CSE 417: Algorithms and Computational Complexity

Winter 2002

Computability & Uncomputability

L. Ruzzo & Bill Pentney

1

A Brief History of Ideas

- From Classical Greece, if not earlier, "logical thought" held to be a somewhat mystical ability
- Mid 1800's: Boolean Algebra and foundations of mathematical logic created possible "mechanical" underpinnings
- 1900: David Hilbert's famous speech outlines program: mechanize all of mathematics? http://mathworld.wolfram.com/HilbertsProblems.html
- 1930's: Gödel, Church, Turing, et al. prove it's impossible

2

What's an "Algorithm"?

Programming System

- "Input": finite (but arbitrarily long) sequence of symbols from a fixed, finite set (e.g., {0,1}, or {a,b,c}, or "ascii")
- "Configuration": a finite (but arbitrarily large) description of intermediate results in the computation
- "Operations": a fixed set of possible operations, each "obviously" mechanical, defined by how they change one config into another
- "Program/Algorithm": finite list of operations (and rules for choosing the order in which they are executed.)

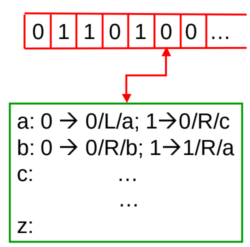
3

Examples

- C/C++/etc.:

```
main() {
    int i; // really an integer
    for (i=0;i<10;i++){
        …
    }
    return 0;
}
```

- The Turing Machine (Alan M. Turing, 1912-54)



4

Turing Machines

- Church-Turing Thesis
  - Any reasonable model of computation that includes all possible algorithms is equivalent in power to a Turing machine
- Evidence
  - Huge numbers of equivalent models to TM's based on radically different ideas

5

Universal Turing Machine

- A Turing machine interpreter U
  - On input the code of a program (or Turing machine) P and an input x, U outputs the same thing as P does on input x
  - Basis for modern stored-program computer
- Notation:
  - We'll write <P> for the code of program P and <P,x> for the pair of the program code and input

6

Halting Problem

- Given: the code of a program P and an input x for P, i.e. given <P,x>
- Output: 1 if P halts on input x and 0 if P does not halt on input x
- Theorem (Turing): There is no program that solves the halting problem
  "The halting problem is undecidable"

7

Diagonal construction

- Suppose there is a program H solving the halting Problem
- Now define a new program D such that
  - D on input x:
    - runs H checking if the program P whose code is x halts when given x as input; i.e. does P halt on input <P>
    - if H outputs 1 then D goes into an infinite loop
    - if H outputs 0 then D halts.

8

Code for D assuming subroutine for H

- Function D(x):
  - **if** H(x,x)=1 **then**
    - **while** (true); /* loop forever */
  - **else**
    - **no-op**; /* do nothing and halt */
  - **endif**

9

Finishing the argument

- Suppose D has code <D> then
  - D halts on input <D>
  - iff (by definition of D)
  - H outputs 0 given program D and input <D>
  - iff (by definition of H)
  - D runs forever on input <D>
- Contradiction!

10

Undecidability of the Halting Problem (alternate proof)

- Suppose that there is a program H that computes the answer to the Halting Problem
- We'll build a table with all the possible programs down one side and all the possible inputs along the other and do a diagonal flip to produce a contradiction

11

input

program code	ε	0	1	00	01	10	11	000	001	010	011	....
ε	0	1	1	0	1	1	1	0	0	0	1	....
0	1	1	0	1	0	1	1	0	1	1	1	....
1	1	0	1	0	0	0	0	0	0	0	1	....
00	0	1	1	0	1	0	1	1	0	1	0	....
01	0	1	1	1	1	1	1	0	0	0	1	....
10	1	1	0	0	0	1	1	0	1	1	1	....
11	1	0	1	1	0	0	0	0	0	0	1	....
000	0	1	1	1	1	0	1	1	0	1	0	....
001	.	.	.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	.	.	.	
.												

