3

Definitions and Technical Preliminaries

3.1 Interactive Proofs

Given a function f mapping $\{0,1\}^n$ to a finite range $\mathcal{R}$, a k-message *interactive proof system* (IP) for f consists of a probabilistic verifier algorithm $\mathcal{V}$ running in time poly(n) and a prescribed ("honest") deterministic prover algorithm $\mathcal{P}$.[1,2] Both $\mathcal{V}$ and $\mathcal{P}$ are given a common input $x \in \{0,1\}^n$, and at the start of the protocol $\mathcal{P}$ provides a value y claimed to equal $f(x)$. Then $\mathcal{P}$ and $\mathcal{V}$ exchange a sequence of messages $m_1, m_2, \ldots, m_k$ that are determined as follows. The IP designates one of the parties, either $\mathcal{P}$ or $\mathcal{V}$, to send the first message m_1. The party

[1] In general, one may consider defining IPs to permit probabilistic prover strategies. However, as explained in Section 3.3, it is without loss of generality to restrict attention to deterministic prover strategies.

[2] The choice of domain $\{0,1\}^n$ in this section is not essential, but rather made by convention and for convenience. One reason $\{0,1\}^n$ is a convenient domain is that, in order to express a proof system's costs (e.g., prover time and verifier time) in terms of the size of the input, we need a well-defined notion of input size, and if the input domain is all n-bit strings, then n is the natural such measure.

sending each message alternates, meaning for example that if $\mathcal{V}$ sends m_1, then $\mathcal{P}$ sends m_2, $\mathcal{V}$ sends m_3, $\mathcal{P}$ sends m_4, and so on.[3]

Both $\mathcal{P}$ and $\mathcal{V}$ are thought of as "next-message-computing algorithms", meaning that when it is $\mathcal{V}$'s (respectively, $\mathcal{P}$'s) turn to send a message m_i, $\mathcal{V}$ (respectively, $\mathcal{P}$) is run on input $(x, m_1, m_2, \ldots, m_{i-1})$ to produce message m_i. Note that since $\mathcal{V}$ is probabilistic, any message m_i sent by $\mathcal{V}$ may depend on both $(x, m_1, m_2, \ldots, m_{i-1})$ and on the verifier's internal randomness.

The entire sequence of k messages $t := (m_1, m_2, \ldots, m_k)$ exchanged by $\mathcal{P}$ and $\mathcal{V}$, along with the claimed answer y, is called a *transcript*. At the end of the protocol, $\mathcal{V}$ must output either 0 or 1, with 1 indicating that the verifier accepts the prover's claim that $y = f(x)$ and 0 indicating that the verifier rejects the claim. The value output by the verifier at the end of the protocol may depend on both the transcript t and the verifier's internal randomness.

Denote by $\mathrm{out}(\mathcal{V}, \mathrm{x}, \mathrm{r}, \mathcal{P}) \in \{0,1\}$ the output of verifier $\mathcal{V}$ on input x when interacting with deterministic prover strategy $\mathcal{P}$, with $\mathcal{V}$'s internal randomness equal to r. For any fixed value r of $\mathcal{V}$'s internal randomness, $\mathrm{out}(\mathcal{V}, \mathrm{x}, \mathrm{r}, \mathcal{P})$ is a deterministic function of x (as we have restricted our attention to deterministic prover strategies $\mathcal{P}$).

Definition 3.1. An interactive proof system $(\mathcal{V}, \mathcal{P})$ is said to have completeness error δ_c and soundness error δ_s if the following two properties hold.

(1) *(Completeness)* For every $x \in \{0,1\}^n$,

$$\Pr_r[\mathrm{out}(\mathcal{V}, \mathrm{x}, \mathrm{r}, \mathcal{P}) = 1] \geq 1 - \delta_{\mathrm{c}}.$$

(2) *(Soundness)* For every $x \in \{0,1\}^n$ and *every* deterministic prover strategy $\mathcal{P}'$, if $\mathcal{P}'$ sends a value $y \neq f(x)$ at the start of the

[3] Without loss of generality, the final message m_k is sent by the prover. There is no point in having the verifier send a message to the prover if the prover is not going to respond to it.

protocol, then

$$\Pr_r[\text{out}(\mathcal{V}, \mathrm{x}, \mathrm{r}, \mathcal{P}') = 1] \leq \delta_{\mathrm{s}}.$$

An interactive proof system is valid if $\delta_c, \delta_s \leq 1/3$.

Intuitively, for any input x, the completeness condition requires that there be a convincing proof for what is the value of f on input x. The soundness condition requires that false statements of the form "$f(x) = y$" for any $y \neq f(x)$ lack a convincing proof. That is, there is no cheating prover strategy $\mathcal{P}'$ that can convince $\mathcal{V}$ to accept a false claim with probability more than $1/3$.

