Chapter 4: How Universal Are Turing Machines?

CS105: Great Insights in Computer Science

Philosophy

- Some fields are ignored by philosophers (civil engineering, physical chemistry).
- Computer Science is not so lucky.
  - What does it mean to “know” something?
  - What is “intelligence”?
  - Do objects “compute” their movements?
Great Humiliations?

- Freud:
  - Astronomy (Galileo): Our world is just another world.
  - Natural History (Darwin): Our species is just another species.
  - Psychoanalysis (Freud): Consciousness is just another mental process.

- Hillis:
  - Computer Science (Turing?): Our mind is just another computing device.

What Can’t Computers Do?

- To fight this idea, philosophers are fond of finding things that computers can’t do.
  - No one gets wet from a computer simulation of a hurricane (Dennett).
  - Translate human languages (Dreyfus).
  - Be self reflective or conscious (Lucas).
Kinds of Computation

Church’s lambda calculus: mathematical function notation

Turing machine: finite-state machine + memory tape

Von Neumann architecture: finite-state machine + memory

Post Correspondence: string extension rules

Chomsky’s context-sensitive grammars: string rewrite rules

programming languages: instructions, loops, subroutines

can simulate

Jonesing for Mario

Nestopia

Dell Inspiron 5100

iEmulator
Church-Turing Thesis

• Usually taken to mean that any machine computation can be carried out on a Turing machine.

• Sometimes expanded to mean that any physical system can be simulated on a Turing machine.

• And, since brains are physical systems, our minds must be equivalent to Turing machines!

The Element of Surprise

• A lot of CS research has gone into understanding the powers, limits, and philosophy of... coin flips!

• Later, I’ll show how computers use a bit of randomness to speed up their computations.

• For now, let’s take a brief sidebar into some probability theory that will be useful.
Using Random Bits

• Since numbers are made of bits, we can generate a random number using random bits.

• If there’s a way to create random bits (coin flips), how make a random number from 0 to 3 (dreidel)?

• How about 0 to 15?

• Tricky: How about 0 to 2?

Examples
State of the Art

- The Mersenne twister algorithm, by Makoto Matsumoto and Takuji Nishimura in 1997, has a colossal period of $2^{19937}-1$ iterations (probably more than the number of computations which can be performed in the future existence of the universe), is proven to be equidistributed in 623 dimensions (for 32-bit values), and runs faster than all but the least statistically desirable generators.

- Python has a package for this generator.

Demo: Guess My Bit

- Six rounds:
  - Guess my bit. If wrong, you’re out.
  - Anyone who can survive all 6 rounds gets extra credit.
Some Probability Theory

- If event has probability $p$, $1/p$ tries before it happens (on average).
- If $n$ distinct events are equally likely, $\sim n \ln n$ tries before we see all $n$ of them (order independent) or $\sim n^2$ (in order).
- If we start with a number $n$, we can cut it in half $\log n$ times before 1 is reached.

Quantum Effects

- Pseudo-random-based encryption always has a chance of being cracked.
- Only source of true randomness: quantum mechanics (the rest of physics is deterministic, if chaotic).
- Einstein didn’t like it. Tough.
Quantum Randomness

Quantum Computer

- A quantum bit (qubit) is simultaneously zero and one (a superposition). \( n \) qubits can represent \( 2^n \) possibilities.
- When you look, one possibility presents itself, according to well understood probabilistic rules. A kind of parallel search.
- **Shor:** A computer with qubits can factor numbers in polynomial time!
If Factoring is Easy...

- quantum computers invalidate standard cryptosystems. No more secrets.
- However, they also open up some wild possibilities.
- *quantum cryptography*: qubits can be completely random and correlated at a distance. Can be used to send absolutely secret messages.
• My only piece of advice for today:
• Don’t listen to anything I say today; it will hurt your head.

• The main topic today is self reference and self contradiction.
• The idea is that “interesting” things happen when something can refer to itself and assert that it has properties that negate its own existence.
Oxymorons

- We’re all familiar with oxymorons: words that harbor two conflicting meanings.

- Top ten list from http://www.oxymoronlist.com/:

  20. Government Organization
  19. Alone Together
  18. Personal Computer
  17. Silent Scream
  16. Living Dead
  15. Same Difference
  14. Taped Live
  13. Plastic Glasses
  12. Tight Slacks
  11. Peace Force

  10. Pretty Ugly
  9. Head Butt
  8. Working Vacation
  7. Tax Return
  6. Virtual Reality
  5. Dodge Ram
  4. Work Party
  3. Jumbo Shrimp
  2. Healthy Tan
  1. Microsoft Works

Surface Contradiction

- Examples seem incongruous, but they all actually make sense.
  - “jumbo shrimp” just means pretty big for a shrimp, which makes perfect sense. Like “pretty fly for a white guy”.

