17. Introduction to Theoretical CS
Introduction to theoretical computer science

Fundamental questions
- What can a computer do?
- What can a computer do with limited resources?

General approach
- Don't talk about specific machines or problems.
- Consider minimal abstract machines.
- Consider general classes of problems.

Surprising outcome. Sweeping and relevant statements about all computers.
Why study theory?

In theory...
• Deeper understanding of computation.
• Foundation of all modern computers.
• Pure science.
• Philosophical implications.

In practice...
• Web search: theory of pattern matching.
• Sequential circuits: theory of finite state automata.
• Compilers: theory of context free grammars.
• Cryptography: theory of computational complexity.
• Data compression: theory of information.
• ...

"In theory there is no difference between theory and practice. In practice there is."

— Yogi Berra
17. Introduction to Theoretical CS

- Regular expressions
- DFAs
- Applications
- Limitations
Pattern matching

Pattern matching problem. Is a given string a member of a given set of strings?

Example 1 (from genomics)

A nucleic acid is represented by one of the letters a, c, t, or g.

A genome is a string of nucleic acids.

A Fragile X Syndrome pattern is a genome having an occurrence of gcg, followed by any number of cgg or agg triplets, followed by ctg.

Note. The number of triplets correlates with Fragile X Syndrome, a common cause of mental retardation.

Q. Does this genome contain a such a pattern?

A. Yes.

```
 gcggcggtgtgtgcgagagagtgggttttaagctggcggaggcggctggcggaggctg
gcgaggcggctg
```

A genome is a string of nucleic acids. A nucleic acid is represented by one of the letters a, c, t, or g.
Pattern matching

Example 2 (from computational biochemistry)

An **amino acid** is represented by one of the characters CAVLIMCRKHDENQSTYFWP.

A **protein** is a string of amino acids.

A **C$_2$H$_2$-type zinc finger domain signature** is

- C followed by 2, 3, or 4 amino acids, followed by
- C followed by 3 amino acids, followed by
- L, I, V, M, F, Y, W, C, or X followed by 8 amino acids, followed by
- H followed by 3, 4, or 5 amino acids, followed by
- H.

**Q.** Is this protein in the C$_2$H$_2$-type zinc finger domain?

**A.** Yes.

\[
\text{CAASCGGPYACGGWAGYHAGWH}
\]
Pattern matching

Example 3 (from commercial computing)

An e-mail address is
• A sequence of letters, followed by
• the character "@", followed by
• the character ".", followed by a sequence of letters, followed by
• [any number of occurrences of the previous pattern]
• "edu" or "com" (others omitted for brevity).

Q. Which of the following are e-mail addresses?

<table>
<thead>
<tr>
<th>A.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:rs@cs.princeton.edu">rs@cs.princeton.edu</a></td>
<td>✓</td>
</tr>
<tr>
<td>not an e-mail address</td>
<td>✗</td>
</tr>
<tr>
<td><a href="mailto:wayne@cs.princeton.edu">wayne@cs.princeton.edu</a></td>
<td>✓</td>
</tr>
<tr>
<td>eve@airport</td>
<td>✗</td>
</tr>
<tr>
<td><a href="mailto:rs123@princeton.edu">rs123@princeton.edu</a></td>
<td>✗</td>
</tr>
</tbody>
</table>

Challenge. Develop a precise description of the set of strings that are legal e-mail addresses.
A **regular expression** (RE) is a notation for specifying sets of strings.

An RE is:
- A sequence of letters or ".".
- The **union** of two REs.
- The **closure** of an RE (any number of occurrences).
- May be delimited by ( ).