The two costs of paramount importance in any interactive proof are $\mathcal{P}$'s runtime and $\mathcal{V}$'s runtime, but there are other important costs as well: $\mathcal{P}$'s and $\mathcal{V}$'s space usage, the total number of bits communicated, and the total number of messages exchanged. If $\mathcal{V}$ and $\mathcal{P}$ exchange k messages, then $\lceil k/2 \rceil$ is referred to as the *round complexity* of the interactive proof system.[4] The round complexity is the number of "back-and-forths" in the interaction between $\mathcal{P}$ and $\mathcal{V}$. If k is odd, then the final "back-and-forth" in the interaction is really just a "back" with no "forth", i.e., it consists of only one message from prover to verifier.

Interactive proofs were introduced in 1985 by Goldwasser *et al.* [133] and Babai [15].[5]

3.2 Argument Systems

Definition 3.2. An *argument system* for a function f is an interactive proof for f in which the soundness condition is only required to hold against prover strategies that run in polynomial time.

[4] Be warned that the literature is not consistent with regard to the meaning of the term "rounds". Vexingly, many papers use the terms rounds and messages interchangeably.

[5] More precisely, [133] introduced IPs, while Babai (with different motivations) introduced the so-called *Arthur-Merlin class hierarchy*, which captures constant-round interactive proof systems, with the additional requirement that the verifier's randomness is public—that is, any coin tossed by $\mathcal{V}$ is made visible to the prover as soon as it is tossed. See Section 3.3 for discussion of public vs. private verifier randomness.

The notion of soundness in Definition 3.2 is called *computational soundness*. Computational soundness should be contrasted with the notion of soundness in Definition 3.1, which is required to hold even against computationally unbounded provers $\mathcal{P}'$ that might be devoting enormous computational resources to trying to trick $\mathcal{V}$ into accepting an incorrect answer. The soundness notion from Definition 3.1 is referred to as *statistical soundness* or *information-theoretic soundness*.

Argument systems were introduced by Brassard *et al.* in 1986 [77]. They are sometimes referred to as *computationally sound proofs*, but in this monograph we will mainly use the term "proof" to refer to statistically sound protocols.[6] Unlike interactive proofs, argument systems are able to utilize cryptographic primitives. While a super-polynomial time prover may be able to break the primitive and thereby trick the verifier into accepting an incorrect answer, a polynomial time prover will be unable to break the primitive. The use of cryptography often allows argument systems to achieve additional desirable properties that are unattainable for interactive proofs, such as reusability (i.e., the ability for the verifier to reuse the same "secret state" to outsource many computations on the same input), public verifiability, etc. These properties will be discussed in more detail later in this survey.

3.3 Robustness of Definitions and the Power of Interaction

At first glance, it may seem that a number of aspects of Definitions 3.1 and 3.2 are somewhat arbitrary or unmotivated. For example, why does Definition 3.1 insist that the soundness and completeness errors be at most 1/3, and not some smaller number? Why does the completeness condition in Definition 3.1 demand that the honest prover is deterministic? And so forth. As we explain in this section, many of these choices are made for convenience or aesthetic reasons—the power of IPs and arguments are largely unchanged if different choices are made in the

[6] The main exception is in Section 18, where we use the term "SNARK proof π" to refer to a string π that convinces the verifier of a non-interactive argument system to accept. This terminology is unambiguous because the acronym SNARK, which is short for Succinct Non-interactive ARgument of Knowledge, clarifies that the protocol at hand is an argument system.

definitions.[7] The remarks in this section are somewhat technical and may be skipped with no loss of continuity.

- (Perfect vs. Imperfect Completeness) While Definition 3.1 required that the completeness error $\delta_c < 1/3$, all of the interactive proofs that we will see in this monograph actually satisfy *perfect* completeness, meaning that $\delta_c = 0$. That is, the honest prover in our IPs and arguments will *always* convince the verifier that it is honest.

  It is actually known [119] that any IP for a function f with $\delta_c \leq 1/3$ can be transformed into an IP for f with perfect completeness, with a polynomial blowup in the verifier's costs (e.g., verifier time, round complexity, communication complexity).[8] We will not need such transformations in this monograph, because the IPs we give will naturally satisfy perfect completeness.

- (Soundness Error) While Definition 3.1 required the soundness error δ_s to be at most $1/3$, the constant $1/3$ is merely chosen by convention. In all of the interactive proofs that we see in this survey, the soundness error will always be proportional to $1/|\mathbb{F}|$, where $\mathbb{F}$ is the field over which the interactive proof is defined. In practice, the field will typically be chosen large enough so that the soundness error is astronomically small (e.g., smaller than, say, 2^{-128}). Such tiny soundness error is essential in cryptographic applications, where a cheating prover successfully tricking a verifier to accept a false claim can have catastrophic effects. Soundness error of any IP or argument can also be generically reduced from δ_s to δ_s^k by

[7] Generally speaking, robustness to tweaks in the definition is a hallmark of a "good" notion or model in complexity theory. If the power of a model is highly sensitive to idiosyncratic or arbitrary choices in its definition, then the model may have limited utility and be unlikely to capture fundamental real-world phenomena. After all, the real world is messy and evolving—the hardware people use to compute is complicated and changes over time, protocols get used in a variety of different settings, etc. Robustness of a model to various tweaks helps ensure that any protocols in the model are useful in a variety of different settings and will not be rendered obsolete by future changes in technology.

