Lecture 38: Modeling Computing

David Evans
University of Virginia cs1120 Fall 2009
http://www.cs.virginia.edu/cs1120

Menu

Exam 2 and Final

- Noncomputability of Malware Detection
- Modeling Computing
  - Turing's Model
  - Universal Machines

Remember to send me your requested site name by **midnight tonight** if you want your site hosted at *name*.cs.virginia.edu.

Is-Malware Problem

Input: A string, *s*, representing a program.

Output: If *s* is malware, **True**; otherwise, **False**.

Is "Is-Malware" computable?

From Paul Graham's "Undergraduation":

My friend Robert learned a lot by writing network software when he was an undergrad. One of his projects was to connect Harvard to the Arpanet; it had been one of the original nodes, but by 1984 the connection had died. Not only was this work not for a class, but because he spent all his time on it and neglected his studies, he was kicked out of school for a year.
... When Robert got kicked out of grad school for writing the Internet worm of 1988, I envied him enormously for finding a way out without the stigma of failure.
... It all evened out in the end, and now he's a professor at MIT. But you'll probably be happier if you don't go to that extreme; it caused him a lot of worry at the time.

3 years of probation, 400 hours of community service, $10,000+ fine

Morris Internet Worm (1988)

p = fingerd
- Program used to query user status
- Worm also attacked other programs

i = "nop^{400} pushl $68732f pushl $6e69622f movl sp,r10 pushl $0 pushl $0 pushl r10 pushl $3 movl sp,ap chmk $3b"

Worm infected several thousand computers (~10% of Internet in 1988)

is_malware("p(i)") should evaluate to **True**

Uncomputability Proof

Suppose we could define **is_malware**. Then we could define **halts**:

```
def halts(s):
  return is_malware (remove_evil(s) +
      """
      do_evil()
      """)
```

Can we make **remove_evil**?

Yes, just replace all externally visible actions (e.g., file writes) in *s* with shadow actions.

Thus, **is_malware** is noncomputable.

Can Anti-Virus programs exist?

"The **Art** of Computer Virus Research and Defense"
Peter Szor, Symantec

"Solving" Noncomputable Problems

- Since the problem is noncomputable, there is no procedure that (1) always gives the correct answer, and (2) always finishes.
- Must give up one of these to "solve" undecidable problems
  - Giving up #2 is not acceptable in most cases
  - Must give up #1
- Or change the problem: e.g., detect file infections during an execution

Actual **is_malware** Programs

- Sometimes give the wrong answer
  - "False positive": say P is a virus when it isn't
  - "False negative": say P is safe when it is
- Database of known viruses: if P matches one of these, it is a virus
- Clever virus authors can make viruses that change each time they propagate
  - Emulate program for a *limited number* of steps; if it doesn't do anything bad, assume it is safe

How convincing is our Halting Problem proof?

```
def paradox():
  if halts('paradox()'):
    while True:
      pass
```

1. **paradox** leads to a contradiction.
2. If we have **halts**, an algorithm that solves the Halting Problem, we can define **paradox**.
3. Therefore, **halts** does not exist.

This "proof" assumes Python exists and is means exactly what it should! Python is too complex to believe this: **we need a simpler and more precise model of computation.**

Should Python implementation convince us that Python exists?

```
def make_huge(n):
  if n == 0: return [0]
  return make_huge(n-1) + make_huge(n-1)
```

```
>>> len(make_huge(10))
1024
>>> len(make_huge(100))
File "C:/Users/David Evans/cs1120/huge.py", line 3, in make_huge
  return make_huge(n-1) + make_huge(n-1)
File "C:/Users/David Evans/cs1120/huge.py", line 3, in make_huge
  return make_huge(n-1) + make_huge(n-1)
...
File "C:/Users/David Evans/cs1120/huge.py", line 3, in make_huge
  return make_huge(n-1) + make_huge(n-1)
MemoryError
```

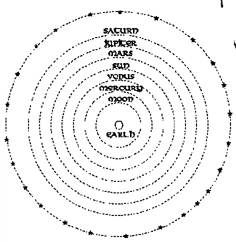
No real interpreter can correctly implement the full semantics of Python!

Solutions

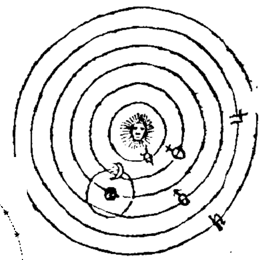
- Option 1: Prove "Python" does exist
  - Show that some ideal interpreter could implement all the evaluation rules (but what is interpreting that ideal interpreter?)
- **Option 2: Find a simpler computing model**
  - Define it precisely
  - Show that the Halting paradox procedure can be defined in this model

Note: our running time analyses also all depend on our computing model!

What makes a good model?



