Announcements

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

• My office hours this week
  – Usual: today immediately after class until 2:50pm
  – Extra office hour: Thursday 11-12

• Homework 8 due Friday
  – Solutions available Friday night-Saturday online on password-protected page

• Final Exam, Monday, June 10, 2:30-4:20 pm MGH 389
  – Topic list and sample final exam problems have been posted
  – Comprehensive final, closed book, closed notes
  – Review session, Sunday, June 9, 4:00 pm EEB 125

Last lecture highlights

Turing machine = Finite control + Recording Medium + Focus of attention

Finite Control:

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>(1,$s_2$)</td>
</tr>
<tr>
<td>$s_2$</td>
<td>(H,$s_3$)</td>
</tr>
<tr>
<td>$s_3$</td>
<td>(H,$s_3$)</td>
</tr>
</tbody>
</table>

Recording Medium

<table>
<thead>
<tr>
<th>Tape Symbol</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ _ 0 0 1 0 0 1 _</td>
<td></td>
</tr>
</tbody>
</table>

input $x$

output

output $P(x)$

• The Universal Turing Machine $U$
  – Takes as input: $(<P,x>)$ where $<P>$ is the code of a program and $x$ is an input string
  – Simulates $P$ on input $x$

• Same as a Program Interpreter
**Last lecture highlights**

Program P

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>(1, $s_2$)</td>
<td>(0, $s_3$)</td>
</tr>
<tr>
<td>$s_2$</td>
<td>(H, $s_3$)</td>
<td>(R, $s_1$)</td>
</tr>
<tr>
<td>$s_3$</td>
<td>(H, $s_2$)</td>
<td>(R, $s_3$)</td>
</tr>
</tbody>
</table>

Universal TM U

Program code $P$:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Programs about Program Properties**

- The Universal TM takes a program code $<P>$ as input, and an input $x$, and interprets $P$ on $x$
  - Step by step by step by step...
- Can we write a TM that takes a program code $<P>$ as input and checks some property of the program?
  - Does $P$ ever return the output “ERROR”?
  - Does $P$ always return the output “ERROR”?
  - Does $P$ halt on input $x$?

**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 \(<D>\)?

\[
\begin{align*}
\text{Function } D(x): \\
&\quad \text{if } H(x,x) = 1 \text{ then} \\
&\quad \quad \text{while (true); /* loop forever */} \\
&\quad \text{else} \\
&\quad \quad \text{no-op; /* do nothing and halt */} \\
&\quad \text{endif}
\end{align*}
\]

\( D \) halts on input \(<D>\)

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

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

\[\iff D \text{ runs forever on input } <D>\]

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

---

**SCOOPING THE LOOP SNOOPER**

A proof that the Halting Problem is undecidable

by Geoffrey K. Pullum (U. Edinburgh)

*No general procedure for bug checks succeeds.*

Now, I won’t just assert that, I’ll show where it leads:
I will prove that although you might work till you drop,
you cannot tell if computation will stop.

For imagine we have a procedure called \( P \)
that for specified input permits you to see
whether specified source code, with all of its faults,
defines a routine that eventually halts.

You feed in your program, with suitable data,
and \( P \) gets to work, and a little while later
(in finite compute time) correctly infers
whether infinite looping behavior occurs...

---

**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

---

SCOOPING THE LOOP SNOOPER

... Here’s the trick that I’ll use -- and it’s simple to do.
I’ll define a procedure, which I will call \( Q \),
that will use \( P \)’s predictions of halting success
to stir up a terrible logical mess.

... And this program called \( Q \) wouldn’t stay on the shelf;
I would ask it to forecast its run on itself.
When it reads its own source code, just what will it do?
What’s the looping behavior of \( Q \) run on \( Q \)?

... Full poem at: [http://www.lel.ed.ac.uk/~gpullum/loopsnoop.html](http://www.lel.ed.ac.uk/~gpullum/loopsnoop.html)
Another view of the proof undecidability of the Halting Problem

- Suppose that there is a program $H$ that computes the answer to the Halting Problem.
- We will build a table with a row for each program (just like we did for uncountability of reals).
- If the supposed program $H$ exists then the $D$ program we constructed as before will exist and so be in the table.
- But $D$ must have entries like the “flipped diagonal”
  - $D$ can’t possibly be in the table.
  - Only assumption was that $H$ exists. That must be false.

(P, x) entry is 1 if program P halts on input x and 0 if it runs forever.
Recall: Code for $D$ assuming subroutine $H$ 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

- If $D$ existed it would have a row different from every row of the table
- $D$ can’t be a program so $H$ cannot exist!

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
- The full story is even worse

The “Always Halting” problem

- **Given:** $<Q>$, the code of a program $Q$
- **Output:** 1 if $Q$ halts on every input 0 if not.

**Claim:** the “always halts” problem is undecidable

**Proof idea:**
- Show we could solve the Halting Problem if we had a solution for the “always halts” problem.
- No program solving for Halting Problem exists $\Rightarrow$ no program solving the “always halts” problem exists
The “Always Halting” problem

Suppose we had a TM A for the Always Halting problem

The “Always ERROR” problem

• Given: <R>, the code of a program R
• Output: 1 if R always prints ERROR
  0 if R does not always print ERROR

Pitfalls

• Not every problem on programs is undecidable!
  Which of these is decidable?
• Input <P> and x
  Output: 1 if P prints “ERROR” on x after less than 100 steps
  0 otherwise
• Input <P> and x
  Output: 1 if P prints “ERROR” on x after more than 100 steps
  0 otherwise