St. Francis Xavier University
Department of Computer Science

CSCI 541: Theory of Computing
Lecture 4: Reducibility
Fall 2023

1 Polynomial-Time Mapping Reductions

As we well know, there are many difficult decision problems in the class NP. We could solve such decision problems through a brute-force search of exponentially many possible solutions, but since we don't yet have an answer to the P vs. NP problem, we do not know whether there exists a much more efficient polynomial-time algorithm that would solve such difficult decision problems.

While we currently have no way of proving definitively that some decision problem A is in NP but not in P, we can be reasonably sure that A is not in P (assuming $\mathsf{P} \neq \mathsf{NP}$) by showing that every problem in NP can be transformed into an instance of A by way of a *mapping reduction* or, more generally, just a *reduction*. In this way, we establish that solving A is at least as hard as solving anything else in NP, and since we have no polynomial-time algorithms for the difficult problems in NP, we conclude that no polynomial-time algorithm is likely to exist to solve A. We could apply the same reduction idea to show that some decision problems are at least as hard as problems in other complexity classes, such as PSPACE.

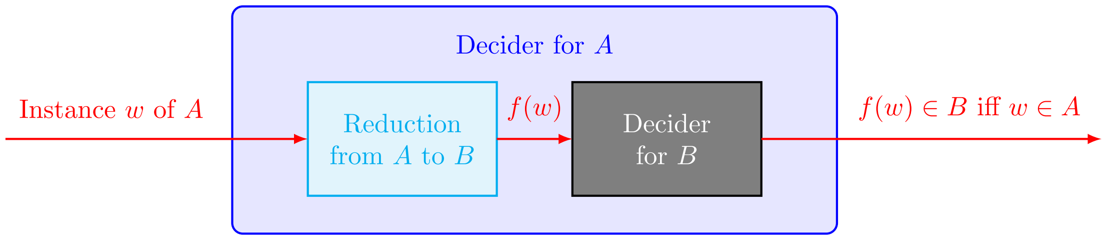
We encounter examples of reductions in real life every day, whether we realize it or not. For example, we can reduce the problem of finding a book in the library to the problem of searching for that book in the library's catalog system. If we use the catalog to find the book, then we can take the solution to that problem (the location of the book as listed in the catalog) and apply it to our original problem (finding the location of the book in the stacks). As another example, students can reduce the problem of staying awake in lectures to the problem of acquiring a coffee from the café.

Computationally speaking, a reduction is a process that converts an instance of some problem A to an equivalent instance of some other problem B. Specifically, this conversion is performed by a computable function.

Definition 1 (Mapping reduction)**.** Given two decision problems A and B, problem A is mapping reducible to problem B if there exists a computable function $f\colon \Sigma^* \to \Sigma^*$ where, for all $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.

In other terms, if A reduces to B, then we can transform every instance w of A to an instance $f(w)$ of B. Since $w \in A$ if and only if $f(w) \in B$, we know that the transformed instance will produce the same output as the original instance. As a result, we can use a reduction along with a decision algorithm for problem B to decide the original problem A.

Diagrammatically, we can visualize a mapping reduction from A to B in the following way:



We denote a mapping reduction from A to B by the notation $A \leq_m B$. Note that the direction of a reduction is important; if $A \leq_m B$, then we say that we reduce *from A to B*.

If we take a general mapping reduction, it's possible that the process of applying the reduction alone may blow up the resource usage of whatever algorithm we're reasoning about. This isn't desirable if, say, we're focused on a complexity class like P or NP where we must limit our algorithm to using only a polynomial amount of time. For this reason, we can place a limitation on our mapping reductions to likewise use only a polynomial amount of time, and to do so we need only change our definition to use polynomial-time computable functions; that is, functions that can be computed on a polynomial-time Turing machine.

Definition 2 (Polynomial-time mapping reduction)**.** Given two decision problems A and B, problem A is polynomial-time mapping reducible to problem B if there exists a polynomial-time computable function $f\colon \Sigma^* \to \Sigma^*$ where, for all $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.

Just like before, if A is polynomial-time reducible to B, then we can transform every instance w of A to an instance $f(w)$ of B in polynomial time. We denote a polynomial-time mapping reduction from A to B by the notation $A \leq_m^P B$.

1.1 Properties

We can prove a number of useful properties about polynomial-time mapping reductions. To begin, we will prove some facts about the mapping reduction relation itself.

Lemma 3. *The polynomial-time mapping reduction relation $\leq_m^P$ is reflexive and transitive.*

Proof. To show that $\leq_m^P$ is reflexive, take $f(x) = x$ as our polynomial-time computable function. Then $A \leq_m^P A$ for all decision problems A.

To show that $\leq_m^P$ is transitive, suppose that $A \leq_m^P B$ by way of some polynomial-time computable function f and $B \leq_m^P C$ by way of some other polynomial-time computable function g. Then $A \leq_m^P C$ by taking $h(x) = g(f(x))$ as our polynomial-time computable function. □

However, the mapping reduction relation is *not* symmetric. This means that if $A \leq_m^P B$ for some decision problems A and B, then it is not always the case that $B \leq_m^P A$ as well.

