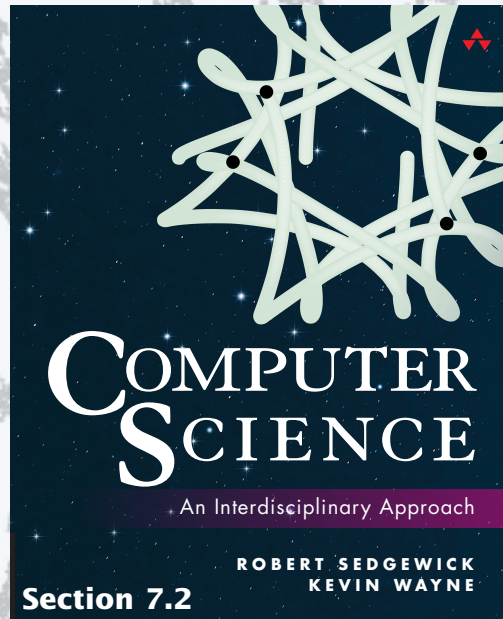


**COMPUTER SCIENCE**  
**SEDGEWICK / WAYNE**

PART II: ALGORITHMS, MACHINES, and THEORY



<http://introcscs.princeton.edu>

# 14. Introduction to Theoretical CS

## 14. Introduction to Theoretical CS

- **Overview**
- Regular expressions
- DFAs
- Applications
- Limitations

# 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*

*— Yogi Berra*

# Abstract machines

---

## Abstract machine

- Mathematical model of computation.
- Each machine defined by specific rules for transforming input to output.
- This lecture: Deterministic finite automata (DFAs).

## Formal language

- A set of strings.
- Each defined by specific rules that characterize it.
- This lecture: Regular expressions (REs).

## Questions for this lecture

- Is a given string in the language defined by a given RE, or not?
- Can a DFA help answer this question?



```
      madam im adam
a man a plan a canal panama
  able i was ere i saw elba
    evil olive
go hang a salami im a lasagna hog
  pull up if i pull up
    ...
```



**COMPUTER SCIENCE**  
S E D G E W I C K / W A Y N E  
PART I: PROGRAMMING IN JAVA

CS.17.A.Theory.Overview

## 14. Introduction to Theoretical CS

- Overview
- **Regular expressions**
- DFAs
- Applications
- Limitations

## Pattern matching

**Pattern matching problem.** Is a given string an element of a given set of strings?

**Example 1** (from computational biochemistry)

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

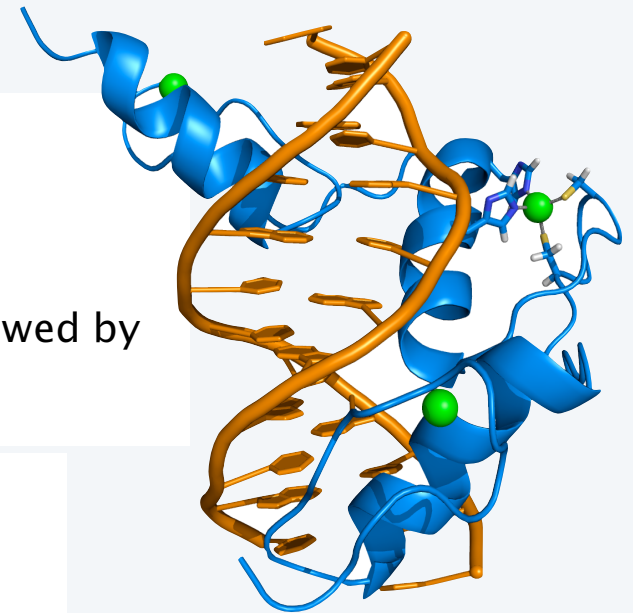
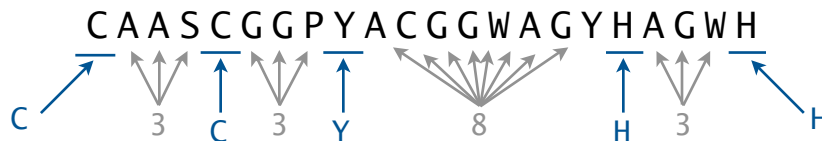
A **protein** is a string of amino acids.

