### CSE 417: Algorithms and Computational Complexity

Winter 2001
Lecture 19
Instructor: Paul Beame

---

#### 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"

---

#### Undecidability of the Halting Problem

- 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

---

#### 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.
  - The row for the program $D$ would be like the flip of the diagonal

---

#### Entries are 1 if program $P$ given by the code halts on input $x$ and 0 if it runs forever
**Code for D assuming subroutine for H**

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

**Finishing the argument**

- D must be different from any program in the list:
  1. Suppose it has code <D> then
  2. D halts on input <D>
  3. iff (by definition of D)
  4. H outputs 0 given program D and input <D>
  5. iff (by definition of H)
  6. D runs forever on input <D>
  7. Contradiction!

**Relating hardness of problems**

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

**Reductions**

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

**Property that makes this correct**

- It better be the case that no matter what x is
  1. L(x)=R(y)
  2. 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.

**Reduction L ≤ R**

- inputs for L
- inputs for R
- T

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

Intuition: L is at least as easy as R or, equivalently, R is at least as hard as L
Example: BFS ≤ 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
  $$<G,v> \to <G,\text{weights},v>$$

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)

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

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 $x$?

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 $P$
  - 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 $<M>$

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.
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 |

A geek joke

- An engineer
  - is placed in a kitchen with an empty kettle on the table and told to boil water; she fills the kettle with water, puts it on the stove, turns on the gas and boils water.
  - she is next confronted with a kettle full of water sitting on the counter and told to boil water; she puts it on the stove, turns on the gas and boils water.

- A mathematician
  - is placed in a kitchen with an empty kettle on the table and told to boil water; he fills the kettle with water, puts it on the stove, turns on the gas and boils water.
  - he is next confronted with a kettle full of water sitting on the counter and told to boil water; he empties the kettle in the sink, places the empty kettle on the table and says, “I’ve reduced this to an already solved problem”.