- The contradiction isn’t very deep.
# You Don’t Say...

- We’re not f-ing bitter!
- I am not in denial.
- Eschew obfuscation.
- When you least expect it, expect it.
- But, I don’t speak English.
- I’m not talking to you.
- I can’t talk right now.
- I’m sorry, but I just don’t care.
- Leave me be, I’m asleep.
- I’m speechless.
- I prefer not to have an opinion.
- As a general rule, we handle everything on a case-by-case basis.
- I told you he’s unpredictable!
- OK is “oll korrect” (1939)!
- Don’t let me pressure you!

# Unquotable

- Know what I hate most? Rhetorical questions. (Camp)
- They all laughed when I said I was going to be a comedian. Well, they're not laughing now. (Munkhouse)
- If there is anything the nonconformist hates worse than a conformist, it's another nonconformist who doesn't conform to the prevailing standard of nonconformity. (Vaughan)
- You know what I hate? Indian givers...no, I take that back. (Phillips)
- Last month I blew $5,000 on a reincarnation seminar. I figured, hey, you only live once. (Shakes)
- I almost had a psychic girlfriend but she left me before we met. Plan to be spontaneous tomorrow. I’d kill for a Nobel Peace Prize. I was trying to daydream, but my mind kept wandering. (Wright)
Untitled by Anonymous

There is nothing which cannot be improved

Escher Parody
Serious Fun
• There’s a weird idea here: You shouldn’t be able to create a statement that, if interpreted properly, results in another statement that contradicts the original statement.
• It’s a “go back in time and kill your own grandfather” sort of thing.
• There are a bunch of deep mathematical insights that come from applying this idea.
• Here’s a quick survey before the main event.

Russell’s Paradox
• In the town of Chelm, there’s a barber. His job is to shave every man in town who does not shave himself.
• Who shaves the barber?
• For the statement to be true:
  - If he shaves himself, then he does not need to shave himself.
  - If he does not shave himself, then he needs to shave himself.
Naive Set Theory

- This example was created to show that there are limitations to how you can create meaningful sets.
- If there is unrestricted self-reference, you can create impossible situations.

Gödel’s Theorem

“This statement cannot be proven.”

- Kurt Gödel showed that any system of mathematics that includes the integers can express this self-referential statement.
  - If it’s true, you can’t prove it! (Incompleteness.)
  - If it’s false, you can prove something that’s false! (Inconsistency.)
- Tough choice.
Kantor’s Diagonalization

- How many fractions are there? *Infinite*.
- How many decimals are there? *Infinite*.
- Are they the same size infinity?
- Well, we can make an infinitely long list that includes every fraction:

List of Fractions

- Start with all the fractions where the numerator and denominator add up to 1, then 2, then 3.
- Every fraction must eventually appear.

<table>
<thead>
<tr>
<th></th>
<th>0/1</th>
<th>0/2</th>
<th>1/1</th>
<th>0/3</th>
<th>1/2</th>
<th>0/4</th>
<th>1/3</th>
<th>2/2</th>
<th>0/5</th>
<th>1/4</th>
<th>2/3</th>
<th>0/6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/5</td>
<td>2/4</td>
<td>3/3</td>
<td>0/7</td>
<td>1/6</td>
<td>2/5</td>
<td>3/4</td>
<td>0/8</td>
<td>1/7</td>
<td>2/6</td>
<td>3/5</td>
<td>4/4</td>
</tr>
</tbody>
</table>
And The Decimals?

- Can we list all the decimals?
- The “add to a constant” trick doesn’t work anymore, since we have decimals like 0.3333... where the digit sum is infinite.
- So, let’s say we can list them all.
- Here’s the list, hypothetically:

On The List?

- Read down the diagonal.
  0.5898032467...
- Add 1 to each digit (with wraparound).
  0.6909143578...
- The resulting decimal is not on the list! (Differs from the ith one in the ith digit.)
Conclusion

- You *can’t* make a list of all the decimals.
- You *can* make a list of all the fractions.
- There are more decimals (real numbers) than fractions (rational numbers)!

