Introduction to Regular Expressions

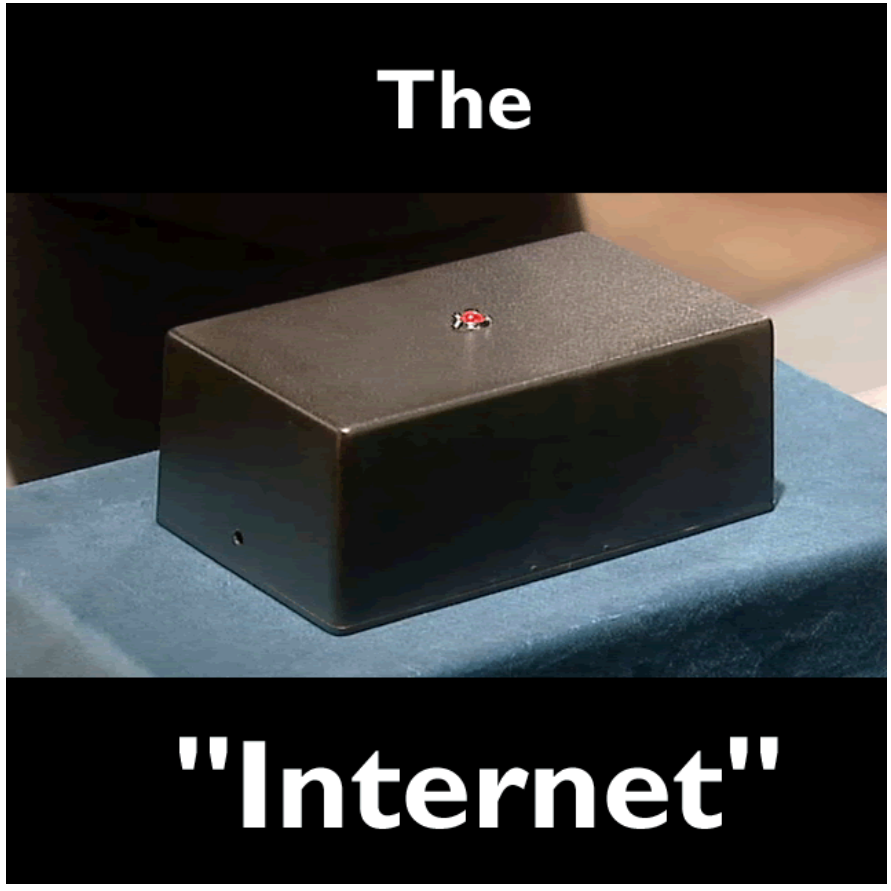
Basil L. Contovounesios
blc@netsoc.tcd.ie

For the
Dublin University Internet Society [NETSOC]

Tuesday 3rd November, 2015

Finite Automata

- Imagine a mysterious black box



Finite Automata

It comprises:

- some form of input
  - seemingly wireless (it is worth watching *The IT Crowd* ☺)
- a reset button; and
- a light labelled 'accept'

Finite Automata

How does it work?

Finite Automata

- Let us imagine the box has two buttons `a` and `b`

Finite Automata

- Let us imagine the box has two buttons `a` and `b`
- We carry out some experiments
  - cue mad pair button smashing

Finite Automata

- Let us imagine the box has two buttons `a` and `b`
- We carry out some experiments
  - cue mad pair button smashing

- The light eventually turns on

Finite Automata

- Let us imagine the box has two buttons `a` and `b`
- We carry out some experiments
  - cue mad pair button smashing

- The light eventually turns on
- We have been accepted as one of its own!

Finite Automata

- Let us imagine the box has two buttons `a` and `b`
- We carry out some experiments
  - cue mad pair button smashing

- The light eventually turns on
- We have been accepted as one of its own!
- More importantly, we have been diligent and recorded the accepted sequence

Finite Automata

- Our luck seems to end there - repeating the same sequence doesn't do the trick

Finite Automata

- Our luck seems to end there - repeating the same sequence doesn't do the trick
- Have you tried resetting it? The sequence works again!