<table>
<thead>
<tr>
<th>operation</th>
<th>example RE</th>
<th>matches (IN the set)</th>
<th>does not match (NOT in the set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>concatenation</td>
<td>aabaab</td>
<td>aabaab</td>
<td>every other string</td>
</tr>
<tr>
<td>wildcard</td>
<td>.u.u.u.</td>
<td>cumulus jugulum</td>
<td>succubus tumultuous</td>
</tr>
<tr>
<td>union</td>
<td>aa</td>
<td>baab</td>
<td>aa baab</td>
</tr>
<tr>
<td>closure</td>
<td>ab*a</td>
<td>aa abbbba</td>
<td>ab ababa</td>
</tr>
<tr>
<td>parentheses</td>
<td>a(a</td>
<td>b)aab</td>
<td>aaaababa</td>
</tr>
<tr>
<td></td>
<td>(ab)*a</td>
<td>a ababababa</td>
<td>aa abbba</td>
</tr>
</tbody>
</table>
### More examples of regular expressions

The notation is surprisingly expressive.

<table>
<thead>
<tr>
<th>regular expression</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>.<em>spb.</em></td>
<td>raspberry crispbread</td>
<td>subspace subspecies</td>
</tr>
<tr>
<td>contains the trigraph spb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a*</td>
<td>(a<em>ba</em>ba<em>ba</em>)*</td>
<td>bbb</td>
</tr>
<tr>
<td>multiple of three b’s</td>
<td>aaa</td>
<td>bb</td>
</tr>
<tr>
<td></td>
<td>bbbababbaa</td>
<td>baabbbaa</td>
</tr>
<tr>
<td>.*0.....</td>
<td>1000234 98701234</td>
<td>111111111 403982772</td>
</tr>
<tr>
<td>fifth to last digit is 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcg(cgg</td>
<td>agg)*ctg</td>
<td>gcgcggctg gcgcgggctg gcgcggaggctg</td>
</tr>
<tr>
<td>fragile X syndrome pattern</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generalized regular expressions

Additional operations further extend the utility of REs.

<table>
<thead>
<tr>
<th>operation</th>
<th>example RE</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>one or more</td>
<td>a(bc)+de</td>
<td>abcde</td>
<td>ade</td>
</tr>
<tr>
<td></td>
<td></td>
<td>abcbcde</td>
<td>bcd</td>
</tr>
<tr>
<td>character class</td>
<td>[A-Za-z][a-z]</td>
<td>lowercase Capitalized</td>
<td>camelCase illegal</td>
</tr>
<tr>
<td>exactly k</td>
<td>[0-9]{5}–[0-9]{4}</td>
<td>08540–1321 19072–5541</td>
<td>111111111 166–54–1111</td>
</tr>
<tr>
<td>negation</td>
<td>[^aeiou]{6}</td>
<td>rhythm</td>
<td>decade</td>
</tr>
<tr>
<td>white space</td>
<td>\s</td>
<td>any whitespace char (space, tab, newline...)</td>
<td>every other character</td>
</tr>
</tbody>
</table>

Note. These operations are all shorthand. They are very useful but not essential.

RE: (a|b|c|d|e)(a|b|c|d|e)*
shorthand: (a–e)+
Example of describing a pattern with a generalized RE

A $C_2H_2$-type zinc finger domain signature is
- C followed by 2, 3, or 4 amino acids, followed by
- C followed by 3 amino acids, followed by
- L, I, V, M, F, Y, W, C, or X followed by 8 amino acids, followed by
- H followed by 3, 4, or 5 amino acids, followed by
- H.

Q. Give a generalized RE for all such signatures.

A. C.$^{\{2,4\}}$C...[LIVMFYW]{$^8$}H.$^{\{3,5\}}$H

"Wildcard" matches any of the letters CAVLIMCRKHDENQSTYFWP
Example of a real-world RE application: PROSITE

PROSITE consists of documentation entries describing protein domains, families and functional sites as well as associated patterns and profiles to identify them [More... / References / Commercial users]. PROSITE is complemented by ProRule, a collection of rules based on profiles and patterns, which increases the discriminatory power of profiles and patterns by providing additional information about functionally and/or structurally critical amino acids [More...].

Forthcoming changes: information can be found here.

