CS311—Computational Structures

Problems, Languages, Machines, Computability, Complexity

Lecture 1

Andrew P. Black
Andrew Tolmach

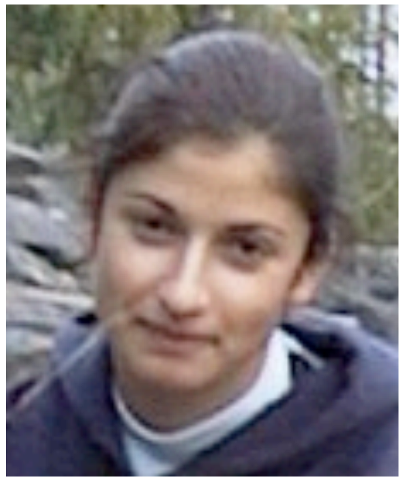
Portland State UNIVERSITY

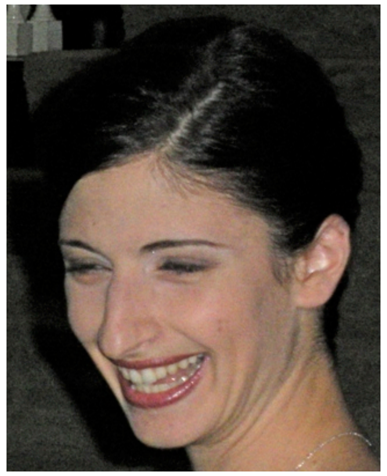
The Geography Game

- Could you write a computer program to play the geography game?

Face Recognition

- Are these the same person?

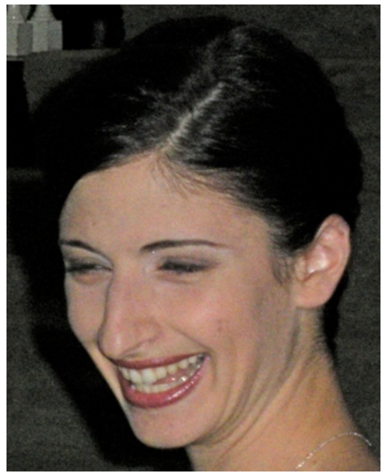




Portland State UNIVERSITY

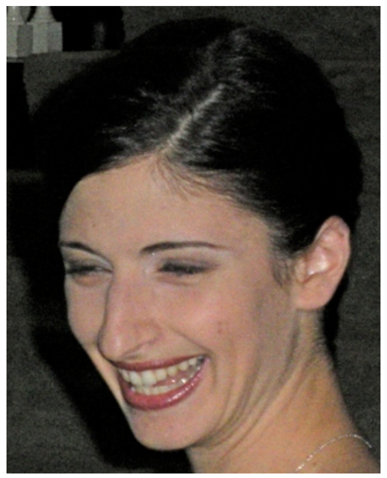
Face Recognition

- Are these the same person?



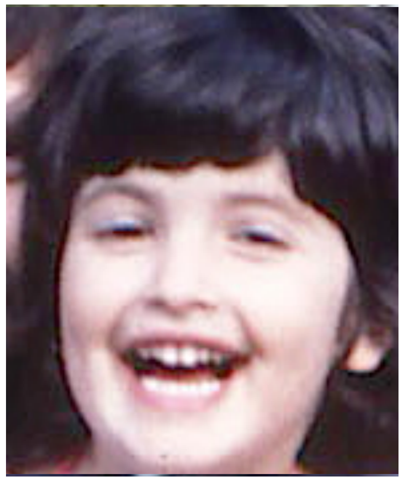
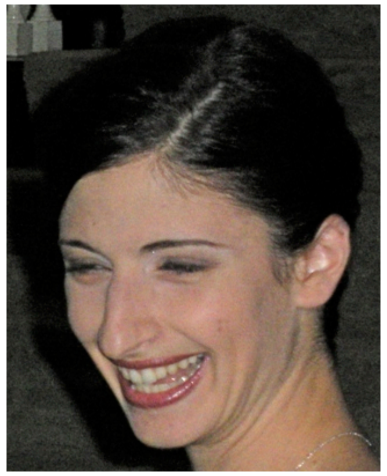
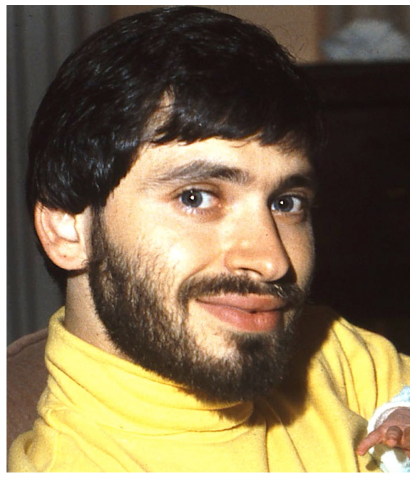
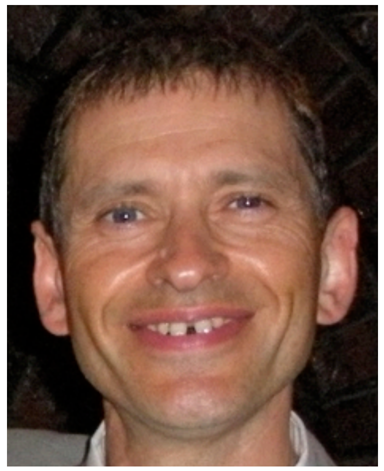
Face Recognition

