Projects Schedule

• By Tuesday, October 18: Project “abstract” due

• By Thursday October 20: Feedback from me

• Week of October 24: Present project abstract to class

• Month of November: Time in class for help on projects

• December 9: Final paper due
Brainstorming on projects
Computation

Motivating questions:

• What does “computation” mean?

• What are the similarities and differences between computation in computers and in natural systems?

• What are the limits of computation? Are there things that cannot be “computed”?
Some history...

Hilbert’s problems (1900)
  – Is mathematics complete?
  – Is mathematics consistent?
  – Is mathematics decidable?

David Hilbert, 1862 - 1943
Some history...

Hilbert’s problems (1900)

– Is mathematics complete?
   Can every mathematical statement be proved or disproved from a finite set of axioms?

– Is mathematics consistent?

– Is mathematics decidable?
Some history...

Hilbert’s problems (1900)

– Is mathematics complete?
Can every mathematical statement be proved or disproved from a finite set of axioms?

– Is mathematics consistent?
Can only the true statements be proved?

– Is mathematics decidable?

David Hilbert, 1862 - 1943
Some history...

Hilbert’s problems (1900)

– Is mathematics complete?
  Can every mathematical statement be proved or disproved from a finite set of axioms?

– Is mathematics consistent?
  Can only the *true* statements be proved?

– Is mathematics decidable?
  Is there a “definite procedure” that can be applied to every statement that will tell us in a finite time whether the statement is true or false?

David Hilbert, 1862 - 1943
Gödel’s incompleteness theorem: Arithmetic (and more complex systems of mathematics) must either be incomplete or inconsistent (or both)
Gödel’s incompleteness theorem: Arithmetic (and more complex systems of mathematics) must either be incomplete or inconsistent (or both)

“This statement is unprovable.”
The “Entscheidungs” (decision) problem

Is there a “definite procedure” that can be applied to every statement that will tell us in a finite time whether the statement is true or false?

What is a “definite procedure”? 
Turing machines

Rules
1...
2...
3....
...

Current state: **start**
Current symbol: 0

Tape head

<table>
<thead>
<tr>
<th>...</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>...</th>
</tr>
</thead>
</table>

Tape

Alan Turing, 1912 - 1954
Simple example: Even or Odd?
Simple example: Even or Odd Number of 1s?

Symbols: 0, 1, blank
States: start, even, odd, halt
Simple example: Even or Odd Number of 1s?

Symbols: 0, 1, blank
States: start, even, odd, halt

1. If you are in the start state and read a 0, then change to the even state, replace the 0 with a blank (i.e., erase the 0), and move one cell to the right.
2. If you are in the even state and read a 1, change to the odd state, replace the 1 with a blank, and move one cell to the right.
3. If you are in the odd state and read a 1, change to the even state, replace the 1 with a blank, and move one cell to the right.
4. If you are in the odd state and read a 0, replace that 0 with a 1 and change to the halt state.
5. If you are in the even state and read a 0, replace that 0 with a 0 (i.e., don’t change it) and change to the halt state.
Turing machine = “definite procedure”
Turing machine applet

http://math.hws.edu/TMCM/java/xTuringMachine/
Encoding a Turing Machine

Goal: encode TM rules as a binary string

—Current State—Current Symbol—New State—New Symbol—Motion—

In this shorthand, Rule 1 above would be written as:

—start—0—even—blank—right—
Encoding a Turing Machine

Goal: encode TM rules as a binary string

—Current State—Current Symbol—New State—New Symbol—Motion—

In this shorthand, Rule 1 above would be written as:

—start—0—even—blank—right—

Possible code:

<table>
<thead>
<tr>
<th>States</th>
<th>Symbols</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>start = 000</td>
<td>‘0’ = 000</td>
<td>move_right = 000</td>
</tr>
<tr>
<td>even = 001</td>
<td>‘1’ = 001</td>
<td>move_left = 111</td>
</tr>
<tr>
<td>odd = 010</td>
<td>‘blank’ = 100</td>
<td></td>
</tr>
<tr>
<td>halt = 100</td>
<td>separator (—) = 111</td>
<td></td>
</tr>
</tbody>
</table>
Encoding a Turing Machine

Goal: encode TM rules as a binary string

---

Current State—Current Symbol—New State—New Symbol—Motion---

In this shorthand, Rule 1 above would be written as:

---start—0—even—blank—right---

<table>
<thead>
<tr>
<th>States</th>
<th>Symbols</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>start = 000</td>
<td>‘0’ = 000</td>
<td>move_right = 000</td>
</tr>
<tr>
<td>even = 001</td>
<td>‘1’ = 001</td>
<td>move_left = 111</td>
</tr>
<tr>
<td>odd = 010</td>
<td>‘blank’ = 100</td>
<td></td>
</tr>
<tr>
<td>halt = 100</td>
<td>separator (—) = 111</td>
<td></td>
</tr>
</tbody>
</table>

What does Rule 1 look like using this code?
Universal Turing Machine

Special “universal” Turing machine $U$

Input on $U$’s tape: code for some other Turing machine $M$, plus input $I$ for that Turing machine

Universal TM $U$ “runs $M$ on $I$”

In other words, $U$ is simply a programmable computer!
Turing’s solution to the *Entscheidungs* problem

Recall the *Entscheidungs* problem:

Is there always a definite procedure that can decide whether a statement is true?

What Turing did:

1. Proposed a particular statement
2. Assume there exists a definite procedure (= Turing machine $H$) for deciding the truth or falsity of this statement
3. Prove that $H$ cannot exist.
Turing’s proof that H cannot exist

• Let $U(M, I)$ denote the results of having universal TM $U$ run $M$ on $I$

• Turing’s key insight: can have $I = M'$
  – That is, can have $U(M, M')$

Example of this in your day-to-day experience?

• Can also have $U(M, M)$

Example?
Turing’s proof that H cannot exist

Proof (by contradiction):

Statement: = “Turing Machine $\textbf{M}$, when run on input $\textbf{l}$, will halt after a finite number of time steps.”

(Example of machine that does not halt?)

Assume $\textbf{H}$ exists, such that $\textbf{H}(\textbf{M}, \textbf{l})$ will answer “yes” if $\textbf{M}$ halts on $\textbf{l}$; “no” if it does not, for any $\textbf{M}, \textbf{l}$. (Will later show this leads to a contradiction.)
• Problem of designing $H$ is called the “Halting Problem”.

• Not clear how to design $H$, but let’s assume it exists.

• Given $H$, let’s create $H'$ as follows:
  – $H'$ takes as input the code of a Turing machine $M$
  – $H'$ then runs $H(M,M)$
  – If $H(M,M)$ answers “yes” (i.e., “yes, $M$ halts on input $M$”), then $H'$ goes into an infinite loop.
  – If $H(M,M)$ answers “no” (i.e., “no, $M$ does not halt on input $M$”), then $H'$ halts.

• Now, Turing asked, does $H'$ halt when given $H'$ as input?
Recap

• In the 1930s, Turing formalized the notion of “definite procedure”

• This formalization opened the door for the invention of programmable computers in the 1940s. Turing machines were blueprints for the “von-Neumann architecture”.

• Turing also showed that there are limits to what can be computed.

• 19th century: all seemed possible in science and mathematics

• 20th century: limits to prediction and determinism (quantum mechanics, chaos); limits to mathematics and computation

• 21st century: ?