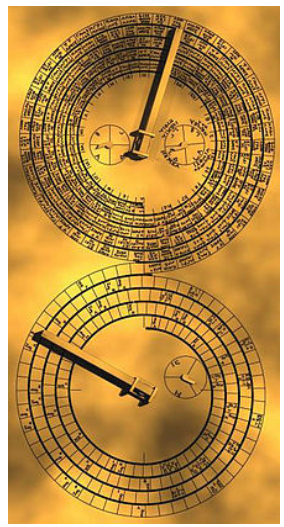
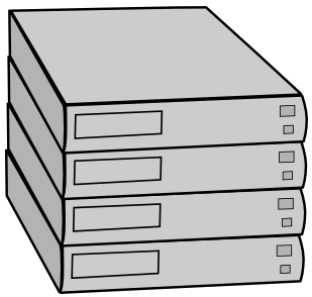
Automata and Formal Languages (1)



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

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

This Course

- the ultimate goal is to implement a small web-browser (really small)

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 to bound 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
5
6  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
5
6  get_page("""http://www.inf.kcl.ac.uk/staff/urbanc/""")
```

slightly more complicated for handling errors:

```
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 }
```

Cookies

Servers from
Dot.com Inc.

GET request

Client

Cookies

Servers from
Dot.com Inc.

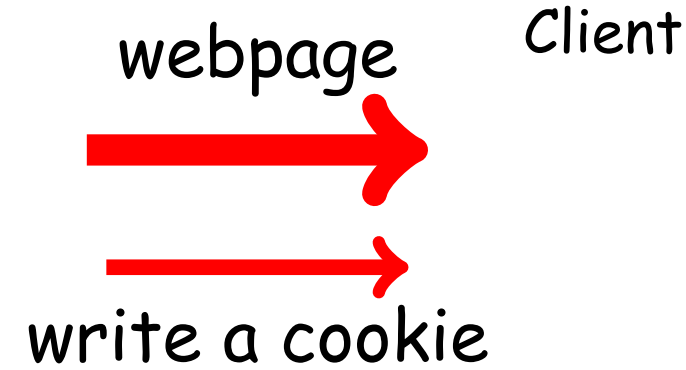
GET request

Client

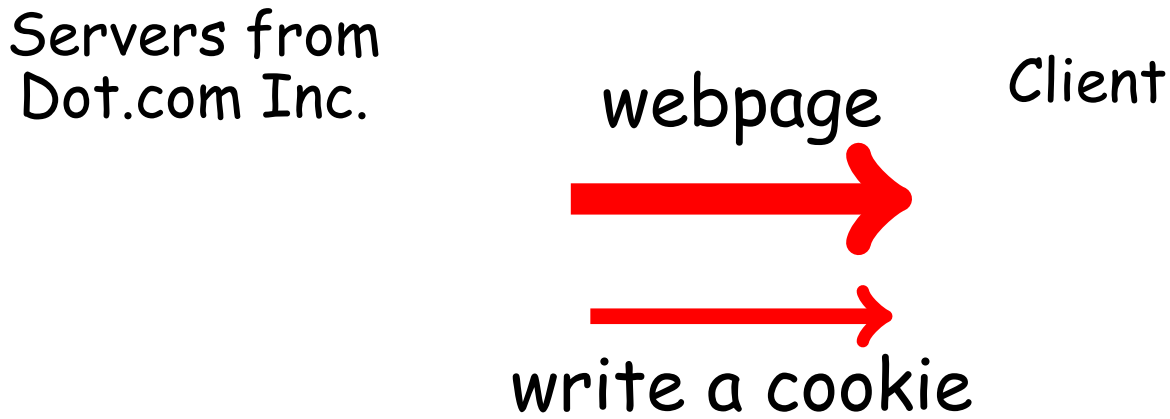
read a cookie

Cookies

Servers from
Dot.com Inc.



Cookies



- cookies: max 4KB data
- cookie theft, cross-site scripting attacks
- session cookies, persistent cookies, HttpOnly cookies, third-party cookies, zombie cookies

Cookies

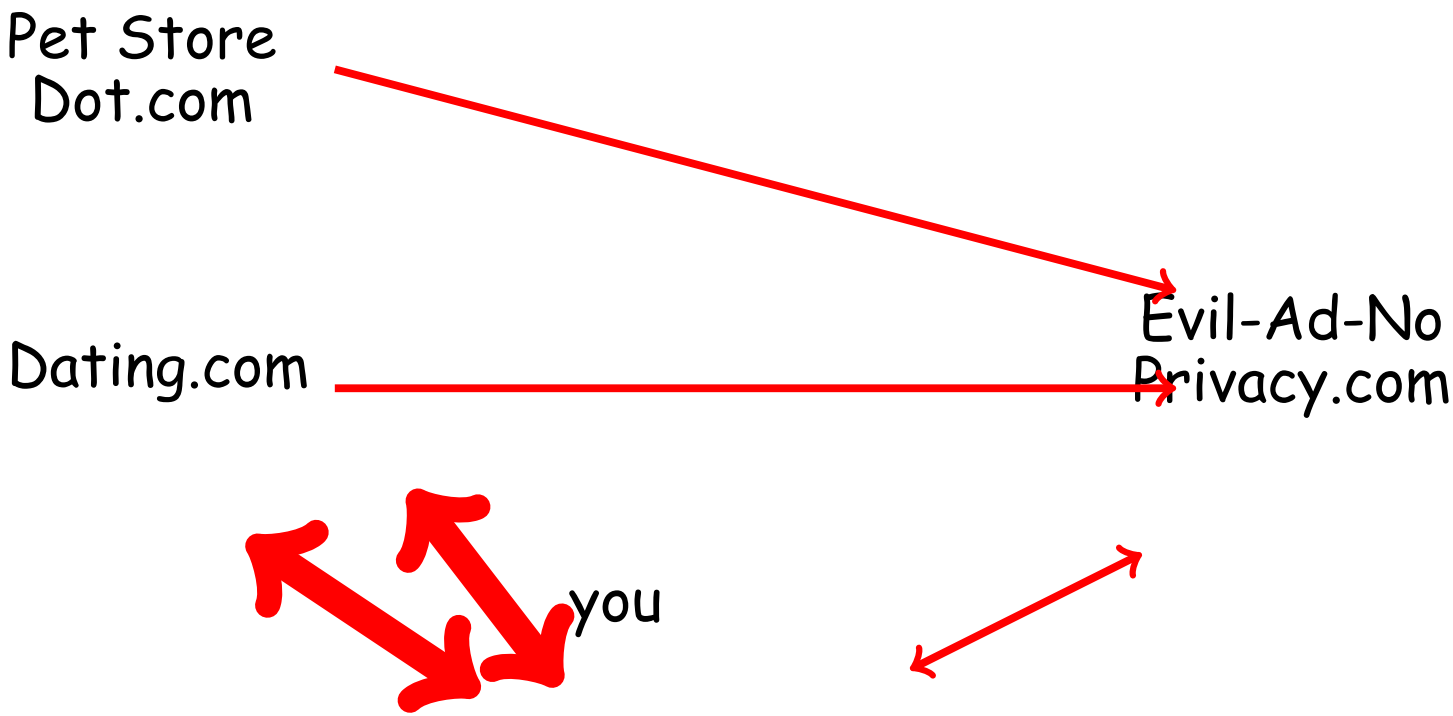
EU Privacy Directive about Cookies:

"In May 2011, a European Union law was passed stating that websites that leave non-essential cookies on visitors' devices have to alert the visitor and get acceptance from them. This law applies to both individuals and businesses based in the EU regardless of the nationality of their website's visitors or the location of their web host. It is not enough to simply update a website's terms and conditions or privacy policy. The deadline to comply with the new EU cookie law was 26th May 2012 and failure to do so could mean a fine of up to £500,000."

→BBC News

- session cookies, persistent cookies, HttpOnly cookies, third-party cookies, zombie cookies

- While cookies are per web-page, this can be easily circumvented.



My First Webapp

GET request:

1. read the cookie from client
2. if none is present, set `visits` to **0**
3. if cookie is present, extract `visits` counter
4. if `visits` is greater or equal **10**,
   print a valued customer message
   otherwise just a normal message
5. increase `visits` by **1** and store new cookie with client

- cookie value encoded as hash

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)
```