Lecture 20

Models of Computation II
(S&G, ch. 10)

Read S&G ch. 11
(Using & Managing Data)
for next week
Ordinary Turing Machine

- We can design a Turing machine $M$ for a specific purpose
- For each allowable input $x$ it produces the corresponding output $y$

Universal Turing Machine

- We can design a Turing machine $U$ that can emulate any Turing machine $M$
- Let $m$ be an encoding of $M$ (e.g., its rules)
- For each allowable input $x$ it produces the corresponding output $y$
Equivalence Between TMs and Other Models of Computation

• If we can use some model of computation to program a UTM, then we can emulate any TM
  – So this model is at least as powerful as TMs
• If can design TM to emulate another kind of universal machine, then UTM can emulate it
  – So other model is no more powerful than TMs
• The way to prove equivalent “power” of different models of computation
• Equivalent in terms of “computability” not space/time efficiency

General-Purpose Computers

• The Universal Turing Machine is theoretical foundation of general purpose computer
• Instead of designing a special-purpose computer for each application
• Design one general-purpose computer:
  – interprets program (virtual machine description) stored in its memory
  – emulates that virtual machine
Church-Turing Thesis

- **CT Thesis**: The set of effectively calculable problems is exactly the set of problems solvable by TMs
- Empirical evidence: All the independently designed models of computation turned out to be equivalent to TM in power
- Easy to see how any calculus can be emulated by a TM
- Easy to see how any (digital) computer can be emulated by a TM (and vice versa)
- But, there is research in non-Turing models of computation

The Limits of Computation
The Liar Paradox

- Epimenides the Cretan (7th cent. BCE) said, “The men of Crete were ever liars …”
- “If you say that you are lying, and say it truly, you are lying.” — Cicero (106–43 BCE)

“I am lying.”

Undecidability of the Halting Problem (Informal)

- Assume we have procedure **Halts** that decides halting problem for any program/input pair
- Let \( P(X) \) represent the execution of program \( P \) on input \( X \)
- **Halts** \( P, X \) = **true** if and only if program \( P \) halts on input \( X \)
- **Halts** \( P, X \) = **false** if and only if program \( P \) doesn’t halt on input \( X \)
- Program \( P \) encoded as string or other legal input to programs
Assumed Turing Machine for Halting Problem

- We can design a Turing machine \texttt{Halts} that can decide, for any Turing machine \( P \) and input \( x \), whether \( P \) halts on \( x \).
- Let \( p \) be an encoding of \( P \) (e.g., its rules).
- If \( P \) halts on \( x \):

\[ p \ x \rightarrow \text{true} \]

Assumed Turing Machine for Halting Problem (2)

- If \( P \) doesn’t halt on \( x \):

\[ p \ x \rightarrow \text{false} \]
Undecidability of the Halting Problem (2)

- Define the “paradoxical procedure” \( Q \):
  1. procedure \( Q(P) \):
  2. if \textbf{Halts}(P, P) then
  3. go into an infinite loop
  4. else // \textbf{Halts}(P, P) is false, so
  5. halt immediately
- Now \( Q \) is a program that can be applied to any program string \( P \)

Turing Machine \( Q \)

- After running TM \textbf{Halts} on \( p \) and \( p \), if result was \textbf{true}, go into an infinite loop
Turing Machine $Q$ (2)

- After running TM Halts on $p$ and $p$, if result was false, halt immediately

```
Halts

\[ Q \]

\[ \text{false} \]
```

```
Halts

\[ Q \]

\[ \text{false} \]
```

TM $Q$ Applied to $q$

- After running TM Halts on $q$ and $q$, if result was true, go into an infinite loop

```
Halts

\[ Q \]

\[ \text{true} \]
```

```
Halts

\[ Q \]

\[ 0000... \]
```
**TM Q Applied to q (2)**

- After running TM **Halts** on q and q, if result was **false**, halt immediately

- **Halts**
  - q q

- **Halts**
  - q q (false)

- **Halts**
  - Q

- **Halts**
  - Q halts!

- **Halts**
  - false

**Undecidability of the Halting Problem (3)**

- What will be the effect of executing Q (Q)?
- If **Halts** (Q, Q) = **true**, then go into an infinite loop, that is, don’t halt
  - But **Halts** (Q, Q) = **true** iff Q (Q) halts
- If **Halts** (Q, Q) = **false**, then halt immediately
  - But **Halts** (Q, Q) = **false** iff Q (Q) doesn’t halt
- So Q (Q) halts if and only if Q (Q) doesn’t halt
- A contradiction!
- Our assumption (that **Halts** exists) was false
Rice’s Theorem (Informal)

• Suppose that \( B \) is any behavior that a program might exhibit on a given input
  – examples: print a 0, open a window, delete a file, generate a beep
• Assume that we have a procedure \texttt{DoesB} \((P, X)\) that decides whether \( P (X) \) exhibits behavior \( B \)
• As in Turing’s proof, we show a contradiction

Rice’s Theorem (2)

• Define a paradoxical procedure \( Q \):
  1. procedure \( Q (P) \):
  2. if \texttt{DoesB} \((P, P)\) then
  3. \texttt{don’t do B}
  4. else
  5. \texttt{do B}
• Note that \( B \) must be a behavior that we can control
Rice’s Theorem (3)

- Consider the result of executing $Q (Q)$
- $Q (Q)$ does $B$ if and only if $Q (Q)$ doesn’t do $B$
- Contradiction shows our assumption of existence of decision procedure $\text{DoesB}$ was false
- A TM cannot decide any “controllable” behavior for all program/input combinations

Gödel’s Incompleteness Theorem (informally)

- By constructing a “paradoxical proposition” that asserts own unprovability, can prove:
- *In any system of formal logic (powerful enough to define arithmetic) there will be a true proposition that be neither proved nor disproved in that system*
- Yet by reasoning outside the system, we can prove it’s true
- Does this imply that human reasoning cannot be captured in a formal system (calculus)? Or reduced to calculation?
- Philosophers have been grappling with this problem since the 1930s
Hypercomputation

- CT Thesis says “effectively calculable” = “Turing-computable”
- Some authors equate “computable” with Turing-computable
- If true, then the limits of the TM are the limits of computation
- Is human intelligence “effectively calculable”?
- Hypercomputation = computation beyond the “Turing limit”