Lecture 24

Andrea LaPaugh

May 2, 1996

Complexity of computations

Have seen that Finite State Acceptors have limited abilities

can't count

Can we prove that more powerful computers have limited ablilities?

Toy Machine

C programs

Are there problems that are intrinsically difficult?

Halting Problem

Write a C program that reads in
another program and its input
and decides whether or not
it goes into an infinite loop

Example 1:

```
while (x != 1)
  if (x > 2) x = x - 2; else x = x + 2;
```

Example 2:

```
while (x != 1)
  if (x % 2) x = 3*x+1; else x = x/2;
```

Halting Problem (cont.)

THEOREM: The halting problem is unsolvable.

Proof: Suppose there is such a program HALTS that takes two inputs:
a binary string that is the ascii of a program P
and a binary string that is the input y to P.
If P halts on input y, HALTS halts and says "yes".
If P does not halt on input y, HALTS halts and says "no".

Build new program CONFUSE:
Given input string x, CONFUSE interprets as a program P_x.

If P_x is not legal C-code, CONFUSE HALTS.

IF P_x is legal C, CONFUSE calls HALTS as a subroutine with inputs x(= P_x) and x
(we use same binary string for input to P_x.)

If HALTS returns "yes", CONFUSE goes into infinite loop;

If HALTS returns "no", CONFUSE halts.

Now try giving CONFUSE itself as input:
(binary string x_C that is ascii of CONFUSE)
CONFUSE calls HALTS as a subroutine with inputs x_C and x_C

If HALTS says CONFUSE halts on x_C,
then CONFUSE does not halt

If HALTS says CONFUSE does not halt on x_C,
then CONFUSE halts

Contradiction!! HALTS can't exist.

Models of computation

But maybe Halting Problem is special

Many other properties of programs unsolvable,

including "Are two programs equivalent?"

But maybe questions about programs are special

Post Correspondence Problem is unsolvable

But maybe C programs are too hard.

Try it on Toy Machine programs.

BUT: If we could solve halting problem

for Toy machine, can construct a solver

for C-program halting problem:

(use C compiler as component)

A Puzzle ("Post's Correspondence Problem")

Given a set of cards

N types of cards, as many of each as needed

each has a top string and a bottom string

Example 1:

BAB	A	AB	BA
A	ABA	B	B

Puzzle: find a way to arrange the cards so that top and bottom strings are the same (or report that it's impossible).

Solution to Example 1:

A	BA	BAB	AB	A
ABA	B	A	B	ABA

Example 2: (no solution)

A	ABA	B	A
BAB	B	A	B

Surprising fact: this puzzle is UNSOLVABLE!

can't write a program to determine whether a given set of cards can be so arranged

Turing Machine

How prove all these problems unsolvable?

Need a "simpler" model of computer

N states (like FSM)

One "tape" used for input, memory, and output

- tape "cells" hold symbols
- tape has as many cells as needed

"head" which reads/writes cells of the tape

- can move right or left, one cell at a time

Initial configuration

- tape contains just input, rest blank
- head at leftmost tape cell
- in state 0

Execution

- read a symbol from the tape
- depending on which symbol, current state
  - write a symbol onto the tape
  - move right or left one cell
  - change to new state
- stop when (and if) state N-1 reached

Equivalence by simulation

Turing machine simulates Toy machine:

Encode PC, registers, memory on tape

Control simulates Toy machine instructions.

Toy machine simulates C:

Compiler translates C into Toy code

Toy machine executes Toy code

C program simulates:

any computer we know how to build!

including Turing machine

Proving problems unsolvable

Turing machine computation has simple encoding

for each move:

tape contents + head position + state

Turing machine move changes very little

tape changes near head position only

no random access memory!

To prove problem unsolvable,

"code" Turing machine into problem

given a TM,

create an instance of the problem

with the property that

if there is a solution to the problem

the corresponding TM halts

Example: Post correspondence problem

"Post Correspondence Problem"

Can mimic computation of Turning Machine using PCP
If could decide PCP, could decide if Turing Machine halts.
Thus: PCP is undecidable.
Idea of construction for a given Turing machine M: (some details missing)
First card must be:

```
#
#S(0)w#  where w is input string
```

A group of cards:

```
a   for each symbol in              #
a    read/write alphabet            #
```

Another group of cards:

```
S(i)a    if when reading 'a' in state S(i)
bS(j)    M writes 'b', goes to state S(j)
         and moves right

cS(i)a    if when reading 'a' in state S(i),
S(j)cb    M writes 'b', goes to state S(j)
          and moves  left
```

Final group of cards:

```
aS(n-1)b          S(n-1)b         aS(n-1)
S(n-1)            S(n-1)          S(n-1)

S(n-1)##
#
```

Church/Turing Thesis

Q. Which problems can a Turing machine solve?

A. *Any* problem *any* computer can solve!

A ``thesis``, not a ``theorem`` -- can't be proved

because we can't precisely define ``computable``.

Evidence in favor

have tried to define TMs with more ``power``

all attempts still can be simulated with a TM

many different ways to define ``computable``

all have been proven equivalent

Implications

Practical (solving problems)

Practical problems unsolvable

Is my program correct?

Recognize and avoid unsolvable problems

Learn from structure

same theory tells us about efficiency

Theoretical (what is computation?)

we "assume" that step-by-step reasoning will solve *any* scientific or technical problem

"not quite" says the halting problem

anything that "is like" (could be) a computer has the same flaw

physical machine (rods/gears, etc.)

genetic material?

matter itself?