A **C<sub>2</sub>H<sub>2</sub>-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<sub>2</sub>H<sub>2</sub>-type zinc finger domain?

**A.** Yes.





## Pattern matching

---

### Example 2 (from commercial computing)

An **e-mail address** is

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

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

	A.
rs@cs.princeton.edu	✓
not an e-mail address	✗
wayne@cs.princeton.edu	✓
eve@airport	✗
rs123@princeton.edu	✗

Oops, need to fix description →

**Challenge.** Develop a precise description of the set of strings that are legal e-mail addresses.

## Pattern matching

---

### Example 3 (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.

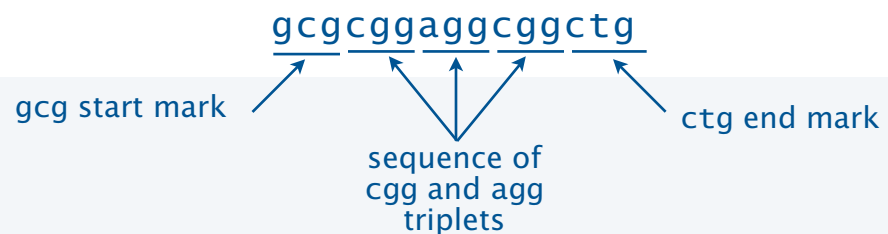
A **Fragile X Syndrome pattern** is a genome having an occurrence of gcg, followed by any number of cgg or agg triplets, followed by ctg.

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?

gcggcgtgtgtgagagagagtgggtttaaagctg gcgcggaggcggctggcgcggaggctg

A. Yes.



## Regular expressions

A **regular expression** (RE) is a notation for specifying a set of strings ( a formal language).

An RE is either

- The empty set
- The empty string
- A single character or wildcard symbol
- An RE enclosed in parentheses
- The *concatenation* of two or more REs
- The *union* of two or more REs
- The *closure* of an RE  
(any number of occurrences)

<i>operation</i>	<i>example RE</i>	<i>matches (IN the set)</i>	<i>does not match (NOT in the set)</i>
<i>concatenation</i>	aabaab	aabaab	every other string
<i>wildcard</i>	.u.u.u.	cumulus jugulum	succubus tumultuous
<i>union</i>	aa   baab	aa baab	every other string
<i>closure</i>	ab*a	aa abbba	ab ababa
<i>parentheses</i>	a(a b)aab	aaaab abaab	every other string
	(ab)*a	a ababababa	aa abbba

## More examples of regular expressions

The notation is surprisingly expressive.

<i>regular expression</i>	<i>matches</i>	<i>does not match</i>
<code>.*spb.*</code> <i>contains the trigraph spb</i>	raspberry crispbread	subspace subspecies
<code>a*   (a*ba*ba*ba*)*</code> <i>multiple of three b's</i>	bbb aaa bbbaababbaa	b bb baabbbaa
<code>.*0....</code> <i>fifth to last digit is 0</i>	1000234 98701234	111111111 403982772
<code>.*gcg(cgg agg)*ctg.*</code> <i>fragile X syndrome pattern</i>	...gcgctg... ...gcgcggctg... ...gcgcggaggctg...	gcgcgg cggcggcggctg gcgcaggctg

## Generalized regular expressions

Additional operations further extend the utility of REs.