Ptolomy

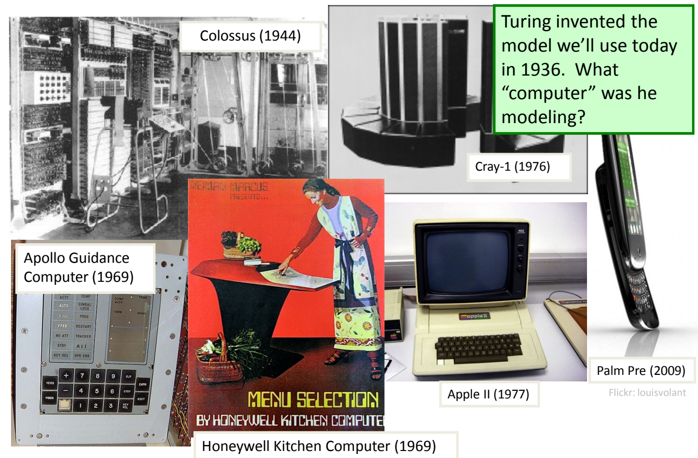


Copernicus

$$F = GM_1M_2 / R^2$$

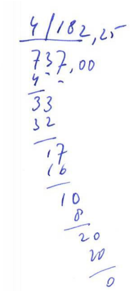
Newton

How should we model a Computer?



"Computers" before WWII

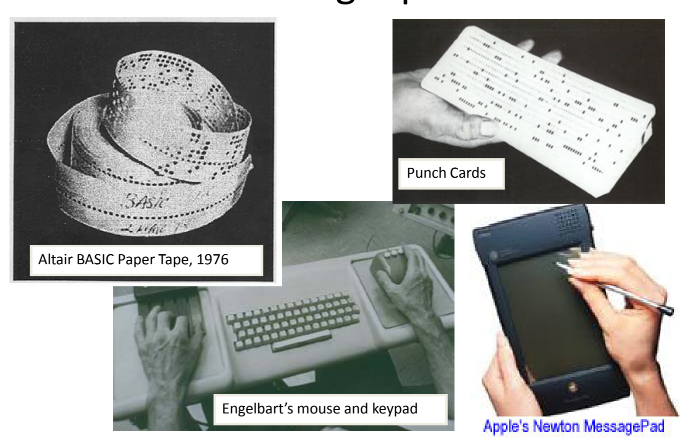
Mechanical Computing



Modeling Computers

- Input
  - Without it, we can't describe a problem
- Output
  - Without it, we can't get an answer
- Processing
  - Need some way of getting from the input to the output
- Memory
  - Need to keep track of what we are doing

Modeling Input

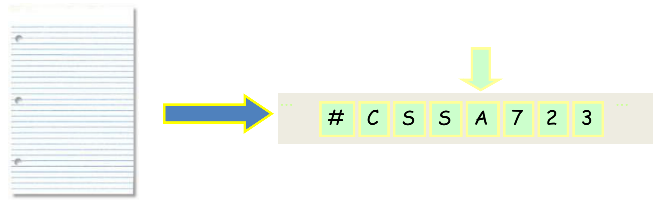


Turing's Model

"Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book."

Alan Turing, *On computable numbers, with an application to the Entscheidungsproblem*, 1936

Modeling Pencil and Paper

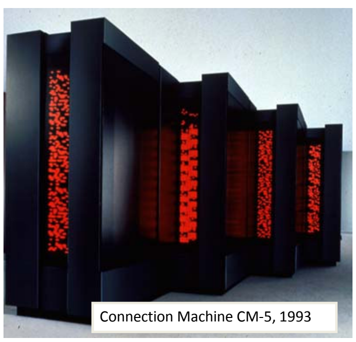


How long should the tape be?

Infinitely long! We are *modeling* a computer, not building one. Our model should not have silly practical limitations (like a real computer does).

Modeling Output

- Blinking lights are cool, but hard to model
- Use the tape: output is what is written on the tape at the end



Connection Machine CM-5, 1993

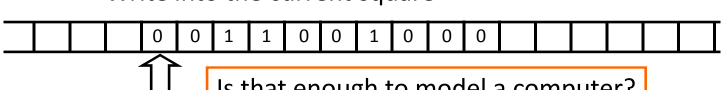
Modeling Processing (Brains)

Look at the current state of the computation

Follow simple rules about what to do next

Modeling Processing

- Evaluation Rules
  - Given an input on our tape, how do we evaluate to produce the output
- What do we need:
  - Read what is on the tape at the current square
  - Move the tape one square in either direction
  - Write into the current square

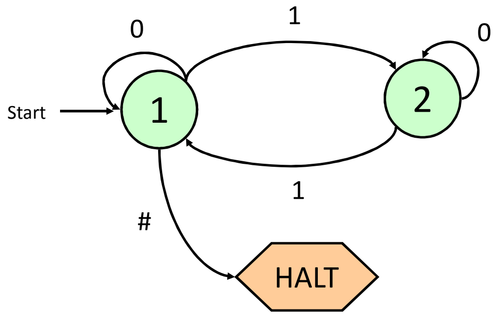


Is that enough to model a computer?

