St. Francis Xavier University
Department of Computer Science

CSCI 541: Theory of Computing
Lecture 6: Provers and Verifiers
Fall 2023

1 Interactive Proof Systems

Previously, when we defined and discussed the class NP, we spoke of the class in the context of a polynomial-time nondeterministic Turing machine that takes an input word w corresponding to an instance of a decision problem L and decides whether or not $w \in L$. This *decider*-based (or *prover*-based) definition is not the only way to define the class NP, though. We could alternatively define the class in terms of a *verifier*, or a polynomial-time deterministic Turing machine that takes both an input word w corresponding to an instance of a decision problem L as well as a polynomial-length *certificate*, or proof, that $w \in L$. The Turing machine then checks the validity of this certificate, which can be done far more efficiently than deciding the problem itself.

We can show that many decision problems belong to the class NP using either the prover-based definition or the verifier-based definition.

Example 1. Recall the Satisfiability decision problem, which is well-known to be NP-complete. Given a Boolean formula in conjunctive normal form, a nondeterministic prover can guess an assignment of values to variables in the formula and check whether the assignment is satisfying in polynomial time. On the other hand, a deterministic verifier can take as input the same Boolean formula as well as an assignment that claims to be satisfying, and check whether the assignment is indeed satisfying in polynomial time.

Example 2. Consider the *graph isomorphism* decision problem:

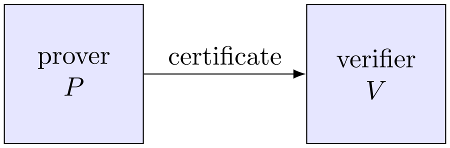
Graph-Isomorphism
Given: two undirected graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$
Determine: whether G and H are isomorphic; that is, whether $|V_G| = |V_H|$, $|E_G| = |E_H|$, and there exists a one-to-one function $f\colon V_G \to V_H$ where $\{u, v\} \in E_G$ if and only if $\{f(u), f(v)\} \in E_H$ for vertices $u, v \in V_G$

If G and H are isomorphic, we sometimes denote this by $G \cong H$.

Given two graphs G and H, a nondeterministic prover can guess such a mapping of vertices and edges and check whether it is an isomorphism in polynomial time. On the other hand, a deterministic verifier can take as input the same graphs G and H as well as a claimed isomorphism, and check whether it is indeed a valid isomorphism in polynomial time.

Remark. The graph isomorphism problem is known to be in NP, but interestingly, it is not known either to be in P or to be NP-complete. Graph isomorphism is an example of a decision problem believed to be NP-*intermediate*, and there are few natural examples of such decision problems.

Let's consider now what happens if we have both a prover and a verifier for a given decision problem. We can think of this setup as a game, where the prover needs to convince the verifier to accept. Here, the prover takes an input instance of the decision problem, computes a solution, and then passes off both the instance and its solution to the verifier. The verifier then checks the validity of the prover's output and either accepts or rejects as appropriate.



This model of computation, where a prover and a verifier exchange messages, is known as an *interactive proof system* (although it's not very "interactive" quite yet). And indeed, if we connect the prover and the verifier in this way, then we can see that the class NP is a very simple example of such an interactive proof system: if $w \in L$, then the prover computes the certificate directly, which is then checked by the verifier; and if $w \notin L$, then nothing the prover can pass off will be accepted by the verifier. In other words, the verifier does all the work of checking certificates, just as in the verifier-based definition of NP. Some complexity theorists, in fact, view NP as the class of decision problems that have "short proofs", or certificates with short (i.e., polynomial) length.

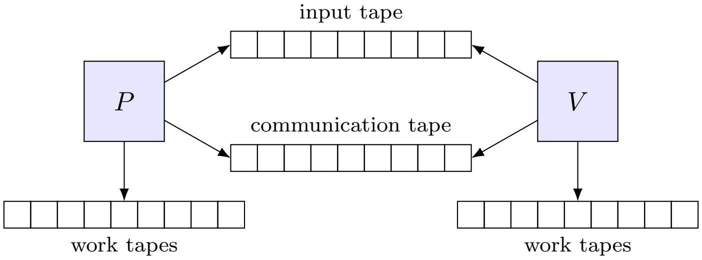
Of course, since the prover can only "talk to" the verifier once, and since the two machines can only "talk" in one direction, we don't get anything too interesting so far.

1.1 Interactivity

Let's change things up slightly by adding the following property to our prover-verifier pair:

Interactivity. The prover and the verifier can engage in two-way communication.

An interactive proof system employs an *interactive protocol*, where the computation proceeds in rounds with only one of the prover or verifier being active during any given round. Both the prover and the verifier have access to a shared read-only input tape, as well as to a shared read-write *communication tape.* In addition to these shared tapes, both the prover and the verifier have their own private read-write work tapes.