[8] The transformation does *not* necessarily preserve the prover's runtime.

repeating the protocol $\Theta(k)$ times in sequence and rejecting unless the verifier accepts in a majority of the repetitions.[9]

- (Public vs. Private Randomness) In an interactive proof system, $\mathcal{V}$'s randomness is internal, and in particular is not visible to the prover. This is referred to in the literature as *private randomness*. One can also consider IPs in which the verifier's randomness is public—that is, any coin tossed by $\mathcal{V}$ is made visible to the prover as soon as it is tossed. We will see that such *public-coin* IPs are particularly useful, because they can be combined with cryptography to obtain argument systems with important properties (see Section 5 on the Fiat-Shamir transformation).

  Goldwasser and Sipser [136] showed that the distinction between public and private coins is not crucial: any private coin interactive proof system can be simulated by a public coin system (with a polynomial blowup in costs for the verifier, and a small increase in the number of rounds). As with perfect vs. imperfect completeness, we will not need to utilize such transformations in this monograph because all of the IPs that we give are naturally public coin protocols.

- (Deterministic vs. Probabilistic Provers) Definition 3.1 demands that the honest prover strategy $\mathcal{P}$ be deterministic, and only requires soundness to hold against deterministic cheating prover strategies $\mathcal{P}'$. Restricting attention to deterministic prover strategies in this manner is done only for convenience, and does not alter the power of interactive proofs.

  Specifically, if there is a probabilistic prover strategy $\mathcal{P}'$ that convinces the verifier $\mathcal{V}$ to accept with probability at least p (with the probability taken over both the prover's internal randomness and the verifier's internal randomness), then there is a deterministic prover strategy achieving the same. This follows from an averaging argument over the prover's randomness: if a probabilistic prover $\mathcal{P}'$ convinces $\mathcal{V}$ to accept a claim "$f(x) = y$" with probability p,

[9] For perfectly complete protocols, the verifier may reject unless *every* repetition of the base protocol leads to acceptance.

there must be at least one setting of the internal randomness r' of $\mathcal{P}'$ such that the deterministic prover strategy obtained by fixing the randomness of $\mathcal{P}'$ to r' also convinces the verifier to accept the claim "$f(x) = y$" with probability p. (Note that the value r' may depend on x). In this monograph, the honest prover in all of our IPs and arguments will naturally be deterministic, so we will have no need to exploit this generic transformation from randomized to deterministic prover strategies.[10]

Interactive Proofs for Languages Versus Functions. Complexity theorists often find it convenient to study *decision problems*, which are functions f with range $\{0,1\}$. We think of decision problems as "yes-no questions", in the following manner: any input x to f is interpreted as a question, namely: "Does $f(x)$ equal 1?". Equivalently, we can associate any decision problem f with the subset $\mathcal{L} \subseteq \{0,1\}^n$ consisting of "yes-instances" for f. Any subset $\mathcal{L} \subseteq \{0,1\}^n$ is called a *language*.

The formalization of IPs for languages differs slightly from that for functions (Definition 3.1). We briefly describe this difference because celebrated results in complexity theory regarding the power of IPs and their variants (e.g., **IP** = **PSPACE** and **MIP** = **NEXP**) refer to IPs for languages.

In an interactive proof for the language $\mathcal{L}$, given a public input $x \in \{0,1\}^n$, the verifier $\mathcal{V}$ interacts with a prover $\mathcal{P}$ in exactly the same manner as in Definition 3.1 and at the end of the protocol $\mathcal{V}$ must output either 0 or 1, with 1 corresponding to "accept" and 0 corresponding to "reject". The standard requirements of an IP for the language $\mathcal{L}$ are:

- **Completeness**. For any $x \in \mathcal{L}$, there is some prover strategy that will cause the verifier to accept with high probability.
- **Soundness**. For any $x \notin \mathcal{L}$, then for *every* prover strategy, the verifier rejects with high probability.

[10] An important caveat is that for most of the *zero-knowledge* proofs and arguments considered in Sections 11–17 in this monograph, the prover will be randomized. This randomization of the proof has no bearing on the completeness or soundness of the protocol, but rather is incorporated as a means of ensuring that the proof leaks no information to the verifier (other than its own validity).

Given a language $\mathcal{L}$, let $f_{\mathcal{L}}\colon \{0,1\}^n \to \{0,1\}$ be the corresponding decision problem, i.e., $f_{\mathcal{L}}(x) = 1$ if x is in $\mathcal{L}$, and $f_{\mathcal{L}}(x) = 0$ if x is not in $\mathcal{L}$. Note that for $x \notin \mathcal{L}$, the above definition of an IP for $\mathcal{L}$ does *not* require that there be a "convincing proof" of the fact that $f_{\mathcal{L}}(x) = 0$. This is in contrast to the definition of IPs for the *function* $f_{\mathcal{L}}$ (Definition 3.1), for which the completeness requirement insists that for *every* input x (even those for which $f_{\mathcal{L}}(x) = 0$), there be a prover strategy that convinces the verifier of the value of $f(x)$.

