8

MIPs and Succinct Arguments

Multi-prover interactive proofs (MIPs) grant the verifier access to more than one untrusted prover, and assume the provers cannot tell each other about what challenges they receive from the verifier. While MIPs are of interest in their own right, they are also important building blocks for constructing succinct arguments. In particular, in this section we give 2-prover MIPs for circuit satisfiability (Section 8.2) and its generalization R1CS-satisfiability (Section 8.4), in which the second prover effectively acts as a polynomial commitment scheme, a notion we introduced in Section 7.3. Accordingly, one can obtain a (single-prover) succinct argument with state-of-the-art performance by replacing the second prover with an appropriate polynomial commitment scheme, practical instantiations of which are covered in detail in Sections 10.4.4, 10.5, and Section 14.[1] In particular, the arguments obtained from the

[1]When an initial version of this monograph was publicly released in the form of lecture notes in 2018, this approach to obtaining succinct arguments had not been previously published; the only published approach to turning MIPs into succinct arguments at that time [55] made use of a cryptographic primitive known as Fully Homomorphic Encryption, which is currently much too computationally intensive to yield practical SNARKs. Since that time, Setty [226] has implemented and extended the MIP-to-succinct-argument approach described in this monograph, with several follow-on works [137], [230].

MIPs of this section have significantly shorter proofs than those of Section 7.

MIPs are also of significant historical importance, and the state-of-the-art MIPs of this section exhibit several ideas that will recur in our coverage of PCPs and IOPs (Sections 9 and 10).

8.1 MIPs: Definitions and Basic Results

Definition 8.1. A k-prover interactive proof protocol for a language $\mathcal{L} \subseteq \{0,1\}^*$ involves $k+1$ parties: a probabilistic polynomial time verifier, and k provers. The verifier exchanges a sequence of messages with each prover; each prover's message is a function of the input and the messages from $\mathcal{V}$ that it has seen so far. The interaction produces a transcript $t = (\mathcal{V}(r), \mathcal{P}_1, \ldots, \mathcal{P}_k)(x)$, where r denotes $\mathcal{V}$'s internal randomness. After the transcript t is produced, $\mathcal{V}$ decides whether to output accept or reject based on r, t, and x. Denote by $\text{out}(\mathcal{V}, \text{x}, \text{r}, \mathcal{P}_1, \ldots, \mathcal{P}_\text{k})$ the output of verifier $\mathcal{V}$ on input x given prover strategies $(\mathcal{P}_1, \ldots, \mathcal{P}_k)$ and that $\mathcal{V}$'s internal randomness is equal to r.

The multi-prover interactive proof system has completeness error δ_c and soundness error δ_s if the following two properties hold.

(1) *(Completeness)* There exists a tuple of prover strategies $(\mathcal{P}_1, \ldots, \mathcal{P}_k)$ such that for every $x \in \mathcal{L}$,

$$\Pr[\text{out}(\mathcal{V}, \text{x}, \text{r}, \mathcal{P}_1, \ldots, \mathcal{P}_\text{k}) = \text{accept}] \geq 1 - \delta_\text{c}.$$

(2) *(Soundness)* For every $x \notin \mathcal{L}$ and every tuple of prover strategies $(\mathcal{P}'_1, \ldots, \mathcal{P}'_k)$,

$$\Pr[\text{out}(\mathcal{V}, \text{x}, \text{r}, \mathcal{P}'_1, \ldots, \mathcal{P}'_\text{k}) = \text{accept}] \leq \delta_\text{s}.$$

Say that a k-prover interactive proof system is valid if $\delta_c, \delta_s \leq 1/3$. The complexity class **MIP** is the class of all languages possessing valid k-prover interactive proof systems, for some $k = \text{poly}(n)$.

The MIP model was introduced by Ben-Or *et al.* [32]. It is crucial in Definition 8.1 that each prover's message is a function only of the input and the messages from $\mathcal{V}$ that it has seen so far. In particular, $\mathcal{P}_i$

cannot tell $\mathcal{P}_j$ what messages $\mathcal{V}$ has sent it, or vice versa, for any $i \neq j$. If such "cross-talk" between $\mathcal{P}_i$ and $\mathcal{P}_j$ were allowed, then it would be possible to simulate any MIP by a single-prover interactive proof, and the classes **MIP** and **IP** would become equal.

As discussed in Section 1.2.3, it can be helpful to think of MIP as follows. The provers are like prisoners who are about to be interrogated. The prisoners get placed in separate interrogation rooms. Prior to going into these rooms, the prisoners can talk amongst themselves, plotting a strategy for answering questions. But once they are placed in the rooms, they can no longer talk to each other, and in particular prover i cannot tell the other provers what questions the verifier is asking it. The verifier is like the interrogator, trying to determine if the prover's stories are consistent with each other, and with the claim being asserted.

The next section shows that, up to polynomial blowups in $\mathcal{V}$'s runtime, 2-prover MIPs are just as expressive as k-prover MIPs, for any $k = \text{poly}(n)$.

8.1.1 What Does a Second Prover Buy?

Non-Adaptivity. In a single-prover interactive proof, the prover $\mathcal{P}$ is allowed to act adaptively, in the sense that $\mathcal{P}$'s response to the ith message m_i sent from $\mathcal{V}$ is allowed to depend on the preceding $i-1$ messages. Intuitively, the reason that MIPs are more expressive than IPs is that the presence of a second prover (who does not know $\mathcal{V}$'s messages to the first prover) prevents the first prover from behaving in this adaptive manner.[2] This can be formalized via the following easy lemma showing that the complexity class **MIP** is equivalent to the class of languages satisfied by polynomial time randomized *oracle machines*. Here, an oracle machine is essentially a computer that has query access to a giant string $\mathcal{O}$ that is fixed at the start of the computer's execution. The string $\mathcal{O}$ may be enormous, but the computer is allowed to look

[2] One may initially have the intuition that, since allowing adaptivity on the part of the prover means allowing "more expressive" prover strategies, prover adaptivity leads to efficient proof systems for more challenging problems. In fact, the opposite is true. Allowing the prover to behave adaptively gives the prover more power to break soundness. Hence, allowing the prover to behave adaptively actually weakens the class of problems that have proof systems with an efficient verifier.

at any desired symbol $\mathcal{O}_i$ (i.e., the ith symbol of $\mathcal{O}$) in unit time. One can think of any query that the computer makes to $\mathcal{O}$ as a question, and $\mathcal{O}_i$ as the answer. Because $\mathcal{O}$ is fixed at the start of the computer's execution, the answers that are returned by $\mathcal{O}$ are non-adaptive in the sense that the answer to the computer's jth question *does not depend on which questions the computer asked previously.*

Lemma 8.2 ([113])**.** Let $\mathcal{L}$ be a language, and M a probabilistic polynomial time oracle Turing Machine. Let $M^{\mathcal{O}}$ denote M when given query access to oracle $\mathcal{O}$. Suppose that $x \in \mathcal{L} \implies \exists$ an oracle $\mathcal{O}$ such that $M^{\mathcal{O}}$ accepts x with probability 1, and $x \notin \mathcal{L} \implies \forall$ oracles $\mathcal{O}$, $M^{\mathcal{O}}$ rejects x with probability at least 2/3. Then there is a 2-prover MIP for $\mathcal{L}$.

Remark 8.1. In Lemma 8.2, one can think of $\mathcal{O}$ as a giant purported proof that $x \in \mathcal{L}$, and machine M only looks at a small (i.e., polynomial) number of symbols of the proof. This is the same notion as a *probabilistically checkable proof*, which we introduce formally in Section 9.1. In this terminology, Lemma 8.2 states that any PCP with a polynomial time verifier can be turned into a 2-prover MIP with a polynomial time verifier.

Proof. We first describe a "subroutine" 2-prover MIP that has perfect completeness and high but bounded soundness error. The final 2-prover MIP simply repeats the subroutine MIP independently several times.