Finite Automata

- Our luck seems to end there - repeating the same sequence doesn't do the trick
- Have you tried resetting it? The sequence works again!
- Continuing these experiments, we discover several inputs which are accepted following a reset:
  - `aa`
  - `aba`
  - `abba`
  - `abbba`

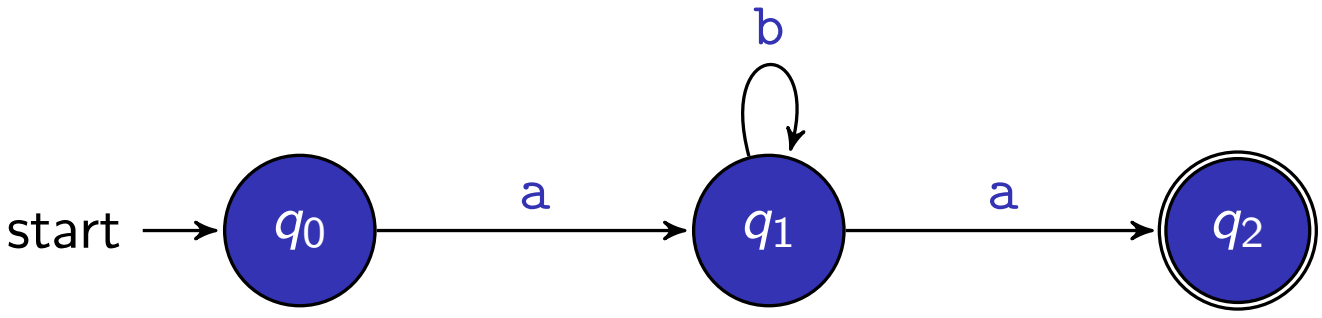
Finite Automata

- Our luck seems to end there - repeating the same sequence doesn't do the trick
- Have you tried resetting it? The sequence works again!
- Continuing these experiments, we discover several inputs which are accepted following a reset:
  - `aa`
  - `aba`
  - `abba`
  - `abbba`
- A pattern is clearly discernible