Entries are 1 if program P given by the code halts on input x and 0 if it runs forever

12

program code / input

	ε	0	1	00	01	10	11	000	001	010	011	....
ε	1	1	1	0	1	1	1	0	0	0	1	....
0	1	0	0	1	0	1	1	0	1	1	1	....
1	1	0	0	0	0	0	0	0	0	0	1	....
00	0	1	1	1	1	0	1	1	0	1	0	....
01	0	1	1	1	0	1	1	0	0	0	1	....
10	1	1	0	0	0	0	1	0	1	1	1	....
11	1	0	1	1	0	0	1	0	0	0	1	....
000	0	1	1	1	1	0	1	0	0	1	0	....
001	.	.	.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	.	.	.	
.												

Want to create a new program whose halting properties are given by the flipped diagonal

13

Diagonal construction

- Suppose H exists
- Now define a new program D such that
  - D on input x:
    - runs H checking if the program P whose code is x halts when given x as input; i.e. does P halt on input <P>
    - if H outputs 1 then D goes into an infinite loop
    - if H outputs 0 then D halts.

14

Relating hardness of problems

- We have one problem that we know is impossible to solve
  - Halting problem
- Showing this took serious effort
- We'd like to use this fact to derive that other problems are impossible to solve
  - don't want to go back to square one to do it

15

Reductions

- Given two problems to solve, L and R.
  - (think Left and Right)
- Suppose you had a translation program T so that the following would correctly solve L (if you happened to have code for R handy)
  - Function L(x)
    - Run program T to translate input x for L into an input y for R
    - Call a subroutine for problem R on input y
    - Output the answer produced by R(y)

16

Property that makes this correct

- It better be the case that no matter what x is

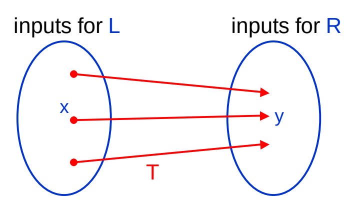
  L(x)=R(y)

  i.e. L(x)=R(T(x))

- T is called a reduction from problem L to problem R
- If such a T exists we write L ≤ R.

17

Reduction L ≤ R



L(x)=R(T(x))

Intuition: L is at least as easy as R or, equivalently, R is at least as hard as L

18

Example: BFS $\leq$ Shortest-Path

- BFS: Given a graph G and a vertex v, output the BFS tree of G started at v
- Shortest-Paths: Given a graph G with non-negative weights on its edges, and a vertex v output the shortest-path tree of G from v
- Reduction T: Given G and v, create weights for all edges in G giving each edge weight 1
  - <G,v> $\xrightarrow[T]{}$ <G,weights,v>

19

Properties of reductions

- Given that I have any reduction T such that L(x)=R(T(x))
  - **If** I had a program that solves R then I would have a program that solves L
- Therefore
  - If there is no program that solves L then there cannot be any program that solves R!
  - (statement is just equivalent to one above)

20

Another undecidable problem

- 1's problem: Given the code of a program M does M output 1 on input 1? If so, answer 1 else answer 0.
- Claim: the 1's problem is undecidable
- Proof: by reduction from the Halting Problem

21

What we want for the reduction

- Halting problem takes as input a pair <P,x>
- 1's problem takes as input <M>
- Given <P,x> can we create an <M> so that M outputs 1 on input 1 exactly when P halts on input x?

22

Yes

- Here is all that we need to do to create M
  - modify the code of P so that instead of reading x, x is hard-coded as the input to P and get rid of all output statements in P
  - add a new statement at the end of P that outputs 1.
- We can write another program T that can do this transformation from <P,x> to <M>

23

How we might do the hard-coding if the code were in C?

- Include an assignment at the start that would place the characters in string x in some array A.
- Replace all scanf's in P with calls to a new function scanA that simulates scanf but gets its data from array A.
- Replace all printf's in P by printB which doesn't actually do anything.

24

Finishing things off

- Therefore we get a reduction
  - Halting Problem ≤ 1's problem
- Since there is no program solving the Halting Problem there must be no program solving the 1's problem.

25

Why the name reduction?

- Weird: it maps an easier problem into a harder one
- Same sense as saying Maxwell reduced the problem of analyzing electricity & magnetism to solving partial differential equations
  - solving partial differential equations in general is a much harder problem than solving E&M problems

26

Quick lessons

- Don't rely on the idea of improved compilers and programming languages to eliminate major programming errors
  - truly safe languages can't possibly do general computation
- Document your code!!!!
  - there is no way you can expect someone else to figure out what your program does with just your code ....since....in general it is provably impossible to do this!

29