The Subroutine MIP. $\mathcal{V}$ simulates M, and every time M poses a query q to the oracle, $\mathcal{V}$ asks the query to $\mathcal{P}_1$, treating $\mathcal{P}_1$'s response as $\mathcal{O}(q)$. At the end of the protocol, $\mathcal{V}$ picks a query q uniformly at random from all queries that were posed to $\mathcal{P}_1$, and poses it to $\mathcal{P}_2$, rejecting if $\mathcal{P}_2$'s response to q does not equal $\mathcal{P}_1$'s.

Completeness of the subroutine is clear: if $x \in \mathcal{L}$, there is some oracle $\mathcal{O}^*$ causing M to accept x with probability 1. If $\mathcal{P}_1$ and $\mathcal{P}_2$ respond to any query q with $\mathcal{O}^*(q)$, then $\mathcal{V}$ will accept x on each of the runs of the protocol with probability 1.

For soundness of the subroutine, observe that since $\mathcal{P}_2$ is only asked a single query, we can treat $\mathcal{P}_2$ as an oracle $\mathcal{O}$. That is, $\mathcal{P}_2$'s answer on

query q is a function only of q. On any run of the protocol on input $x \notin \mathcal{L}$, let $q_1, \ldots, q_\ell$ denote the queries that $\mathcal{V}$ poses to $\mathcal{P}_1$ on input x. On the one hand, if $\mathcal{P}_1$ ever answers a query q_i differently than $\mathcal{O}(q_i)$, the verifier will pick that query to pose to $\mathcal{P}_2$ with probability at least $1/\ell$, and in this case the verifier will reject. On the other hand, if $\mathcal{P}_1$ answers every query q_i with $\mathcal{O}(q_i)$, then $\mathcal{V}$ will reject with probability at least $2/3$ because $M^{\mathcal{O}}$ rejects with that probability. Therefore, $\mathcal{V}$ rejects on each run of the protocol with probability at least $1/\ell$.

The Final MIP. The final MIP repeats the subroutine protocol independently and sequentially 3ℓ times, where ℓ is (an upper bound on) the number of queries that M poses to the oracle on any input $x \in \{0,1\}^n$ (note that ℓ is at most polynomial in the input size n, since M runs in polynomial time). $\mathcal{V}$ accepts only if all instances accept. Since the subroutine MIP has perfect completeness, so does the final MIP. Since the subroutine has soundness error at most $1 - 1/\ell$, sequentially repeating it k times with independently chosen verifier queries on each repetition ensures that, when given an input $x \notin \mathcal{L}$, $\mathcal{V}$ rejects on at least one run of the subroutine with probability at least $1 - (1 - 1/\ell)^{3\ell} > 2/3$. □

The same argument implies that any k-prover MIP (with completeness error at most $\delta_c \leq 1/(9\ell)$, where ℓ is the total number of queries asked) can be simulated by a 2-prover MIP [32]. In the simulation, $\mathcal{V}$ poses all of the questions from the k-prover MIP to $\mathcal{P}_1$, then picks a question at random and poses it to $\mathcal{P}_2$, rejecting if the answers do not agree. $\mathcal{P}_2$ can be treated as an oracle since $\mathcal{P}_2$ is only posed a single question, and hence has no opportunity to behave adaptively. And if $\mathcal{P}_1$ answers even a single query q_i "adaptively" (i.e., different than how $\mathcal{P}_2$ would answer), the probability this is detected is at least $1/\ell$. The whole 2-prover protocol must be repeated $\Omega(\ell)$ times to drive the soundness error from $1 - 1/\ell$ down to $1/3$.

In summary, one can both force non-adaptivity and reduce the number of provers to 2 by posing all queries to $\mathcal{P}_1$ and choosing one of the queries at random to pose to $\mathcal{P}_2$. While this conveys much of the intuition for why MIPs are more expressive than IPs, the technique

is very expensive in practice, due to the need for $\Omega(\ell)$ repetitions—typically, ℓ is on the order of $\log n$, and can easily be in the hundreds in implementations. Fortunately, the MIP that we describe in Section 8.2 requires only two provers without the need for repetition to force non-adaptivity or reduce the number of provers to 2.

But What Does Non-Adaptivity Buy? We will see in Section 8.2 that non-adaptivity buys *succinctness for NP statements*. That is, we will give an MIP for arithmetic circuit *satisfiability* (as opposed to circuit evaluation) in which the total communication and verifier runtime is sublinear in the size of the witness w.

This should not be surprising, as we saw the same phenomenon in Section 7. There, we used a polynomial commitment scheme to cryptographically *bind* the prover to a multilinear polynomial $\tilde{w}$ that was fixed at the start of the interaction with the verifier. In particular, the polynomial commitment scheme enforced non-adaptivity, i.e., the prover must tell the verifier $\tilde{w}(r)$, and is not able to "change its answer" based on the interaction with the verifier. The addition of a second prover in a 2-prover MIP has exactly the same effect. Indeed, we will see that the second prover in the MIP of Section 8.2 essentially functions as a polynomial commitment scheme. Indeed, we will ultimately obtain a (single-prover) succinct argument from the MIP with state-of-the-art practical performance by replacing the second prover with a polynomial commitment scheme; see Section 8.3.

8.2 An Efficient MIP For Circuit Satisfiability

Warmup: A 2-Prover MIP for Low-Depth Arithmetic Circuit Satisfiability. The succinct argument from Section 7 can be directly adapted to yield a 2-prover MIP. The idea is to use the second prover to function as the polynomial commitment scheme.

In more detail, the verifier uses the first prover to apply the GKR protocol to the claim $\mathcal{C}(x, w) = y$. As explained in Section 7.3.2.1, at the end of this protocol, the prover makes a claim about $\tilde{w}(r)$.

In Section 7, this claim was checked by forcing the prover to reveal $\tilde{w}(r)$ via the polynomial commitment protocol (which itself involved a vector-commitment combined with a low-degree test).

In the MIP, the verifier simply uses the second prover to play the role of the polynomial commitment scheme. This means that the second prover provides claimed value for $\tilde{w}(r)$ that does not depend on the questions the verifier asked to the first prover, and executes a low-degree test. Roughly speaking, this combination ensures that the claimed value $\tilde{w}(r)$ is indeed consistent with the multilinear extension $\tilde{w}$ of some witness w that was fixed at the start of the protocol—in particular, w does not depend on the point r chosen by the verifier.

For example, if the low-degree test used is the point-versus-line test, then the verifier picks a random line λ in $\mathbb{F}^{\log n}$ containing r, and sends λ to the second prover, who is asked to respond with a univariate polynomial of degree $\log n$ claimed to equal $\tilde{w}$ restricted to λ. Since r is on the line λ, this univariate polynomial implicitly specifies $\tilde{w}(r)$, and the verifier checks that this value matches the first prover's claim about $\tilde{w}(r)$.

A downside of the warm-up 2-prover MIP for arithmetic circuit satisfiability is that the communication cost and the verifier's runtime grow linearly with the circuit depth d, i.e., is $O(d \log S)$. Hence, the protocol does not save the verifier time for deep, narrow circuits. Arguably, this is not a major downside, because Section 6.5 explained that *any* computer program running in time T can be turned into an equivalent instance of arithmetic circuit satisfiability where the circuit is short and wide rather than long and narrow (specifically, the circuit has depth roughly $O(\log T)$ and size $\tilde{O}(T)$).

Still, even for reasonably small-depth circuits, the proofs from the warm-up GKR-derived MIP can be rather large. In this section, we give an MIP whose proof length is smaller than the warm-up GKR-based MIP by a factor of close to the circuit depth, i.e., $O(\log S)$ rather than $O(d \log S)$, which can be a substantial concrete improvement even for circuits of quite small depth. In so doing, we will see some ideas that recur in later sections when we study argument systems based on PCPs, IOPs, and linear PCPs.

The 2-Prover MIP described in the remainder of this section is a refinement of one given by Blumberg *et al.* [61], which they called Clover. It combines several new ideas with techniques from the original **MIP** = **NEXP** proof of [17], as well as the GKR protocol [135] and its refinements by Cormode *et al.* [102].

