A Primer on Reductions

Manuel Eberl

28th January 2022

The purpose of this document is to give some more explanations about what reduction mean and how to do reduction proofs.

1 Definition

Recall that for languages $A, B \subseteq \Sigma^\star$ we defined that $A \leq B$ if there exists a computable function $f : \Sigma^\star \to \Sigma^\star$ such that

$$f(x) \in B \iff x \in A\,.$$

The function f here is a *reduction function*, and it serves as a kind of translation from the problem "is a given word in A¿' to the problem "is a given word in B¿'.

Note that for a reduction function, every word in A must be mapped to a word in B, and every word not in A must be mapped to a word not in B.

2 Consequences

In particular, $A \leq B$ means that if we know how to (computationally) solve B, then we also know how to solve A:

- if B is recursive, then A is also recursive.
- if B is recursively enumerable, then A is also recursively enumerable.
- if A is not recursive, then B is not recursive either
- if A is not recursively enumerable, then B is not recursively enumerable either

If you have trouble remembering what the notation $A \leq B$ means, just think of the $\leq$ as "is easier than": if $A \leq B$ holds, then A cannot be harder than B (at least not in the sense of what is computable).

3 Finding a Reduction Function

When you are trying to prove a reduction $A \leq B$, some creativity is required to come up with the reduction function f. One possible approach here is to think of it like this: suppose Prof. Kaliszyk gives you a word $w \in \Sigma^\star$ and asks you to decide whether $w \in A$. However, you are not allowed to simply say "yes" or "no": instead, you are given a magic oracle machine that decides whether a given word is in B and then tells the result of that to Prof. Kaliszyk for you.

The only freedom you have here is in which word w' you feed into the magic machine, and you have to choose it in such a way that the answer to "Is w' in B?" that the machine answers happens to be exactly the same answer as to "Is w in A?". The process of how you get from w to w' is exactly the reduction function (and of course, it has to be a computable process, i.e. something that you could program as a computer algorithm).

4 Examples

1) Let us show that $A \leq B$ where $A = \{0, 1, 01\}$ and $B = \{w \mid |w| = 5\}$. Here our reduction function is to check if the input word is one of the three words 0, 1, 01. If it is, we return some word with length 5, e.g. 00000. Otherwise, we return some word with length not 5, e.g. 0.

$$f : \Sigma^\star \to \Sigma^\star,\ f(w) = \begin{cases} 00000 & \text{if } w = 0 \vee w = 1 \vee w = 01 \\ 0 & \text{otherwise} \end{cases}$$

2) Analogously, we can show $A \leq \text{HP}$ with $A = \{0, 1, 01\}$ where

$$\text{HP} = \{M\#x \mid M \text{ halts on input } x\}$$

is the halting problem from the lecture. Again, we check if the input word is one of 0, 1, 01 and if yes, we return some word in HP, e.g. to M#0 where M is the encoding of a Turing Machine that always halts.

Otherwise, we must return some word that is not in HP, e.g. M'#0 where M' is the encoding of a machine that immediately loops. Another possibility would be to return a "garbage" word like 000 that does not even contain the separating character # and is therefore also not in HP.

$$f : \Sigma^\star \to \Sigma^\star,\ f(w) = \begin{cases} M\#0 & \text{if } w = 0 \vee w = 1 \vee w = 01 \\ 000 & \text{otherwise} \end{cases}$$

3) Define $\text{HP}_0 = \{M \mid M \text{ halts on input } 0\}$. This is a more restricted version of the halting problem where we do not get to choose the input word (it is instead fixed to 0), so it looks like HP_0 should be intuitively easier than the normal halting problem HP.

We can prove this by showing the reduction $\text{HP}_0 \leq \text{HP}$. Following the explanation above, we essentially are given a word w and must decide whether the Turing Machine M that it encodes halts on the input 0. We are also given an oracle that decides whether some arbitrary Turing Machine M' halts on some arbitrary input x (which we may choose). The obvious choice is $M' = M$ and $x = 0$ – and indeed that is our reduction function:

$$f : \Sigma^\star \to \Sigma^\star,\ f(M) = M\#0$$

4) Interestingly enough, we can however also prove the reduction $\text{HP} \leq \text{HP}_0$. That means that contrary to what one might expect, HP and HP_0 are actually equally difficult! The reduction function here does, however, require some more creativity.

   The situation is that we are given a word w and must decide whether $w \in \text{HP}$. If w does not contain exactly one #, it is obviously not in HP by definition. Otherwise we can split $w = u\#x$ and let M be the Turing Machine encoded by u.

   We must now decide whether M halts on the input x. Our oracle allows us to decide whether some arbitrary machine M' (that we get to choose) halts on the input 0. We now essentially need to trick the oracle into deciding our original problem for us.

   To do this, we construct a machine M' that deletes its tape, writes x onto the tape, and then runs M.[1]

   It is clear that if we now run M' on any input, the behaviour will be the same as running M on input x. Thus, if we feed M' to our oracle, we get the correct answer to "Is x in HP?".

   The only missing part now is that we need to map all the "garbage words" that do not contain exactly one # to something not in HP_0. This can be done by simply mapping them to a Turing Machine M_{loop} that always loops.

$$f : \Sigma^\star \to \Sigma^\star,\ f(w) = \begin{cases} M' & \text{if } w \text{ is of the form } u\#x \\ M_{\text{loop}} & \text{otherwise} \end{cases}$$

5) As an example of an incorrect reduction proof, consider the following reduction function:

$$f : \Sigma^\star \to \Sigma^\star,\ f(M) = \begin{cases} 1 & \text{if } M \text{ halts on input } 0 \\ 0 & \text{otherwise} \end{cases}$$

   This reduces HP_0 (which is not recursive, as we have shown above) to the (obviously recursive) set $\{0\}$, which can clearly not be. The problem with this "proof" is that the reduction function f is not computable.

[1] The details of how to actually construct that machine M' are not important, as long as the process is computable. You can think of it as simply adding a few extra states to the beginning of M that perform the deletion of the tape and writing x onto it.