For our next property, we will show that if we have a polynomial-time mapping reduction between two decision problems A and B, then we also have a reduction between the complements of these two problems. This may come in handy if, for example, a reduction between the problems themselves may be conceptually difficult, but the complementary problems are comparatively easier to work with.

Lemma 4. *Given two decision problems A and B, $A \leq_m^P B$ if and only if $\overline{A} \leq_m^P \overline{B}$.*

Proof. Since $A \leq_m^P B$, we know by the definition of a polynomial-time mapping reduction that there exists some polynomial-time computable function f where, for all w, $w \in A$ if and only if $f(w) \in B$.

If $w \in \overline{A}$, then $w \notin A$, so $f(w) \notin B$ and thus $f(w) \in \overline{B}$. Similarly, if $w \notin \overline{A}$, then $w \in A$, so $f(w) \in B$ and thus $f(w) \notin \overline{B}$. Therefore, $w \in \overline{A}$ if and only if $f(w) \in \overline{B}$, and the same function f gives us the polynomial-time mapping reduction $\overline{A} \leq_m^P \overline{B}$. □

Lastly, recall that every decision problem corresponds to a language where the words in that language correspond to "yes" instances of the decision problem. Since decision problems and languages effectively use the same underlying idea, we can reduce a decision problem to a language.

Since we are using polynomial-time reductions, if our decision problem is already in the class P, then it turns out that we can reduce it to *any* non-trivial language (where "non-trivial" means that not all inputs produce the same output).

Lemma 5. *Given a decision problem A, if $A \in \mathsf{P}$, then A is polynomial-time reducible to any language other than $\emptyset$ and Σ^*.*

Proof. If $A \in \mathsf{P}$, then we can solve any instance of A directly in polynomial time. Then, for any non-trivial language B, we use the solution to our instance of A to choose the appropriate output: if $w \in A$, then we choose some element contained in B; and if $w \notin A$, then we choose some element not contained in B.

To see why we cannot reduce to the trivial language $\emptyset$, suppose $A \neq \emptyset$ and suppose there exists some polynomial-time computable function f where, for all w, $w \in A$ implies $f(w) \in \emptyset$. This immediately produces a contradiction, since $f(w) \in \emptyset$ can never be true.

To see why we cannot reduce to the trivial language Σ^*, we use a similar argument as for the language $\emptyset$. Suppose $A \neq \Sigma^*$. Then for any function f, it is possible that $w \notin A$, but $f(w) \in \Sigma^*$ is always true. □

Unfortunately, this fact presents an issue for us if we want to use mapping reductions on decision problems within the class P or any of its subclasses, like L. We will see how to overcome this difficulty later.

1.2 Closure

One other useful property of mapping reductions arises when we know in advance that some decision problem B belongs to a particular complexity class C. For certain complexity classes, we can prove that any decision problem A that reduces to some decision problem $B \in \mathsf{C}$ must also belong to C itself. If this is the case, then we say the class C is *closed* under mapping reductions.

Definition 6 (Closure under polynomial-time mapping reductions)**.** A complexity class C is closed under polynomial-time mapping reductions if, whenever $A \leq_m^P B$ and $B \in \mathsf{C}$, we have that $A \in \mathsf{C}$ as well.

The property of closure gives us a nice way of proving that a certain decision problem A belongs to some complexity class. If we start with a decision problem B that we know is in some complexity class, all we need is to come up with a reduction $A \leq_m^P B$, and this serves as our proof that A is in the same complexity class as B.

However, observe that we said closure holds only for *particular* complexity classes. While it is not the case that all complexity classes are closed under polynomial-time mapping reductions, we do have closure for the major complexity classes.

Lemma 7. *The classes* P*,* NP*,* PSPACE*, and* EXP *are closed under polynomial-time mapping reductions.*

Proof. We will prove closure for the class NP; proofs for the other classes are similar.

Let A and B be decision problems. Suppose that $A \leq_m^P B$ by way of some polynomial-time computable function f, where f is computed by a deterministic Turing machine $\mathcal{M}_f$ in polynomial time $p(n)$. Further suppose that $B \in \mathsf{NP}$; that is, suppose some nondeterministic Turing machine $\mathcal{N}$ recognizes B in polynomial time $q(n)$.

Construct a nondeterministic Turing machine $\mathcal{N}'$ that takes a word w as input and performs the following steps:

1. Simulate the computation of $\mathcal{M}_f$ on the input w, and write the result $f(w)$ to a work tape.
2. Simulate the computation of $\mathcal{N}$ on the input $f(w)$.
   (a) If $\mathcal{N}$ accepts, then accept.
   (b) Otherwise, reject.

Since $f(w)$ was computed by the Turing machine $\mathcal{M}_f$, we know that $|f(w)| \leq p(|w|)$, since $\mathcal{M}_f$ can write at most one symbol to its output tape per computation step. From this, we can conclude that the computation time of $\mathcal{N}'$ is upper-bounded by $p(|w|) + q(p(|w|))$, where the first term comes from the simulation of $\mathcal{M}_f$ and the second term comes from the simulation of $\mathcal{N}$.

Altogether, the Turing machine $\mathcal{N}'$ recognizes A by accepting w if and only if the Turing machine $\mathcal{N}$ accepts $f(w)$, and $\mathcal{N}'$ performs its computation in nondeterministic polynomial time. Thus, $A \in \mathsf{NP}$. □