The motivation behind the above formalization of IPs for languages is as follows. One may think of inputs in the language $\mathcal{L}$ as *true statements*, and inputs not in the language as *false statements*. The above completeness and soundness properties require that all true statements have convincing proofs, and all false statements do not have convincing proofs. It is natural *not* to require that false statements have convincing refutations (i.e., convincing proofs of their falsity).

While the notions of interactive proofs for languages and functions are different, they are related in the following sense: given a *function* f, an interactive proof for f is equivalent to an interactive proof for the *language* $\mathcal{L}_f := \{(x,y)\colon y = f(x)\}$.

As indicated above, in this monograph we will primarily be concerned with interactive proofs for functions instead of languages. We only talk about interactive proofs for languages when referring to complexity classes such as **NP** and **IP**, defined next.

NP and IP. Let **IP** be the class of all languages solvable by an interactive proof system with a polynomial time verifier. The class **IP** can be viewed as an interactive, randomized variant of the classical complexity class **NP** (**NP** is the class obtained from **IP** by restricting the proof system to be non-interactive and deterministic, meaning that the completeness and soundness errors are 0).

We will see soon that the class **IP** is in fact equal to **PSPACE**, the class of all languages solvable by algorithms using polynomial space (and possibly exponential time). **PSPACE** is believed to be a vastly bigger class of languages than **NP**, so this is one formalization of the statement that "interactive proofs are far more powerful than classical static (i.e, **NP**) proofs".

By Your Powers Combined, I am IP. The key to the power of interactive proofs is the *combination* of randomness and interaction. If randomness is disallowed (equivalently, if perfect soundness $\delta_s = 0$ is required), then interaction is pointless, because the prover can predict the verifier's messages with certainty, and hence there is no reason for the verifier to send the messages to the prover. In more detail, the proof system can be rendered non-interactive by demanding that the (non-interactive) prover send a transcript of the interactive protocol that would cause the (interactive) verifier to accept, and the (non-interactive) verifier can check that indeed the (interactive) verifier would have accepted this transcript. By perfect soundness of the interactive protocol, this non-interactive proof system is perfectly sound.

On the other hand if no interaction is allowed, but the verifier is allowed to toss random coins and accept an incorrect proof with small probability, the resulting complexity class is known as **MA** (short for *Merlin-Arthur*). This class is widely believed to be equal to **NP** (see for example [155]), which as stated above is believed by many researchers to be a much smaller class of problems than **IP** = **PSPACE**.[11]

3.4 Schwartz-Zippel Lemma

Terminology. For an m-variate polynomial g, the degree of a term of g is the sum of the exponents of the variables in the term. For example if $g(x_1, x_2) = 7x_1^2x_2 + 6x_2^4$, then the degree of the term $7x_1^2x_2$ is 3, and the degree of the term $6x_2^4$ is 4. The total degree of g is the maximum of the degree of any term of g, which in the preceding example is 4.

[11] More precisely, it is widely believed that for every non-interactive randomized proof system $(\mathcal{V}, \mathcal{P})$ for a language $\mathcal{L}$, there is a non-interactive deterministic proof system $(\mathcal{V}', \mathcal{P}')$ for $\mathcal{L}$ in which the runtime of the deterministic verifier $\mathcal{V}'$ is at most polynomially larger than that of the randomized verifier $\mathcal{V}$. This would not necessarily mean that the deterministic verifier $\mathcal{V}'$ is *just as fast* as the randomized verifier $\mathcal{V}$. See for example Freivald's non-interactive randomized proof system for matrix multiplication in Section 2.2—the verifier there runs in $O(n^2)$ time, which is faster than any known deterministic verifier for the same problem, but "only" by a factor of about $O(n^{0.3728639})$, which is a (small) polynomial in the input size. This is in contrast to the transformation of the preceding paragraph from deterministic interactive proofs to non-interactive proofs, which introduces no overhead for either the verifier or the prover.

The lemma itself. Interactive proofs frequently exploit the following basic property of polynomials, which is commonly known as the Schwartz-Zippel lemma [224], [260].

Lemma 3.3 (Schwartz-Zippel Lemma)**.** Let $\mathbb{F}$ be any field, and let g: $\mathbb{F}^m \to \mathbb{F}$ be a nonzero m-variate polynomial of total degree at most d. Then on any finite set $S \subseteq \mathbb{F}$,

$$\Pr_{x \leftarrow S^m}[g(x) = 0] \leq d/|S|.$$

Here, $x \leftarrow S^m$ denotes an x drawn uniformly at random from the product set S^m, and $|S|$ denotes the size of S. In words, if x is chosen uniformly at random from S^m, then the probability that $g(x) = 0$ is at most $d/|S|$. In particular, any two distinct polynomials of total degree at most d can agree on at most a $d/|S|$ fraction of points in S^m.