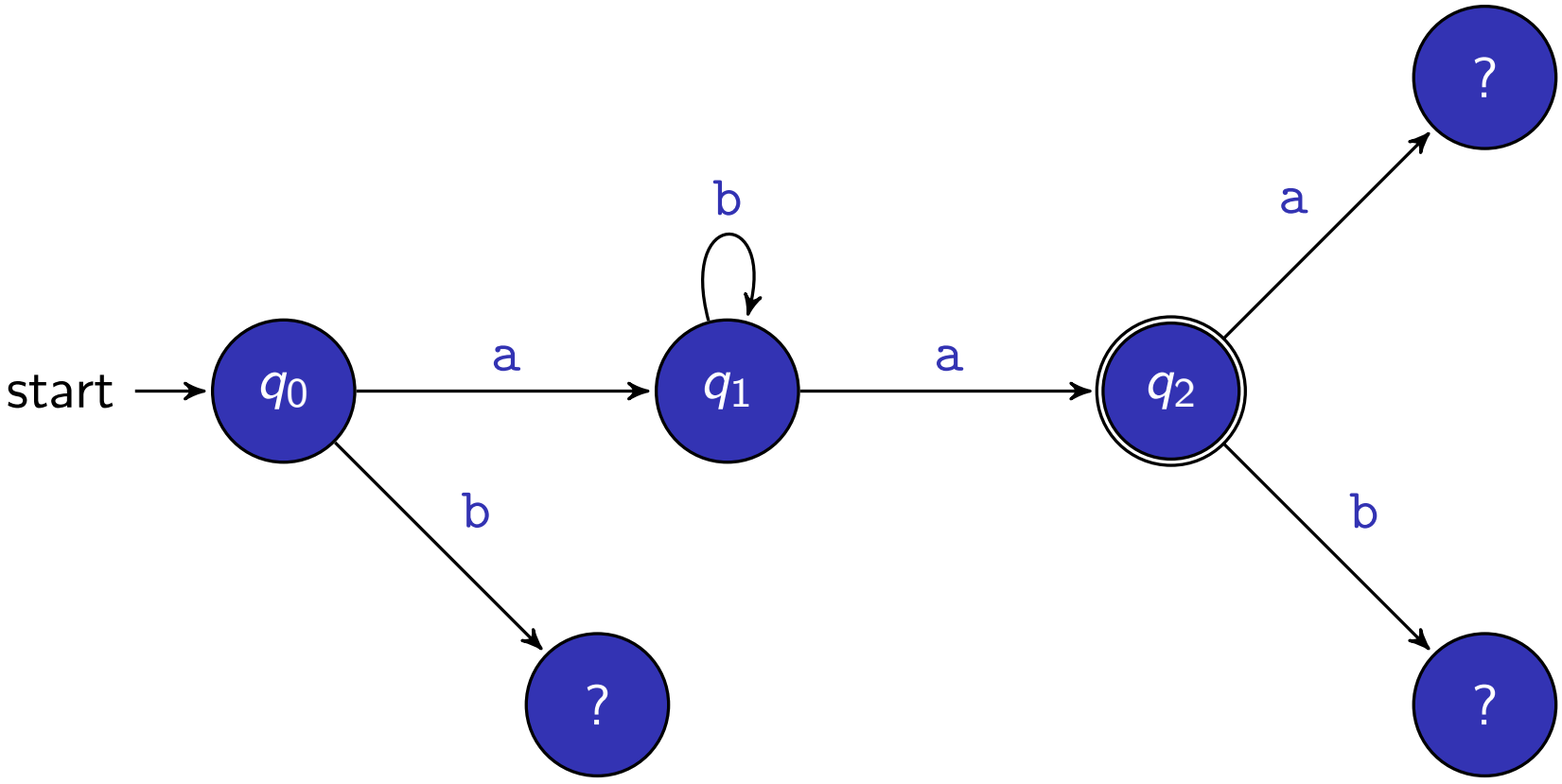
Finite Automata

- Our luck seems to end there - repeating the same sequence doesn't do the trick
- Have you tried resetting it? The sequence works again!
- Continuing these experiments, we discover several inputs which are accepted following a reset:
  - `aa`
  - `aba`
  - `abba`
  - `abbba`
- A pattern is clearly discernible
- How far can we go?