8.2.1 Protocol Summary

8.2.1.1 Terminology

Let $\mathcal{C}$ be an arithmetic circuit over a field $\mathbb{F}$ taking an explicit input x and a non-deterministic input w. Let $S = 2^k$ denote the number of gates in $\mathcal{C}$, and assign each gate in $\mathcal{C}$ a binary label in $\{0,1\}^k$. Refer to an assignment of values to each gate of $\mathcal{C}$ as a *transcript* of $\mathcal{C}$, and view the transcript as a function $W\colon \{0,1\}^k \to \mathbb{F}$ mapping gate labels to their values.

Given a claim that $\mathcal{C}(x,w) = y$, a *correct transcript* is a transcript in which the values assigned to the input gates are those of x, the intermediate values correspond to the correct operation of each gate in $\mathcal{C}$, and the values assigned to the output gates are y. The arithmetic circuit satisfiability problem on instance $\{\mathcal{C}, x, y\}$ is equivalent to determining whether there is a correct transcript for $\{\mathcal{C}, x, y\}$. See Figure 8.1 for an example.

8.2.1.2 Overview of the MIP

The MIP works by having $\mathcal{P}_1$ claim to "hold" an extension Z of a correct transcript W for $\{\mathcal{C}, x, y\}$. If the prover is honest, then Z will equal $\widetilde{W}$, the multilinear extension of W. The protocol then identifies a polynomial $g_{x,y,Z}\colon \mathbb{F}^{3k} \to \mathbb{F}$ (which depends on x, y, and Z) satisfying the following property: $g_{x,y,Z}(a,b,c) = 0$ for all Boolean inputs $(a,b,c) \in \{0,1\}^{3k}$ $\iff$ Z is indeed an extension of a correct transcript W.

To check that $g_{x,y,Z}$ vanishes at all Boolean inputs, the protocol identifies a related polynomial $h_{x,y,Z}$ such that $g_{x,y,Z}$ vanishes at all Boolean inputs $\iff$ the following equation holds:

$$\sum_{(a,b,c)\in\{0,1\}^{3k}} h_{x,y,Z}(a,b,c) = 0. \tag{8.1}$$

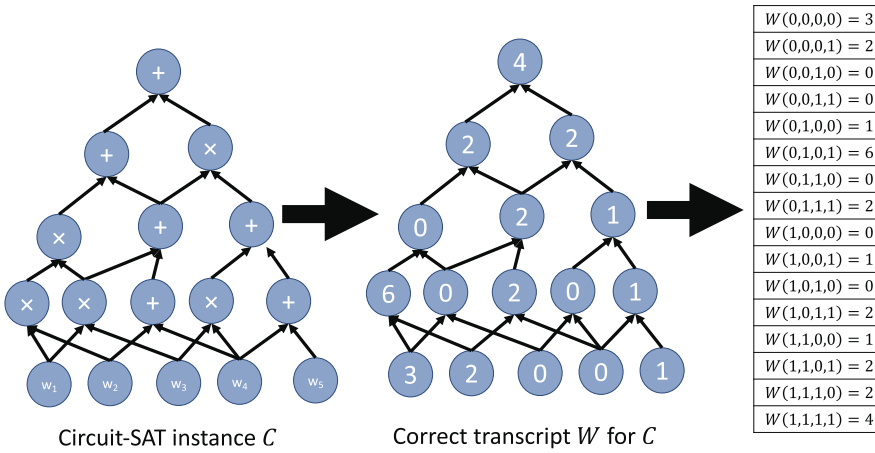


Figure 8.1: The leftmost image depicts an arithmetic circuit $\mathcal{C}$ over field $\mathbb{F}$ of size 16, with no public input x, and non-deterministic input $w = (w_1, w_2, w_3, w_4, w_5) \in \mathbb{F}^5$. The middle image depicts a correct transcript W for $\mathcal{C}$ producing output $y = 4$. The rightmost image is the evaluation table of W when viewed as a function mapping $\{0,1\}^4$ to $\mathbb{F}$.

Strictly speaking, the polynomial $h_{x,y,Z}$ is randomly generated, and there is a small chance over the random choice of $h_{x,y,Z}$ that Equation (8.1) holds even though $g_{x,y,Z}$ does not vanish at all Boolean inputs. The MIP applies the sum-check protocol to the polynomial $h_{x,y,Z}$ to compute this sum. Note that if Z is a low-degree polynomial, then so is $h_{x,y,Z}$, as is required both to control costs and guarantee soundness in the sum-check protocol.

At the end of the sum-check protocol, $\mathcal{V}$ needs to evaluate $h_{x,y,Z}$ at a random point, which in turn requires evaluating Z at a random point $r \in \mathbb{F}^k$. Unfortunately, $\mathcal{V}$ cannot compute $Z(r)$, since $\mathcal{V}$ does not have access to the polynomial Z (as Z only "exists" in $\mathcal{P}_1$'s head). Instead, $\mathcal{V}$ asks $\mathcal{P}_2$ to send her $Z(r)$, using the point-vs-line low-degree test (see Section 7.3.2.2). Specifically, $\mathcal{P}_2$ is asked to send Z restricted to a line Q, where Q is chosen to be a random line in $\mathbb{F}^k$ containing r. This forces $\mathcal{P}_2$ to implicitly make a claim about $Z(r)$ (note that $\mathcal{P}_2$ does not know

which point in Q is r); $\mathcal{V}$ rejects if $\mathcal{P}_1$ and $\mathcal{P}_2$'s claims about $Z(r)$ are inconsistent, and accepts otherwise.[3]

The low-degree test cannot guarantee that Z itself is a low-degree polynomial, since $\mathcal{V}$ only ever inspects Z at a small number of points. Hence it is impossible to argue that $h_{x,y,Z}$ itself satisfies Equation (8.1): the soundness analysis for the sum-check protocol breaks down if the polynomial to which it is applied has large degree. However, the low-degree test *does* guarantee that if $\mathcal{P}_1$ and $\mathcal{P}_2$'s claims about $Z(r)$ are consistent with non-negligible probability over the random choice of r, then Z is close to a low-degree polynomial Y, in the sense that $Y(r') = Z(r')$ for a large fraction of points $r' \in \mathbb{F}^k$. Since $h_{x,y,Y}$ *is* low-degree, it is straightforward to tweak the soundness analysis of the sum-check protocol to argue that $h_{x,y,Y}$ satisfies Equation (8.1), and hence that Y extends a correct transcript for $\{\mathcal{C}, x, y\}$ (cf. Theorem 8.4).

Remark 8.2. The fact that the low-degree test only guarantees that the function Z is *close to* rather than *exactly equal to* a low-degree polynomial substantially complicates the soundness analysis of the MIP (Section 8.2.1.4). In (single-prover) succinct arguments derived from the MIP, the second prover can be replaced with a polynomial commitment scheme that ensures Z is *exactly equal to* a multilinear polynomial, and these complications go away. We cover such polynomial commitment schemes in Sections 10.4.2, 10.5, and Section 14. Accordingly, readers primarily interested in the resulting succinct arguments rather than the MIPs themselves can skip the detailed soundness analysis of Section 8.2.1.4.

Identical complications arose in Section 7, because the polynomial commitment scheme given there used a low-degree test and hence only bound the prover to a function close to a low-degree polynomial.

[3] Blumberg *et al.* [61] actually use a different low-degree test called the point-vs-plane test, despite the fact that it leads to asymptotically larger proofs and larger runtime for $\mathcal{P}_2$. They made this choice because the known soundness analyses of the point-vs-line test involve huge constant factors, and hence yield good soundness only over impractically large fields. The constant factors in known analyses of the point-vs-plane test are more reasonable [195], enabling the use of reasonably sized fields in the MIP of [61].

