Announcements

- Reading
  - 7th edition: p. 201 and 13.5
  - 6th edition: p. 177 and 12.5

- Topic list and sample final exam problems have been posted

- Final exam, Monday, June 10
  - 2:30-4:20 pm MGH 389.

Last lecture highlights

- Cardinality
- A set $S$ is *countable* iff we can write it as $S = \{s_1, s_2, s_3, \ldots\}$ indexed by $\mathbb{N}$
- Set of rationals is countable
  - “dovetailing”
- $\Sigma^*$ is countable
  - $\{0,1\}^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots\}$
- Set of all (Java) programs is countable

Last lecture highlights

- The set of real numbers is not countable
  - “diagonalization”
  - $r_1^{D=0. \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad \ldots}$
  - $r_2 \quad 0 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad \ldots$
  - $r_3 \quad 0 \quad 1 \quad 4 \quad 5 \quad 2 \quad 8 \quad 5 \quad 7 \quad 1 \quad 4 \quad \ldots$
  - $r_4 \quad 0 \quad 1 \quad 4 \quad 1 \quad 5 \quad 1 \quad 9 \quad 2 \quad 6 \quad 5 \quad \ldots$
  - $r_5 \quad 0 \quad 1 \quad 2 \quad 1 \quad 2 \quad 5 \quad 1 \quad 2 \quad 2 \quad \ldots$
  - $r_6 \quad 0 \quad 2 \quad 5 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \ldots$
  - $r_7 \quad 0 \quad 7 \quad 1 \quad 8 \quad 2 \quad 8 \quad 1 \quad 8 \quad 5 \quad 2 \quad \ldots$
  - $r_8 \quad 0 \quad 6 \quad 1 \quad 8 \quad 0 \quad 3 \quad 3 \quad 9 \quad 4 \quad 5 \quad \ldots$
  - $\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots$
- Why doesn’t this show that the rationals aren’t countable?
Last lecture highlights

• There exist functions that cannot be computed by any program
  – The set of all functions $f : \mathbb{N} \to \{0,1,...,9\}$ is not countable
  – The set of all (Java/C/C++) programs is countable
  – So there are simply more functions than programs

Do we care?

• Are any of these functions, ones that we would actually want to compute?
  – The argument does not even give any example of something that can’t be done, it just says that such an example exists
• We haven’t used much of anything about what computers (programs or people) can do
  – Once we figure that out, we’ll be able to show that some of these functions are really important

Before Java...more from our Brief History of Reasoning

• 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
    • $\mu$-recursive functions (Kleene)
      – alternative functional programming basis

Turing Machines

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
Components of Turing’s Intuitive Model of Computers

- **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

What is a Turing Machine?

- **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 state of mind 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

Sample Turing Machine

<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>(1,$s_2$)</td>
<td>(1,$s_3$)</td>
<td>(0,$s_1$)</td>
</tr>
<tr>
<td>$s_2$</td>
<td>(H,$s_1$)</td>
<td>(R,$s_1$)</td>
<td>(R,$s_1$)</td>
</tr>
<tr>
<td>$s_3$</td>
<td>(H,$s_3$)</td>
<td>(R,$s_3$)</td>
<td>(R,$s_3$)</td>
</tr>
</tbody>
</table>
What is a Turing Machine?

Turing Machine ≡ Ideal Java/C Program

• Ideal C/C++/Java programs
  – Just like the C/C++/Java you’re used to programming with, except you never run out of memory
    • constructor methods always succeed
    • malloc 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 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”
• You already are used to this idea:
  – We’ll write $\langle P \rangle$ for the code of program $P$
  – i.e. $\langle P \rangle$ is the program text as a sequence of ASCII symbols and $P$ is what actually executes

Turing’s Idea: A Universal Turing Machine

• A Turing machine interpreter $U$
  – On input $\langle P \rangle$ and its input $x$, $U$ outputs the same thing as $P$ does on input $x$
  – At each step it decodes which operation $P$ would have performed and simulates it.
• One Turing machine is enough
  – Basis for modern stored-program computer
    • Von Neumann studied Turing’s UTM design
Halting Problem

• **Given:** the code of a program \( P \) and an input \( x \) for \( P \), i.e. given \( (\langle P \rangle, x) \)
• **Output:** 1 if \( P \) halts on input \( x \)  
  0 if \( P \) does not halt on input \( x \)

**Theorem** (Turing): There is no program that solves the halting problem  
“The halting problem is undecidable”

Proof by contradiction

• Suppose that \( H \) is a Turing machine that solves the Halting problem

  **Function** \( D(x) \):
  
  - if \( H(x,x)=1 \) then  
    - while (true); /* loop forever */
  - else  
    - no-op; /* do nothing and halt */
  - endif

• What does \( D \) do on input \( \langle D \rangle \)?  
  – Does it halt?

**That’s it!**

• We proved that there is no computer program that can solve the Halting Problem.

• This tells us that there is no compiler that can check our programs and guarantee to find any infinite loops they might have

---

**Does \( D \) halt on input \( \langle D \rangle \)?**

\( D \) halts on input \( \langle D \rangle \)

\[ \iff H \text{ outputs 1 on input } (\langle D \rangle, \langle D \rangle) \]

[since \( H \) solves the halting problem and so \( H(\langle D \rangle, x) \) outputs 1 iff \( D \) halts on input \( x \)]

\[ \iff D \text{ runs forever on input } \langle D \rangle \]

[since \( D \) goes into an infinite loop on \( x \) iff \( H(x,x)=1 \)]