Release 20.97, of 08-Nov-2013 (1673 documentation entries, 1308 patterns, 1056 profiles and 1062 ProRule)

PROSITE access

- by documentation entry
- by ProRule description
- by taxonomic scope
- by number of positive hits

Type an RE here
Another example of describing a pattern with a generalized RE

An e-mail address is

- A sequence of letters, followed by
- the character "@", followed by
- the character ".", followed by a sequence of letters, followed by
- [any number of occurrences of the previous pattern]
- "edu" or "com" (others omitted for brevity).

Q. Give a generalized RE for e-mail addresses.

A. \([a-z]+@[a-z]+\.[a-z]+(edu|com)\)

Exercise. Extend to handle rs123@princeton.edu, more suffixes such as .org, and any other extensions you can think of.

Next. Determining whether a given string matches a given RE.
Self-assessment 1 on REs

Q. Which of the following strings match the RE \( a^*bb(ab|ba)^* \)?

is in the set it describes

1. abb
2. aaba
3. abba
4. bbaaab
5. cbb
6. bbababbab
Self-assessment 2 on REs

Q. Give an RE for *genes*
   • Characters are a, c, t or g.
   • Starts with atg (a *start codon*).
   • Length is a multiple of 3.
   • Ends with tag, taa, or ttg (a *stop codon*).
17. Introduction to Theoretical CS

- Regular expressions
- DFAs
- Applications
- Limitations
A **DFA** is an abstract machine that solves a pattern matching problem.

- A string is specified on an input tape (no limit on its length).
- The DFA reads each character on input tape once, moving left to right.
- The DFA lights "YES" if it **recognizes** the string, "NO" otherwise.

Each DFA defines a **set** of strings (all the strings that it recognizes).
A **DFA** is an abstract machine with a finite number *states*, each labeled Y or N, and *transitions* between states, each labelled with a symbol. One state is the *start* state.

- Begin in the *start* state (denoted by an arrow from nowhere).
- Read an input symbol and move to the indicated state.
- Repeat until the last input symbol has been read.
- Turn on the "YES" or "NO" light according to the label on the current state.

Does this DFA recognize this string?
A **DFA** is an abstract machine with a finite number of states, each labeled Y or N and transitions between states, each labelled with a symbol. One state is the **start** state.

- Begin in the **start** state.
- Read an input symbol and move to the indicated state.
- Repeat until the last input symbol has been read.
- Turn on the "YES" or "NO" light according to the label on the current state.

Does this DFA recognize this string?
Simulating the operation of a DFA

```java
public class DFA {
    private int state;
    private int start;
    private String[] action;
    private ST<Character, Integer>[] next;

    public DFA(In in) {
        /* Fill in data structures */
    }

    public String simulate(String input) {
        state = start;
        for (int i = 0; i < input.length(); i++) {
            state = next[state].get(input.charAt(i));
        }
        return action[state];
    }

    public static void main(String[] args) {
        DFA dfa = new DFA(new In(args[0]));
        while (!StdIn.isEmpty()) {
            input = StdIn.readString();
            StdOut.println(dfa.simulate(input));
        }
    }
}
```

### Table

<table>
<thead>
<tr>
<th>action[]</th>
<th>next[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0 0 1</td>
</tr>
<tr>
<td>b</td>
<td>1 1 2</td>
</tr>
<tr>
<td>c</td>
<td>2 2 0</td>
</tr>
</tbody>
</table>

### Diagram

```
   Y 0
   b   N 1
    b  N 2
```

### more b3.txt

```
% more b3.txt

Yes 0 1
No 1 2
No 2 0
```

### java DFA b3.txt

```
bababa
Yes
bb
No
```

### alphabet

```
ab
0
```
Self-assessment 1 on DFAs

Q. Which of the following strings does this DFA accept?

1. Bitstrings that end in 1
2. Bitstrings with an equal number of occurrences of 01 and 10
3. Bitstrings with more 1s than 0s
4. Bitstrings with an equal number of occurrences of 0 and 1
5. Bitstrings with at least one 1
**Self-assessment 2 on DFAs**

Q. Which of the following strings does this DFA accept?

1. Bitstrings with at least one 1
2. Bitstrings with an equal number of occurrences of 01 and 10
3. Bitstrings with more 1s than 0s
4. Bitstrings with an equal number of occurrences of 0 and 1
5. Bitstrings that end in 1
Kleene's theorem

**Two ways to define a set of strings**
- Regular expressions (REs).
- Deterministic finite automata (DFAs).

**Remarkable fact.** DFAs and REs are *equivalent*.

**Equivalence theorem (Kleene)**
Given any RE, there exists a DFA that accepts the same set of strings.
Given any DFA, there exists an RE that matches the same set of strings.