Effectively, interactivity allows the verifier to ask questions of the prover before it comes to a decision. Adding interactivity is akin to students asking questions of the professor in a lecture, compared to students simply reading the professor's lecture notes on their own.

In addition to having unlimited computational capability, the prover is allowed to act dishonestly. The verifier, meanwhile, has bounded computational capability, but always acts honestly. This sets the stage for a more interesting game between the two machines: as the prover-verifier pair engages interactively with one another, the prover repeatedly tries to get the verifier to accept by performing a computation and then writing a suitable output to the communication tape at the end of each prover round.

Both the prover and the verifier must also adhere to two properties in order to collectively define a decision problem.

Definition 3 (Interactive proof system)**.** A prover-verifier pair (P, V) defines an interactive proof system for a decision problem L if the following properties are satisfied:

- *Completeness.* If $w \in L$, then there exists a communication strategy where the prover P can always make the verifier V accept.
- *Soundness.* If $w \notin L$, then for any communication strategy used by the prover P, the verifier V will always reject.

As it turns out, even if we introduce interactivity, this procedure still only models the class NP! We can see why this is the case by considering inputs $w \in L$ and inputs $w \notin L$.

- If $w \in L$, then the prover can always make the verifier accept by simply writing the certificate showing that $w \in L$ to the communication tape.
- If $w \notin L$, then no matter how the prover acts, it can never provide a valid certificate to get the verifier to accept, as no such certificate exists.

In either case, having multiple rounds of interaction doesn't help, since if $w \in L$ then the prover can convince the verifier to accept in one round, and if $w \notin L$ then the prover can never convince the verifier regardless of the number of rounds.

1.2 Interactivity and Randomness

Therefore, let's change things up once more by adding another property to our prover-verifier pair:

Randomness. The verifier is a polynomial-time probabilistic Turing machine.

Here, our prover-verifier pair setup is largely the same as we had before, with both machines having access to some subset of tapes. But now, the verifier has access to an additional private read-only random tape.

With probabilistic computation being added into the mix, we can now associate probabilities of acceptance with each of the outcomes $w \in L$ and $w \notin L$, and we can modify our definitions of completeness and soundness appropriately.

Definition 4 (Interactive proof system—probabilistic def'n)**.** A prover-verifier pair (P, V) defines an interactive proof system for a decision problem L if the following properties are satisfied:

- *Completeness.* If $w \in L$, then there exists a communication strategy where $\mathbb{P}[(P, V)$ accepts $w] \geq 2/3$.
- *Soundness.* If $w \notin L$, then for any communication strategy, $\mathbb{P}[(P, V)$ accepts $w] \leq 1/3$.

Note that, as with our other discussions about probabilities of acceptance, the exact probabilities do not matter; most authors choose $2/3$ and $1/3$ as we used in our definition, but any probabilities can be used so long as there is an appropriate probability gap between correctly and incorrectly accepting input words.

You may wonder at this point why we added randomness to our prover-verifier pair in an attempt to gain more power. Thinking back to our observation that interactivity allowed the verifier to ask questions of the prover, having interactivity alone provides no benefit: since the verifier was deterministic, the prover could simply predict the verifier's questions in advance and include the answers in its certificate. A verifier that acts probabilistically allows us to sidestep this issue.

For another perspective of why randomness leads to an increase in power, consider a special interaction where the prover sends an empty certificate to the verifier. Then, since the prover is of no help to the verifier, the verifier can only handle decision problems $L \in \mathsf{P}$ as it has bounded (i.e., deterministic) computational capability. However, if the verifier is a probabilistic Turing machine, then it can handle decision problems $L \in \mathsf{BPP}$, and we know—as far as we know—that $\mathsf{P} \subseteq \mathsf{BPP}$. Thus, assuming $\mathsf{P} \neq \mathsf{BPP}$, randomization gives the verifier in our prover-verifier pair more computational power!

At the same time, why did we make only the verifier act randomly, and not the prover? It turns out that allowing the prover to act randomly doesn't change what we can compute with our prover-verifier pair—for any decision problem L, if a probabilistic prover can make the verifier accept with some probability, then the prover could instead use its unlimited computational capability to compute probabilities of the verifier accepting different outputs and choose its output deterministically in each round in such a way as to maximize the probability of the verifier accepting.

1.3 Decision Problems with Interactive Protocols: IP

Using the notion of an interactive proof system with a probabilistic verifier, we can define a special complexity class of all decision problems that can be solved using an interactive protocol.

Definition 5 (The class IP)**.** A decision problem L belongs to the complexity class IP if there exists a prover-verifier pair that engages in polynomially-many rounds of interaction and satisfies the properties of completeness and soundness as specified in Definition 4.