- Are these the same person?



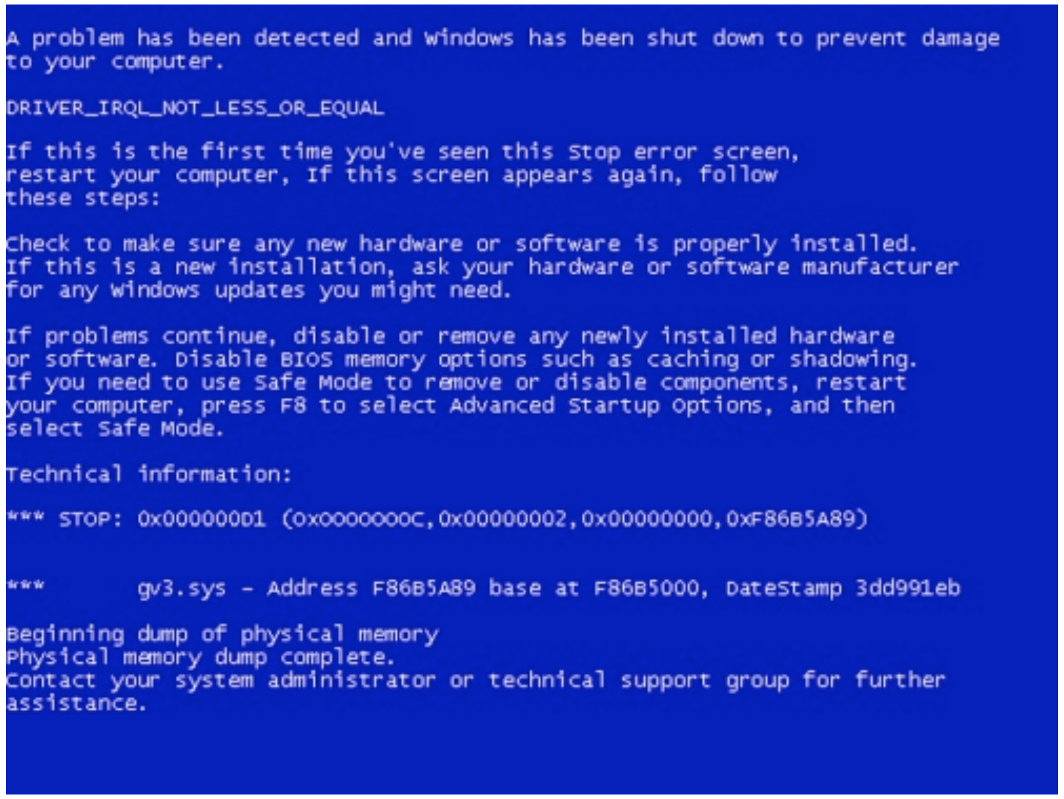
Face Recognition

- Are these the same person?
- Can you program a computer to recognize people?

Computer Programs

- Can you write a computer program that looks at a windows driver and decides if it will terminate within 100 ms?



```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

DRIVER_IRQL_NOT_LESS_OR_EQUAL

If this is the first time you've seen this Stop error screen,
restart your computer, If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x000000D1 (0x0000000C,0x00000002,0x00000000,0xF86B5A89)


***       gv3.sys - Address F86B5A89 base at F86B5000, DateStamp 3dd991eb

Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.
```

Windows 95

An error occured whilst trying to
load the application, "bluescreen.exe"

A fatal e VXB UMM (82)
88018E35

* Press an
* Press CT . You will
lose any
* Buy a Mac

Error Error

An error occured whilst trying to
load the previous error.