<i>operation</i>	<i>example RE</i>	<i>matches</i>	<i>does not match</i>
one or more	<code>a(bc)+de</code>	abcde abcbcde	ade bcde
character class	<code>[A-Za-z][a-z]*</code>	lowercase Capitalized	camelCase 4illegal
exactly j	<code>[0-9]{5}-[0-9]{4}</code>	08540-1321 19072-5541	111111111 166-54-1111
between j and k	<code>a.{2,4}b</code>	abcb abcxcb	ab aaaaaab
negation	<code>[^aeiou]{6}</code>	rhythm	decade
whitespace	<code>\s</code>	<i>any whitespace char</i> (space, tab, newline...)	<i>every other character</i>

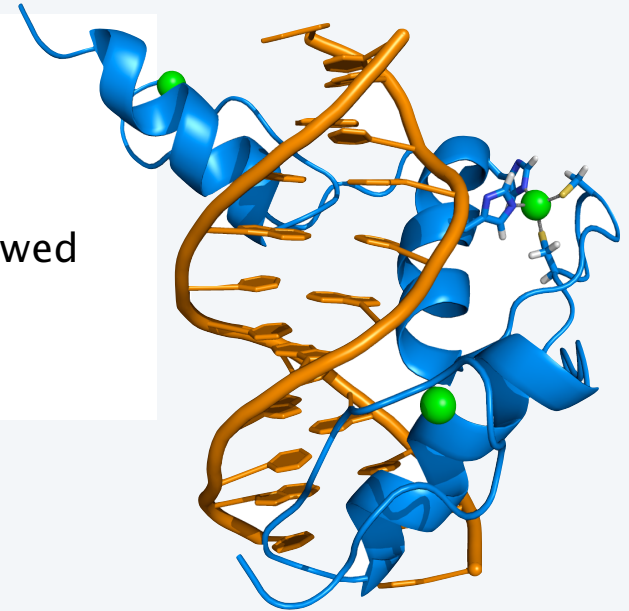
**Note.** These operations are all *shorthand*.  
They are very useful but not essential.

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

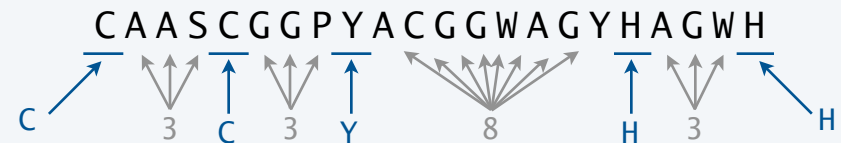
- C followed by 2, 3, or 4 amino acids, followed by
- C followed by 3 amino acids, followed by
- L, I, V, M, F, Y, W, C, or X followed by 8 amino acids, followed by
- H followed by 3, 4, or 5 amino acids, followed by



Q. Give a generalized RE for all such signatures.

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

"Wildcard" matches any of the letters  
CAVLIMCRKHDENQSTYFWP



# Example of a real-world RE application: PROSITE

**proSite** Database of protein domains, families and functional sites

Home ScanProsite ProRule Documents Downloads Links Funding

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

e.g. PDCC00022, PS50089, SH3, zinc finger  
  add wildcard "\*\*"

Browse:

- [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 nonempty sequence of lowercase 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]+\.)+(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.



## Pop quiz 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. bbbaab
5. cbb
6. bbababbab

## Pop quiz 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*).





**COMPUTER SCIENCE**  
SE D G E W I C K / W A Y N E  
PART I: PROGRAMMING IN JAVA

*Image sources*

[http://en.wikipedia.org/wiki/Homology\\_modeling#/media/File:DHR57B\\_homology\\_model.png](http://en.wikipedia.org/wiki/Homology_modeling#/media/File:DHR57B_homology_model.png)

