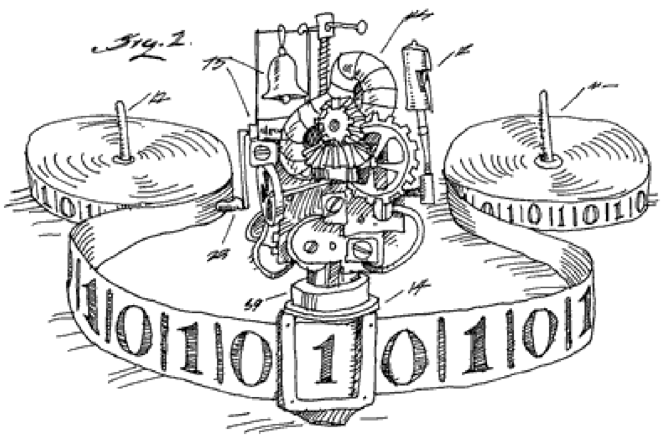
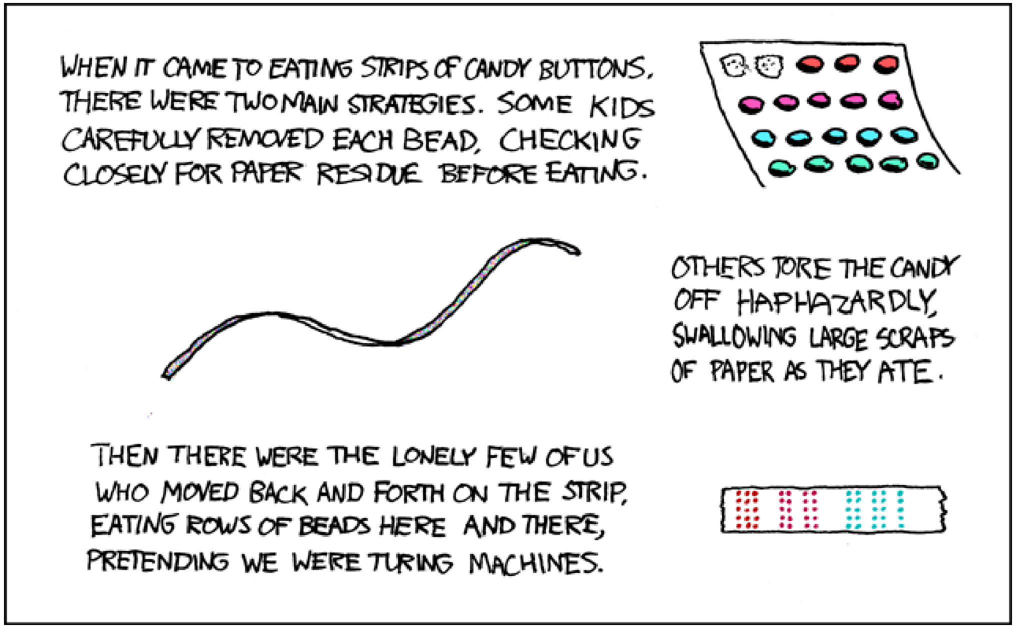
cse 311: foundations of computing

Fall 2015

Lecture 29: Reductions and Turing machines





the Halting problem is undecidable

Given: - CODE(**P**) for any program **P**

- input **x**

Output: **true** if **P** halts on input **x**

false if **P** does not halt on input **x**

(or CODE(**P**) is not a valid program)

It isn't possible to write a program that solves the Halting Problem.

Does D(CODE(D)) halt?

```
public static void D(x) {
    if (H(x,x) == true) {
        while (true); /* don't halt */
    }
    else {
        return;       /*   halt   */
    }
}
```

H solves the halting problem implies that
H(CODE(D),x) is **true** iff D(x) halts, H(CODE(D),x) is **false** iff not

Suppose D(CODE(D)) **halts**.
Then, we must be in the **second** case of the if.
So, H(CODE(D), CODE(D)) is **false**
Which means D(CODE(D)) **doesn't halt**

Suppose D(CODE(D)) **doesn't halt**.
Then, we must be in the **first** case of the if.
So, H(CODE(D), CODE(D)) is **true**.
Which means D(CODE(D)) **halts**.

Contradiction!

proving that a language is undecidable

Consider a language $L \subseteq \Sigma^*$

We say that L is **undecidable** if there is no Java program that takes $x \in \Sigma^*$ as input and outputs

true	if $x \in L$
false	otherwise

HALTING = { $(C, x) : C = \text{CODE}(P)$ and P halts on input x }

Theorem: HALTING is **undecidable**.