OK

Press any key to continue _

Hey, it looks like you're having
an error!

What’s the course about?

- What it really means to be a computer
  - Problems as “formal” language “recognition”
  - Algorithms as “formal” machine “programs”
- Different models of computation.
- What computers can do, and what they *can’t* do
- What’s easy to compute, and what’s hard

What’s the course about?

- What it really means to be a computer
    - Problems as “formal” language “recognition”
    - Algorithms as “formal” machine “programs”
- Different models of computation.
- What computers can do, and what they *can’t* do
- What’s easy to compute, and what’s intractable

Major Technical Topics

- Regular languages and regular expressions, deterministic and nondeterministic automata, closure properties, pumping lemma.
- Context-free languages and pushdown automata, parsing, closure properties, pumping lemma.
- Turing machines, Church-Turing thesis, equivalent models of computation.
- Computability: decidable and undecidable problems.
- Complexity classes: P, NP, completeness.

Portland State UNIVERSITY

Official Course Objectives

Upon the successful completion of this course students will be able to:

1. Find regular grammars and context-free grammars for simple languages whose strings are described by given properties.
2. Apply algorithms to: transform regular expressions to NFAs, NFAs to DFAs, and DFAs to minimum-state DFAs; construct regular expressions from NFAs or DFAs; and transform between regular grammars and NFAs.
3. Apply algorithms to transform: between PDAs that accept by final state and those that accept by empty stack; and between context-free grammars and PDAs that accept by empty stack.
4. Describe LL(*k*) grammars; perform factorization if possible to reduce the size of *k*; and write recursive descent procedures and parse tables for simple LL(1) grammars.
5. Transform grammars by removing all left recursion and by removing all possible productions that have the empty string on the right side.

6. Apply pumping lemmas to prove that some simple languages are not regular or not context-free.
7. State the Church-Turing Thesis and solve simple problems with some of the following models of computation: Turing machines (single-tape and multi-tape); while-loop programs; partial recursive functions; Markov algorithms; Post algorithms; the lambda calculus; and Post systems.
8. Describe the concepts of unsolvable and partially solvable; state the halting problem and prove that it is unsolvable and partially solvable; and use diagonalization to prove that the set of total computable functions cannot be enumerated.
9. Describe the hierarchy of languages and give examples of languages at each level that do not belong in a lower level.
10. Describe the complexity classes P, NP, and PSPACE.
11. Use an appropriate programming language as an experimental tool for testing properties of computational structures.

Course Tools

- What you need for this course:
  - Knowledge from the prerequisite courses (esp. CS250)
  - Textbook — Hopcroft, Motwani & Ullman
  - Pencil and paper
  - Brain and time
- What you don't need for this course:
  - A computer (except just a little bit now and then)
  - An attitude of fear and loathing
- This is a theoretical subject...embrace it!

Things to Do

- Read the syllabus
- Register for the course!
- Attend class and take notes
- Read the class web page
    - http://www.cs.pdx.edu/~black/cs311
- Sign up for the email list
- Read Chs. 1,2.1-2 of the IALC textbook
- Take the Entrance Exam

Gradiance

- On-line tutoring and quiz tool from Addison-Wesley
- Set up an account and connect to this course:
    - http://www.gradiance.com/pearson/servlet/SSOConnectorLogin
    - Course Token is TBA
    - This token is NOT on the web site, so write it down!

Warning!

- I expect you to read the class web page
- I expect you to read the class email list
- I expect you to read *and work problems from* the textbook!

If you can't or won't or don't have time to work independently, you will have trouble with this course.

Review of languages

- **Languages** are **sets** of **strings** of **symbols** drawn from some **alphabet**
- Some language examples:
  - $L_1 = \{a^n b^n \mid n \leq 2\}$
  - $L_2 = \{a^n b^m \mid n, m \geq 0\}$
  - $L_3 = \{a^n b^n \mid n \geq 0\}$
  - $L_4 = \{a^n b^m c^p \mid n, m, p \geq 0\}$
  - $L_5 = \{a^n b^n c^n \mid n \geq 0\}$
- $L1 \subseteq L3 \subseteq L2 \subseteq L4 \qquad L5 \subseteq L4$

Problems as Languages

- Why do we care about languages (in this sense)?
- They show up everywhere in CS
  - programming languages, HTML and XML documents, ...
  - (They also show up in human linguistics...)
- But the big reason for this course: we can use formal languages to describe **problems**
- In particular, we can reduce decision problems (is P true?) to language membership problems (is the encoding of P *in* some language L?)