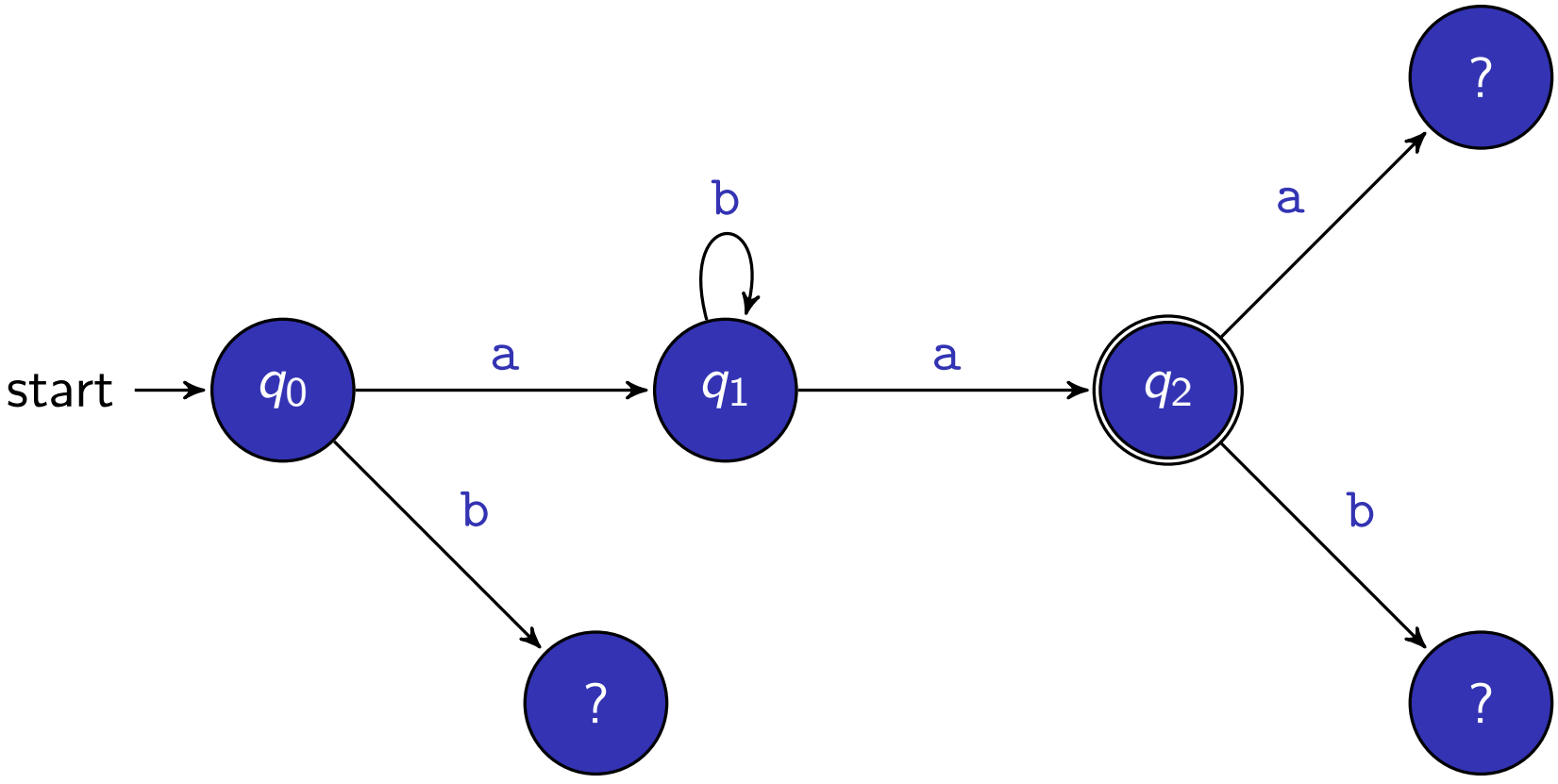
Finite Automata



Finite Automata



Finite Automata



This set of states is not even uniquely correct

Finite Automata

- The title gives it away - this is a finite automaton (FA), or finite-state machine
- Model of computation, like Turing machines and λ-calculus, but less powerful
- State machines are everywhere, e.g. vending machines, combination locks and, of course, regular expressions!

Enter Chomsky

- The set of all symbol combinations, or strings/words, accepted by a FA (e.g. `abba`) form a *regular* language
- The two are practically inseparable concepts, with the latter often being defined in terms of the former
- The title refers to a hierarchical classification of formal languages and their grammars in terms of their expressiveness and recognition

Enter Regular Expressions (finally)

- Succinct notation for expressing regular languages
- Regular expressions are readily understood by humans, e.g. when describing text patterns
- There are algorithms for transforming regular expressions to minimal (optimal) FAs which can be efficiently implemented on a computer