We will not prove the lemma above, but it is easy to find a proof online (see, e.g., the wikipedia article on the lemma, or an alternative proof due to Moshkovitz [194]). An easy implication of the Schwartz-Zippel lemma is that for any two distinct m-variate polynomials p and q of total degree at most d over $\mathbb{F}$, $p(x) = q(x)$ for at most a $d/|\mathbb{F}|$ fraction of inputs. Section 2.1.1 on Reed-Solomon fingerprinting exploited precisely this implication in the special case of univariate polynomials (i.e., $m = 1$).

3.5 Low Degree and Multilinear Extensions

Motivation and Comparison to Univariate Lagrange Interpolation. In Section 2.4, we considered any *univariate* function f mapping $\{0, 1, \ldots, n-1\}$ to $\mathbb{F}_p$, and studied the univariate low-degree extension of f. This was the unique univariate polynomial g over $\mathbb{F}_p$ of degree at most $n-1$ such that $g(x) = f(x)$ for all $x \in \{0, 1, \ldots, n-1\}$. In this section, we consider *multivariate* functions f, more specifically defined over the v-variate domain $\{0,1\}^v$. Note that when $v = \log n$, the domain $\{0,1\}^v$ has the same size as the univariate domain $\{0, 1, \ldots, n-1\}$.

As we will see, functions defined over the domain $\{0,1\}^v$ have extension polynomials that have much lower degree than in the univariate case. Specifically, any function f mapping $\{0,1\}^v \to \mathbb{F}$ has an extension

polynomial that is *multilinear*, meaning it has degree at most 1 in each variable. This implies that the *total degree* of the polynomial is at most v, which is logarithmic in the domain size 2^v. In contrast, univariate low-degree extensions over a domain of size n require degree $n-1$. Multivariate polynomials with ultra-low degree in each variable turn out to be especially useful when designing interactive proofs with small communication and fast verification.

Details of Polynomial Extensions for Multivariate Functions. Let $\mathbb{F}$ be any finite field, and let $f\colon \{0,1\}^v \to \mathbb{F}$ be any function mapping the v-dimensional Boolean hypercube to $\mathbb{F}$. A v-variate polynomial g over $\mathbb{F}$ is said to be an *extension* of f if g agrees with f at all Boolean-valued inputs, i.e., $g(x) = f(x)$ for all $x \in \{0,1\}^v$. Here, the domain of the v-variate polynomial g over $\mathbb{F}$ is $\mathbb{F}^v$, and 0 and 1 are respectively associated with the additive and multiplicative identity elements of $\mathbb{F}$.

As with univariate low-degree extensions, one can think of a (low-degree) extension g of a function $f\colon \{0,1\}^v \to \mathbb{F}$ as a distance-amplifying encoding of f: if two functions $f, f'\colon \{0,1\}^v \to \mathbb{F}$ disagree at even a single input, then any extensions g, g' of total degree at most d must differ *almost everywhere*, assuming $d \ll |\mathbb{F}|$.[12] This is made precise by the Schwartz-Zippel lemma above, which guarantees that g and g' agree on at most $d/|\mathbb{F}|$ fraction of points in $\mathbb{F}^v$. As we will see throughout this survey, these distance-amplifying properties give the verifier surprising power over the prover.[13]

[12] As with Footnote 10 in Section 2.4, the univariate setting, precisely how small d must be for a degree-d extension polynomial g to be called "low-degree" is deliberately left vague and may be context-dependent. At a minimum, d should be less than $|\mathbb{F}|$ to ensure that the probability $d/|\mathbb{F}|$ appearing in the Schwartz-Zippel lemma is less than 1; otherwise, the Schwartz-Zippel lemma is vacuous. When a low-degree extension g is used in interactive proofs or arguments, various costs of the protocol, such as proof size, verifier time, or prover time, often grow linearly with the degree d of g, and hence the smaller d is, the lower these costs are.

[13] In fact, the use of low-degree extensions in many of the interactive proofs and arguments we describe in this survey could in principle be replaced with different distance-amplifying encodings that do not correspond to polynomials at all (see for example [190], [215] for papers in this direction). However, we will see that low-degree extensions have nice structure that enables the prover and verifier to run especially efficiently when we use low-degree extensions rather than general distance-amplifying

Definition 3.4. A multivariate polynomial g is *multilinear* if the degree of the polynomial in each variable is at most one.

For example, the polynomial $g(x_1, x_2) = x_1x_2 + 4x_1 + 3x_2$ is multilinear, but the polynomial $h(x_1, x_2) = x_2^2 + 4x_1 + 3x_2$ is not. Throughout this survey, we will frequently use the following fact.

Fact 3.5. Any function $f\colon \{0,1\}^v \to \mathbb{F}$ has a unique *multilinear* extension (MLE) over $\mathbb{F}$, and we reserve the notation $\widetilde{f}$ for this special extension of f.

That is, $\widetilde{f}$ is the unique multilinear polynomial over $\mathbb{F}$ satisfying $\widetilde{f}(x) = f(x)$ for all $x \in \{0,1\}^v$. See Figures 3.1 and 3.2 for an example of a function and its multilinear extension.