## 14. Introduction to Theoretical CS

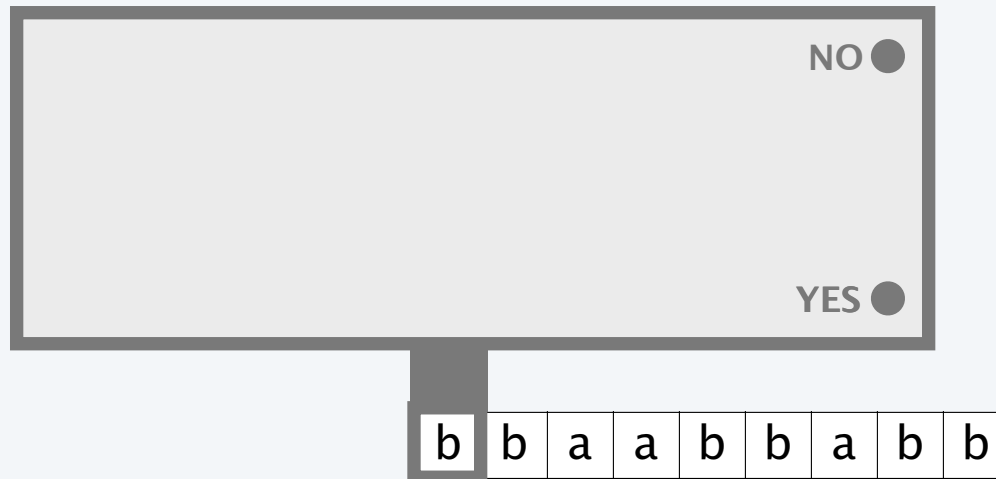
- Overview
- Regular expressions
- **DFAs**
- Applications
- Limitations

## Deterministic finite automata (DFA)

---

A **DFA** is an abstract machine that solves a pattern matching problem.

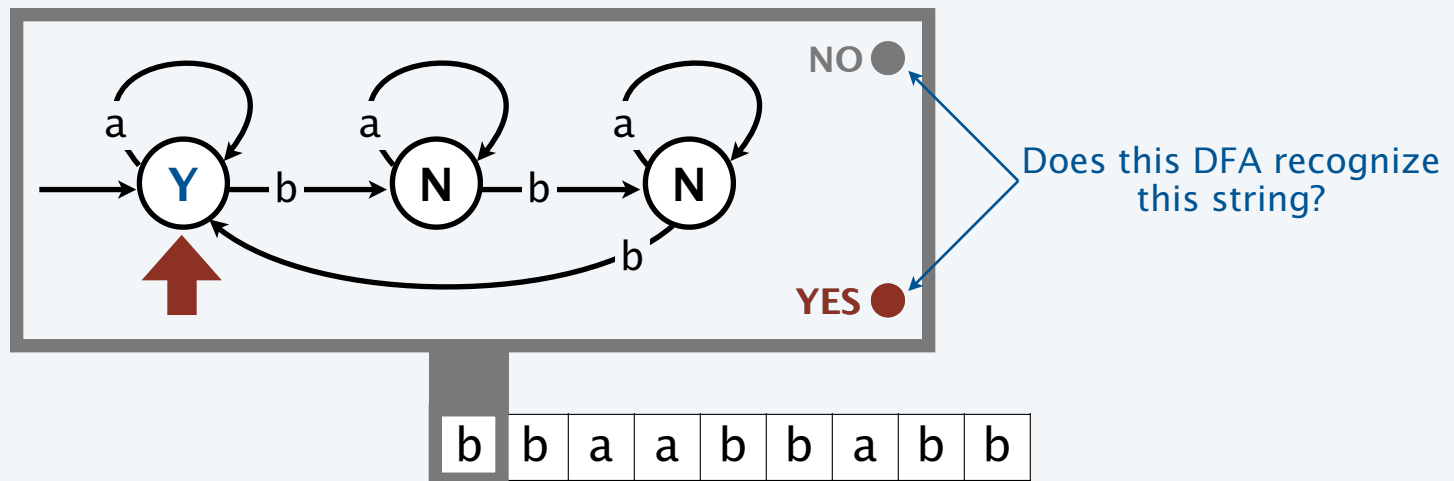
- A string is specified on an input tape (no limit on its length).
  - The DFA reads each character on input tape once, moving left to right.
  - The DFA lights "YES" if it *recognizes* the string, "NO" otherwise.
- Each DFA defines a *language* (the set of strings that it recognizes).



## Deterministic finite automata details and example

A **DFA** is an abstract machine with a finite number *states*, each labeled Y or N, and *transitions* between states, each labeled with a symbol. One state is the *start* state.

