Announcements

• Reading
  – 7th edition: p. 201 and 13.5
  – 6th edition: p. 177 and 12.5
  – 5th edition: p. 222 and 11.5
• Topic list and sample final exam problems have been posted
• Comprehensive final, roughly 67% of material post midterm
• Review session, Saturday, December 8, 10 am – noon, EEB 125
• Final exam, Monday, December 10
  – 2:30-4:20 pm or 4:30-6:20 pm, Kane 220.

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
    – $\{0,1\}^* = \{0,1,00,10,11,000,001,010,011,100,101,\ldots\}$
  • Set of all (Java) programs is countable

Last lecture highlights

• The set of real numbers is not countable
  – “diagonalization”
  \[
  \begin{array}{cccccccccccc}
  0 & . & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots \\
  r_1 & 0 & 1 & 4 & 2 & 5 & 8 & 5 & 7 & 1 & 4 & \ldots \\
  r_2 & 0 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & \ldots \\
  r_3 & 0 & 1 & 4 & 1 & 5 & 1 & 2 & 6 & 5 & \ldots \\
  r_4 & 0 & 1 & 2 & 1 & 2 & 2 & 2 & 2 & 2 & \ldots \\
  r_5 & 0 & 2 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots \\
  r_6 & 0 & 7 & 1 & 8 & 2 & 2 & 1 & 8 & 5 & \ldots \\
  r_7 & 0 & 6 & 1 & 8 & 0 & 3 & 3 & 3 & 9 & \ldots \\
  \end{array}
  \]
  – Why doesn’t this show that the rationals aren’t countable?

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
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>s1</th>
<th>s2</th>
<th>s3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

What is a Turing Machine?

- Components of Turing’s Intuitive Model of Computers
- Turing Machines
- Church-Turing Thesis
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 $<P>$ for the code of program $P$
  - i.e. $<P>$ is the program text as a sequence of ASCII symbols and $P$ is what actually executes

Halting Problem

- Given: the code of a program $P$ and an input $x$ for $P$, i.e. given $(<P>,x)$
- 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 $<D>$?
  - Does it halt?

Does $D$ halt on input $<D>$?

$D$ halts on input $<D>$

$\Leftrightarrow$ $H$ outputs 1 on input $(<D>,<D>)$

[since $H$ solves the halting problem and so $H(<D>,x)$ outputs 1 iff $D$ halts on input $x$]

$\Leftrightarrow$ $D$ runs forever on input $<D>$

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