Strategy: To show that some other language L is undecidable, we could show that if we could decide L, we could also decide HALTING. Since HALTING is undecidable, it must be that L is also undecidable!

a useful tool: hardcoding

Define the language:

HaltsNoInput

$= \{\, P : P$ is a program that halts when run with no input $\}$

Goal: Show that if we could decide HaltsNoInput, we could also decide HALTING.

a useful tool: hardcoding

HaltsNoInput

$= \{ P : P$ is a program that halts when run with no input $\}$

equivalence of programs is undecidable

EQUIV $= \{\ (P, Q)$: programs P and Q have same behavior on every input $\}$

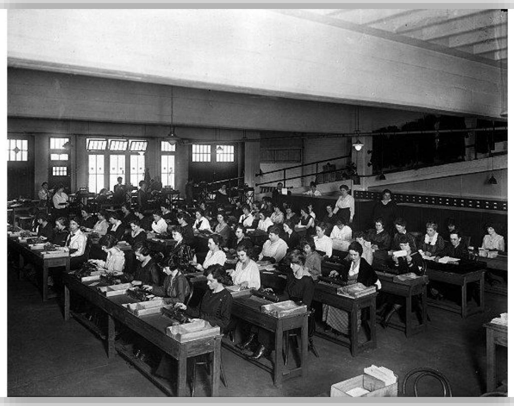
division by zero

$\text{DivByZero} = \{ (Q, x) : Q \text{ attempts to divide by 0 when run on input } x \}$

computers and algorithms

- Does Java (or any programming language) cover all possible computation? Every possible algorithm?

- There was a time when computers were people who did calculations on sheets paper to solve computational problems



- Computers as we known them arose from trying to understand everything these people could do.

1930's:

How can we formalize what algorithms are possible?

- **Turing machines** (Turing, Post)
  - basis of modern computers
- **Lambda Calculus** (Church)
  - basis for functional programming
- **μ-recursive functions** (Kleene)
  - alternative functional programming basis

Church-Turing Thesis:

Any reasonable model of computation that includes all possible algorithms is equivalent in power to a Turing machine

Evidence

- Intuitive justification
- Huge numbers of equivalent models to TM's based on radically different ideas

Turing machines

- Finite Control
  - Brain/CPU that has only a finite # of possible “states of mind”
- Recording medium
  - An unlimited supply of blank “scratch paper” on which to write & read symbols, each chosen from a finite set of possibilities
  - Input also supplied on the scratch paper
- Focus of attention
  - Finite control can only focus on a small portion of the recording medium at once
  - Focus of attention can only shift a small amount at a time

Turing machines

- Recording medium
  - An infinite read/write “tape” marked off into cells
  - Each cell can store one symbol or be “blank”
  - Tape is initially all blank except a few cells of the tape containing the input string
  - Read/write head can scan one cell of the tape - starts on input

- In each step, a Turing machine
  - Reads the currently scanned symbol
  - Based on current state and scanned symbol

    Overwrites symbol in scanned cell

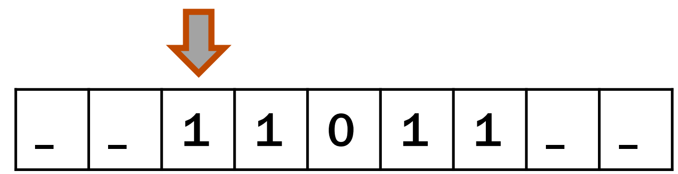
    Moves read/write head left or right one cell

    Changes to a new state

- Each Turing Machine is specified by its finite set of rules

Turing machines

	_	0	1
s_1	$(1, L, s_3)$	$(1, L, s_4)$	$(0, R, s_2)$
s_2	$(0, R, s_1)$	$(1, R, s_1)$	$(0, R, s_1)$
s_3			
s_4			



Turing machines

Ideal Java/C programs:

- Just like the Java/C you're used to programming with, except you never run out of memory
  - Constructor methods always succeed **malloc** in C never fails

Equivalent to Turing machines except a lot easier to program:

- Turing machine definition is useful for breaking computation down into simplest steps
- We only care about high level so we use programs

Turing's big idea: machines as data

Original Turing machine definition:

- A different "machine" **M** for each task
- Each machine **M** is defined by a finite set of possible operations on finite set of symbols

M has a finite description as a sequence of symbols, its "code" denoted **<M>**

You already are used to this idea with the notion of the program code or text but this was a new idea in Turing's time.

Turing's big idea: a universal TM

- A Turing machine interpreter **U**
  - On input **<M>** and its input **x**, **U** outputs the same thing as **M** does on input **x**
  - At each step it decodes which operation **M** would have performed and simulates it.
- One Turing machine is enough
  - Basis for modern stored-program computer

    Von Neumann studied Turing's UTM design

Rice's theorem ("can't tell a book by its cover")

Not *every* problem on programs is undecidable!

Which of these is decidable?