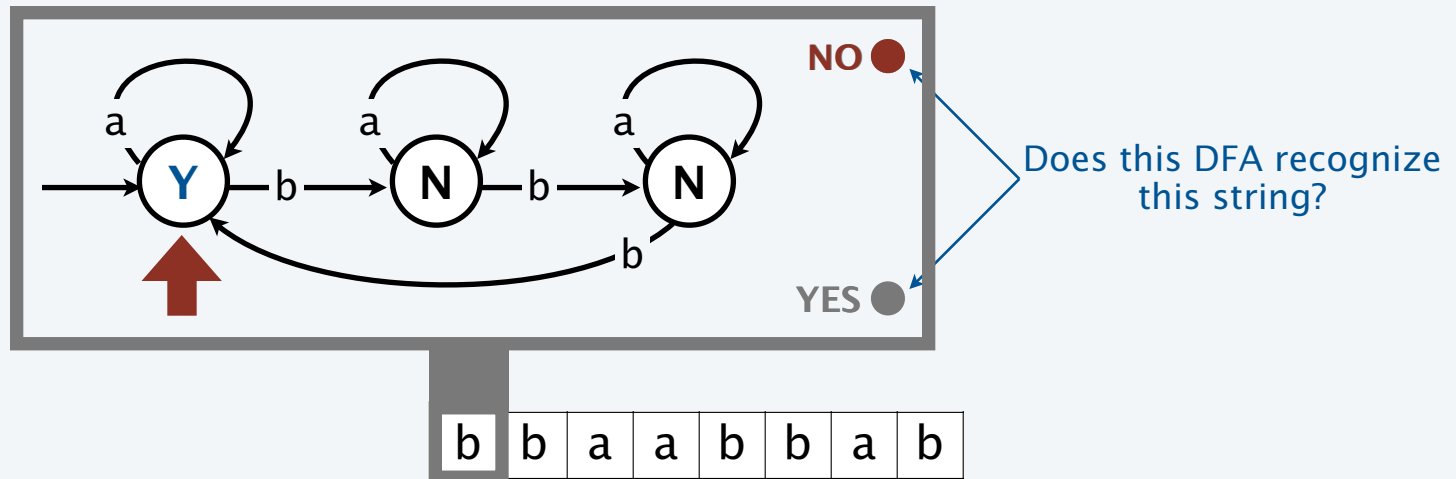
- Begin in the *start* state.
- Read an input symbol and move to the indicated state.
- Repeat until the last input symbol has been read.
- Turn on the "YES" or "NO" light according to the label on the current state.



## Deterministic finite automata details and example

A **DFA** is an abstract machine with a finite number *states*, each labeled Y or N, and *transitions* between states, each labeled 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.



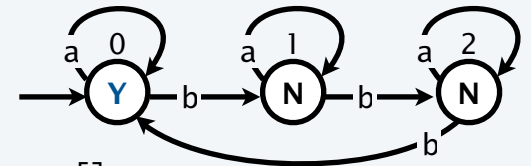
## Simulating the operation of a DFA

```

public class DFA
{
    private int start;
    private boolean[] action;
    private ST<Character, Integer>[] next;
    public DFA(String filename)
    { /* Fill in data structures */ }
    public boolean recognizes(String input)
    {
        int 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(args[0]);
        while (!StdIn.isEmpty())
        {
            input = StdIn.readString();
            if (dfa.recognizes(input)) StdOut.println("Yes");
            else StdOut.println("No");
        }
    }
}

```

symbol table to map  
chars a, b, ... to next  
state 0, 1, ...



	action[]	next[]		
			a	b
0	True	0	0	1
1	False	1	1	2
2	False	2	2	0

# states →  
alphabet →  
start state →

```

% more b3.txt
3
ab
0
True  0 1
False 1 2
False 2 0
% java DFA b3.txt
bababa
Yes
bb
No
abbabbababbbabaaa
Yes
abbabbababbba
No

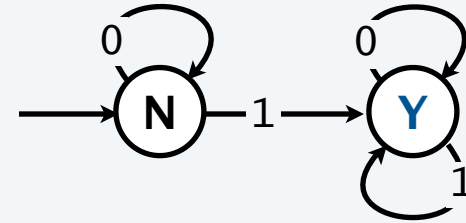
```