The first step in the proof of Fact 3.5 is to establish the existence of a multilinear polynomial extending f. In fact, we give an explicit expression for this polynomial, via Lagrange interpolation. This is analogous to Lemma 2.3 in Section 2.3, which considered the case of univariate rather than multilinear polynomials.

Lemma 3.6 (Lagrange Interpolation of Multilinear Polynomials)**.** Let $f\colon \{0,1\}^v \to \mathbb{F}$ be any function. Then the following multilinear polynomial $\widetilde{f}$ extends f:

$$\widetilde{f}(x_1, \ldots, x_v) = \sum_{w \in \{0,1\}^v} f(w) \cdot \chi_w(x_1, \ldots, x_v), \tag{3.1}$$

where, for any $w = (w_1, \ldots, w_v)$,

$$\chi_w(x_1, \ldots, x_v) \coloneqq \prod_{i=1}^{v} (x_i w_i + (1 - x_i)(1 - w_i)). \tag{3.2}$$

The set $\{\chi_w\colon w \in \{0,1\}^v\}$ is referred to as the set of *multilinear Lagrange basis polynomials* with interpolating set $\{0,1\}^v$.

Proof. For any vector $w \in \{0,1\}^v$, χ_w satisfies $\chi_w(w) = 1$, and $\chi_w(y) = 0$ for all other vectors $y \in \{0,1\}^v$. To see that the latter property holds,

encodings. It remains an important research direction to obtain IPs and arguments with similar (or better!) efficiency by using non-polynomial encodings—Section 10.5 of this survey covers one result in this vein.

	0	1
0	1	2
1	1	4

Figure 3.1: All evaluations of a function f mapping $\{0,1\}^2$ to the field $\mathbb{F}_5$.

observe that if $w_i \neq y_i$, then either $w_i = 1$ and $y_i = 0$ or $w_i = 0$ and $y_i = 1$. Either way, the ith term on the right hand side of Equation (3.2), namely $(x_i w_i + (1 - x_i)(1 - w_i))$, equals 0. This ensures that the entire product on the right hand side of Equation (3.2) equals 0.

It follows that $\sum_{w \in \{0,1\}^v} f(w) \cdot \chi_w(y) = f(y)$ for all Boolean vectors $y \in \{0,1\}^v$. In addition, the right hand side of Equation (3.1) is a multilinear polynomial in $(x_1, \ldots, x_v)$, as each term of the sum is clearly a multilinear polynomial, and a sum of multilinear polynomials is itself multilinear. Putting these two statements together, the right hand side of Equation (3.1) is a multilinear polynomial extending f. □

Lemma 3.6 demonstrated that for any function $f\colon \{0,1\}^v \to \mathbb{F}$, there is some multilinear polynomial that extends f. To complete the proof of Fact 3.5, we must establish that there is only one such polynomial.

	0	1	2	3	4
0	1	2	3	4	0
1	1	4	2	0	3
2	1	1	1	1	1
3	1	3	0	2	4
4	1	0	4	3	2

Figure 3.2: All evaluations of the multilinear extension, $\widetilde{f}$ of f over $\mathbb{F}_5$. Via Lagrange interpolation (Lemma 3.6), $\widetilde{f}(x_1, x_2) = (1 - x_1)(1 - x_2) + 2(1 - x_1)x_2 + x_1(1 - x_2) + 4x_1x_2$.

Completing the Proof of Fact 3.5. To show that there is a unique multilinear polynomial extending f, we show that if p and q are two multilinear polynomials such that $p(x) = q(x)$ for all $x \in \{0,1\}^v$, then p and q are in fact the same polynomial, i.e., $p(x) = q(x)$ for all $x \in \mathbb{F}^v$. Equivalently, we want to show that the polynomial $h \coloneqq p - q$ is the identically 0 polynomial.

Observe that h is also multilinear, because it is the difference of two multilinear polynomials. Furthermore, the assumption that $p(x) = q(x)$ for all $x \in \{0,1\}^v$ implies that $h(x) = 0$ for all $x \in \{0,1\}^v$. We now show that any such polynomial is identically 0.

Assume that h is a multilinear polynomial that vanishes on $\{0,1\}^v$, meaning that $h(x) = 0$ for all $x \in \{0,1\}^v$. If h is not the identically zero polynomial, then consider any term t in h of minimal degree. h must have at least one such term since h is not identically 0. For example, if $h(x_1, x_2, x_3) = x_1x_2x_3 + 2x_1x_2$, then the term $2x_1x_2$ is of minimal degree, since it has degree 2, and h has no terms of degree 1 or 0.

Now consider the input z obtained by setting all of the variables in t to 1, and all other variables to 0 (in the example above, $z = (1,1,0)$). At input z, term t is nonzero because all of the variables appearing in term t are set to 1. For instance, in the example above, the term $2x_1x_2$ evaluates to 2 at input $(1,1,0)$).

Meanwhile, by multilinearity of h, all other terms of h contain at least one variable that is not in term t (otherwise, t would not be of minimal degree in h). Since z sets all variables not in t to 0, this means that all terms in h other than t evaluate to 0 at z. It follows that $h(z) \neq 0$ (e.g., in the example above, $h(z) = 2$).