Preview: The importance of checking that a polynomial vanishes on a designated subspace. The problem of checking that a certain polynomial $g_{x,y,Z}$ vanishes on a designated subspace plays a central role in many MIPs and PCPs. The problem is sometimes referred to as checking a *Vanishing Reed-Solomon or Reed-Muller code* [50]. This problem will arise several more times in this survey, including in state of the art PCPs, IOPs, and linear PCPs described in Sections 9, 10, and 17. One difference is that in the PCPs, IOPs, and linear PCPs of later sections, the polynomial $g_{x,y,Z}$ is *univariate*, instead of $(3 \log S)$-variate as in the MIP considered here.

Comparison to the GKR Protocol. While the GKR protocol verifies the claim $\mathcal{C}(x, w) = y$ layer by layer, with a different instance of the sum-check protocol required for each layer of $\mathcal{C}$, the MIP of this section verifies the whole circuit in one shot, using a single invocation of the sum-check protocol. The reason the GKR protocol must work layer-by-layer is that the verifier must force the prover to make a claim about (the multilinear extension of) *the input alone*, since the verifier never materializes the intermediate gates of the circuit. This is not necessary in the multi-prover setting: in the MIP, $\mathcal{P}_1$ makes a claim about an extension Z of *the entire transcript*. $\mathcal{V}$ cannot check this claim independently, but that is okay because there is a second prover to ask for help.

8.2.1.3 Protocol Details

Notation. Let $\text{add}, \text{mult}\colon \{0,1\}^{3k} \to \{0,1\}$ denote the functions that take as input three gate labels (a, b, c) from $\mathcal{C}$ and outputs 1 if and only if gate a adds (respectively, multiplies) the outputs of gates b and c. While the GKR protocol had separate functions add_i and mult_i for each layer of $\mathcal{C}$, the MIP of this section arithmetizes all of $\mathcal{C}$ at once. We also add a third wiring predicate, which has no analog within the GKR protocol: let $\text{io}\colon \{0,1\}^{3k} \to \{0,1\}$ denote the function that returns 1 when gate a is either a gate from the explicit input x or one of the output gates, and gates b and c are the in-neighbors of a (input gates have in-neighbors $b = c = \mathbf{0}$).

Notice that add, mult, and io are independent of the inputs x and purported outputs y. The final function that plays a role in the MIP does depend on x and y. Define $I_{x,y}\colon \{0,1\}^k \to \mathbb{F}$ such that $I_{x,y}(a) = x_a$ if a is the label of an input gate, $I_{x,y}(a) = y_a$ if a is the label of an output gate, and $I_{x,y}(a) = 0$ otherwise.

Lemma 8.3. For $G_{x,y,W}(a,b,c)\colon \{0,1\}^{3k} \to \mathbb{F}$ defined as below, $G_{x,y,W}(a,b,c) = 0$ for all $(a,b,c) \in \{0,1\}^{3k}$ if and only if W is a correct transcript for $\{C,x,y\}$:

$$\begin{aligned} G_{x,y,W}(a,b,c) = \mathrm{io}(a,b,c) \cdot (I_{x,y}(a) - W(a)) + \mathrm{add}(a,b,c) \\ \cdot (W(a) - (W(b) + W(c))) + \mathrm{mult}(a,b,c) \\ \cdot (W(a) - W(b) \cdot W(c)). \end{aligned}$$

Proof. If W is not a correct transcript, there are five cases:

(1) Suppose $a \in \{0,1\}^k$ is the label of an input gate. If $W(a) \neq x_a$, then $G_{x,y,W}(a,0,0) = I_{x,y}(a) - W(a) = x_a - W(a) \neq 0$.

(2) Suppose $a \in \{0,1\}^k$ is the label of a non-output addition gate with in-neighbors b and c. If $W(a) \neq W(b)+W(c)$, then $G_{x,y,W}(a,b,c) = W(a) - (W(b) + W(c)) \neq 0$.

(3) Suppose $a \in \{0,1\}^k$ is the label of a non-output multiplication gate with in-neighbors b and c. If $W(a) \neq W(b) \cdot W(c)$, then $G_{x,y,W}(a,b,c) = W(a) - (W(b) \cdot W(c)) \neq 0$.

(4) Suppose $a \in \{0,1\}^k$ is the label of an output addition gate with in-neighbors b and c. If $y_a \neq W(b) + W(c)$, then $G_{x,y,W}(a,b,c) = I_{x,y}(a) - W(a) + (W(a) - (W(b)+W(c))) = y_a - (W(b)+W(c)) \neq 0$.

(5) Suppose $a \in \{0,1\}^k$ is the label of an output multiplication gate with in-neighbors b and c. If $y_a \neq W(b) \cdot W(c)$, then $G_{x,y,W}(a,b,c) = I_{x,y}(a) - W(a) + (W(a) - (W(b) \cdot W(c))) = y_a - (W(b) \cdot W(c)) \neq 0$.

On the other hand, if W is a correct transcript then it is immediate from the definition of $G_{x,y,W}$ that $G_{x,y,W}(a,b,c) = 0$ for all $(a,b,c) \in \{0,1\}^{3k}$. □

For any polynomial $Z\colon \mathbb{F}^k \to \mathbb{F}$, define the associated polynomial:

$$g_{x,y,Z}(a,b,c) = \widetilde{\text{io}}(a,b,c) \cdot (\widetilde{I}_{x,y}(a) - Z(a)) + \widetilde{\text{add}}(a,b,c) \cdot (Z(a) - (Z(b) + Z(c))) + \widetilde{\text{mult}}(a,b,c) \cdot (Z(a) - Z(b) \cdot Z(c)).$$

It follows from Lemma 8.3 that Z extends a correct transcript W if and only if $g_{x,y,Z}$ vanishes on the Boolean hypercube. We now define a polynomial $h_{x,y,Z}$ such that $g_{x,y,Z}$ vanishes on the Boolean hypercube if and only if $\sum_{u\in\{0,1\}^{3k}} h_{x,y,Z}(u) = 0$.

Defining $h_{x,y,Z}$. As in Lemma 4.9 of Section 4.6.7.1, let $\beta_{3k}(a,b) : \{0,1\}^{3k} \times \{0,1\}^{3k} \to \{0,1\}$ be the function that evaluates to 1 if $a = b$, and evaluates to 0 otherwise, and define the formal polynomial

$$\widetilde{\beta}_{3k}(a,b) = \prod_{j=1}^{3k} ((1-a_j)(1-b_j) + a_j b_j)\,.$$

It is straightforward to check that $\widetilde{\beta}_{3k}$ is the multilinear extension β_{3k}. Indeed, $\widetilde{\beta}_{3k}$ is a multilinear polynomial. And for $a, b \in \{0,1\}^{3k}$, it is easy to check that $\widetilde{\beta}_{3k}(a,b) = 1$ if and only if a and b are equal coordinate-wise.

Consider the polynomial

$$p(X) \coloneqq \sum_{u\in\{0,1\}^{3k}} \widetilde{\beta}_{3k}(X,u) \cdot g_{x,y,Z}(u).$$

Clearly p is multilinear since $\widetilde{\beta}$ is, and p vanishes on all inputs in $\{0,1\}^{3k}$ if and only if $g_{x,y,Z}$ does. Since the multilinear extension on domain $\{0,1\}^{3k}$ is unique, this means that p is the *identically zero* polynomial if and only if $g_{x,y,Z}$ vanishes on all inputs in $\{0,1\}^{3k}$. For the verifier to check that p is indeed the zero-polynomial, it is enough for the verifier to pick a random input $r \in \{0,1\}^{3k}$ and confirm that $p(r) = 0$, because if p is any *nonzero* polynomial of total degree at most d, the Schwartz-Zippel lemma implies that $p(r)$ will equal 0 with probability at most $d/|\mathbb{F}|$.

Hence, we define

$$h_{x,y,Z}(Y) \coloneqq \widetilde{\beta}_{3k}(r, Y) \cdot g_{x,y,Z}(Y). \tag{8.2}$$

This definition ensures that $p(r) = \sum_{u \in \{0,1\}^{3k}} h_{x,y,Z}(u)$.