## Pop quiz 1 on DFAs

---

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

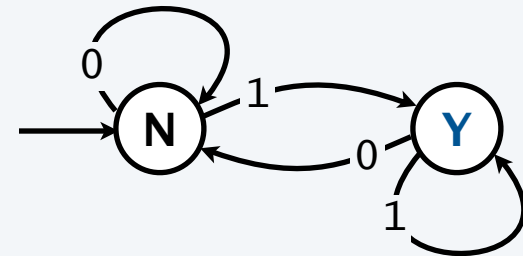


1. Bitstrings that end in 1
2. Bitstrings with an equal number of occurrences of 01 and 10
3. Bitstrings with more 1s than 0s
4. Bitstrings with an equal number of occurrences of 0 and 1
5. Bitstrings with at least one 1

## Pop quiz 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 (language)

- Regular expressions (REs).
- Deterministic finite automata (DFAs).

Remarkable fact. DFAs and REs are *equivalent*.

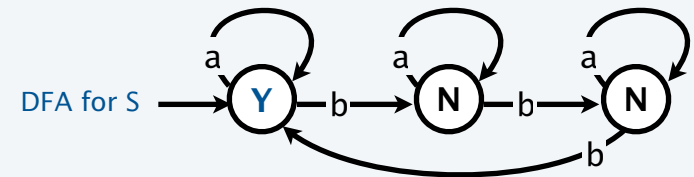
## Equivalence theorem (Kleene)

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



RE for  $S$   $a^* | (a^*ba^*ba^*ba^*)^*$



Steven Kleene  
1909–1994



**COMPUTER SCIENCE**  
S E D G E W I C K / W A Y N E  
PART I: PROGRAMMING IN JAVA

*Image sources*

<http://math.library.wisc.edu/images/skleene.gif>

## 14. Introduction to Theoretical CS

- Overview
- Regular expressions
- DFAs
- **Applications**
- Limitations

## GREP: a solution to the RE pattern matching problem

---

"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.

**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.



Interested in details? Take a course in algorithms.



Ken Thompson  
1983 Turing Award



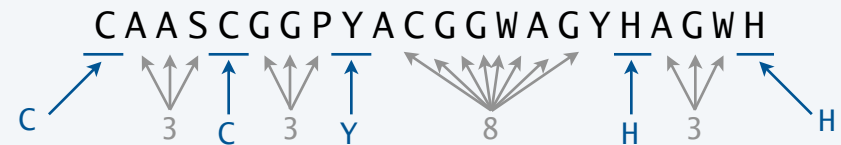
## REs in Java

Java's String class implements GREP.

public class String	
...	
boolean matches(String re)	<i>does this string match the given RE?</i>
...	

```
String re = "C.{2,4}C...[LIVMFYWC].{8}H.{3,5}H";  
String zincFinger = "CAASC GGPYACGGWAGYHAGWH";  
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));
        }
    }
}
```

### Applications

- Scientific research.
- Compilers and interpreters.
- Internet commerce.
- ...

### Does a given string match a given RE?

- Take RE from command line.
- Take strings from StdIn.

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

*C<sub>2</sub>H<sub>2</sub> type zinc finger domain*

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

*legal Java identifier*

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

*valid email address (simplified)*

need quotes to  
"escape" the shell



## Beyond matching

Java's `String` class contains other useful RE-related methods.

- RE search and replace
- RE delimited parsing

public class String	
...	
<code>String replaceAll(String re, String to)</code>	<i>replace all occurrences of substrings matching RE with to</i>
<code>String[] split(String re)</code>	<i>split the string around matches of the given RE</i>
...	

Examples using the RE `"\\s+"` (matches one or more whitespace characters). Tricky notation (typical in string processing): `\\` signals "special character" so `\\` means `\` and `\\s` means `\s`

Replace each sequence of at least one whitespace character with a single space.

```
String s = StdIn.readAll();  
s = s.replaceAll("\\s+", " ");
```

Create an array of the words in `StdIn` (basis for `StdIn.readAllStrings()` method)

```
String s = StdIn.readAll();  
String[] words = s.split("\\s+");
```

## Way beyond matching

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

public class Pattern	
...	
static Pattern compile(String re)	<i>parse the re to construct a Pattern</i>
Matcher matcher(String input)	<i>create a Matcher that can find substrings matching the pattern in the given input string</i>
...	
public class Matcher	
...	
boolean find()	<i>set internal variable match to the next substring that matches the RE in the input. If none, return false, else return true</i>
String group()	<i>return match</i>
String group(int k)	<i>return the kth group (identified by parens within RE) in match</i>
...	

