def}}{=} & (\text{der c } r_1) + (\text{der c } r_2) \\
\text{der c } (r_1 \cdot r_2) & \stackrel{\text{def}}{=} & ((\text{der c } r_1) \cdot r_2) + \\
 & & \quad (\text{if nullable } r_1 \text{ then der c } r_2 \text{ else } \varnothing) \\
\text{der c } (r^*) & \stackrel{\text{def}}{=} & (\text{der c } r) \cdot (r^*) \\
\text{ders [] r} & \stackrel{\text{def}}{=} & r \\
\text{ders (c::s) r} & \stackrel{\text{def}}{=} & \text{ders s (der c r)}
\end{array}
$$

The Derivative

```
1  def deriv (r: Rexp, c: Char) : Rexp = r match {
2    case NULL => NULL
3    case EMPTY => NULL
4    case CHAR(d) => if (c == d) EMPTY else NULL
5    case ALT(r1, r2) => ALT(deriv(r1, c), deriv(r2, c))
6    case SEQ(r1, r2) =>
7      if (nullable(r1)) ALT(SEQ(deriv(r1, c), r2), deriv(r2, c))
8      else SEQ(deriv(r1, c), r2)
9    case STAR(r) => SEQ(deriv(r, c), STAR(r))
10 }
```

The Matcher

```
1  def matches(r: Rexp, s: String) : Boolean =
2    nullable(derivs(r, s.toList))
3
4
5  /* Examples */
6
7  println(matches(SEQ(SEQ(CHAR('c'), CHAR('a')), CHAR('b')),"cab"))
8  println(matches(STAR(CHAR('a')),"aaa"))
9
10 /* Convenience using implicits */
11 implicit def string2rexp(s : String) : Rexp = {
12   s.foldRight (EMPTY: Rexp) ( (c, r) => SEQ(CHAR(c), r) )
13 }
14
15 println(matches("cab" ,"cab"))
16 println(matches(STAR("a"),"aaa"))
17 println(matches(STAR("a"),"aaab"))
```

This Course

We will have a look at

- regular expression / regular expression matching
- a bit of sets (of strings)
- automata
- the Myhill-Nerode theorem
- parsing
- grammars
- a small interpreter / webbrowser

Exam

- The question "Is this relevant for the exams?" is not appreciated!

  Whatever is in the homework sheets (and is not marked optional) is relevant for the exam.
  No code needs to be written.

Maps in Scala

- `map` takes a function, say f, and applies it to every element of the list:

```
List(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
List(1, 4, 9, 16, 25, 36, 49, 64, 81)
```