Announcements:

- PS #6 due Friday December 2, 11:59PM
    - Submissions open until grading starts, as usual
    - No earlier than Monday night 11:59PM
- FINAL EXAM on Monday December 12
- Tournament and review session on Sunday December 11
    - Time and place to be announced

- What have we covered in CS3110?
- Tools for solving difficult computational problems
    - Abstraction, specification, design
    - Functional programming
    - Concurrency
    - Reasoning about programs
    - Data structures and algorithms

- My personal view of computer scientists versus computer programmers
    - Note that there are 100x as many programmers
- At any time there are some existing programs
- And some programs that don't exist but clearly could
    - Example: problem set (before anyone solves it)
    - Ukrainian spellchecker for Android

- Computer programmers write such programs
- This can be hard work, and well paid
- Always clear that such a program exists,
    - but not necessarily trivial to write it within resource constraints (programmer time, running time/space)

- Computer scientists expand the set of programs we know how to write
- Write programs whose existence is not at all clear
    - Can we make a car that drives itself?
    - Distinguish pictures of cats from dogs?
    - Find broken bones in x-ray images?
    - Create synthetic pictures that look as good as real ones?

- Sometimes we fail
    - Quite often, in fact
    - “If you aren’t occasionally failing, then you are working on problems that are too easy.”
- Sometimes we discover that a problem is fundamentally hard
    - It wasn’t just that the person who tried it wasn’t smart enough
- This is the topic of our final lecture

We're going to show that the set of all programs is *countable*.

DEFINITION:

We'll say that two sets A and B are *the same size* if there is an exact pairing between them; that is, if there is a set R of pairs (a b) such that every element of A occurs on the left-hand side of *exactly one* pair in R and every element of B occurs on the right-hand side of *exactly one* pair in R.

Example: the sets {0,1,2} and {2,4,6} are the same size because we can pair them up as follows: (0 2), (1 4), (2 6). This definition goes for infinite sets as well.

A set S is **countable** if it is the same size as the natural numbers N = {0,1,2,...}; that is, if there is a way to pair the elements of S with 0,1,2,...

Do you think this should be possible for every infinite set? It's not!

Countable sets are all the same size; their "size" is the size of N, countably infinite, $\aleph_0$. This is the smallest infinite size. There are strictly larger infinite sets, as we'll see.

So, a set S is countable if the elements can be listed out: s0, s1, s2,...

- For example, N is countable: take the identity pairing (n n)
- The set of integers Z = {...,-3,-2,-1,0,1,2,3,...} is countable:
    - pair n with 2n, -n with 2n+1.

Wait a minute. Something's weird. N is a *proper subset* of Z. So how can it be the same size?

But it is. You can pair up the elements of Z with the elements of N exactly.

The even numbers E = {0,2,4,6,8,...} are countable: pair n with 2n. Again, E is a proper subset of N, so how can it be the same size? But it is.

The *rationals* are countable.

1/1	1/2	1/3	1/4	1/5
2/1	2/2	2/3	2/4	2/5
3/1	3/2	3/3	3/4	3/5
4/1	4/2	4/3	4/4	4/5
5/1	5/2	5/3	5/4	5/5

Number these in a diagonal zigzag:

1	2	4	6	10
3	*	7	*	
5	8	*		
9	*			
11				

etc... Skip over duplicates (marked in the table above with "*").

There are many programs (in any language, or all of them!):

* A program is a finite string of ASCII characters.

* We can list out all the finite ASCII strings.

0 - empty string strings of length 0

1 - a strings of length 1

2 - b .

... .

26 - z .

27 - aa strings of length 2

28 - ab .

etc. .

* Now, not all of these are legal programs, but all legal programs are in this list.

* We can recognize the legal OCaml programs and skip over the others.

* So, there are (only) countably many OCaml programs.

So far it looks like everything is countable (we can put any set in 1-1 correspondence with N, or with a subset of N).

This is true for any set whose elements are finite – this is basically what the above argument about programs showed.

But it's easy to show that real numbers in [0,1) are not countable. A real number can be thought of as a function f from N to {0,1,…,9}, where f(m) is the m^{th} digit. Example: π-3 = f, where f(0) = 1, f(1) = 4, f(2) = 1, f(3) = 5, f(4) = 9, etc.

But functions from N to {0,1,…,9} are not countable. It's easiest to consider simpler functions from N to {0,1} (binary representation of a real).

If these functions were countable we could list them in a table, where each function is a row. Let's call the first function f0, the next one f1, etc.

	inputs									
	0	1	2	3	4	5	6	7	8	9 ...
f0	#f	#t	#f	#t	#f	#t	#f	#t	#f	#t ...
f1	#f	#f	#t	#t	#f	#t	#f	#t	#f	#f ...
f2	#t	#f	#t	#f	#t	#f	#t	#f	#t	#f ...
f3	#f	#f	#f	#f	#t	#f	#f	#f	#f	#f ...
f4	#f	#t	#f	#f	#t	#t	#t	#f	#f	#t ...

