What computers just *cannot* do. (Part II)

COS 116: 3/5/2008

Sanjeev Arora

Administrivia

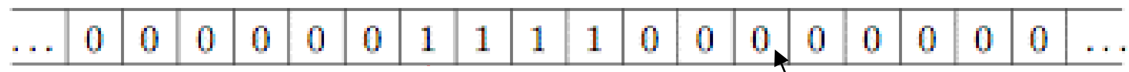
- Midterm - in-class 3/13
  - Review session Tues and Wed during lab slot.
  - 2006 midterm linked under "extras" on web

Recap from last time

...	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	...

- Turing-Post computational model:
  - Greatly simplified model
  - Infinite tape, each cell contains 0/1
  - Program = finite sequence of instructions (only 6 types!)
  - Unlike pseudocode, no conditionals or loops, only "GOTO"
  - code(*P*) = binary representation of program *P*

Example: doubling program



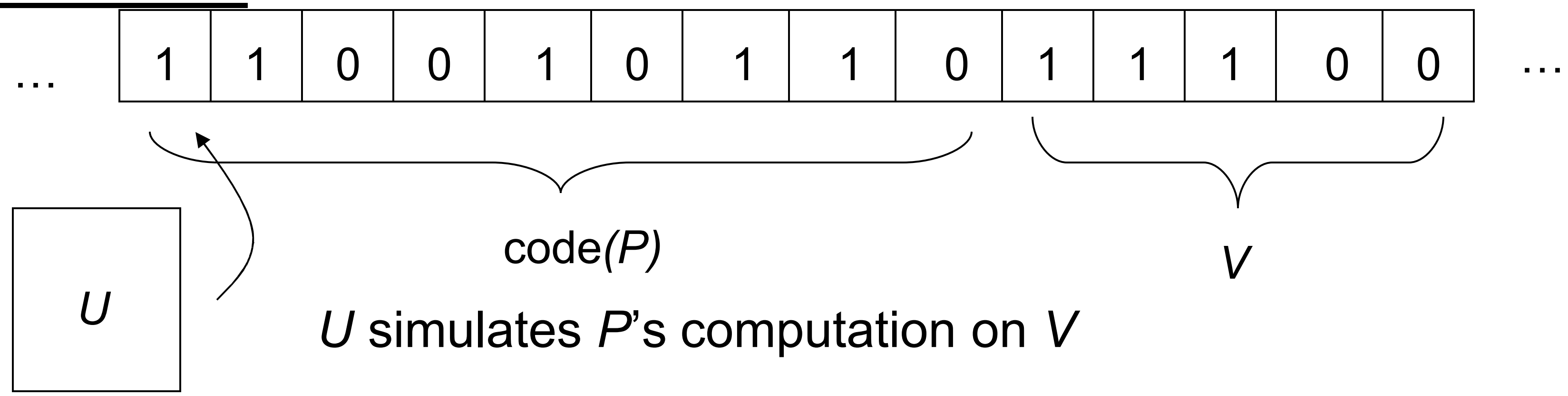
1. PRINT 0
2. GO LEFT
3. GO TO STEP 2 IF 1 SCANNED
4. PRINT 1
5. GO RIGHT
6. GO TO STEP 5 IF 1 SCANNED
7. PRINT 1
8. GO RIGHT
9. GO TO STEP 1 IF 1 SCANNED
10. STOP

Program halts on this input data if STOP is executed in a finite number of steps

Some facts

- Fact 1: Every pseudocode program can be written as a T-P program, and vice versa

- Fact 2: There is a universal T-P program __________



Discussion Time

Is there a universal pseudocode program ?

How would you write it?

What are some examples of universal programs in real life?

Halting Problem

$$\ldots \quad \underbrace{1\;1\;0\;0\;1\;0\;1\;1\;0}_{\text{code}(P)}\;\underbrace{1\;1\;1\;0\;0}_{V} \quad \ldots$$

- Decide whether *P* halts on *V* or not
- Cannot be solved! Turing proved that *no Turing-Post program can solve Halting Problem for all inputs (code(P), V).*

Makes precise something quite intuitive: “Impossible to demonstrate a negative”

Suppose program P halts on input V. How can we detect this in finite time?