Looping Forever

- Each of these scripts counts *i* backwards from 10 to 1 and pops 10 times.
- For each, is there an initial value we can set *i* to be that would cause the script to loop forever?
The Halting Problem

- Looping forever is one of the most annoying classes of programming errors.
- Would be great if a tool could automatically detect whether a program always halted.
- We’d like a subroutine `haltTester` that takes a program as input and returns `true` if the program halts on all inputs and `false` if some input makes it loop forever.

haltTester

- Scratch doesn’t let you set a variable to a program, but many languages do.
- Let’s imagine that we can.
- Let’s imagine that `haltTester` is a Scratch script that expects a program in a variable called “prog”.
- After it’s done its test, it sets a variable “answer” to 1 (True) if prog always halts and 0 (False) otherwise.
Contrary

• If a **haltTester** subroutine exists, can use it in other scripts.

  ![Contrary Diagram]

• For example, this script takes a program “prog” as input, and, if prog is the program **contrary** and **haltTester** says it always halts, **contrary** loops forever.

  • Otherwise it halts.

Contrary Analysis

• What does **contrary** do if **prog** is set to **contrary**?
  • If **haltTester** answers 1, that means **contrary** halts on all inputs, so it should halt on itself as input.
  • But, in this case, **contrary** loops forever!

• So, it must be that **haltTester** returns 0, meaning that **contrary** loops forever on some input.
  • But, in this case, note that **contrary** halts for any input, including **contrary**!

  • It can’t happen!
Halting Summary

- If `haltTester` says `contrary` always halts, then it purposefully loops forever when `prog = contrary`.
- If `haltTester` says `contrary` sometimes doesn’t halt, then it never loops, no matter what `prog` is.
- As with the Barber paradox, the problem here is our assumption, specifically, that a program like `haltTester` can be written.
- So, `haltTester` is a well-defined problem that no program can solve: It is incomputable.

The Answer Man

by Michael Littman

A story about computability and the halting problem to the tune of Billy Joel’s “Piano Man”

Midi file from http://ekn.net/midi/Billy-Joe/Piano_Man.mid.
Created for CS105, Rutgers University,
What’s Up?

• The “Answer Man” is the haltTester. The singer is contrary. Asking whether or not a someone drinks tequilla with salt is like asking whether or not a program halts.

• Note that it is not impossible to write a program that solves the halting problem for some programs but not others. The impossible bit is something that works for all programs.

CS Implications

• There are many problems that turn out to be incomputable.

• All involve computations that might take an infinite number of comparisons to solve and you’re never quite sure when to stop.

• An open problem I posed in my thesis (finding optimal policies for partially observable Markov decision processes) was later shown to be incomputable.
Philosophical Implication

• Some have argued that since people can tell if programs halt but programs can’t tell if programs halt, people are fundamentally more powerful / intelligent than computers.
• Hogwash.

3x+1 Problem

• Take a number. Half it if it’s even. Otherwise, triple and add 1. Continue until 1 is reached.
• Any power of 2 will be brought to 1 quickly.
• Some take awhile: 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
• No one knows if it always halts!
• We can’t (easily) solve the halting problem.
Here’s an odd little aside. For many formal self-reference-based proofs, programs need to be able to refer to themselves. How do you do that?

Consider a subroutine `subber`.

It takes a string as input and produces a new string as output.

The output is essentially a copy, but some special characters are converted (subbed).

None of these characters are in `subber`, by the way.

```
def subber(q):
    o = ""
    for i in q:
        if i == str(1+1): o = o + ""
        elif i == str(1+1+1): o = o + q
        elif i == str(1+1+1+1): o = o + "\n"
        elif i == str(1+1+1+1+1): o = o + "\t"
        elif i == str(1+1+1+1+1+1): o = o + "\\\n"
        else: o = o + i
    return o
```
Self-Referential Program

- Running this program causes it to print precisely the program itself!

  ```python
def selfPrint():
    print subber("def selfPrint():45print subber(232)")
  ```

- Can even include subber, too:

  ```python
def selfContained():
    print subber("def selfContained():45print subber(232)")
  ```

Weird?

- Something weird and “birth”-like here. The program has a string, which is the program, which somehow has the string, which is the program...

- Should be infinitely big, but it’s not via clever use of variables and substitution.
• “Since computers can’t solve the Halting Problem, we are smarter.”: Uh, no.
• “Since the halting problem can’t be solved, one cannot reliably predict the future, even when the future is the future execution of a deterministic program. ‘Free Will’ is consequence of the Halting Problem.”: Hmm!