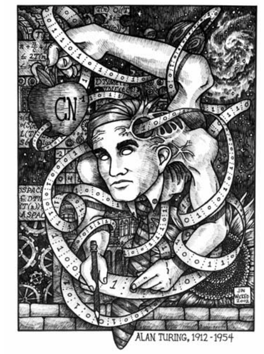
Class 26: Modeling Computing



CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/evans

How convincing is our Halting Problem proof?

```
(define (contradict-halts x)
  (if (halts? contradict-halts null)
      (loop-forever)
      #t))
```

contradicts-halts cannot exist. Everything we used to make it except halts? does exist, therefore halts? cannot exist.

This "proof" assumes Scheme exists and is consistent!

DrScheme

- Is DrScheme a proof that Scheme exists?

From Lecture 13...

```
> (time (permute-sort <= (rand-int-list 7)))
cpu time: 261 real time: 260 gc time: 0
(6 7 35 47 79 82 84)
> (time (permute-sort <= (rand-int-list 8)))
cpu time: 3585 real time: 3586 gc time: 0
(4 10 40 50 50 58 69 84)
> (time (permute-sort <= (rand-int-list 9)))
```

Crashes!

Solutions

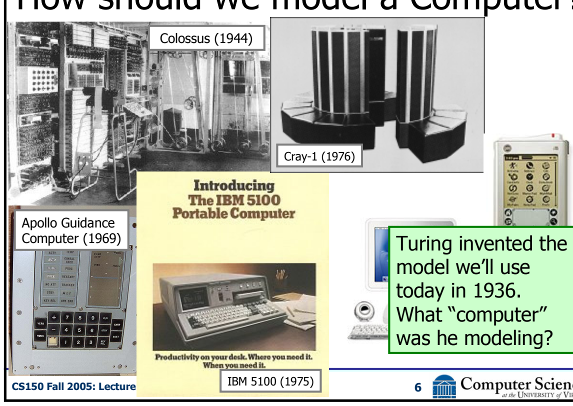
- Option 1: Prove "Idealized Scheme" does exist
  - Show that we could implement all the evaluation rules
- Option 2: Find some simpler computing model
  - Define it precisely
  - Show that "contradict-halts" can be defined in this model

Modeling Computation

- For a more convincing proof, we need a more precise (but simple) model of what a computer can do
- Another reason we need a model:

  Does complexity really make sense without this? (how do we know what a "step" is? are they the same for all computers?)

How should we model a Computer?



Turing invented the model we'll use today in 1936. What "computer" was he modeling?

Turing's "Computer"

"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."

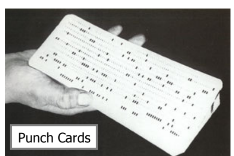
Alan Turing, *On computable numbers, with an application to the Entscheidungsproblem*, 1936"

Modeling Computers

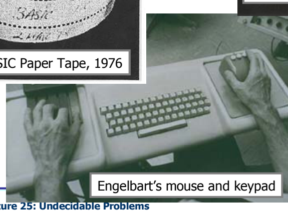
- Input
  - Without it, we can't describe a problem
- Output
  - Without it, we can't get an answer
- Processing
  - Need some way of getting from the input to the output
- Memory
  - Need to keep track of what we are doing

Modeling Input

Altair BASIC Paper Tape, 1976



Punch Cards



Engelbart's mouse and keypad

Apple's Newton MessagePad

Simplest Input

- Non-interactive: like punch cards and paper tape
- One-dimensional: just a single tape of values, pointer to one square on tape

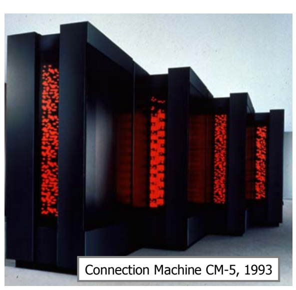
			0	0	1	1	0	0	1	0	0	0							

⇧ 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
- Output is what is written on the tape at the end of a computation



Connection Machine CM-5, 1993

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

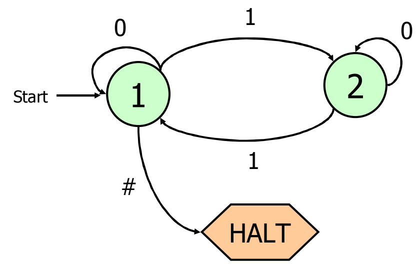
			0	0	1	1	0	0	1	0	0	0							

⇧ Is that enough to model a computer?