Example: tic-tac-toe

```
O _ X      _ _ _      O _ X
O _ X      _ _ _      O _ _
_ O _      _ _ _      _ _ _
```

- Decision problem: for a given configuration, can the next player always win?
- Encode configurations as strings, e.g.

  OBXOBXBOB BBBBBBBBB OBXOBBBBB

- Define language L = set of configuration strings for which the next player can always win.
- For configuration C, we ask "is the encoding of C in L?"

Example: arithmetic

- Decision problem: given $p,q,r \in \mathbb{N}$, is $p \times q = r$?
    - E.g. $6 \times 7 = 42$? $7 \times 9 = 64$? $8 \times 8 = 91$?
- Encoding is simple
    - E.g. 6B7B42 7B9B64 8B8B91
- Define $L = \{pBqBr \mid p,q,r \in \{0..9\}^* \text{ and } p \times q = r\}$
- For a particular p,q,r, ask "is $pBqBr \in L$?"

- Is restriction to **decision** problems too limiting?

$\mathbb{N}$ and its cardinality

- $\mathbb{N} = \{0, 1, 2, 3, 4, \dots\}$
- How can we define $\mathbb{N}$ without "…" ?

$\mathbb{N}$ and its cardinality

- $\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}$
- How can we define $\mathbb{N}$ without "…" ?

  By induction

  1. $0 \in \mathbb{N}$
  2. if $x \in \mathbb{N}$, then $x+1 \in \mathbb{N}$
  3. these are the only elements of $\mathbb{N}$

Cardinality of Sets

Cardinality of Sets

- How can we decide if two sets are of the same size?

Cardinality of Sets

- How can we decide if two sets are of the same size?
- Set up a *bijection* (one-to-one and onto mapping) between the elements of the sets

Cardinality of Sets

- How can we decide if two sets are of the same size?
- Set up a *bijection* (one-to-one and onto mapping) between the elements of the sets
  - Is there a bijection between $\mathbb{N}$ and $\mathbb{N}_{\text{even}}$?

Cardinality of Sets

- How can we decide if two sets are of the same size?
- Set up a *bijection* (one-to-one and onto mapping) between the elements of the sets
  - Is there a bijection between $\mathbb{N}$ and $\mathbb{N}_{\text{even}}$?
  - Sure!

Portland State UNIVERSITY

Countability

- A **countable** set is one that is either finite or has the same cardinality as $\mathbb{N}$
  - Any subset of a countable set is countable
  - $\mathbb{N}$, $\mathbb{N}_{\text{even}}$ are countable
- We reason about countable sets using **induction**

Proofs by Induction

Proofs by Induction

- For example, to prove some property P(n) for all natural numbers n, it suffices to prove
  - Base case: P(0) is true.
  - Inductive step: If P(n) is true, then P(n+1) is true, forall n ≥ 0.

Proofs by Induction

- For example, to prove some property P(n) for all natural numbers n, it suffices to prove
  - Base case: P(0) is true.
  - Inductive step: If P(n) is true, then P(n+1) is true, forall $n \geq 0$.
- Why is this enough?
  - Every natural number can be produced by starting with 0 and repeatedly adding 1.
  - So, we can prove P(n) for any n by applying inductive step n times, then base case.

Prove by Induction

- Theorem: $0 + 1 + 2 + ... + n = n(n+1)/2$.
- State the induction hypothesis:
  - let $H(n)$ be $0 + 1 + 2 + ... + n = n(n+1)/2$
- Prove the base case: $H(0)$

  $H(0)$ is $0 = 0\,(1)/2$

  This is true because 0 is the zero of multiplication

A simple inductive proof

- Prove the inductive step:
  - We have to show that H(n) ⇒ H(n + 1)
  - H(n) is 0 + 1 + 2 + ... + n = n(n+1)/2

  add (n+1) to both sides:

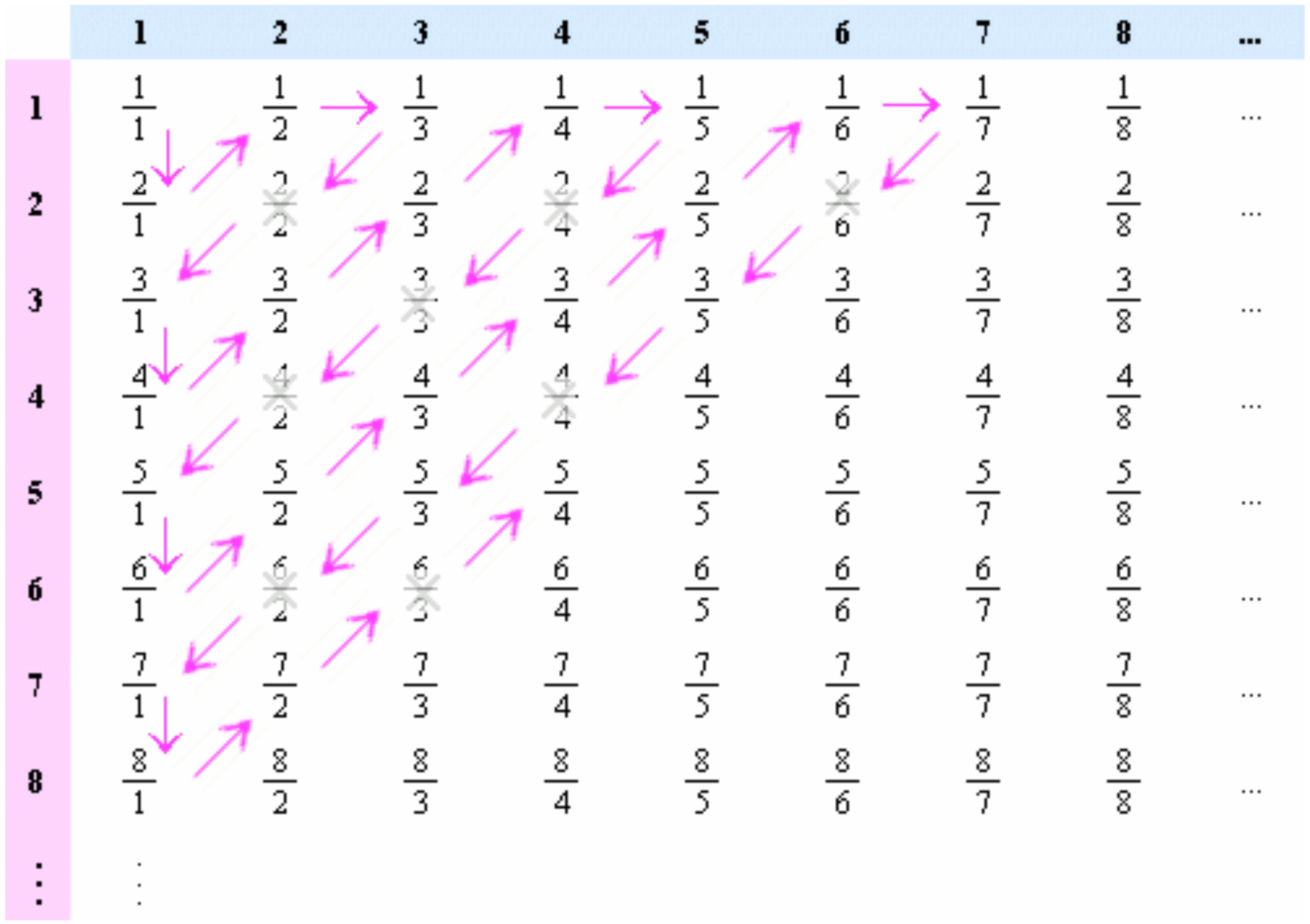
$$\begin{aligned} 0+1+2+\ldots+n + (n+1) &= n(n+1)/2 + (n+1) \\ &= n(n+1)/2 + 2(n+1)/2 \\ &= (n+2)(n+1)/2 \\ &= (n+1)((n+1)+1)/2 \end{aligned}$$

H(n + 1)

QED.