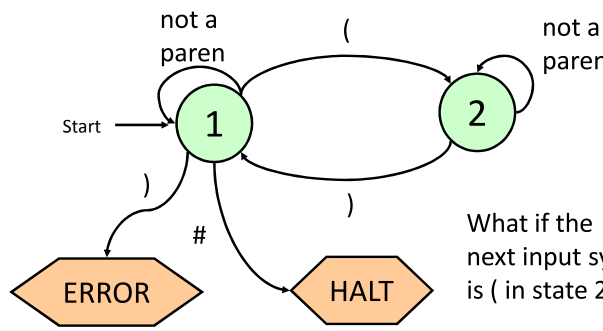
Modeling Processing

- Read, write and move is not enough
- We also need to keep track of what we are doing:
  - How do we know whether to read, write or move at each step?
  - How do we know when we're done?
- What do we need for this?

Finite State Machines

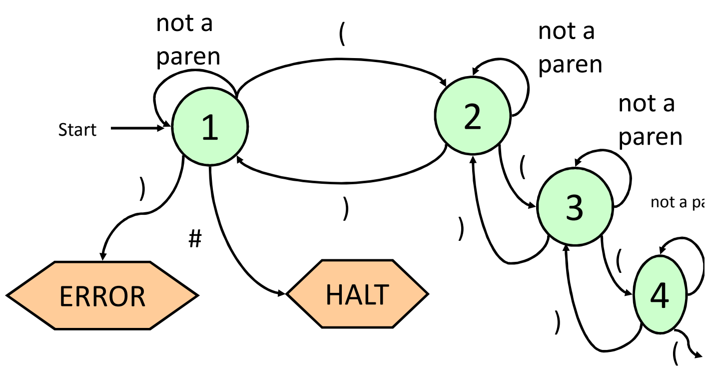


Hmmm…maybe we don't need those infinite tapes after all?



What if the next input symbol is (in state 2?

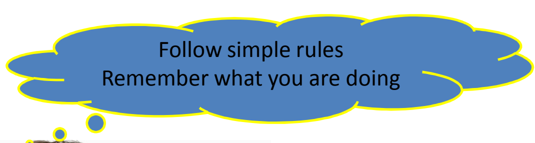
How many states do we need?

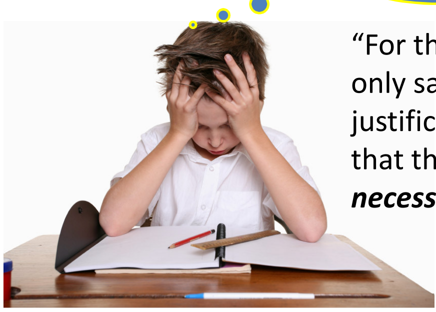


Finite State Machine

- There are lots of things we can't compute with only a finite number of states
- Solutions:
  - "Infinite" State Machine
    - Hard to define, draw, and reason about
  - **Add an infinite tape to the Finite State Machine**

Modeling Processing (Brains)





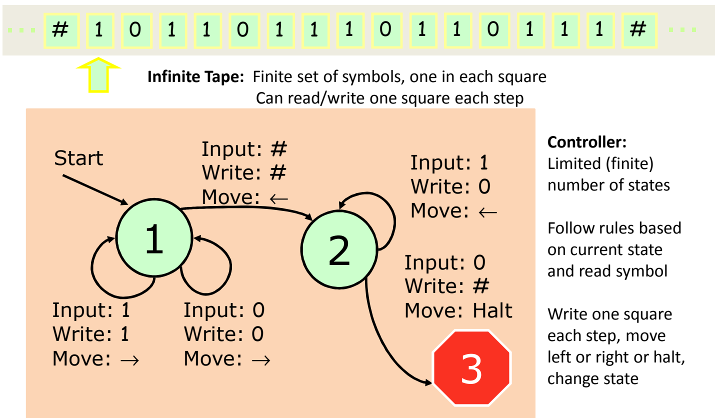
"For the present I shall only say that the justification lies in the fact that the ***human memory is necessarily limited***."

Alan Turing

FSM + Infinite Tape

- Start:
  - FSM in Start State
  - Input on Infinite Tape
  - Pointer to start of input
- Step:
  - Read one input symbol from tape
  - Write symbol on tape, and move L or R one square
  - Follow transition rule from current state
- Finish:
  - Transition to **halt** state

Turing's Model: Turing Machine



Charge

- If you want us to host your site, remember to send me your site name before midnight tonight!
- Wednesday:
  - Busy Beavers
  - Alternate Computing Models