But we can easily create a function that isn't in this table. It simply differs from fm in its response to the input m.

This argument is called diagonalization, and is one of the greatest mathematical ideas of all time. Cantor, Godel, Russel, Turing, etc!

All the following sets are all the same size (they can be put into one-to-one correspondence with each other), and they are all uncountable:

- all Boolean valued functions of one argument
- all infinite binary strings, e.g. 01101001010010...

  (0 in position n if f(n)=#f, 1 in position n if f(n)=#t)
- all real numbers in the interval [0,1]

  (take the binary expansion .01101001010010...)

  (there are some duplicates here, e.g. .000111111... and .001000000....

  but only countably many)

  (and uncountable minus countable is still uncountable)
- all paths in the infinite complete binary tree

  (0=go left, 1=go right)
- all subsets of N

- Boolean-valued functions (true/false) are generally pretty easy to write programs for. Examples: prime, even, etc.
- By diagonalization there are more of these than programs
    - So there must be ones for which there is no program!
    - In fact, this is true for (almost) all of them
- Consider the following question: does a function of one argument terminate or run forever, given this input?
    - halts(f,a) will be true or false depending on if f(a) halts
    - **Boolean-valued** function
- Note that we aren't going to write in OCaml because types get in the way
- Now consider a new **Boolean-valued** function safely(g)
    - First we check if halts(g,g), and if so we return not(g(g))
    - Otherwise we just return false
    - In pseudocode (NOT in ML) we have

```
safely(g) = if halts(g,g) then not(g,g) else false
```

    - Ignoring type checking you can do things like:

```
safely(fun(f)->f(24) != 42)
```

- OK, now what is the value of safely(safely) ?
- It's the value of not(safely(safely)). Oops!

- Resolution: you can't have a function like halts.
- In any language, no matter how smart you are.
- Determining whether or not a program halts is undecidable
- The only way you can figure out what a program does it to run it!

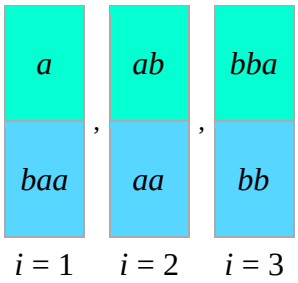
- This has huge real-life consequences.
    - Microsoft design of plug-ins (requiring burglars to sign in)
    - Virtualization
    - Virus issues

- Computer scientists tend to informally say that all programming languages are the same,
    - i.e. anything you can do in one language you can do in another
- There is a mathematically precise way to express this
    - Turing equivalence, see CS3810
    - Taught by John Hopcroft, Turing-award winner
- Weaker languages can actually be better
    - PDF versus postscript

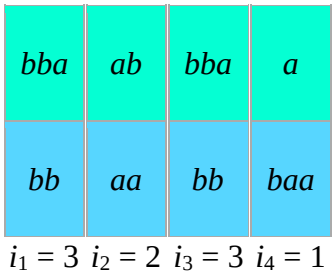
- How do you tell if a problem is undecidable?
- It's not always obvious, though there is one great (sound) heuristic

- Consider the following child's game:
    - We are given types of blocks over symbols, such as a,b,c
    - Infinite set of blocks of each type
    - Find a sequence of blocks so that the top symbols and the bottom ones are the same

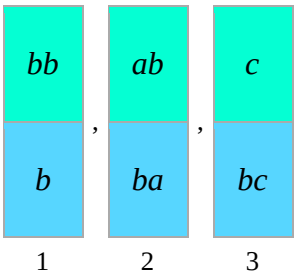
- Example 1:



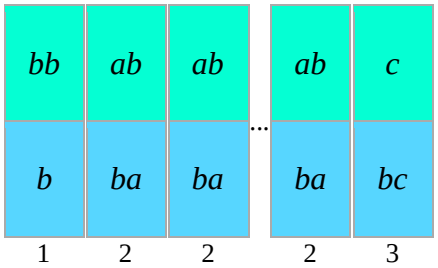
- Solution: 3,2,3,1



- Example 2:



- Solution: 1, any number of 2, 3



- Can we write a program to solve this? It depends!
- For a binary alphabet, it is decidable (first example)
- For an alphabet with 7 or more characters it is undecidable
- For 3 (second example) or more characters it is unknown!

- Suppose we can use no more than k blocks (including copies). Is it decidable?
- Yes – it is finite!
- But it is actually NP-hard, so can't do better than brute force

- Another example: Wang tiling of the plane