“Just simulate.”

Intuitive difficulty: If P does not actually halt, no obvious way to detect this after just a finite amount of time.

Turing’s proof makes this intuition concrete.

Ingredients of the proof…..

Ingredient 1: “Proof by contradiction”

Fundamental assumption:
A mathematical statement is either true or false

“When something’s not right, it’s wrong.”

Bob Dylan

Aside: Epimenides Paradox

- *Κρῆτες ἀεί ψεῦσται*
- “Cretans, always liars!”
- But Epimenides was a Cretan!’

  (can be resolved…)

- More troubling: “This sentence is false.”

Ingredient 2:

Discussion
Time

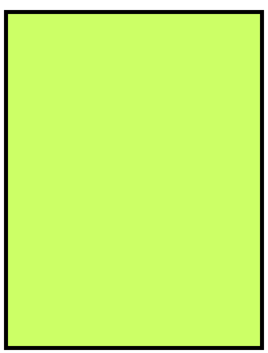
Suppose you are given some T-P program P

How would you turn P into a T-P program that does NOT halt on all inputs that P halts on?

Finally, the proof…

Suppose program H solves Halting Problem on ALL inputs of the form code(P), V.

H



Consider program D

- On input V, check if it is code of a T-P program.
- If no, HALT immediately.
- If yes, use doubling program to create the bit string V, V and simulate H on it.
- If H says “Doesn’t Halt”, HALT immediately.
- If H says “Halts”, go into infinite loop

If H halts on every input, so does D

Gotcha! Does D halt on the input code(D)?

Lessons to take away

- Computation is a very simple process
  (can arise in unexpected places)

- Universal Program

- No real boundary between hardware, software, and data

- No program that decides whether or not mathematical statements are theorems.

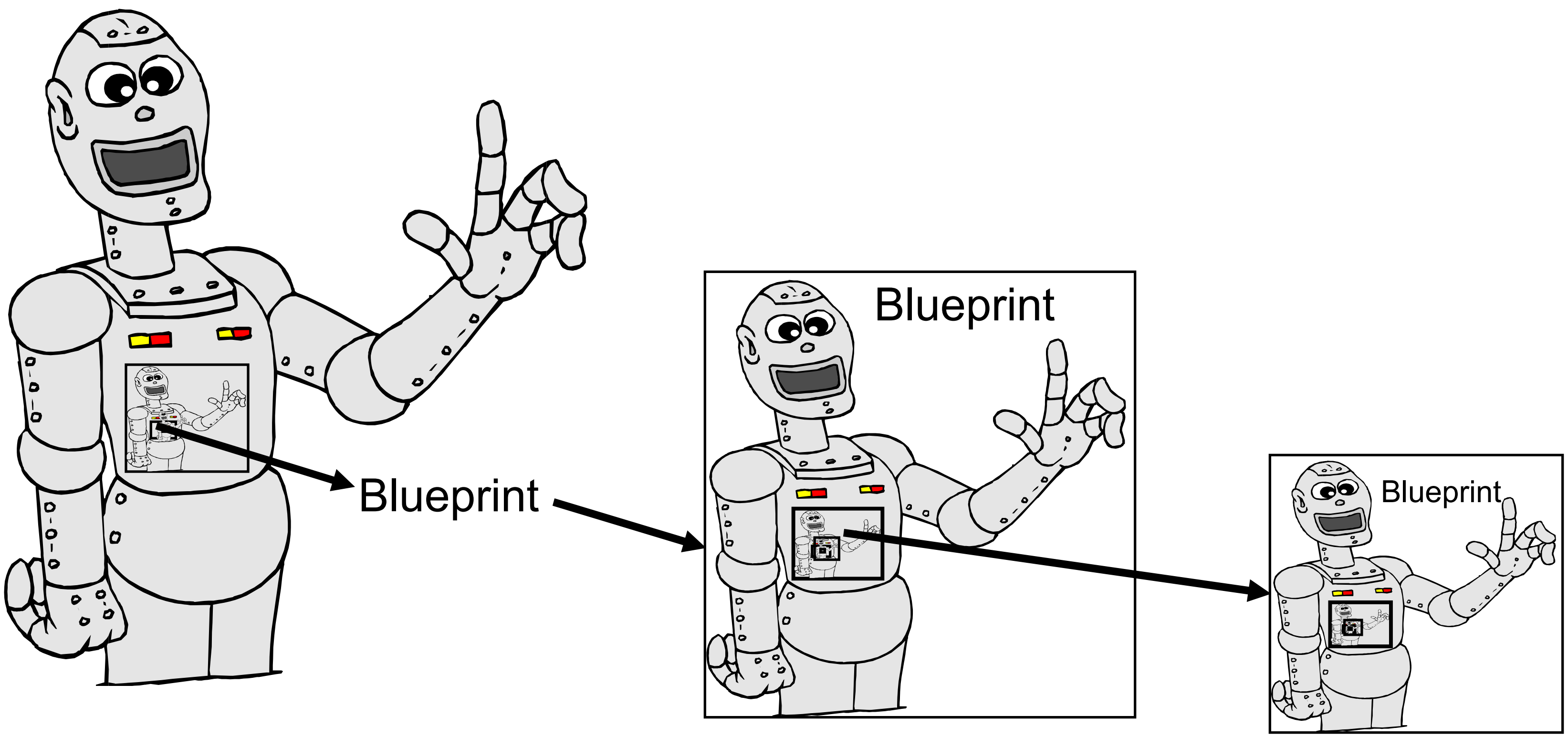
- Many tasks are uncomputable; e.g. “If we start Game of life in this configuration, will cell (100, 100) ever have a critter?”