Applications

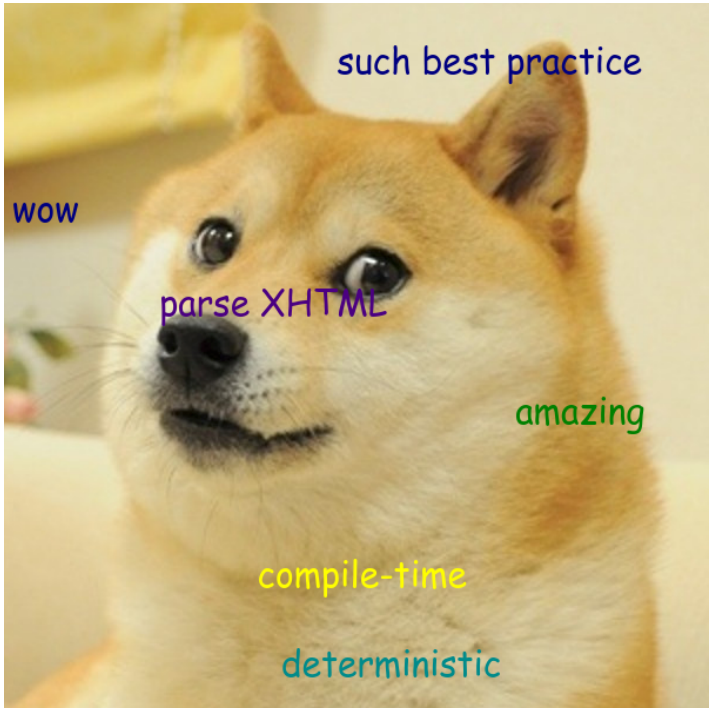
- Text processing

Applications

- Text processing
  - String matching, e.g. URL validation

Applications

- Text processing
  - String matching, e.g. URL validation



Applications

- Text processing
  - String matching, e.g. URL validation
  - Filtering, e.g. logs

Applications

- Text processing
  - String matching, e.g. URL validation
  - Filtering, e.g. logs

Applications

- Text processing
  - String matching, e.g. URL validation
  - Filtering, e.g. logs
  - Search and replace,
    e.g. batch file renaming

Applications

- Text processing
  - String matching, e.g. URL validation
  - Filtering, e.g. logs
  - Search and replace,
    e.g. batch file renaming
- Formal language theory,
  compiler design, etc.

Mechanics

- The following is a standard set of rules for constructing regexes:

Mechanics

- The following is a standard set of rules for constructing regexes:
  - The constants $\varnothing$, $\{\epsilon\}$ and each literal character in the given alphabet are regular expression

Mechanics

- The following is a standard set of rules for constructing regexes:
  - The constants $\varnothing$, $\{\epsilon\}$ and each literal character in the given alphabet are regular expression
  - If `x` and `y` are regular expressions, so are their *concatenation* `xy` and *alternation* `x|y`
    - ★ sometimes described as *summation*, `x+y`

Mechanics

- The following is a standard set of rules for constructing regexes:
  - The constants $\varnothing$, $\{\epsilon\}$ and each literal character in the given alphabet are regular expression
  - If `x` and `y` are regular expressions, so are their *concatenation* `xy` and *alternation* `x|y`
    - ★ sometimes described as *summation*, `x+y`
  - If `x` is a regular expression, so is its *iteration* or *Kleene star* $\texttt{x}^*$

Mechanics

- The following is a standard set of rules for constructing regexes:
  - The constants $\varnothing$, $\{\epsilon\}$ and each literal character in the given alphabet are regular expression
  - If `x` and `y` are regular expressions, so are their *concatenation* `xy` and *alternation* `x|y`
    - ★ sometimes described as *summation*, `x+y`
  - If `x` is a regular expression, so is its *iteration* or *Kleene star* $\texttt{x}^*$
- So regular expressions comprise a combination of *literal* (*a*) and *meta*-characters (|)

Mechanics

- The following is a standard set of rules for constructing regexes:
  - The constants $\varnothing$, $\{\epsilon\}$ and each literal character in the given alphabet are regular expression
  - If `x` and `y` are regular expressions, so are their *concatenation* `xy` and *alternation* `x|y`
    - ★ sometimes described as *summation*, `x+y`
  - If `x` is a regular expression, so is its *iteration* or *Kleene star* $\texttt{x}^*$
- So regular expressions comprise a combination of *literal* (*a*) and *meta*-characters (|)
- Most modern implementations vastly extend this standard definition—varied notation