Portland State UNIVERSITY

Countability of ℚ



	1	2	3	4	5	6	7	8	...
1	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{7}$	$\frac{1}{8}$	...
2	$\frac{2}{1}$	$\cancel{\frac{2}{2}}$	$\frac{2}{3}$	$\cancel{\frac{2}{4}}$	$\frac{2}{5}$	$\cancel{\frac{2}{6}}$	$\frac{2}{7}$	$\frac{2}{8}$	...
3	$\frac{3}{1}$	$\frac{3}{2}$	$\cancel{\frac{3}{3}}$	$\frac{3}{4}$	$\frac{3}{5}$	$\frac{3}{6}$	$\frac{3}{7}$	$\frac{3}{8}$	...
4	$\frac{4}{1}$	$\cancel{\frac{4}{2}}$	$\frac{4}{3}$	$\cancel{\frac{4}{4}}$	$\frac{4}{5}$	$\frac{4}{6}$	$\frac{4}{7}$	$\frac{4}{8}$	...
5	$\frac{5}{1}$	$\frac{5}{2}$	$\frac{5}{3}$	$\frac{5}{4}$	$\frac{5}{5}$	$\frac{5}{6}$	$\frac{5}{7}$	$\frac{5}{8}$	...
6	$\frac{6}{1}$	$\cancel{\frac{6}{2}}$	$\cancel{\frac{6}{3}}$	$\frac{6}{4}$	$\frac{6}{5}$	$\frac{6}{6}$	$\frac{6}{7}$	$\frac{6}{8}$	...
7	$\frac{7}{1}$	$\frac{7}{2}$	$\frac{7}{3}$	$\frac{7}{4}$	$\frac{7}{5}$	$\frac{7}{6}$	$\frac{7}{7}$	$\frac{7}{8}$	...
8	$\frac{8}{1}$	$\frac{8}{2}$	$\frac{8}{3}$	$\frac{8}{4}$	$\frac{8}{5}$	$\frac{8}{6}$	$\frac{8}{7}$	$\frac{8}{8}$	...
⋮	⋮								

- it's countable because we can line up its elements in order and count them
- Why do we do it in this wierd zig-zag fashion?

Portland State UNIVERSITY

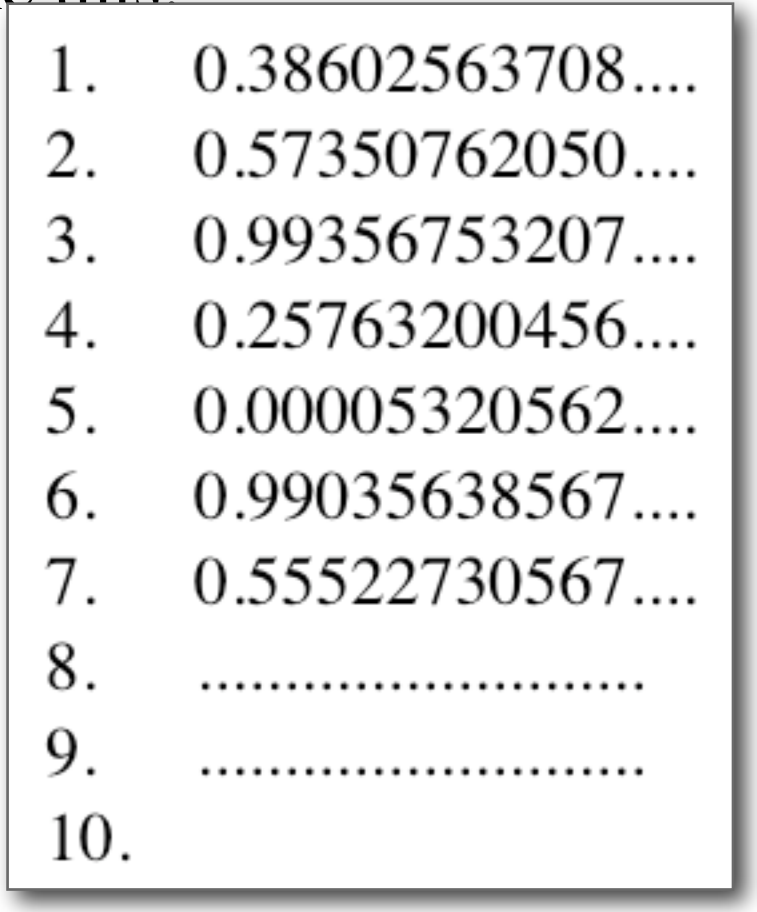
Uncountability of $\mathbb{R}$

To Prove: *The set of real numbers* $\{x \in \mathbb{R} \mid 0 < x < 1\}$, *is uncountable*.

Proof by contradiction: Suppose that the set of reals is countable. Then we can claim to make an enumeration of all of them, and that it looks something like this:

1. 0.38602563708....
2. 0.57350762050....
3. 0.99356753207....
4. 0.25763200456....
5. 0.00005320562....
6. 0.99035638567....
7. 0.55522730567....
8. ....
9. ....
10.

Now we claim to have listed every decimal between 0 and 1. But you can always give me a decimal which is not in my table! Do it like this:



1.	0.38602563708....
2.	0.57350762050....
3.	0.99356753207....
4.	0.25763200456....
5.	0.00005320562....
6.	0.99035638567....
7.	0.55522730567....
8.	....
9.	....
10.	