Age-old mystery: Self-reproduction.

How does the seed encode the whole?

Self-Reproduction

Fallacious argument for impossibility:

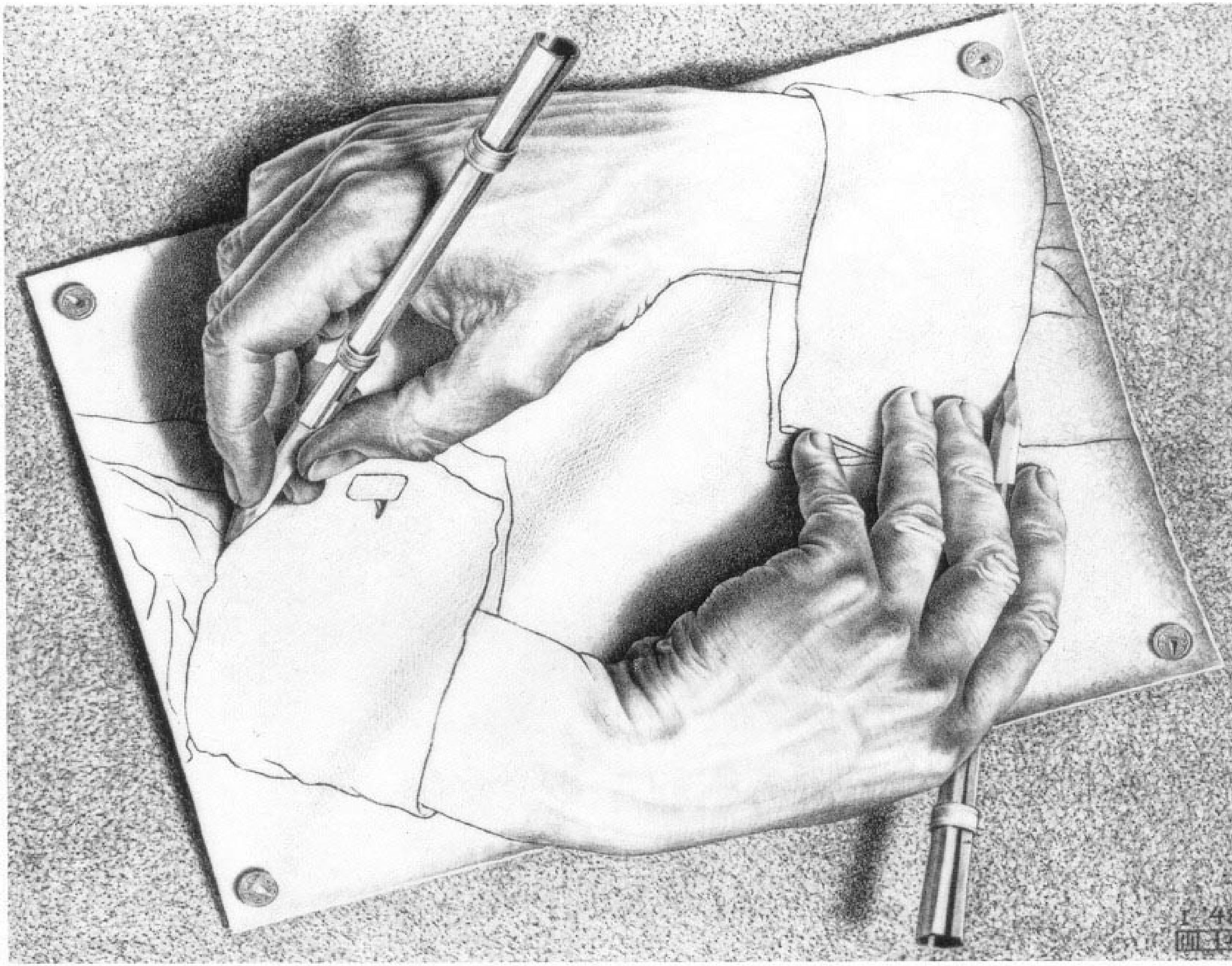


M.C. Escher

Print Gallery

Fallacy Resolved: “Blueprint” can involve some computation; need not be an exact copy!

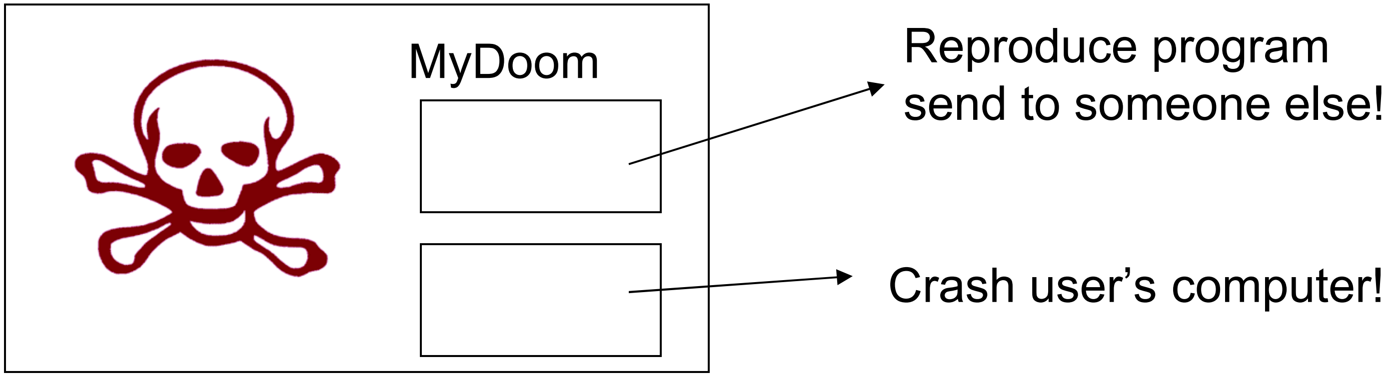
Print this sentence twice, the second time in quotes. “Print this sentence twice, the second time in quotes.”



High-level description of program that self-reproduces

A	Print 0 Print 1 . . . Print 0	Prints binary code of B
B	. . . .	Takes binary string on tape, and in its place prints (in English) the sequence of statements that produce it, followed by the translation of the binary string into English.

Self-reproducing programs



- Fact: for every program *P*, there exists a program *P'* that has the exact same functionality except at the end it also prints code(*P'*) on the tape