In summary, in the MIP, $\mathcal{V}$ chooses r uniformly at random from the set $\mathbb{F}^{3k}$, defines $h_{x,y,Z}$ based on r as per Equation (8.2), and is convinced that Z extends a correct transcript for $\{\mathcal{C}, x, y\}$ as long as

$$0 = \sum_{u \in \{0,1\}^{3k}} h_{x,y,Z}(u).$$

More formally, if $g_{x,y,Z}$ has total degree at most d, then with probability at least $1 - (d+1)/|\mathbb{F}|$ over the random choice of r, if $g_{x,y,Z}$ does not vanish on the Boolean hypercube then $\sum_{u \in \{0,1\}^{3k}} h_{x,y,Z}(u) \neq 0$. For simplicity, the remainder of the presentation ignores the $(d+1)/|\mathbb{F}|$ probability of error in this step (the $(d+1)/|\mathbb{F}|$ can be absorbed into the soundness error of the entire MIP).

Applying the Sum-Check Protocol to $h_{x,y,Z}$. $\mathcal{V}$ applies the sum-check protocol to $h_{x,y,Z}$, with $\mathcal{P}_1$ playing the role of the prover in this protocol. To perform the final check in this protocol, $\mathcal{V}$ needs to evaluate $h_{x,y,Z}$ at a random point $r \in \mathbb{F}^{3k}$. Let r_1, r_2, r_3 denote the first, second, and third k entries of r. Then evaluating $h_{x,y,Z}(r)$ requires evaluating $K_q(r)$, $\widetilde{\text{io}}(r)$, $\widetilde{\text{add}}(r)$, $\widetilde{\text{mult}}(r)$, $\widetilde{I}_{x,y}(r_1)$, $Z(r_1)$, $Z(r_2)$, and $Z(r_3)$. $\mathcal{V}$ can compute the first five evaluations without help in $O(\log(T))$ time, assuming that $\widetilde{\text{add}}$ and $\widetilde{\text{mult}}$ can be computed within this time bound (see Section 4.6.6 for further discussion of this assumption). However, $\mathcal{V}$ cannot evaluate $Z(r_1)$, $Z(r_2)$, or $Z(r_3)$ without help, because $\mathcal{V}$ does not know Z. To deal with this, the verifier first uses the technique from Section 4.5.2 to reduce the evaluation of Z at the three points r_1, r_2, and r_3, to the evaluation of Z at a single point $r_4 \in \mathbb{F}^k$. This reduction forces $\mathcal{P}_1$ to make a claim regarding the value of $Z(r_4)$. Unfortunately, $\mathcal{V}$ does not know Z and hence cannot evaluate $Z(r_4)$ unaided. To obtain the evaluation $Z(r_4)$, $\mathcal{V}$ turns to $\mathcal{P}_2$.

The Low-Degree Test. $\mathcal{V}$ sends $\mathcal{P}_2$ a random line Q in $\mathbb{F}^k$ passing through r_4, and demands that $\mathcal{P}_2$ reply with a univariate polynomial of degree at most k, claimed to equal Z restricted to Q. Note that $\mathcal{P}_2$

does not know *which* of the $|\mathbb{F}|$ many points on the line Q equals r_4. Since r_4 lies on Q, $\mathcal{P}_2$'s response implicitly specifies a value for $Z(r_4)$. $\mathcal{V}$ accepts if this value equals that claimed by $\mathcal{P}_1$ and rejects otherwise.

8.2.1.4 MIP Soundness Analysis

Theorem 8.4. Suppose that $\mathcal{P}_1$ and $\mathcal{P}_2$ convince the MIP verifier to accept with probability $\gamma > .5 + \varepsilon$ for $\varepsilon = \Omega(1)$. Then there is some polynomial Y such that $h_{x,y,Y}$ satisfies Equation (8.1).

Detailed Sketch. Let Z^* denote the function that on input r_4 outputs $\mathcal{P}_1$'s claimed value for $Z(r_4)$.[4] If $\mathcal{P}_1$ and $\mathcal{P}_2$ pass the low-degree test with probability at least γ, known analyses of the low-degree test guarantee that, if working over a sufficiently large field $\mathbb{F}$, there is some polynomial Y of total degree at most k such that Z^* and Y agree on a $p \geq \gamma - o(1)$ fraction of points. Since Y has total degree at most k, $h_{x,y,Y}$ has total degree at most $6k$.

Suppose that $h_{x,y,Y}$ does not satisfy Equation (8.1). Let us say that $\mathcal{P}_1$ *cheats* at round i of the sum-check protocol if he does not send the message that is prescribed by the sum-check protocol in that round, when applied to the polynomial $h_{x,y,Y}$. The soundness analysis of the sum-check protocol (Section 4.1) implies that if $\mathcal{P}_1$ falsely claims that $h_{x,y,Y}$ does satisfy Equation (8.1), then with probability at least $1-(3k)\cdot(6k)/|\mathbb{F}| = 1-o(1)$, $\mathcal{P}_1$ will be forced to cheat at all rounds of the sum-check protocol including the last one. This means that in the last round, $\mathcal{P}_1$ sends a message that is inconsistent with the polynomial Y.

If $\mathcal{P}_1$ does cheat in the last round, the verifier will reject unless, in the final check of the protocol, the verifier winds up choosing a point in $\mathbb{F}^{3k}$ at which $h_{x,y,Y}$ and h_{x,y,Z^*} disagree. This only happens if $\mathcal{V}$ picks a point $r_4 \in \mathbb{F}^k$ for use in the low-degree test such that $Y(r_4) \neq Z(r_4)$. But this occurs with probability only $1 - p = 1 - \gamma + o(1)$. In total, the probability that $\mathcal{P}_1$ passes all tests is therefore at most $1 - \gamma + o(1)$.

[4] In principle, $\mathcal{P}_1$'s claim about $Z(r_4)$ could depend on other messages sent to $\mathcal{P}_1$ by $\mathcal{V}$, namely r_1, r_2, and r_3. This turns out not to help $\mathcal{P}_1$ pass the verifier's checks. In our proof sketch, we simply assume for clarity and brevity that $\mathcal{P}_1$'s claim about $Z(r_4)$ depends on r_4 alone.

If $\gamma > \frac{1}{2}$, this contradicts the fact that $\mathcal{P}_1$ and $\mathcal{P}_2$ convince the MIP verifier to accept with probability at least γ. □

Recall that if $h_{x,y,Y}$ satisfies Equation (8.1), then $g_{x,y,Y}$ vanishes on the Boolean hypercube, and hence Y is an extension of a correct transcript for $\{\mathcal{C}, x, y\}$. So Theorem 8.4 implies that if the MIP verifier accepts with probability $\gamma > \frac{1}{2}$, then there is a correct transcript for $\{\mathcal{C}, x, y\}$.

Although the soundness error can reduced from $\frac{1}{2} + o(1)$ to an arbitrarily small constant with $O(1)$ independent repetitions of the MIP, this would be highly expensive in practice. Fortunately, it is possible to perform a more careful soundness analysis that establishes that the MIP itself, *without repetition*, has soundness error $o(1)$.

The bottleneck in the soundness analysis of Theorem 8.4 that prevents the establishment of soundness error less than $\frac{1}{2}$ is that, if the prover's pass the low-degree test with probability $\gamma < \frac{1}{2}$, then one can only guarantee that there is a polynomial Y that agrees with Z on a γ fraction of points. The verifier will choose a random point r in the sum-check protocol at which Y and Z disagree with probability $1 - \gamma > \frac{1}{2}$, and in this case all bets are off.

