UNIT 14C
The Limits of Computing:
Non-computable Functions

Problem Classifications

- **Tractable Problems**
  - Problems that have reasonable, polynomial-time solutions

- **Intractable Problems**
  - Problems that may have no reasonable, polynomial-time solutions

- **Noncomputable Problems**
  - Problems that have no algorithms at all to solve them
Today’s Lecture

• We will look the Halting Problem that is a canonical problem in the study of limits of computing.
• We will show using proof by contradiction that it cannot be solved
• Along the way, we will think about termination and programs that have some form of self-reference.

The Barber Paradox

• Suppose there is a town with just one barber, who is male. In this town, every man keeps himself clean-shaven, and he does so by doing exactly one of two things:
  1. Shaving himself, or
  2. Going to the barber.

  • Another way to state this is: The barber is a man in town who shaves those and only those men in town who do not shave themselves.

  • Who shaves the barber?
Program Termination

- Can we determine if a program will terminate given a valid input?
- Example:
  ```python
def mystery1(x):
    while (x != 1) do
        x = x - 2
    end
end
```
  - Does this algorithm terminate when \( x = 15 \)?
  - Does this algorithm terminate when \( x = 110 \)?

Another Example

```python
def mystery2(x):
    while (x != 1) do
        if x % 2 == 0 then
            x = x / 2
        else
            x = 3 * x + 1
        end
    end
end
```
- Does this algorithm terminate when \( x = 15 \)?
- Does this algorithm terminate when \( x = 110 \)?
- Does this algorithm terminate for any positive \( x \)?
The Halting Problem

• Does a universal program $H$ exist that can take any program $P$ and any input $I$ for program $P$ and determine if $P$ terminates/halts when run with input $I$?
• Alan Turing showed that such a universal program $H$ cannot exist.
  – This is known as the Halting Problem.

Proof by Contradiction (example)

Suppose you want to prove the proposition “One cannot get an A in this course without doing the homeworks”.
1. You first assume the opposite: “One can get an A in this course without doing the homeworks”.
2. From that assumption and using what you know about the course you arrive at a conclusion, which is not true (e.g. Homeworks are worth less than 10%).
3. Since you know that this conclusion is false (contradicts with what is known), the initial assumption must be wrong.
   “One can get an A in this course without doing the homeworks”. Must be false
Proof by Contradiction (first step)

- Assume a program \( H \) exists that requires a program \( P \) and an input \( I \).
  - \( H \) determines if program \( P \) will halt when \( P \) is executed using input \( I \).

- We will show that \( H \) cannot exist by showing that if it did exist we would get a logical contradiction.

\[
\begin{array}{ccc}
\text{\( P \)} & \xrightarrow{\text{HALT CHECKER}} & \text{\( H \)} \\
\downarrow & & \downarrow \\
\text{\( I \)} & & \text{\( \begin{cases} \text{YES} & \text{if } P \text{ halts when run with input } I \\ \text{NO} & \text{if } P \text{ does not halt when run with input } I \end{cases} \)}
\end{array}
\]

Programs Computing with Their Own Representation

- A compiler is a program that takes as its input a program that needs to be translated from a high-level language (e.g. Ruby) to a low-level language (e.g. machine language).
  - In general, a program can process any data, so it can have a program as its input to process.

- Can a compiler compile itself? \( \text{YES!} \)
Proof (cont’d)

• Let $D$ be a program that takes input $<M>$ where $<M>$ is a program description.
• $D$ asks the halt checker $H$ what happens if $M$ runs with itself $<M>$ as input?
• If $H$ answers that $M$ will halt if it runs with itself as input, then $D$ goes into an infinite loop (and does not halt).
• If $H$ answers that $M$ will not halt if it runs with itself as input, then $D$ halts.

How $D$ Works

$D$ asks $H$ what happens if we run program $M$ on with input $<M>$. Loops if it says yes. Stops and returns OK if it says no.
D gets evil

• What happens if $D$ tests itself?
  – If $H$ answers yes ($D$ halts), then $D$ goes into an infinite loop and does not halt.

Proof By Contradiction (last step)

• What happens if $D$ tests itself?
  – If $D$ does not halt on $<D>$, then $D$ halts on $<D>$.
  – If $D$ halts on $<D>$, then $D$ does not halt on $<D>$.
Contradiction

• No matter what $H$ answers about $D$, $D$ does the opposite, so $H$ can never answer the halting problem for the specific program $D$.
  – Therefore, a universal halting checker $H$ cannot exist.
• We can never write a computer program that determines if ANY program halts with ANY input.
  – It doesn’t matter how powerful the computer is.
  – It doesn’t matter how much time we devote to the computation.

Why Is Halting Problem Special?

• One of the first problems to be shown to be noncomputable (i.e. undecidable, unsolveable)
• A problem can be shown to be noncomputable by transforming the halting problem into that problem
  – For example, a virus detection software cannot detect if a program is a virus for all possible programs. To be computable, they need to give up correctness for some cases.
What Should You Know?

• The fact that there are limits to what we can compute at all and what we can compute efficiently.
  – What do we mean when we call a problem tractible/intractable?
  – What do we mean when we call a problem solveable (i.e. computable) vs. unsolveable (noncomputable)?
• What the question N vs. NP is about.
• Name some NP-complete problems and reason about the work needed to solve them using brute-force algorithms.
• The fact that Halting Problem is unsolveable and that there are many others that are unsolveable.