- Input CODE(P) and x
  Output: true if P prints "ERROR" on input x
  after less than 100 steps
  false otherwise
- Input CODE(P) and x
  Output: true if P prints "ERROR" on input x
  after more than 100 steps
  false otherwise

Compilers Suck Theorem (informal):

Any "non-trivial" property the **input-output behavior** of Java programs is undecidable.

takeaway

- Can't rely on the idea of improved compilers and programming languages to eliminate major programming errors
  - truly safe languages can't possibly do general computation
- Document your code
  - there is no way you can expect someone else to figure out what your program does with just your code; since in general it is provably impossible to do this!

What's next?

Foundations II: Probability, statistics, and uncertainty.

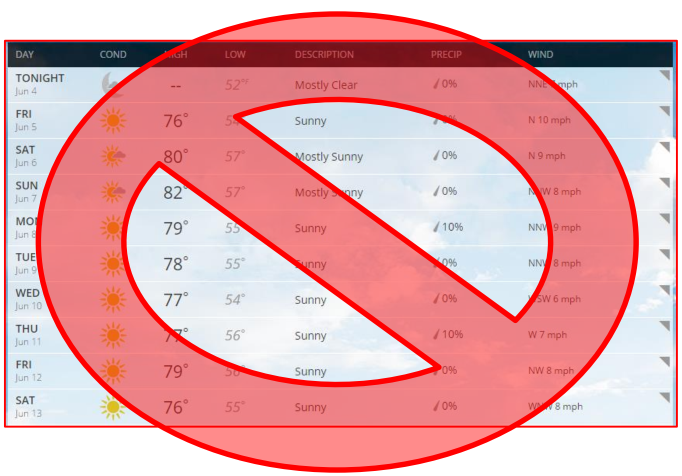
The **final exam** is Monday, Dec 14, 2015

Notes: One page of notes allowed, front and back.

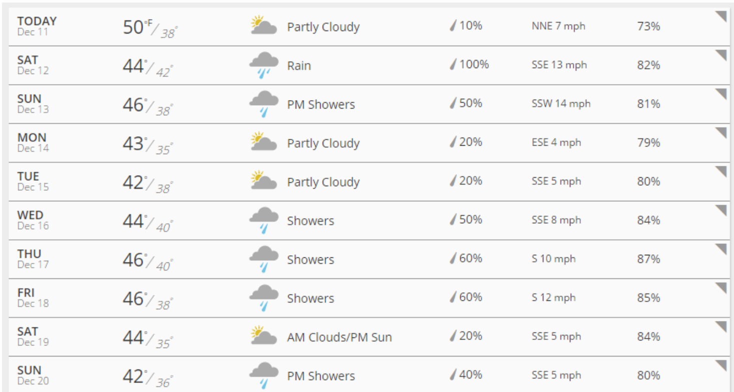
Review session:

- Sunday at 2pm in EEB 105

And then…



DAY	COND	HIGH	LOW	DESCRIPTION	PRECIP	WIND
TONIGHT Jun 4		--	52°F	Mostly Clear	0%	NNE [illegible] mph
FRI Jun 5		76°	[illegible]	Sunny	[illegible]	N 10 mph
SAT Jun 6		80°	57°	Mostly Sunny	0%	N 9 mph
SUN Jun 7		82°	57°	Mostly [illegible]	0%	[illegible] 8 mph
[illegible]		79°	[illegible]	Sunny	10%	[illegible] 9 mph
TUE Jun 9		78°	55°	[illegible]	[illegible]	[illegible] 8 mph
WED Jun 10		77°	54°	Sunny	0%	[illegible] 6 mph
THU Jun 11		[illegible]	56°	Sunny	10%	W 7 mph
FRI Jun 12		79°	[illegible]	Sunny	0%	NW 8 mph
SAT Jun 13		76°	55°	Sunny	0%	[illegible] 8 mph



TODAY Dec 11	50°F/38°	Partly Cloudy	10%	NNE 7 mph	73%
SAT Dec 12	44°/42°	Rain	100%	SSE 13 mph	82%
SUN Dec 13	46°/38°	PM Showers	50%	SSW 14 mph	81%
MON Dec 14	43°/35°	Partly Cloudy	20%	ESE 4 mph	79%
TUE Dec 15	42°/38°	Partly Cloudy	20%	SSE 5 mph	80%
WED Dec 16	44°/40°	Showers	50%	SSE 8 mph	84%
THU Dec 17	46°/40°	Showers	60%	S 10 mph	87%
FRI Dec 18	46°/38°	Showers	60%	S 12 mph	85%
SAT Dec 19	44°/35°	AM Clouds/PM Sun	20%	SSE 5 mph	84%
SUN Dec 20	42°/36°	PM Showers	40%	SSE 5 mph	80%