This contradicts the assumption that $h(x) = 0$ for all $x \in \{0,1\}^v$. We conclude that any multilinear polynomial h that vanishes on $\{0,1\}^v$ must be identically zero, as desired. □

While any function $f\colon \{0,1\}^v \to \mathbb{F}$ has many polynomials that extend it, Fact 3.5 states that exactly one of those extension polynomials is multilinear. For example, if $f(x) = 0$ for all $x \in \{0,1\}^v$, then the multilinear extension of f is just the 0 polynomial. But $p(x_1, \ldots, x_v) = x_1 \cdot (1 - x_1)$ is one example of a non-multilinear polynomial that also extends f.

Algorithms for Evaluating the Multilinear Extension of f. Suppose that the verifier is given as input the values $f(w)$ for all $n = 2^v$ Boolean vectors $w \in \{0,1\}^v$. Equation (3.1) yields two efficient methods for evaluating $\widetilde{f}$ at any point $r \in \mathbb{F}^v$, The first method was described in [103]: it requires $O(n \log n)$ time, and allows $\mathcal{V}$ to make a single streaming pass over the $f(w)$ values while storing just $v + 1 = O(\log n)$ field elements. The second method is due to Vu *et al.* [241]: it shaves a logarithmic factor off of $\mathcal{V}$'s runtime, bringing it down to linear time, i.e., $O(n)$, but increases $\mathcal{V}$'s space usage to $O(n)$.

Lemma 3.7 ([103]). Fix a positive integer v and let $n = 2^v$. Given as input $f(w)$ for all $w \in \{0,1\}^v$ and a vector $r \in \mathbb{F}^{\log n}$, $\mathcal{V}$ can compute $\widetilde{f}(r)$ in $O(n \log n)$ time and $O(\log n)$ words of space[14] with a single streaming pass over the input (regardless of the order in which the $f(w)$ values are presented).

Proof. $\mathcal{V}$ can compute the right hand side of Equation (3.1) incrementally from the stream by initializing $\widetilde{f}(r) \leftarrow 0$, and processing each update $(w, f(w))$ via:

$$\widetilde{f}(r) \leftarrow \widetilde{f}(r) + f(w) \cdot \chi_w(r).$$

$\mathcal{V}$ only needs to store $\widetilde{f}(r)$ and r, which requires $O(\log n)$ words of memory (one for each entry of r). Moreover, for any w, $\chi_w(r)$ can be computed in $O(\log n)$ field operations (see Equation (3.2)), and thus $\mathcal{V}$ can compute $\widetilde{f}(r)$ with one pass over the stream, using $O(\log n)$ words of space and $O(\log n)$ field operations per update. □

The algorithm of Lemma 3.7 computes $\widetilde{f}(r)$ by evaluating each term on the right hand side of Equation (3.1) independently in $O(v)$ time and summing the results. This results in a total runtime of $O(v \cdot 2^v)$. The following lemma gives an even faster algorithm, running in time $O(2^v)$. Its speedup relative to Lemma 3.7 is obtained by *not* treating each term of the sum independently. Rather, using dynamic programming, Lemma 3.8 computes $\chi_w(r)$ *for all* 2^v vectors $w \in \{0,1\}^v$ in time $O(2^v)$.

[14] A "word of space" refers to the amount of data processed by a machine in one step. It is often 64 bits on modern processors. For simplicity, we assume throughout that a field element can be stored using a constant number of machine words.

Lemma 3.8 ([241]). Fix a positive integer v, and let $n = 2^v$. Given as input $f(w)$ for all $w \in \{0,1\}^v$ and a vector $r = (r_1, \ldots, r_v) \in \mathbb{F}^{\log n}$, $\mathcal{V}$ can compute $\widetilde{f}(r)$ in $O(n)$ time and $O(n)$ space.

Proof. Notice the right hand side of Equation (3.1) expresses $\widetilde{f}(r)$ as the inner product of two n-dimensional vectors, where (associating $\{0,1\}^v$ and $\{0, \ldots, 2^v - 1\}$ in the natural way) the w'th entry of the first vector is $f(w)$ and the w'th entry of the second vector is $\chi_w(r)$. This inner product can be computed in $O(n)$ time given a table of size n whose wth entry contains the quantity $\chi_w(r)$. Vu *et al.*, show how to build such a table in time $O(n)$ using memoization.

The memoization procedure consists of $v = \log n$ stages, where Stage j constructs a table $A^{(j)}$ of size 2^j, such that for any $(w_1, \ldots, w_j) \in \{0,1\}^j$, $A^{(j)}[(w_1, \ldots, w_j)] = \prod_{i=1}^{j} \chi_{w_i}(r_i)$. Notice $A^{(j)}[(w_1, \ldots, w_j)] = A^{(j-1)}[(w_1, \ldots, w_{j-1})] \cdot (w_j r_j + (1 - w_j)(1 - r_j))$, and so the jth stage of the memoization procedure requires time $O(2^j)$. The total time across all $\log n$ stages is therefore $O(\sum_{j=1}^{\log n} 2^j) = O(2^{\log n}) = O(n)$. An example of this memoization procedure for $v = 3$ is given in Figure 3.3.