1. Take the first digit = 7 (not 3); Decimal = 0.7.....
2. Take the second digit = 3 (not 7); Decimal = 0.73.....
3. Take the third digit = 6 (not 3); Decimal = 0.736.....
4. Take the fourth digit = 7 (not 6); Decimal = 0.7367.....
5. Take the fifth digit = 4 (not 5); Decimal = 0.73674.....
6. Take the sixth = 4 (not 6); Decimal = 0.736744.....
7. .... 0.736744?.....
8. .... 0.736744??.....
9. ....
10.

The rule is: make sure that the k^{th} digit of the new decimal is not equal to the k^{th} digit of the k^{th} number in my original list. (We avoid using 9 and 0.)

Complete the Argument

- If the author of the enumeration says: the number that you wrote is already in my list, at position 153, you can say:
- No! My number differs from that in the 153rd decimal digit.
- So, the number you wrote in not in the list.
- Therefore, the list was not a complete enumeration of all the real numbers.
- But we can repeat this argument for *any* supposed enumeration
- So there is *no effective enumeration* of $\mathbb{R}$

Uncomputability

- Is there anything a computer **cannot** do, given sufficient time, memory, etc.?
- **Yes**, and we can prove it!
    - Informally today
        - By a counting argument
        - By showing a particular uncomputable problem
    - More formally in the remainder of the course

More countability facts

- For any alphabet Σ, the set of all **finite** strings Σ^* over that alphabet is countable
  - E.g., if $\Sigma = \{a,b,c\}$, then $\Sigma^* = \{\varepsilon,a,b,c,aa,ab,ac,ba,bb,bc,ca,cb,cc,aaa,...\}$
- But the set of **infinite** strings over even the very simple alphabet $\{0,1\}$ is uncountable
  - By simple modification of **diagonalization** argument for reals

So many problems...

- Each language A over Σ corresponds to a unique infinite string X_A over $\{0,1\}$, its **characteristic sequence**:
  - Enumerate Σ^* as $\{s_1,s_2,s_3,...\}$
  - Define $X_i = 1$ if $s_i \in A$, $X_i = 0$ otherwise
  - $X_A = X_1X_2X_3...$
- So the set of all languages over Σ is uncountable
- Since each language corresponds to a problem, there are an **uncountable** number of problems

Portland State UNIVERSITY

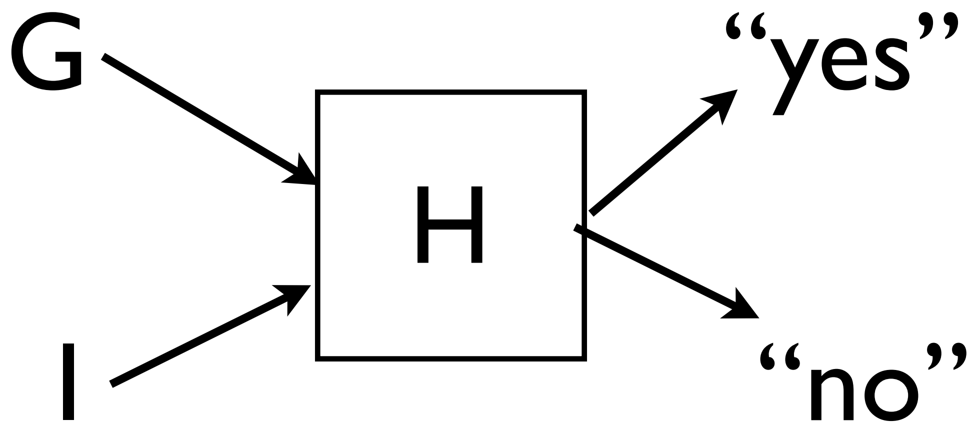
... so few solutions

- Let’s assume that every algorithm can be written down as a finite sequence of symbols over a fixed alphabet
  - For example as a computer program written in ASCII
- Then the number of algorithms is **countable**
- Since there are uncountably many decision problems and only countably many algorithms, there must be some decision problems for which there is no algorithm!
  - Indeed, uncountably many of them...

A specific uncomputable problem

- Suppose we'd like to write a program H that behaves as follows:
    - H reads as input the text of another program G and an input string I.
    - H answers the following question: when G is run on input I, does it print the string "Hello World!" ?
    - If so, H prints the string "yes"; otherwise it prints "no"
    - (Assume all programs are written in the C language; it doesn't really matter.)

Graphically...