The key to the stronger analysis is to use a stronger guarantee from the low-degree test, known as a *list-decoding guarantee*. Roughly speaking, the list-decoding guarantee ensures that if the oracles pass the low-degree test with probability γ, then there is a "small" number of low-degree polynomials $Q_1, Q_2, \ldots$ that "explain" essentially all of the tester's acceptance, in the sense that for almost all points r at which the low-degree test passes, $Z^*(r)$ agrees with $Q_i(r)$ for at least one i. This allows one to argue that even if the provers pass the low-degree test with probability only $\gamma < \frac{1}{2}$, the sum-check protocol will still catch $\mathcal{P}_1$ in a lie with probability very close to 1. Here is a very rough sketch of the analysis. For each polynomial Q_i individually, if $\mathcal{P}_1$ were to claim at the end of its interaction with $\mathcal{V}$ that $Z(r_4) = Q_i(r_4)$, then the probability of $\mathcal{P}_1$ passing all of the verifier's checks is negligible. As there are only a small number of Q_i's, a union bound over all Q_i implies that the probability $\mathcal{P}_1$ passes all of the verifier's checks and is able to claim that $Z(r_4) = Q_i(r_4)$ for *some* Q_i is still negligible. Meanwhile, if $\mathcal{P}_1$ does *not*

claim that $Z(r_4) = Q_i(r_4)$ for some Q_i, the list-decoding guarantee of the low-degree test states that that the provers will fail the low-degree test except with tiny probability.

8.2.1.5 Protocol Costs

Verifier's Costs. $\mathcal{V}$ and $\mathcal{P}_1$ exchanges two messages for each variable of $h_{x,y,Z}$, and where $\mathcal{P}_2$ exchanges two messages in total with $\mathcal{V}$. This is $O(\log S)$ messages in total. Each message from $\mathcal{P}_1$ is a polynomial of degree $O(1)$, while the message from $\mathcal{P}_2$ is a univariate polynomial of total degree $O(\log S)$. In total, all messages can be specified using $O(\log S)$ field elements. As for $\mathcal{V}$'s runtime, the verifier has to process the provers' messages, and then to perform the last check in the sum-check protocol, she must evaluate $\widetilde{\text{add}}$, $\widetilde{\text{mult}}$, $\widetilde{\text{io}}$, and $\widetilde{I}$ at random points. The verifier requires $O(\log S)$ time to process the provers' messages, and Lemma 3.8 implies that $\mathcal{V}$ can evaluate $\widetilde{I}$ at a random point in $O(n)$ time. We assume here that $\widetilde{\text{add}}$, $\widetilde{\text{mult}}$, and $\widetilde{\text{io}}$ can be evaluated at a point in time polylog(S) as well— see Section 4.6.6 and the end of Section 6.5 for discussion of this assumption, and what to do if it does not hold.

Prover's Costs. Blumberg *et al.* [61] showed that, using the techniques developed to implement the prover in the GKR protocol (Section 4.6), specifically Method 2 described there, $\mathcal{P}_1$ can be implemented in $O(S \log S)$ time. In fact, using more advanced techniques (e.g., Lemma 4.5), it is possible to implement the first prover in $O(S)$ time. $\mathcal{P}_2$ needs to specify $\widetilde{W} \circ Q$, where Q is a random line in $\mathbb{F}^k$. Since $\widetilde{W} \circ Q$ is a univariate polynomial of degree $\log S$, it suffices for $\mathcal{P}_2$ to evaluate $\widetilde{W}$ at $1 + \log S$ many points on the line Q—using Lemma 3.8, this can be done in $O(S)$ time per point, resulting in a total runtime of $O(S \log S)$.

8.3 A Succinct Argument for Deep Circuits

Using any polynomial commitment scheme, one can turn the MIP of the previous section into a succinct argument for deep and narrow arithmetic

circuits.[5] Specifically, one gets rid of the second prover, and instead just had the first prover commit to $\tilde{W}$ at the start of the protocol. At the end of the verifier's interaction with the first prover in the MIP above, the first prover makes a claim about $\tilde{W}(r_4)$, which the verifier checks directly by having the prover reveal it via the polynomial commitment protocol.

This succinct argument has an advantage over the approach to succinct argument from Section 7 that was based directly on the GKR protocol: namely, the argument system based on the MIP of the previous section is succinct with a nearly-linear time verifier *even for deep and narrow circuits*. In fact, the MIP-based argument system will have shorter proofs by a factor roughly equal to the depth of the circuit, which can be a significant savings even when the depth is quite small.

The *disadvantage* of the argument system from the previous section is that it applies the polynomial commitment scheme to the entire *transcript extension* $\tilde{W}\colon \mathbb{F}^{\log|\mathcal{C}|} \to \mathbb{F}$, whereas the argument system of Section 7 applied the polynomial commitment scheme only to the multilinear extension of the *witness* $\tilde{w}$. The expense of applying a commitment scheme to $\tilde{w}$ will be much smaller than the expense of applying it to $\tilde{W}$ if the the witness size $|w|$ is much smaller than the circuit size $|\mathcal{C}|$.

Existing polynomial commitment schemes are still the concrete bottlenecks for the prover and verifier in argument systems that use them [226]. Since the witness w can be much smaller than circuit $\mathcal{C}$, applying the polynomial commitment scheme to $\tilde{w}$ can be significantly less expensive than applying it to $\tilde{W}$ (so long as the witness makes up only a small fraction of the total number of gates in the circuit). Besides, we've seen that short, wide circuits are "universal" in the context of succinct arguments, since any RAM running in time T can be turned into an instance of arithmetic circuit satisfiability of size close to T and depth close to $O(\log T)$. In summary, which approach yields a superior argument system for circuit satisfiability in practice depends on many

[5] The polynomial commitment scheme should be extractable in addition to binding. See Section 7.4 for details.

factors, including witness size, circuit depth, the relative importance of proof length vs. other protocol costs, etc.

Remark 8.3. Bitansky and Chiesa [55] gave a different way to transform MIPs into succinct arguments, but their transformation used multiple layers of fully homomorphic encryption, rendering it highly impractical. Unlike the MIP-to-argument transformation in this section, Bitansky and Chiesa's transformation works for *arbitrary* MIPs. The transformation in this section exploits additional structure of the specific MIP of this section, specifically the fact that the sole purpose of the second prover in the MIP is to run a low-degree test. In the setting of succinct arguments, this role played by the second prover can be replaced with a polynomial commitment scheme. In summary, while Bitansky and Chiesa's transformation from MIPs to arguments is more general—applying to arbitrary MIPs, not just those in which the second prover is solely used to run a low-degree test—it is also much less efficient than the transformation of this section.

8.4 Extension from Circuit-SAT to R1CS-SAT

Section 6 gave techniques for turning computer programs into equivalent instances of arithmetic circuit satisfiability, and Section 7 and this section gave succinct non-interactive arguments for arithmetic circuit satisfiability. Arithmetic circuit satisfiability is an example of an *intermediate representation*, a term that refers to any model of computation that is directly amenable to application of interactive proof or argument systems.

A related intermediate representation that has proven popular and convenient in practice is *rank-1 constraint system* (R1CS) instances (also sometimes referred to as *Quadratic Arithmetic Programs* (QAPs)). An R1CS instance is specified by three $m \times n$ matrices A, B, C with entries from a field $\mathbb{F}$ and is satisfiable if and only if there is a vector $z \in \mathbb{F}^n$ with $z_1 = 1$ such that

$$(A \cdot z) \circ (B \cdot z) = C \cdot z. \tag{8.3}$$

Here, $\cdot$ denotes matrix-vector product, and $\circ$ denotes entrywise (a.k.a. Hadamard) product. Any vector z satisfying Equation (8.3) is analogous

to the notion of a "correct transcript" in the context of arithmetic circuit satisfiability (Section 8.2.1.1). We require that the first entry z_1 of z be fixed to 1 because otherwise the all-zeros vector would trivially satisfy *any* R1CS instance, and to ensure that there are efficient transformations from circuit-SAT to R1CS-SAT (see Section 8.4.1).

8.4.1 Relationship Between R1CS-SAT and Arithmetic Circuit-SAT

The R1CS-SAT problem can be thought of as a generalization of the Arithmetic Circuit-SAT problem in the following sense: any instance of Arithmetic Circuit-SAT can be efficiently transformed into instances of R1CS-SAT. The number of rows and columns of the matrices appearing in the resulting R1CS instance is proportional to the number of gates in $\mathcal{C}$, and the number of nonzero entries in any row of the matrices is bounded above by the fan-in of the circuit $\mathcal{C}$. For fan-in two circuits, this means that the equivalent R1CS-SAT instances are sparse, and hence we will ultimately seek protocols where the prover(s) run in time proportional to the number of nonzero entries of these matrices.

