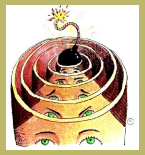


Lecture 31: Reductions

Aims:

- To discuss the idea of a reduction and reducibility;
- To discuss the idea of polynomial-time reductions, and to see what we can learn from them;
- To see what we can learn from other reductions too.

31.1. Reductions

- *Reductions* of one problem to another are going to be very important in several upcoming lectures. The following joke might give you an intuition about what a reduction is:

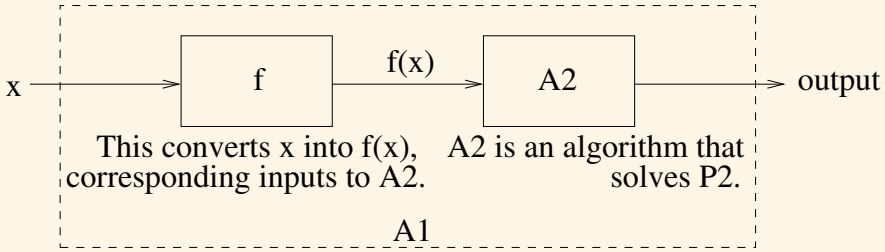
> *What's the difference between a mathematician and an engineer?*
> *Put an empty kettle in the middle of the kitchen floor and tell them to boil some water.*
> *The engineer will fill the kettle with water, put it on the stove, and turn the stove on. The mathematician will do the same thing.*
> *Next, put the kettle already filled with water on the stove, and ask them to boil some water.*
> *The engineer will turn the stove on.*
> *The mathematician will empty the kettle and put it in the middle of the kitchen floor. . . thereby reducing the problem to one that has already been solved!*

- Suppose you have a problem P_2 which you know how to solve, e.g. by using algorithm A_2.
- Suppose you are given another problem P_1 that seems similar to P_2. How might you solve P_1?
  - You could try to solve P_1 from scratch.
  - You could try to borrow elements of A_2.
  - You could try to find a *reduction* from P_1 to P_2.
- A reduction of P_1 to P_2:

– transforms inputs to P_1 into inputs to P_2;
– runs A_2 (which solves P_2) as a 'black-box'; and
– interprets the outputs from A_2 as answers to P_1.

More formally,

> A problem P_1 is *reducible* to a problem P_2 if there is a function f that takes any input x to P_1 and transforms it to an input $f(x)$ of P_2, such that the solution to P_2 on $f(x)$ is the solution to P_1 on x.



A1 is an algorithm that solves P1.

- Here's a simple example.

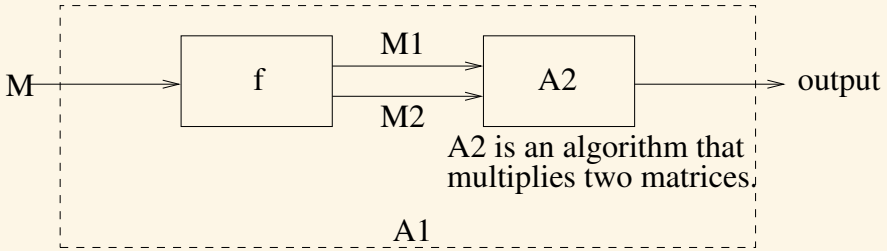
  Suppose we already have an algorithm for solving the following problem:

Problem 31.1. *Matrix Multiplication*
Parameters: *Two matrices, M_1 and M_2.*
Returns: *The result of multiplying M_1 and M_2 together.*

Suppose we are now asked to solve the following problem:

Problem 31.2. *Squaring a Matrix*
Parameters: *A matrix, M.*
Returns: *The result of squaring M.*

Here's how we would do it using a reduction:



A1 is an algorithm that squares a matrix.

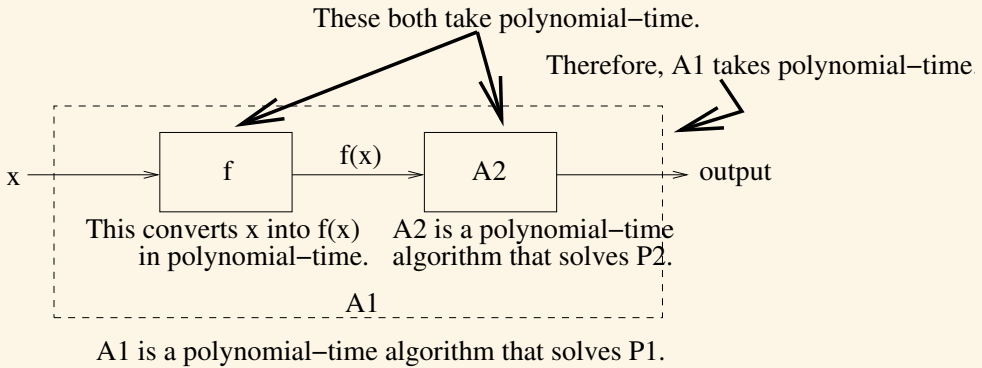
Class Exercise

What does f need to do?