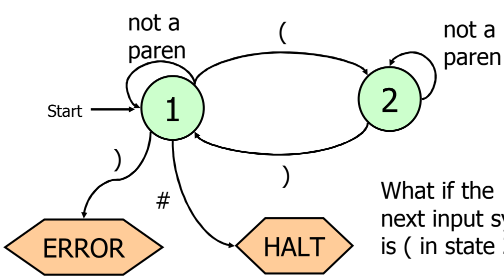
Modeling Processing

- 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

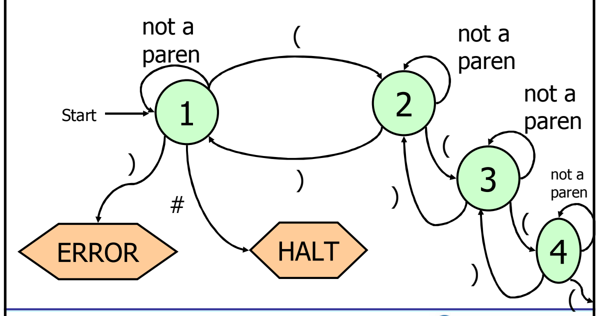


Hmmm…maybe we don't need those infinite tapes after all?



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 describe and draw
  - **Add an infinite tape to the Finite State Machine**

Turing's Explanation

"We have said that the computable numbers are those whose decimals are calculable by finite means. ... For the present I shall only say that the justification lies in the fact that the human memory is necessarily limited."

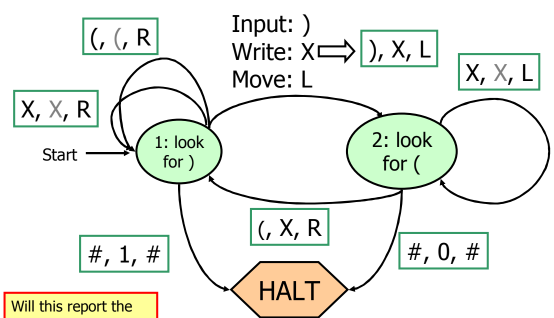
FSM + Infinite Tape

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

Matching Parentheses

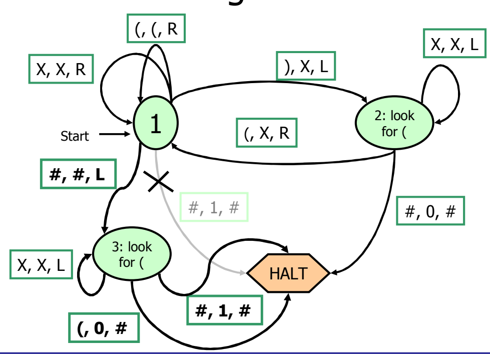
- Find the leftmost)
  - If you don't find one, the parentheses match, write a 1 at the tape head and halt.
- Replace it with an X
- Look left for the first (
  - If you find it, replace it with an X (they matched)
  - If you don't find it, the parentheses didn't match – end write a 0 at the tape head and halt

Matching Parentheses

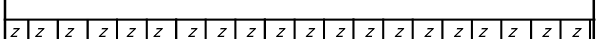


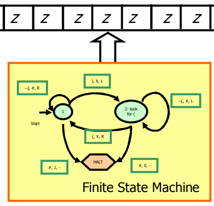
Will this report the correct result for (()?

Matching Parentheses



Turing Machine (1936)





TuringMachine ::= < *Alphabet, Tape, FSM* >
Alphabet ::= { *Symbol** }
Tape ::= < *LeftSide, Current, RightSide* >
OneSquare ::= *Symbol* | #
Current ::= *OneSquare*
LeftSide ::= [*Square**]
RightSide ::= [*Square**]

Everything to left of *LeftSide* is **#**.
Everything to right of *RightSide* is **#**.

Charge

- Wednesday:
  - Universal Turing Machines
- Friday:
  - Lambda Calculus (another simple model of computation, and the basis for Scheme)
- Monday: PS6 Due