To see this, consider an instance $\{\mathcal{C}, x, y\}$ of arithmetic circuit-SAT, i.e., where the prover wants to convince the verifier that there is a w such that $\mathcal{C}(x, w) = y$. We need to construct matrices A, B, C such that there exists a vector z such that Equation (8.3) holds if and only if the preceding sentence is true.

Let N be the sum of the lengths of x, y, and w, plus the number of gates in $\mathcal{C}$, and let $M = N - |w|$. The R1CS-SAT instance will consist of three $M \times (N + 1)$ matrices A, B, and C. We will fix the first entry z of z to 1, and associate each remaining entry of z with either an entry of x, y, or w, or a gate of $\mathcal{C}$.

For an entry j of z corresponding to an entry x_i of x, we define the jth row of A, B, C to capture the constraint the z_j must equal x_i. That is, we set the jth row of A to be the standard basis vector $e_1 \in \mathbb{F}^{N+1}$, the jth row of B to be the standard basis vector $e_j \in \mathbb{F}^{N+1}$, and the jth row of C to be $x_i \cdot e_1$. This means that the jth constraint in the R1CS system asserts that $z_j - x_i = 0$ which is equivalent to demanding that $z_j = x_i$. We include an analogous constraint for each entry j of z corresponding to an entry of y.

For each entry j of z corresponding to an addition gate of $\mathcal{C}$ (with in-neighbors indexed by $j', j'' \in \{2, \ldots, N+1\}$), we define the jth row of A, B, C to capture the constraint that z_j must equal the sum of the two inputs to that addition gate. That is, we set the jth row of A to be the standard basis vector $e_1 \in \mathbb{F}^{N+1}$, the jth row of B to be $e_{j'} + e_{j''} \in \mathbb{F}^{N+1}$, and the jth row of C to be e_j. This means that the jth constraint in the R1CS system asserts that $(z_{j'} + z_{j''}) - z_j = 0$ which is equivalent to demanding that $z_j = z_{j'} + z_{j''}$.

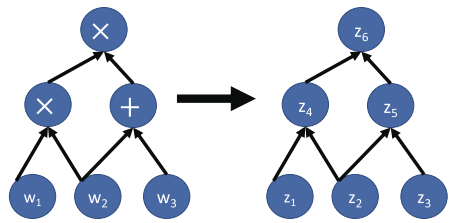
Finally, for each entry j of z corresponding to a multiplication gate of $\mathcal{C}$ (with in-neighbors indexed by $j', j'' \in \{2, \ldots, N+1\}$), we define the jth row of A, B, C to capture the constraint the z_j must equal the product of the two inputs to that gate. That is, we set the jth row of A to be the standard basis vector $e_{j'} \in \mathbb{F}^{N+1}$, the jth row of B to be the standard basis vector $e_{j''} \in \mathbb{F}^{N+1}$, and the jth row of C to be e_j. This means that the jth constraint in the R1CS system asserts that $(z_{j'} \cdot z_{j''}) - z_j = 0$ which is equivalent to demanding that $z_j = z_{j'} \cdot z_{j''}$.

Figure 8.2 has an example circuit and the R1CS instance resulting from the above transformation.

8.4.2 An MIP for R1CS-SAT

As observed in [226], we can apply the ideas of this section to give an MIP and associated succinct argument for R1CS instances. View the matrices A, B, C as functions $f_A, f_B, f_C \colon \{0,1\}^{\log_2 m} \times \{0,1\}^{\log_2 n} \to \mathbb{F}$ in the natural way as per Sections 4.3 and 4.4. Just as in the MIP of this section (Section 8.2.1.2), the prover claims to hold an extension polynomial Z of a correct transcript z for the R1CS instance. Observe that a polynomial $Z \colon \mathbb{F}^{\log_2 m} \times \mathbb{F}^{\log_2 n} \to \mathbb{F}$ extends a correct transcript z for the R1CS instance if and only if the following equation holds for all $a \in \{0,1\}^{\log_2 m}$:

$$\left(\sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_A(a,b) \cdot Z(b) \right) \cdot \left(\sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_B(a,b) \cdot Z(b) \right) - \left(\sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_C(a,b) \cdot Z(b) \right) = 0. \tag{8.4}$$



$$\left(\begin{bmatrix}0&1&0&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&0&1&0&0\\1&0&0&0&0&0&0\end{bmatrix}\begin{bmatrix}1\\z_1\\z_2\\z_3\\z_4\\z_5\\z_6\end{bmatrix}\right)\circ\left(\begin{bmatrix}0&0&1&0&0&0&0\\0&0&1&1&0&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}\begin{bmatrix}1\\z_1\\z_2\\z_3\\z_4\\z_5\\z_6\end{bmatrix}\right)=\begin{bmatrix}0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\y&0&0&0&0&0&0\end{bmatrix}\begin{bmatrix}1\\z_1\\z_2\\z_3\\z_4\\z_5\\z_6\end{bmatrix}$$

Equivalently,

$$\begin{aligned}z_1\cdot z_2&=z_4\\1\cdot(z_2+z_3)&=z_5\\z_4\cdot z_5&=z_6\\1\cdot z_6&=y\end{aligned}$$

Figure 8.2: An arithmetic circuit and an equivalent R1CS instance. Knowing a witness w such that $\mathcal{C}(w)=y$ is equivalent to knowing a vector z that satisfies the constraints of the R1CS. The R1CS instance is expressed in both matrix form and, for readability, as a list of constraints.

Let g_Z denote the $(\log_2(m))$-variate polynomial

$$g_Z(X)=\left(\sum_{b\in\{0,1\}^{\log_2 n}}\tilde{f}_A(X,b)\cdot Z(b)\right)\cdot\left(\sum_{b\in\{0,1\}^{\log_2 n}}\tilde{f}_B(X,b)\cdot Z(b)\right)-\left(\sum_{b\in\{0,1\}^{\log_2 n}}\tilde{f}_C(X,b)\cdot Z(b)\right)\tag{8.5}$$

This polynomial has degree at most 2 in each variable (i.e., it is multi-quadratic), and Equation (8.4) holds if and only if g_Z vanishes at all inputs in $\{0,1\}^{\log_2 m}$.

We obtain an MIP for checking that g_Z vanishes over the Boolean hypercube in a manner analogous to Section 8.2.1.3. Specifically, we can define a related polynomial h_Z by picking a random point $r\in\{0,1\}^{\log_2 m}$

and, in analogy with Equation (8.2), defining

$$h_Z(Y) = \widetilde{\beta}_{\log_2 m}(r, Y) \cdot g_Z(Y).$$

Following the reasoning preceding Equation (8.2), by the Schwartz-Zippel Lemma, it holds that, up to a negligible soundness error (at most $\log_2(m)/|\mathbb{F}|$), g_Z vanishes on the Boolean hypercube if and only if

$$\sum_{a \in \{0,1\}^{\log_2 m}} h_Z(a) = 0.$$

The verifier can compute this last expression by applying the sum-check protocol to the polynomial

$$h_Z(Y) = \widetilde{\beta}_{\log_2 m}(r, Y) \cdot g_Z(Y).$$

After applying the sum-check protocol to $h_Z(Y)$, the verifier needs to evaluate $h_Z(Y)$ at a random input $r' \in \mathbb{F}^{\log_2 m}$. To evaluate $h_Z(r')$, it is enough for the verifier to evaluate $\widetilde{\beta}_{\log_2 m}(r, r')$ and $g_Z(r')$. The former quantity can be evaluated by the verifier in $O(\log_2 m)$ operations in $\mathbb{F}$ using Equation (4.19). The verifier cannot efficiently evaluate $g_Z(r')$ on its own, but by definition (Equation (8.5)), this quantity equals:

$$\left(\sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_A(r', b) \cdot Z(b) \right) \cdot \left(\sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_B(r', b) \cdot Z(b) \right) - \left(\sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_C(r', b) \cdot Z(b) \right). \tag{8.6}$$

This means that to compute $g_Z(r')$, it suffices to apply the sum-check protocol three more times, to the following three $(\log_2(n))$-variate polynomials:

$$p_1(X) = \tilde{f}_A(r', X) \cdot Z(X).$$
$$p_2(X) = \tilde{f}_B(r', X) \cdot Z(X).$$
$$p_3(X) = \tilde{f}_C(r', X) \cdot Z(X).$$

This is because applying the sum-check protocol to $p_1(X)$ computes

$$\left(\sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_A(r', b) \cdot Z(b) \right)$$

and similarly applying the sum-check protocol to p_2 and p_3 computes the remaining two quantities appearing in Equation (8.6). As a concrete optimization, all three invocations of sum-check can be executed in parallel, using the same randomness in each of the three invocations.