- Our treatment of this topic so far makes it sound as if reductions are all about saving programming effort.
- But in theoretical Computer Science, reductions give us information. Under certain circumstances that we'll now investigate in detail, information we know about P_1 might apply to P_2, or information we know about P_2 might apply to P_1. Let's look at the details of this.

31.2. Polynomial-Time Reducibility

- The notion of reductions become especially useful in Complexity Theory when we can place a bound on the complexity of the transformation, f. Suppose, for example, that we know f takes polynomial time.
- We say that a problem P_1 is *polynomial-time reducible* to a problem P_2 if there is a function f, computable in polynomial time, that takes any input x to P_1 and transforms it to an input $f(x)$ of P_2, such that the solution to P_2 on $f(x)$ is the solution to P_1 on x.
- Shorthand $P_1 \stackrel{\text{poly}}{\longrightarrow} P_2$
- Suppose $P_1 \stackrel{\text{poly}}{\longrightarrow} P_2$.
  - This rules out the possibility that P_1 is intractable while P_2 is tractable.
  - Highly informally, it means that P_2 is 'as hard as' P_1.
  - It also means if P_2 is tractable, then P_1 is tractable.
  - Equivalently (the contrapositive): if P_1 is proved to be intractable, then P_2 is also intractable.
- Let's see why.
- **Theorem:** Suppose $P_1 \stackrel{\text{poly}}{\longrightarrow} P_2$. If P_2 is tractable, then P_1 is tractable.
- *Proof:* If P_2 tractable, then we will have a polynomial-time algorithm A_2 for solving P_2. From this, we can construct a polynomial-time algorithm, A_1, for solving P_1, as follows:



The conversion, f, takes polynomial-time and A_2 takes polynomial-time, so A_1 (which comprises f followed by A_2 in sequence) must take polynomial-time.

Hence P_1 would be tractable too.

31.2.1. Proving a problem to be tractable

- Example 1
  - As an example, let's think about matrix multiplication and matrix squaring again.
  - We've already seen that Squaring a Matrix reduces to Matrix Multiplication.

    And, in fact, Squaring a Matrix $\stackrel{\text{poly}}{\longrightarrow}$Matrix Multiplication.
  - Suppose Matrix Multiplication is tractable.
  - What do you conclude about Squaring a Matrix and why?
- Example 2.

– Consider a decision problem P. (It returns YES or NO.)

– The *complement* of P returns YES whenever P returns NO, and returns NO whenever P returns YES.

– E.g.:

Problem 31.3.

Parameters: *An integer x and a finite-length list of integers, L.*

Returns: *YES if x is a member of L and NO otherwise.*

– Its complement is

Problem 31.4.

Parameters: *An integer x and a finite-length list of integers, L.*

Returns: *NO if x is a member of L and YES otherwise.*

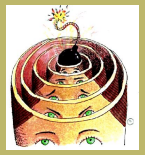
– Suppose P_2 is a decision problem and that P_2 is tractable.

– Suppose P_1 is the complement of P_2.

– **Class Exercise:** Show that P_1 is tractable.

31.2.2. Proving a problem to be intractable

- Suppose $P_1 \xrightarrow{\text{poly}} P_2$.
- We have been using this:



  - If P_2 is tractable, then P_1 is tractable.

- But we can also use this (the contrapositive):

  - If P_1 is proved to be intractable, then P_2 is also intractable.

  Note which way round this is!!!!

- **Class Exercise:** Explain why this is so.

- In fact, any lower bound we prove for P_1 may apply to P_2.

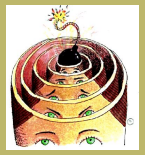
31.3. Non-computability

- We have been using reductions to give us results in Complexity Theory. But we will see reductions again in the next part of this module, when we discuss computability.
- Reductions are a powerful tool for proving a problem to be non-computable.
- Suppose P_1 reduces to P_2. (Here we no longer require polynomial-time reducibility.)
- If P_1 is non-computable, then P_2 is also non-computable.
- **Class Exercise:** Explain why this is so.

Acknowledgements:

The idea of introducing reductions using the old joke about mathematicians and engineers comes from [Man89].

Clip Art (of head with bomb) licensed from the Clip Art Gallery on DiscoverySchool.com.



References

[Man89] U. Manber. *Introduction to Algorithms: A Creative Approach.* Addison Wesley, 1989.