Conceptually, the above algorithm in Stage 1 evaluates all one-variate multilinear Lagrange basis polynomials at the input r_1. There are two such basis polynomials, namely $\chi_0(x_1) = x_1$ and $\chi_1(x_1) = (1 - x_1)$, and hence the algorithm in Stage 1 computes and stores two values: r_1 and $(1 - r_1)$. In Stage 2, the algorithm evaluates all two-variate multilinear Lagrange basis polynomials at the input (r_1, r_2). There are four such values, namely $r_1 r_2$, $r_1(1-r_2)$, $(1-r_1)r_2$ and $(1-r_1)(1-r_2)$. In general,

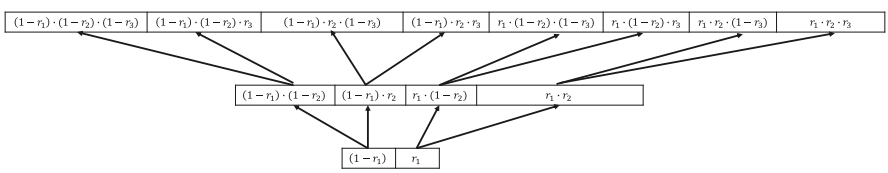

Figure 3.3: Evaluating all eight three-variate Lagrange basis polynomials at input $r = (r_1, r_2, r_3) \in \mathbb{F}^3$ via the memoization procedure in the proof of Lemma 3.8. The algorithm uses 12 field multiplications in total. In contrast, the algorithm given in Lemma 3.7 independently evaluates each Lagrange basis polynomial at r independently. This requires 2 field multiplications per basis polynomial, or $8 \cdot 2 = 16$ multiplications in total.

Stage i of the algorithm evaluates all i-variate multilinear Lagrange basis polynomials at the input $(r_1, r_2, \ldots, r_i)$. Figure 3.3 illustrates the entire procedure when the number of variables is $v = 3$. □

3.6 Exercises

Exercise 3.1. Let A, B, C be $n \times n$ matrices over a field $\mathbb{F}$. In Section 2.2, we presented a randomized algorithm for checking that $C = A \cdot B$. The algorithm picked a random field element r, let $x = (r, r^2, \ldots, r^n)$, and output EQUAL if $Cx = A \cdot (Bx)$, and output NOT-EQUAL otherwise. Suppose instead that each entry of the vector x is chosen independently and uniformly at random from $\mathbb{F}$. Show that:

- If $C_{ij} = (AB)_{ij}$ for all $i = 1, \ldots, n, j = 1, \ldots, n$, then the algorithm outputs EQUAL for every possible choice of x.
- If there is even one $(i, j) \in [n] \times [n]$ such that $C_{ij} \neq (AB)_{ij}$, then the algorithm outputs NOT-EQUAL with probability at least $1 - 1/|\mathbb{F}|$.

Exercise 3.2. In Section 2.1, we described a communication protocol of logarithmic cost for determining whether Alice's and Bob's input vectors are equal. Specifically, Alice and Bob interpreted their inputs as degree-n univariate polynomials p_a and p_b, chose a random $r \in \mathbb{F}$ with $|\mathbb{F}| \gg n$, and compared $p_a(r)$ to $p_b(r)$. Give a different communication protocol in which Alice and Bob interpret their inputs as multilinear rather than univariate polynomials over $\mathbb{F}$. How large should $\mathbb{F}$ be to ensure that the probability Bob outputs the wrong answer is at most $1/n$? What is the communication cost in bits of this protocol?

Exercise 3.3. Let $p = 11$. Consider the function $f\colon \{0,1\}^2 \to \mathbb{F}_p$ given by $f(0,0) = 3$, $f(0,1) = 4$, $f(1,0) = 1$ and $f(1,1) = 2$. Write out an explicit expression for the multilinear extension $\tilde{f}$ of f. What is $\tilde{f}(2,4)$?

Now consider the function $f\colon \{0,1\}^3 \to \mathbb{F}_p$ given by $f(0,0,0) = 1$, $f(0,1,0) = 2$, $f(1,0,0) = 3$, $f(1,1,0) = 4$, $f(0,0,1) = 5$, $f(0,1,1) = 6$, $f(1,0,1) = 7$, $f(1,1,1) = 8$. What is $\tilde{f}(2,4,6)$? How many field multiplications did you perform during the calculation? Can you work

through a calculation of $\tilde{f}(2,4,6)$ that uses "just" 20 multiplication operations? Hint: see Lemma 3.8.

Exercise 3.4. Fix some prime p of your choosing. Write a Python program that takes as input an array of length 2^ℓ specifying all evaluations of a function $f\colon \{0,1\}^\ell \to \mathbb{F}_p$ and a vector $\mathbf{r} \in \mathbb{F}_p^\ell$, and outputs $\tilde{f}(\mathbf{r})$.