At the end of these three final invocations of the sum-check protocol, the verifier needs to evaluate each of p_1, p_2, p_3 at a random input r''. To accomplish this, it suffices for the verifier to evaluate $\tilde{f}_A(r', r'')$, $\tilde{f}_B(r', r'')$, $\tilde{f}_C(r', r'')$, and $Z(r'')$.

At this point, the situation is exactly analogous to the MIP for arithmetic circuit-SAT of Section 8.2.1.3, with $\tilde{f}_A$, $\tilde{f}_B$, and $\tilde{f}_C$ playing the roles of the "wiring predicates" $\widetilde{\text{add}}$ and $\widetilde{\text{mult}}$. That is, for many natural R1CS systems, the verifier can evaluate $\tilde{f}_A$, $\tilde{f}_B$, and $\tilde{f}_C$ in logarithmic time unaided, and $Z(r'')$ can be obtained from the second prover using a low-degree test

Regarding the prover's runtime, we claim that the first prover in the MIP, if given a satisfying assignment $z \in \mathbb{F}^n$ for the R1CS instance, can be implemented in time proportional to the number K of nonzero entries of the matrices A, B, and C. Here, we assume without loss of generality that this number is at least $n + m$, i.e., no row or column of any matrix is all zeros.

We begin by showing that in the first invocation of the sum-check protocol within the MIP, to the polynomial $h_Z(Y)$, the prover can be implemented in time proportional to the number of nonzero entries of A, B, and C. This holds by the following reasoning. First, observe that

$$\begin{aligned} h_Z(Y) = \widetilde{\beta}_{\log_2 m}(r, Y) \cdot g_Z(Y) = \widetilde{\beta}_{\log_2 m}(r, Y) \cdot q_1(Y) \cdot q_2(Y) \\ - \widetilde{\beta}_{\log_2 m}(r, Y) \cdot q_3(Y), \end{aligned} \tag{8.7}$$

where

$$q_1(Y) = \left(\sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_A(Y, b) \cdot Z(b) \right),$$

$$q_2(Y) = \left(\sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_B(Y, b) \cdot Z(b) \right),$$

$$q_3(Y) = \left(\sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_C(Y, b) \cdot Z(b) \right).$$

We wish to apply Lemma 4.5 to conclude that the prover in the sum-check protocol applied the h_Z can be implemented quickly. Since $\widetilde{\beta}_{\log_2 m}(r, Y)$, $q_1(Y)$, $q_2(Y)$, and $q_3(Y)$ are all multilinear polynomials in the variables Y, to apply Lemma 4.5, it is enough to show that all four of these multilinear polynomials can be evaluated at all inputs $a \in \{0,1\}^{\log_2 m}$ in time proportional to the number of nonzero entries of A, B, and C.[6]

First, we observe that $\widetilde{\beta}_{\log_2 m}(r, a)$ can be evaluated by the prover at all inputs $a \in \{0,1\}^{\log_2 m}$ in $O(m)$ total time, as this task is equivalent to evaluating all $(\log m)$-variate Lagrange basis polynomials at input $r \in \mathbb{F}^{\log m}$, which the proof of Lemma 3.8 revealed is possible to achieve in $O(m)$ time.

Second, we turn to the claim that q_1, q_2, and q_3 can be evaluated at all inputs $a \in \{0,1\}^{\log_2 m}$ in the requisite time bound. This holds because, if we interpret $a \in \{0,1\}^{\log_2 m}$ as a number in $\{1, ..., m\}$ and let A_a, B_a, and C_a respectively denote the ath row of A, B, and C, then $q_1(a)$ is simply $A_a \cdot z$, and similarly $q_2(a) = B_a \cdot z$ and $q_3(a) = C_a \cdot z$. Hence all three polynomials can be evaluated at *all* $a \in \{0,1\}^{\log_2(m)}$ in time proportional to the number of nonzero entries of the three matrices A, B, and C.

Similar observations reveal that the prover in the three invocations of the sum-check protocol applied to p_1, p_2, and p_3 can also be implemented in time proportional to the number of nonzero entries of A, B, and C. For example, $p_1(X)$ is a product of two multilinear polynomials $\tilde{f}_A(r', X)$ and $Z(X)$. To apply Lemma 4.5, the evaluations of $Z(X)$ at all inputs $b \in \{0,1\}^{\log_2 n}$ are directly given by the satisfying assignment vector z for the R1CS instance. Turning to $\tilde{f}_A(r', X)$, let $v \in \mathbb{F}^n$ denote the vector of all $(\log_2 n)$-variate Lagrange basis polynomials evaluated at r'. Note that the proof of Lemma 3.8 shows that the vector v can be computed in $O(n)$ time. It can be seen that for $b \in \{0,1\}^{\log_2 n}$, $\tilde{f}_A(r', b)$

[6] Equation (8.7) represents h_Z as a *sum* of products of $O(1)$ multilinear polynomials, while Lemma 4.5 as stated applies only to products of $O(1)$ multilinear polynomials directly. But the lemma extends easily to sums of polynomials, because the honest prover's messages in the sum-check protocol applied to a sum of two polynomials p and q is just the sum of the messages when the sum-check protocol is applied to p and q individually.

is just the inner product of v with the b'th column of A, which (given v) can be computed in time proportional to the number of nonzero entries in this column of A. This completes the explanation of why $p_1(X)$ can be evaluated at all inputs $b \in \{0,1\}^{\log_2 n}$ in time proportional to the number of nonzero entries of A, and similarly for $p_2(X)$ and $p_3(X)$ (with A replaced with B and C respectively).

8.5 MIP = NEXP

In the MIP for arithmetic circuit-SAT of Section 8.2, if the circuit has size S then the verifier's runtime is $\text{poly}(\log S)$, plus the time required to evaluate $\widetilde{\text{add}}$, $\widetilde{\text{mult}}$, and $\widetilde{\text{io}}$ at random inputs. The transformation from computer programs to circuit-SAT instances sketched in Section 6.5 transforms any non-deterministic Random Access Machine running in time T into an arithmetic circuit of size $\tilde{O}(T)$ in which $\widetilde{\text{add}}$, $\widetilde{\text{mult}}$, and $\widetilde{\text{io}}$ can be evaluated at any desired point in time $O(\log T)$. This means that the verifier in the MIP applied to the resulting circuit runs in polynomial time as long as $T \leq 2^{n^c}$ for some constant $c > 0$. In other words, the class of problems solvable in non-deterministic exponential time (**NEXP**) is contained in **MIP**, the class of languages solvable by a multi-prover interactive proof with a polynomial time verifier [17].

The other inclusion, that **MIP** $\subseteq$ **NEXP**, follows from the following simple reasoning. Given any multi-prover interactive proof system for a language $\mathcal{L}$ and input x, one can in non-deterministic exponential time calculate the acceptance probability of the optimal strategy available to provers attempting to convince the verifier to accept, as follows. First, non-deterministically guess the optimal strategies of the provers. Second, compute the acceptance probability that the strategy induces by enumerating over all possible coin tosses of the verifier and seeing how many lead to acceptance when interacting with the optimal prover strategy [113]. Since the multi-prover interactive proof system is a valid MIP for $\mathcal{L}$ this acceptance probability is at least $2/3$ if and only if $x \in \mathcal{L}$.