**Consequence: A way to solve the RE pattern matching problem**
- Build the DFA corresponding to the given RE.
- Simulate the operation of the DFA.

$S =$ the set of $ab$ strings where the number of occurrences of $b$ is a multiple of $3$

**DFA for $S**

**RE for $S**

$\text{RE for } S \quad a^* \mid (a^*ba^*ba^*)^*$
17. Introduction to Theoretical CS

- Regular expressions
- DFAs
- Applications
- Limitations
GREP: a solution to the RE pattern matching problem

An algorithm for the RE pattern matching problem?
• Build the DFA corresponding to the given RE.
• Simulate the operation of the DFA.

Practical difficulty: The DFA might have *exponentially* many states.

A more efficient algorithm: use Nondeterministic Finite Automata (NFA)
• Build the NFA corresponding to the given RE.
• Simulate the operation of the NFA.

"GREP" (Generalized Regular Expression Pattern matcher).
• Developed by Ken Thompson, who designed and implemented Unix.
• Indispensable programming tool for decades.
• Found in most development environments, including Java.
REs in Java

Java's String class implements GREP.

```
public class String {
    ...
    boolean matches(String re) {
        // does this string match the given RE?
    }
    ...
}
```

String re = "C.{2,4}C...[LIVMFYWC].{8}H.{3,5}H";
String zincFinger = "CAASCGGYPACGGAAGYHAGAH";
boolean test = zincFinger.matches(re);
true!
Java RE client example: Validation

public class Validate {
    public static void main(String[] args) {
        String re = args[0];
        while (!StdIn.isEmpty()) {
            String input = StdIn.readString();
            StdOut.println(input.matches(re));
        }
    }
}

Does a given string match a given RE?
• Take RE from command line.
• Take strings from StdIn.

Applications
• Scientific research.
• Compilers and interpreters.
• Internet commerce.
• ...

C$_2$H$_2$ type zinc finger domain

need quotes to "escape" the shell

% java Validate "C.{2,4}C...[LIVMFYWC].[8]H.{3,5}H"
true
CAASCGPYYACGGAAGYHAGAH
false

% java Validate "[$_A-Za-z]$_A-Za-z0-9]*"
ident123
true
123ident
false

% java Validate "[a-z]+@[a-z]+\.(edu|com)"
wayne@cs.princeton.edu
true
eve@airport
false

legal Java identifier
valid email address (simplified)
Beyond matching

Java's `String` class contains other useful RE-related methods.
- RE search and replace
- RE delimited parsing

```java
public class String {
  ...
  String replaceAll(String re, String to) {
    // replace all occurrences of substrings matching RE with to
  }
  String[] split(String re) {
    // split the string around matches of the given RE
  }
  ...
}
```

Tricky notation (typical in string processing): \ signals "special character" so "\"" means "\"
and "\s" means "\s"

Examples using the RE "\\s+" (matches one or more whitespace characters).

Replace each sequence of at least one whitespace character with a single space.
```java
String s = StdIn.readString();
s = s.replaceAll("\\s+", " ");
```

Create an array of the words in StdIn (basis for StdIn.readString() method)
```java
String s = StdIn.readString();
String[] words = s.split("\\s+");
```
Way beyond matching

Java's Pattern and Matcher classes give fine control over the GREP implementation.