Examples of how H works

- For example, if H is given the program G =

```
void main() {
    if (getc() == ‘a’) printf(“Hello world!\n”);
}
```

  it should surely output “yes” if the input I begins with an ‘a’ and “no” otherwise

- Of course, sometimes it will be harder, e.g., G =

```
void main() {
  if (foo(bar(baz(...)))  /* will have to look at these */
    printf(“Hello world!\n”);
}
```

Another H example

- Still worse, we might give it something like G =

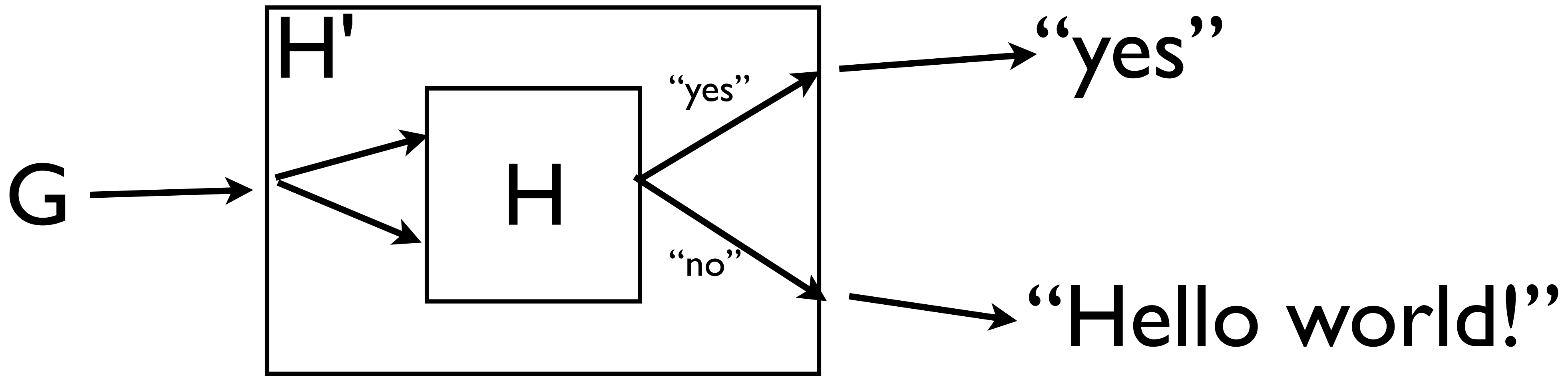
```
void main() {
  int n, total, x,y,z;
  scanf("%d",&n);  /* read n from standard input */
  for (total = 3;; total++) {
     for (x = 1; x <= total-2; x++)
        for (y = 1; y <= total-x-1; y++) {
          z = total - x - y;
          if (pow(x,n) + pow(y,n) == pow(z,n))
            printf("Hello,world!\n"); }}}
```

- Now G has to be at least as smart as 300+ years of human mathematicians
  - from Fermat (1637) to Wiles (1995)
- But still, why not?

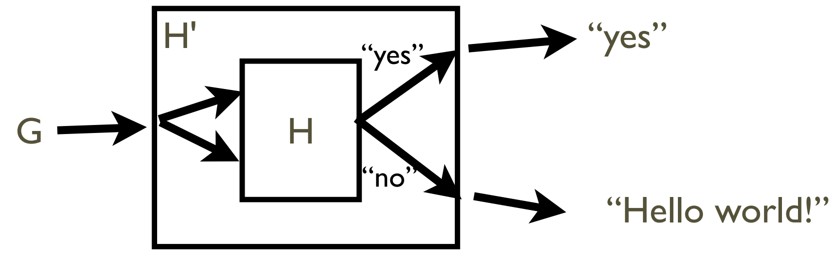
Why H is impossible

- Let’s assume we have H, and show that this leads to an impossibility.
- If we have H, we can easily modify it in two ways, producing a new program H'.
    - Instead of printing “no”, H' prints “Hello world!”
    - Instead of reading both G and I as input, it uses the same input G twice — as program to test **and** as input for that program

Graphically



Self-application



- Now, what will H' do if it is given **itself** as input?
- If it would print "Hello, world!" then it prints "yes" — oops!
- If it would not print "Hello, world!" then it prints "Hello, world!" — oops!
- Paradoxical situation!
- So H' can't exist. So H can't exist either.

Complaints?

- Not a very **realistic** problem?
  - Once we have one unsolvable problem about programs, we can easily produce others. In fact, every non-trivial property of programs is undecidable
- The argument was too **vague**?
  - We'll spend most of the course learning to be more careful
- So there's no C program. But does that mean there's no **algorithm**?
  - Can't **prove** this, but we'll give convincing evidence for the so-called Church-Turing Hypothesis