Why not a constructor?  
Good question.

[A sophisticated interface designed for pros, but very useful for everyone.]

## Java pattern matcher client example: Harvester

```
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());
    }
}
```

### Harvest information from input stream

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

```
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
```

```
gcgcggcggcggcggcggcggctg
gcgctg
gcgctg
gcgcggcggcggaggcggaggcggctg
```

← harvest patterns from DNA

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

```
...
rs@cs.princeton.edu
...
wayne@cs.princeton.edu
...
```

← harvest email addresses from web for spam campaign.  
(no email addresses on that site any more)

## 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*



**COMPUTER SCIENCE**  
S E D G E W I C K / W A Y N E  
PART I: PROGRAMMING IN JAVA

*Image sources*

[http://en.wikipedia.org/wiki/Ken\\_Thompson#/media/File:Ken\\_n\\_dennis.jpg](http://en.wikipedia.org/wiki/Ken_Thompson#/media/File:Ken_n_dennis.jpg)

## 14. Introduction to Theoretical CS

- Overview
- 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



### 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 (stay tuned).
- 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 (see next slide).
- 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 scanning the 0s in that string.
- Delete the substring of 0s between visits of that state.
- 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	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	3	5	.	.	.															

## Another basic question

can recognize more sets of strings

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]



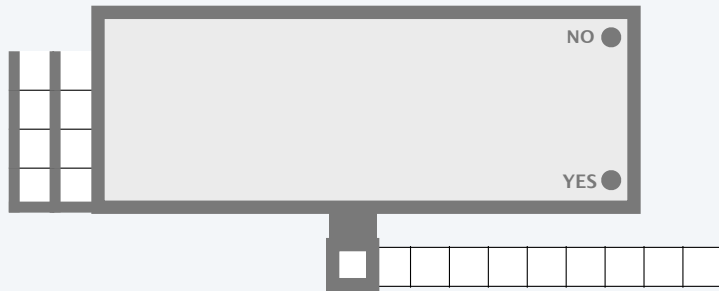
## One last basic question

---

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]



*two machines with equal computational power*



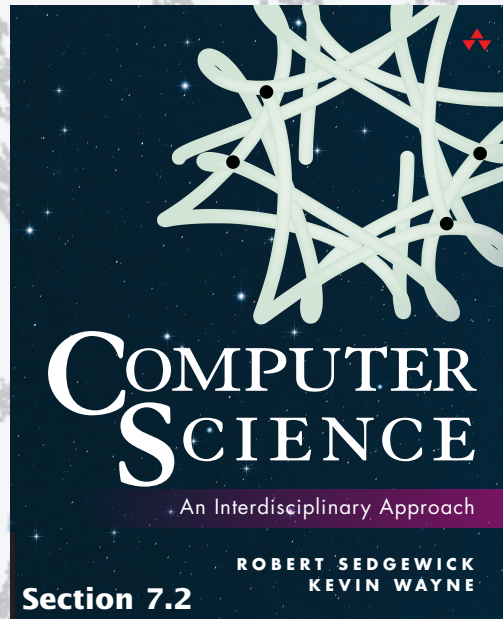
**COMPUTER SCIENCE**  
S E D G E W I C K / W A Y N E  
PART I: PROGRAMMING IN JAVA

*Image sources*

<https://openclipart.org/detail/211418/thenanobel-programming>

**COMPUTER SCIENCE**  
**SEDGEWICK / WAYNE**

PART II: ALGORITHMS, MACHINES, and THEORY



<http://introcs.cs.princeton.edu>

# 14. Introduction to Theoretical CS