<table>
<thead>
<tr>
<th>public class Pattern</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>parse the re to construct a Pattern</td>
</tr>
<tr>
<td>static Pattern compile(String re)</td>
<td>create a Matcher that can find substrings matching the pattern in the given input string</td>
</tr>
<tr>
<td>Matcher matcher(String input)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public class Matcher</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>set internal variable match to the next substring that matches the RE in the input. If none, return false, else return true</td>
</tr>
<tr>
<td>boolean find()</td>
<td>return match</td>
</tr>
<tr>
<td>String group()</td>
<td>return the kth group (identified by parens within RE) in match</td>
</tr>
<tr>
<td>String group(int k)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

[A sophisticated interface designed for pros, but very useful for everyone.]
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class Harvester {
    public static void main(String[] args) {
        String re = args[0];
        In in = new In(args[1]);
        String input = in.readAll();
        Pattern pattern = Pattern.compile(re);
        Matcher matcher = pattern.matcher(input);
        while (matcher.find())
            StdOut.println(matcher.group());
    }
}

% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgccggccggccggccggctg
gcgctg
gcgctg
gcgccggccggccggccggccggctg

% java Harvester "([a-z]+@[a-z]+$)+.+(edu|com)" http://www.cs.princeton.edu/people/faculty
...rs@cs.princeton.edu
...
wayne@cs.princeton.edu
...

Harvest information from input stream
- Take RE from command line.
- Take input from file or web page.
- Print all substrings matching RE.

harvest patterns from DNA
harvest email addresses from web for spam campaign.
Java pattern matcher real-world example: Parsing a data file

A typical situation
- An institution publishes data on the web to be shared by all.
- The data is published in human-readable form.
- You want to strip out everything but the raw data in order to process it.

Example: National Center for Biotechnology Information genome data

```
LOCUS AC146846 128142 bp DNA linear HTG 13-NOV-2003
DEFINITION Ornithorhynchus anatinus clone CLM1-393H9,
ACCESSION AC146846
VERSION AC146846.2 GI:38304214
KEYWORDS HTG; HTGS_PHASE2; HTGS_DRAFT.
SOURCE Ornithorhynchus anatinus (platypus)
ORIGIN
```

```
// a comment
```

```
 1  tgtatattcat ttgaccgtgc tggttttttcc cggttttttca gtaggggtttt agggagccac
 61  gtattcttgt ttgtttttatg ctgcgaata gctggtctgt gaaatctgtc atacagct
121  gccgcaggg gaataatcac aatggtgatg acaaaaatgta ggaagctgt ttctctataa
...
128101  gaaatgcga cccccacgcct aatgacgc ttcttttagat tg
```
Java pattern matcher real-world example: Parsing a data file

Key challenge: Develop an appropriate RE.

```
[ ]*[0-9]+([actg]*)*
```

Parens identify a *group* that includes only the data (a, c, t, g, or spaces).

Ignore everything else

Slight glitch: Need to remove spaces afterwards.

Extract data after spaces followed by a line number.

```
LOCUS AC146846 128142 bp DNA linear HTG 13-NOV-2003
DEFINITION Ornithorhynchus anatinus clone CLM1-393H9,
ACCESSION AC146846
VERSION AC146846.2 GI:38304214
KEYWORDS HTG; HTGS_PHASE2; HTGS_DRAFT.
SOURCE Ornithorhynchus anatinus (platypus)
ORIGIN

1 tgtatattcat ttgaccgtgc tggtttttccc cgggttttcc gtacggtttt agggagccac
61 gtgattctct gtgttttatcg tggcggaaaa acgtgtctcgaa gaaatctctagg atagcagc
121 gccgcagggga gaatgacca gttctgatgt gaaaaatgta ggaagctgt ttcttcataa
... 128101 ggaatgacca cccccacgct aatgtacgct ttctttagat tg
//
```
Java pattern matcher real-world example: Parsing a data file

```java
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class ParseNCBI
{
    public static void main(String[] args)
    {
        String re = "\[[\][0-9]+([actg ]*)\].*";
        Pattern pattern = Pattern.compile(re);
        In in = new In(args[0]);
        while (in.hasNext Line())
        {
            String line = in.readLine();
            Matcher matcher = pattern.matcher(line);
            if (matcher.find())
                StdOut.print(matcher.group(1).replaceAll(" ", ",");
        }
        StdOut.println();
    }
}
```

