Reductions

Note Title 1/31/2014

In mathematics, we often <u>reduce</u> one problem to another.

<u>Example</u> (details <u>not</u> important — no direct connection to computation — just a mathematical example to get the idea)

2 problems, P and Q:

- P : given n, compute $\sum_{k=1}^{n} k2^k$ (i.e. $1\times 2^1 + 2\times 2^2 + 3\times 2^3 + \dots + n2^n$)
- Q : given n, compute $\sum_{k=1}^{n} 2^k$ (i.e. $2^1 + 2^2 + 2^3 + \dots + 2^n$)

[You probably know formulas for one or both problems. This is irrelevant. Pretend you do not know how to solve either problem.]

We are interested in the relationship between the two problems. e.g. if we know how to solve one of them, would that let us easily solve the other one?

Vague definition (more precise one later): <u>P reduces to Q</u> if we can solve P by using a solution to Q.

We will now prove that P red. to Q:

Fix n. Let
$$S = \sum_{k=1}^{n} k2^k = 1\cdot 2 + 2\cdot 2^2 + 3\cdot 2^3 + \dots + n2^n \quad \text{--- (1)}$$

Then
$$2S = 1\cdot 2^2 + 2\cdot 2^3 + 3\cdot 2^4 + \dots + (n-1)2^n + n2^{n+1} \quad \text{--- (2)}$$

Subtracting (1) from (2) [line up the powers of 2 (see the arrows) to see this], we get

$$2S - S = \underbrace{-\left(2^2 + 2^3 + \dots 2^n\right)}_{\text{from the arrows}} + \underbrace{\left(n2^{n+1} - 2\right)}_{\text{other stuff}}$$

So,

$$\underset{\text{the answer to } P}{S} = -\underbrace{\left(\sum_{k=1}^{n} 2^k\right)}_{\text{the answer to } Q} + \left(n2^{n+1} - 2\right) \quad \text{--- } (\star)$$

Thus, given solution to Q, we can solve P also. Hence, P reduces to Q as claimed.

[Note: by rearranging ($\star$), we can also see that Q reduces to P. But let's ignore this]

Note that we now have 2 options for solving P. either: (a) solve Q, then use (*)
or: (b) solve P directly

In this sense, P is currently an "easier" problem than Q. (Our only option for Q is to solve it directly, but that would immediately yield a solution to P also.)

This notion (P "easier" than Q if P red. to Q) is the <u>opposite</u> of the way mathematicians think about reductions. Mathematicians usually use reductions to reduce a 'hard' problem P to an 'easy' problem Q that they already know how to solve. Theoretical comp. scientists, however, are usually considering problems P, Q that they <u>don't</u> know how to solve. An extreme case of this is when P and/or Q is undecidable (i.e. we <u>know</u> that they are impossible to solve).

Very important example: suppose P & Q are as above (so P red. to Q). Suppose P is undecidable.
[It isn't! Just <u>suppose</u>]. Then what does this tell us about Q?

Answer: Q is undecidable too. (Proof: Suppose not. Then we can solve Q. That yields a solution to P, since P red. to Q. This contradicts undecidability of P.)

So to prove some problem Q is undecidable, our strategy is:

1. Choose another undecidable problem P (e.g. halting problem, or AlwaysYes)
2. Show that P reduces to Q (i.e. show that if you had a solution to Q, you could construct a solution to P).

[Note: most textbooks use the notation '$P \leq Q$' for "P reduces to Q". This is a confusing notation, because when you 'reduce' something, it gets smaller, not larger. So we won't use this notation, but you should be aware of it].

<u>formal definition</u> [In terms of Python programs] Problem P reduces to problem Q if you can write a Python program to solve P, assuming you can import a function that implements Q.

[In terms of TMs] Problem P reduces to problem Q if you can construct a TM that solves P, given a TM that solves Q.

Note: it's usually easier to do reductions in terms of TMs rather than Python programs.

Four variants of the Halting Problem

We showed in an earlier class that AlwaysHalts is undecidable. This is just one variant of the Halting Problem. There are at least four common variants (all undecidable):

- halts on all inputs
- halts on empty input
- halts on a given input I
- halts on some input

You can feel free to use any of these halting problems as the basis for reduction proofs.

Example Prove that CanHaltWithABC is undecidable (CanHaltWithABC ≡ determine whether a given program ever halts with output 'ABC' – for any input).

Solution: We will reduce the 'any input' variant of the Halting Problem to CanHaltWithABC.
So, assume we have a TM A that takes as input the description of a machine M, and accepts M iff M can halt with output 'ABC'.
Now construct a TM H that does the following:

- take as input description of TM M
- change M to be a different description M′:
  - from any halting state in M, erase tape contents, write 'ABC', then halt.
- feed M′ into A

By construction, M halts on some input iff M′ halts with output 'ABC' on some input.
Thus H decides the halting problem. #.

Example 2 Show that PrintsX is undecidable. (PrintsX ≡ machine writes an 'X' on the tape at any time during the computation, given empty input).

Solution: We will reduce the 'empty input' version of the Halting Problem to PrintsX.
Therefore, assume we have a TM P that takes as input a description of a TM, M, and accepts if M ever writes an 'X' on its tape (on empty input).

Now create a new machine H, by altering P as follows:

- take as input description of machine M that never reads or writes an 'X'.
- alter any halting states of M so that they write an 'X', then halt. This yields a new machine M'.
- feed M' as the input to P.

By construction, M halts on empty input iff M' prints an 'X'. Thus, H accepts iff M halts. So, H solves the Halting Problem (on the restricted set of machines that don't use the symbol 'X', but this is immaterial).

Strategy guide for reduction proofs:

this can mean Python program, or Turing machine, or any other kind of computer program

Remember, there are 4 different programs involved:

P – the one you want to show is impossible, and will reduce <u>to</u>
(P stands for 'problem')

H – the one you know is undecidable, and will reduce <u>from</u>
(H stands for 'hard' or 'halting', since this will definitely be a hard problem, and is often the halting problem)

M – the input to H. M is a description of a program. It could be a string containing a Python program or a Turing Machine description. There might also be other parts of the input, like I.

M' – a transformed version of M that will be given as input to P. (Possibly along with I and some other info too.)