

Reducibility

The purpose of this short handout is to give you additional intuition towards using the notion of reducibility for showing that some problems are hard.

Reductions were introduced using the notion of *mapping reducibility*:

Definition 1 (Mapping reducibility) *Language A is mapping reducible to language B , denoted $A \leq_m B$, if there exists a computable function f , where for every $x \in A$, $x \in A$ if and only if $f(x) \in B$. The function f is called the reduction of A to B .*

Because of the link between the language that f introduces (note the “if and only if” constraint), we can use the following logic to show that some languages are decidable or undecidable. More precisely, we have:

- If $A \leq_m B$ and B is decidable, then A is decidable.
- If $A \leq_m B$ and A is undecidable, then B is undecidable.
- If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.
- If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

The most common way for us to show that a language B is undecidable was by a proof by contradiction. That is, given an undecidable language A , we assume that a decider M_B exists for B and use it to construct a decider M_A for A . This results in a contradiction, which means that our initial assumption about B was wrong and B is undecidable.

Another approach is to construct the function f directly and show that both directions of the “if and only if” requirement hold. That is, given languages A and B , where A is known to be undecidable, we construct f and show that for every x in A , $x \in A \Rightarrow f(x) \in B$ and $f(x) \in B \Rightarrow x \in A$. This approach might be easier to think about in certain scenarios.

Let’s, for example, look at the problem of testing whether the language accepted by a TM is empty: $E_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$. Both approaches above involve (explicitly or implicitly) constructing the reduction function f . The solution we looked at used the first method performing reduction from language A_{TM} to E_{TM} , which is given below.

Assume by contradiction that decider M_B exists for E_{TM} . We use it to construct decider M_A for A_{TM} as follows:

$M_A =$ “On input $\langle M, x \rangle$:

1. Use M and x to construct a TM M_x that first checks its input. If the input is x , it works just like M . If the input is different from x , it automatically rejects it.
2. Run M_B on $\langle M_x \rangle$.
3. If M_B accepts, reject. If M_B rejects, accept.

Since M_A decides A_{TM} , we arrive at contradiction and conclude that E_{TM} is undecidable.

The second approach would be to just construct the reduction and prove that it works. That is, for this example, our reduction function takes M and x , $\langle M, x \rangle$, as input and outputs description of M_x , $\langle M_x \rangle$. We then have that if $\langle M, x \rangle \in A_{\text{TM}}$, then $L(M_x) = \{x\} \neq \emptyset$; and if $\langle M, x \rangle \notin A_{\text{TM}}$, then $L(M_x) = \emptyset$. Conversely, if $L(M_x) = \emptyset$, the only possible way for M_x to reject all strings is

when M rejects x , therefore $\langle M, x \rangle \notin A_{\text{TM}}$. Similarly, if $L(M_x) \neq \emptyset$, the only possibility for M_x to have any strings in the language is when M accepts x (since M_x is instructed to reject all other strings), thus $\langle M, x \rangle$ must be in A_{TM} . Thus, we showed that both directions of the requirement on f hold, with the exception of one problem: the decision whether $\overline{M_x}$ is in E_{TM} is the reverse of the decision of whether $\langle M, x \rangle$ is in A_{TM} . That is, our function maps $\overline{A_{\text{TM}}}$ to E_{TM} rather than A_{TM} to E_{TM} . Since $\overline{A_{\text{TM}}}$ is also undecidable, the claim that E_{TM} is undecidable will hold as well, but this should be explicitly stated/proved. Thus, the first method is easier in this case.

Now let's look at polynomial time reducibility. We had the following definition:

Definition 2 (Polynomial time reducibility) *Language A is polynomial time reducible to language B , denoted $A \leq_p B$, if there exists a polynomial time computable function f , where for every $x \in A$, $x \in A$ if and only if $f(x) \in B$. The function f is called the polynomial time reduction of A to B .*

With this definition in place, we had the following logic for classes P and NP:

- If $A \leq_p B$ and $B \in \text{P}$, then $A \in \text{P}$.
- If $A \leq_p B$ for $B \in \text{NP}$ and A is NP-complete, then B is NP-complete.

To show NP-completeness, we didn't use proofs by contradiction, but rather always constructed the mapping f and showed its properties, i.e., the second method above. The reason why proofs by contradictions don't work here is because we wouldn't arrive at contradiction. That is, using the logic of proofs of undecidability, we would assume that B is decidable in polynomial time and we can use its decider to construct a polynomial time decider for A (which implies a polynomial time solution for every problem in NP). But since we don't know whether $\text{P} = \text{NP}$ or not, we cannot claim that we arrived at contradiction. Thus, constructing a mapping that can be computed in polynomial time was the approach we took.