COS 126 Lecture 17: Computability

As we make machines more powerful,
we can recognize more languages

Are there languages
that no machine can recognize?
Are there limits on the power of
machines that we can imagine?

Abstract models of computation help us learn
* nature of machines needed to solve problems
* relationship between languages and machines
* intrinsic difficulty of problems

Automata

Languages

Computability

Complexity

Pioneering work in the 1930's

A Puzzle ("Post's Correspondence Problem")

Given a set of cards

N types of cards, as many 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

How do we know?

Turing Machine

Simple machine with N states
- start in state 0
- read a bit
- depending on current state, input bit
  - write a bit, move tape R or L
  - move to a new state
- stop when last bit read
- ACCEPT if in state X, REJECT otherwise

Various equivalent definitions
- * different conventions for ACCEPT/REJECT
- * chars, not bits on tape
- * one-way tape

Ex: [Ref. Harel, Algorithmics]

```
                0      1      2
              -----  -----  -----
mark    0     3 0 L  1 0 R  2 0 R
move1   1     5 0 L  1 1 R  1 2 R
move2   2     6 0 L  2 1 R  2 2 R
YES     3
NO      4     4 0 L  4 1 L  4 2 L
test1   5     3 0 L  7 0 L  4 2 L
test2   6     3 0 L  4 1 L  7 0 L
back    7     0 0 R  7 1 L  7 2 L
```

Sample Turing Machine

Stops iff input is a palindrome

Tape:

```
   1      . . . 0 0 1*2 2 1 0 0 . . . mark
   2      . . . 0 0 0 2*2 1 0 0 . . . move1
   3      . . . 0 0 0 2 2*1 0 0 . . . move1
   4      . . . 0 0 0 2 2 1*0 0 . . . move1
   5      . . . 0 0 0 2 2 1 0*0 . . . move1
   6      . . . 0 0 0 2 2 1*0 0 . . . test1
   7      . . . 0 0 0 2 2*0 0 0 . . . back
   8      . . . 0 0 0 2*2 0 0 0 . . . back
   9      . . . 0 0 0*2 2 0 0 0 . . . back
   10     . . . 0 0 0 2*2 0 0 0 . . . mark
   11     . . . 0 0 0 0 2*0 0 0 . . . move2
   12     . . . 0 0 0 0 2 0*0 0 . . . move2
   13     . . . 0 0 0 0 2*0 0 0 . . . test2
   14     . . . 0 0 0 0*0 0 0 0 . . . back
   15     . . . 0 0 0 0 0*0 0 0 . . . mark
   16     . . . 0 0 0 0*0 0 0 0 . . . YES
```

C Program to Simulate Turing Machines

Three-character alphabet (0 is "blank"");
Input: description of machine (9 ints/state)

    next state, output char, tape move if 0 read
    next state, output char, tape move if 1 read
    next state, output char, tape move if 2 read

```
  int next[3][100], out[3][100], move[3][100];
  char tape[2000]; int head = 1000;
  ... /* read in machine */
  while ((tape[head++] = getchar()) != EOF) ;
  state = 0; head = 1000;
  while (state != N-1)
    {
      in = tape[head] -'0';
      state = next[in][state];
      tape[head] = out[in][state];
      head += move[in][state];
    }
  printf("Halt ");
}
```

Detail(?) missing: might run off end of tape

Machine equivalence

Turing machines are more powerful than
- FSA, PDA, LBA because of
- "infinite" tape memory
- control triggered from memory

CLAIM: Turing machines are as powerful
- as real machines

proof sketch:
- encode state of memory, PC, etc. on TM tape
- develop TM states for each instruction
- can do because all instructions
  - examine current state
  - make well-defined changes
    - depending on current state
- could simulate at gate level, machine level,..

CLAIM: Turing machines are equivalent
- to C programs

proof sketch:
- C program -> TOY program -> TM
- TM -> C program

Works for all programs and machines ?

Universal Turing Machine

A specific Turing machine (UTM) that can
simulate the operation of any Turing Machine

encode machine on tape

0 3 0 L 1 0 R 2 0 R 1 5 0 L 1 1 R 1 2 R ...

build Turing machine corresponding to C simulator

anything that can be computed by any TM
can be computed by UTM

Profound implications

Example:

- given new supercomputer X
- write "universal" simulator UX
- build TM for UX
- UTM can compute anything X can

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 "solving" a "problem" (computability)

Evidence in favor
- imagine Turing machines with more "power"
  - more tapes
  - 2D tapes
  - nondeterminism(!)
- still can simulate with a TM

More evidence in favor
- different ways to define "computable"
  - universal TM
  - lambda calculus
  - Post production system
  - "recursive" functions
- all have been proven equivalent

If a problem can't be solved by a TM, we ASSUME that it can't be solved by any other computer
If a problem can't be solved by *any* specific particular machine, we ASSUME that it can't be solved by any other computer

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

Program 1:

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

Ex:

8 6 4 2 4 2 4 2 4 2 4 2

9 7 5 3 1

Halts iff x odd

Program 2:

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

Ex:

8 4 2 1

7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

???

Halting Problem (cont.)

THEOREM: The halting problem is unsolvable.

Proof:

Consider only programs that print "YES" or "NO"

Assume there exists a program DECIDE that

- takes any program P as input
- prints "YES" if P prints "YES"
- prints "NO" if P does not print "YES"
  - that is, if prints "NO" or doesn’t halt

Change "YES" to "NO" and "NO" to "YES" every-
where in DECIDE to get a program CONFUSE

The program CONFUSE

- takes any program P as input
- prints "NO" if P prints "YES"
- prints "YES" if P prints "NO" or doesn’t halt

Give CONFUSE *itself* as input

- if CONFUSE prints "NO"
  - that means <CONFUSE prints "YES">
- if CONFUSE prints "YES"
  - that means <CONFUSE doesn’t print "YES"">

Contradiction: DECIDE can’t exist

Sketch of proof for halting problem: change
"print YES" to "print anything and halt" and
"print NO" to "never print anything"

Unsolvable Problems

Halting Problem not "artificial"
- reduced to simplest terms to simplify proof
- closely related to some practical problems

Very profound implications

Unsolvability of Halting Problem can be
- used to show other problems to be unsolvable

Technique: "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

Examples

- Post correspondence problem
- Do two programs produce the same input?
- Thue's word problem
- Equivalence of context-free grammars

Implications

Practical

- work with limitations
  - recognize and avoid unsolvable problems
- learn from structure
  - same theory tells us about efficiency

Philosophical

- (caveat: ask a philosopher)
- 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.)
    - human brain?
    - matter itself?
    - universe?