Standard Examples

- We want to match (and catch)
  `Charmander`, `Charmeleon` and `Charizard`

Standard Examples

- We want to match (and catch) `Charmander`, `Charmeleon` and `Charizard`
- We could enumerate the options: `Charmander|Charmeleon|Charizard`

Standard Examples

- We want to match (and catch) `Charmander`, `Charmeleon` and `Charizard`
- We could enumerate the options: `Charmander|Charmeleon|Charizard`
- `Char(mander|meleon|izard)` Elegance++

Standard Examples

- We want to match (and catch) `Charmander`, `Charmeleon` and `Charizard`
- We could enumerate the options: `Charmander|Charmeleon|Charizard`
- `Char(mander|meleon|izard)` Elegance++
- `()` Grouping of tokens

Standard Examples

- We want to ~~match (and catch)~~ `(m|c)atch` ☺
  `Charmander`, `Charmeleon` and `Charizard`
- We could enumerate the options:
  `Charmander|Charmeleon|Charizard`
- `Char(mander|meleon|izard)` Elegance++
- `()` Grouping of tokens

Quantification

- We want to match `banana` and `bananananana`

Quantification

- We want to match `banana` and `bananananana`
- We have seen the *Kleene star*: `ba(na)*`

Quantification

- We want to match `banana` and `bananananana`
- We have seen the *Kleene star*: `ba(na)*`
- What about `ba`?

Quantification

- We want to match `banana` and `bananananana`
- We have seen the *Kleene star*: `ba(na)*`
- What about `ba`?
- *Kleene star* can be read as "zero or more occurences of the previous token"

Quantification

- We want to match `banana` and `bananananana`
- We have seen the *Kleene star*: `ba(na)*`
- What about `ba`?
- *Kleene star* can be read as "zero or more occurences of the previous token"
- There is also the `+` quantifier, meaning "one or more occurences"

Quantification

- We want to match `banana` and `bananananana`
- We have seen the *Kleene star*: `ba(na)*`
- What about `ba`?
- *Kleene star* can be read as "zero or more occurences of the previous token"
- There is also the `+` quantifier, meaning "one or more occurences"
- $\Rightarrow$ `ba(na)+`

Moar Quantification

- `?` - “zero or one occurences”
  - e.g. `win(ner)?`
- `{n}` - “n many occurences”
  - e.g. `ba(na){2}`
- `{n, }` - “at least n many occurences”
- `{n, m}` - “between n and m occurences”

Some Syntax

- Meta-characters (`|`, `(`, etc.) can be escaped with a backslash

Some Syntax

- Meta-characters (`|`, `(`, etc.) can be escaped with a backslash
- In fact some standards require their escaping (more later)

Some Syntax

- Meta-characters (`|`, `(`, etc.) can be escaped with a backslash
- In fact some standards require their escaping (more later)
- In `sed`, `Perl`, `ECMAScript` and others, regexes are contained or delimited by forward slashes
  - e.g. `s/speling/spelling/g`

Standards

- "The good thing about standards is there are so many to choose from."
- POSIX distinguishes between basic and extended regular expressions
- In practice it boils down to the engine you are working with
  - If `grep` does not work, try `egrep`

Moar Syntax

- `.` - (ostensibly) matches any character
- `[ ]` - matches any one contained atom
  - Complement with hat `[^ ]`
  - Define ranges with hyphen, e.g. `[a-z]` or `[^a-z]`
- `^` - matches start of string
- `$` - matches end of string

Character Classes

https://en.wikipedia.org/wiki/Regular_expression#Character_classes

References


Dewdney, A.K. “2. Finite Automata.”
The New Turing Omnibus: 66 Excursions in Computer Science.
New York: Holt, 1993 8-13. Print.

Basil L. Contovounesios

blc@netsoc.tcd.ie

http://blc.netsoc.ie/slides/