Remark. Note that if we allow the prover and probabilistic verifier to engage only in a constant number of rounds of interaction, we can define the so-called *Merlin-Arthur* and *Arthur-Merlin* protocols, denoted MA and $\mathsf{AM}[k]$, respectively. In the Merlin-Arthur protocol, the prover communicates with the verifier once, while in the Arthur-Merlin protocol, the prover and verifier can communicate k times. We can devote pages to each of these types of protocols alone, so we won't go into too much depth here other than to observe that $\bigcup_{k\geq 1} \mathsf{AM}\left[n^k\right] = \mathsf{IP}$.

As we observed, combining interactivity and randomness in our prover-verifier pair gives us more computational power. But how much power do we gain, exactly? Having interactivity on its own simply allowed us to model the class NP, so clearly we have that $\mathsf{NP} \subseteq \mathsf{IP}$. Less clearly, it's possible for us to show that $\mathsf{IP} \subseteq \mathsf{PSPACE}$. A remarkable result proved by the Israeli cryptographer Adi Shamir in 1990 shows that the power of interactive proof systems falls on the upper end of this complexity hierarchy.

Theorem 6 (Shamir's theorem)**.** $\mathsf{IP} = \mathsf{PSPACE}$.

Proof. Omitted. □

Shamir's theorem has quite an involved proof, and so we won't cover the proof here. We will, however, take a more in-depth look at what the class IP is capable of solving.

As a concrete example of the power of interactive proof systems, let's consider an example of a decision problem widely believed *not* to be in NP: the *graph nonisomorphism* problem.

Graph-Nonisomorphism
Given: two undirected graphs $G = (V, E_G)$ and $H = (V, E_H)$
Determine: whether G and H are not isomorphic

As before, if G and H are nonisomorphic, we sometimes denote this by $G \ncong H$. Note that if the vertex sets of G and H have different cardinalities, then they are clearly nonisomorphic, so we set $V_G = V_H = V$ to restrict ourselves to considering only "interesting" instances of the problem.

We can see immediately that the graph nonisomorphism problem is the complement of the graph isomorphism problem, and since graph isomorphism is in NP, graph nonisomorphism is therefore in coNP. While nobody yet knows whether graph nonisomorphism itself belongs also to NP, it is rather easy for us to prove the following.

Theorem 7. Graph-Nonisomorphism *is in* IP.

Proof. The following interactive protocol solves the graph nonisomorphism problem:

V1. Verifier randomly selects one of G and H, and applies a random permutation π to the vertex set.

P1. Prover receives permuted graph and computes which of G or H it received.

V2. Verifier accepts if prover guessed correctly.

We can prove the correctness of this interactive protocol by establishing both the completeness and soundness properties.

- To establish completeness, suppose $G \ncong H$. Then the permuted graph sent by the verifier can be isomorphic to only one of G or H. With unlimited computational capability, the prover will always be able to determine which of G or H it received.
- To establish soundness, suppose $G \cong H$. Then the permuted graph sent by the verifier will always be isomorphic to both G and H, and for a random permutation π, the distributions of the graphs $\pi(G)$ and $\pi(H)$ are the same. Since the prover has no distinguishing information to work from, it can do no better than guessing, and so the probability of acceptance is at most $1/2$. □

You may have noticed in our proof of correctness for the graph nonisomorphism interactive protocol that the probability of acceptance when $G \cong H$ was $1/2$, and not at most $1/3$ as we specified in Definition 4. Fortunately, this is not a problem; we can just run the interactive protocol twice and require that the verifier accept on both runs, and this would make the probability of acceptance $(1/2)^2 = 1/4 \leq 1/3$ as required.

2 Zero-Knowledge Proofs

In an interactive proof system, the prover tries to convince the verifier to accept by giving it some information. This means that, necessarily, the prover must give up some information it has computed in order to achieve the outcome it desires. Indeed, interaction is essential for us to be able to handle decision problems in PSPACE; without it, we would be limited to problems from the much smaller class BPP due to the verifier having to work alone. But what if the prover wants to convince the verifier to accept while keeping its information secret—is such a thing even possible with interactivity?

It's not too difficult to think of scenarios where one party might want to keep information secret from another party; in fact, this is pretty much the basis of all of cryptography! But, for the sake of a toy example to motivate ourselves, consider the following scenario.

Example 8. Two people, Peggy and Victor, work in a ring-shaped building with the entrance on one end and a locked door blocking the passage at the other end. Peggy wishes to prove to Victor that she knows the passcode to this door, but she doesn't want to share the passcode with Victor. Therefore, Peggy proposes the following experiment:

- Peggy will enter the building and walk down either the left or the right passageway without Victor watching.
- Victor, standing outside, will shout either "left" or "right" into the entrance.
- Assuming Peggy knows the passcode to the door, she will be able to exit the building from either the left or the right passageway.

