1 Regular expressions: key idea

- \texttt{rm *.*} We all know what it means

- regular expressions are a generalization of this

- Idea is to capture a pattern

- Basic tool in most string handling. Like a vector in math. Everything starts here.

2 We have seen this before

- What is the expected time until you see “abracadabra” in a collection of random letters?
  - Naive: $26^{11}$
  - Correct: $26^{11} + 26^4 + 26$
– Best proof: martingales
– For our purposes: Markov chains

• Look at interesting points: “”, “a”, “ab”, “abr”, “abra”, “abracadabra”

• Solution: nodes and labelled edges between them

• Basically a DFA

• Goal: write down a DFA without all the graphical structure

3 Regular expression language

• Problem: look for \begin{itemize} and \end{itemize} pairs.

• Subproblem: look for \begin{itemize}. Since \ is the escape character in regular expressions, we have to escape it.

• So: \begin{itemize}.

• Note: How to type this in the shell? you need to escape the escapes. So 4 in all!

• So to match pairs: \begin{itemize}.\end{itemize}.*\begin{itemize}.

• Oops: Doesn’t quite work. But might be good enough.

4 Relationship to DFA

• Consider (here|there). It matches either “here” or “there”.
Equivalent DFA: Two paths: (1) here (2) there.

Alternative form: Two paths (1) t (2) empty, then “here”.

But what about a .* in the middle?

- try: a.*b.
- same as: a[^b]*b. (as far as yes/no is concerned)
- Need to match a, then a cycle of non-”b”’s followed by a b.

So a DFA has to figure out the magic.

5 Relationship to NFA

- NFA allows more than one direction
- have more than one path.
- Requires doing search

6 Where do you find regex?

- Linux: grep / egrep / sed, etc
- emacs: but then it has everything
- perl: built on regex
- R: included
> grep("d", "abcdefg")
[1] 1

> grep("z", "abcdefg")
integer(0)