% java ParseNCBI platypus.txt
ttattttcatagggtcgtgttttctccgggttttcagttacgtttagggagccacgtgtgtgtttgatgagcaggtgcgcagggagaaatgaccagttttgatgatctctgcatagacagctgccgcagggagaatgaccagttttgatgacaaaatgtaggaaaaatgtcgtttttttcataa...

remove the spaces
Applications of REs

Pattern matching and beyond.
• Compile a Java program.
• Scan for virus signatures.
• Crawl and index the Web.
• Process natural language.
• Access information in digital libraries.
• Search-and-replace in a word processors.
• Process NCBI and other scientific data files.
• Filter text (spam, NetNanny, ads, Carnivore, malware).
• Validate data-entry fields (dates, email, URL, credit card).
• Search for markers in human genome using PROSITE patterns.
• Automatically create Java documentation from Javadoc comments.

GREP and related facilities are built in to Java, Unix shell, PERL, Python ...

virtually every computing environment
WHENEVER I LEARN A NEW SKILL, I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!

BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!

IT'S HOPELESS!

EVEYBOY STAND BACK.

I KNOW REGULAR EXPRESSIONS.

http://xkcd.com/208/
17. Introduction to Theoretical CS

- Regular expressions
- DFAs
- Applications
- Limitations
Summary

Programmers
- Regular expressions are a powerful pattern matching tool.
- Equivalent DFA/NFA paradigm facilitates implementation.
- Combination greatly facilitates real-world string data processing.

Theoreticians
- REs provide compact descriptions of sets of strings.
- DFAs are abstract machines with equivalent descriptive power.
- Are there languages and machines with more descriptive power?

You
- CS core principles provide useful tools that you can exploit now.
- REs and DFAs provide an introduction to theoretical CS.
Basic questions

Q. Are there sets of strings that cannot be described by any RE?
A. Yes.
   • Bitstrings with equal number of 0s and 1s.
   • Strings that represent legal REs.
   • Decimal strings that represent prime numbers.
   • DNA strings that are Watson-Crick complemented palindromes.
   • ...

Q. Are there sets of strings that cannot be described by any DFA?
A. Yes.
   • Bit strings with equal number of 0s and 1s.
   • Strings that represent legal REs.
   • Decimal strings that represent prime numbers.
   • DNA strings that are Watson-Crick complemented palindromes.
   • ...

The same question, by Kleene's theorem
A limit on the power of REs and DFAs

**Proposition.** There exists a set of strings that cannot be described by any RE or DFA.

**Proof sketch.** No DFA can recognize the set of bitstrings with equal number of 0s and 1s.

- *Assume that you have such a DFA*, with $N$ states.
- It recognizes the string with $N + 1$ 0s followed by $N + 1$ 1s.
- Some state is revisited when recognizing that string.
- Delete the substring between visits.
- DFA recognizes that string, too.
- It does not have equal number of 0s and 1s.
- *Proof by contradiction:* the assumption that such a DFA exists must be false.

| Ex. $N = 10$ | 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 | 0 3 5 9 8 7 5 . . . |
|             | 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 | 0 3 5 . . . |
Another basic question

Q. Are there abstract machines that are more powerful than DFAs?
A. Yes. A 1-stack DFA can recognize
• Bitstrings with equal number of 0s and 1s.
• Strings that represent legal REs.

Proof. [details omitted]
Yet another basic question

Q. Are there abstract machines that are more powerful than a 1-stack DFA?
A. Yes. A 2-stack DFA can recognize
   • Decimal strings that represent prime numbers.
   • Strings that represent legal Java programs.
   • ...

[stay tuned for next lecture]
One last basic question

Q. Are there machines that are more powerful than a 2-stack DFA?
A. No! Not even a roomful of supercomputers (!!!)

[stay tuned for next lecture]
17. Introduction to Theoretical CS