If Peggy knows the passcode, then no matter what Victor shouts, she will be able to exit the building through the passageway Victor specified. If Peggy didn't know the passcode, however, then she would only be able to exit the building through the correct passageway with probability $1/2$, and repeating this experiment only a small number of times would reveal whether Peggy truly knows the passcode. Finally, and importantly, at no point during the experiment does Victor ever learn the passcode itself!

This scenario involving Peggy and Victor is an example of a special type of interactive protocol involving a *zero-knowledge proof.* In a zero-knowledge proof, the prover must prove to the verifier that it knows the solution to some decision problem without revealing any information about the solution itself; that is, the verifier can only learn that the decision problem *has* a valid solution, but not *what* that solution is.

Remark. Note that a zero-knowledge proof isn't really a "proof" in the mathematical sense, since the verifier is still probabilistic and may (with small probability) incorrectly accept a word that it shouldn't accept. However, just as we saw with our interactive protocol for graph nonisomorphism, we can repeat the number of interactions between the prover and the verifier to minimize this *soundness error.*

An interactive protocol that uses zero-knowledge proofs is sometimes referred to as a *zero-knowledge protocol.* To formalize our notion of zero-knowledge, all we need is to add one property to our existing definition of an interactive proof system.

Definition 9 (Zero-knowledge interactive proof system)**.** A zero-knowledge prover-verifier pair (P, V) defines a zero-knowledge interactive proof system for a decision problem L if the following properties are satisfied:

- *Completeness.* If $w \in L$, then there exists a communication strategy where $\mathbb{P}[(P, V) \text{ accepts } w] \geq 2/3$.
- *Soundness.* If $w \notin L$, then for any communication strategy, $\mathbb{P}[(P, V) \text{ accepts } w] \leq 1/3$.

- *Zero-knowledge.* There exists a polynomial-time probabilistic Turing machine S where, for any $w \in L$, S computes a distribution of tuples of words $(x_1, \dots, x_n)$ such that the distribution of the contents of the communication tape of (P, V) is equal to the distribution produced by S.

We can satisfy the zero-knowledge property in other terms by showing that there exists some *simulator* S that takes as input whatever is to be proved, is given no access to the prover P, and produces an output that resembles an interaction between P and V. The existence of this simulator suggests that the verifier V does not gain any knowledge from the prover P, since S can produce the same output as (P, V) with no access whatsoever to P.

For an example of a zero-knowledge interactive proof system, let's return to our familiar graph isomorphism problem. Since graph isomorphism is in NP, it's not surprising that we can devise an interactive protocol for this problem.

Theorem 10. Graph-Isomorphism *is in* IP.

Proof. The following interactive protocol solves the graph isomorphism problem:

P1. Prover guesses one of G and H as well as a random permutation π, and applies π to the vertex set of the chosen graph.

V1. Verifier receives permuted graph and guesses which of G or H it received.

P2. Prover finds a permutation σ such that, when σ is applied to the verifier's chosen graph, we obtain the prover's chosen graph.

V2. Verifier accepts if their chosen graph permuted with σ matches the prover's chosen graph.

We can prove the correctness of this interactive protocol by establishing both the completeness and the soundness properties.

- To establish completeness, suppose $G \cong H$. Then both graphs are isomorphic to the prover's permuted graph, and the prover can always find a permutation σ that works.
- To establish soundness, suppose $G \ncong H$. Then only one of G or H is isomorphic to the prover's permuted graph. After the verifier guesses, the prover has probability at most $1/2$ to find a permutation σ that works. □

Once again, with this interactive protocol, we can reduce the soundness error from $1/2$ to $1/4$ by simply running the protocol twice and requiring the verifier to accept both times.

What may be more surprising than the fact that the graph isomorphism problem has an interactive protocol is the fact that it has a zero-knowledge protocol! To see why this is the case, observe that over the course of the computation, the communication tape contains three key pieces of information:

- the prover's randomly permuted graph (from step P1);
- the verifier's guessed graph (from step V1); and
- the prover's permutation σ (from step P2).

Note that at no point during the computation is the prover's original permutation π revealed, and so the verifier gains no knowledge about the prover's chosen graph. Indeed, we can define a polynomial-time probabilistic Turing machine S that acts as a simulator in the following way: S writes to its output tape a random permutation σ and a random value denoting a "guess", and with this we can apply σ to either of the graphs G or H depending on the "guess". Since S guesses both the random permutation and the random value in the same way that the prover guesses one of G or H together with π at random, evidently the distributions of these guesses are the same. Therefore, S is capable of producing the same output as the interactive proof system, and so the verifier gains no additional knowledge from the prover during their interactions.