9

PCPs and Succinct Arguments

9.1 PCPs: Definitions and Relationship to MIPs

In an MIP, if a prover is asked multiple questions by the verifier, then the prover can behave adaptively, which means that the prover's responses to any question can depend on the earlier questions asked by the verifier. This adaptivity was potentially bad for soundness, because the prover's ability to behave adaptively makes it harder to "pin down" the prover(s) in a lie. But, as will become clear below, it was potentially good for efficiency, since an adaptive prover can be asked a sequence of questions, and only needs to "think up" answers to questions that are actually asked.

In contrast, Probabilistically Checkable Proofs (PCPs) have non-adaptivity baked directly into the definition, by considering a verifier $\mathcal{V}$ who is given oracle access to a static proof string π. Since π is static, $\mathcal{V}$ can ask several queries to π, and π's response to any query q_i can depend only on q_i, and not on q_j for $j \neq i$.

Definition 9.1. A probabilistically checkable proof system (PCP) for a language $\mathcal{L} \subseteq \{0,1\}^*$ consists of a probabilistic polynomial time verifier $\mathcal{V}$ who is given access to an input x, and oracle access to a proof string

$\pi \in \Sigma^\ell$. The PCP has completeness error δ_c and soundness error δ_s if the following two properties hold.

(1) *(Completeness)* For every $x \in \mathcal{L}$, there exists a proof string $\pi \in \Sigma^\ell$ such that $\Pr[\mathcal{V}^\pi(x) = \text{accept}] \geq 1 - \delta_c$.

(2) *(Soundness)* For every $x \notin \mathcal{L}$ and every proof string $\pi \in \Sigma^\ell$, $\Pr[\mathcal{V}^\pi(x) = \text{accept}] \leq \delta_s$.

ℓ is referred to as the *length* of the proof, and Σ as the alphabet used for the proof. We think of all of these parameters as functions of the input size n. We refer to the time required to *generate* the honest proof string π as the prover time of the PCP.

Remark 9.1. The PCP model was introduced by Fortnow *et al.* [113], who referred to it as the "oracle" model (we used this terminology in Lemma 8.2). The term Probabilistically Checkable Proofs was coined by Arora and Safra [11].

Remark 9.2. Traditionally, the notation $\mathbf{PCP}_{\delta_c,\delta_s}[r, q]_\Sigma$ is used to denote the class of languages that have a PCP verifier with completeness error δ_c, soundness error δ_s, and in which the verifier uses at most r random bits, and makes at most q queries to a proof string π over alphabet Σ. This notation is motivated in part by the importance of the parameters r, q, and Σ in applications to hardness of approximation. In the setting of verifiable computing, the most important costs are typically the verifier's and prover's runtime, and the total number q of queries (since, when PCPs are transformed into succinct arguments, the proof length of the argument is largely determined by q). Note however that the proof length ℓ is a *lower bound* on the prover's runtime in any PCP system since it takes time at least ℓ to write down a proof of length ℓ. Hence, obtaining a PCP with a small proof length is necessary, but not sufficient, for developing a PCP system with an efficient prover.

PCPs and MIPs are closely related: any MIP can turned into a PCP, and vice versa. However, both transformations can result in a substantial increase in costs. The easier direction is turning an MIP into a PCP. This simple transformation dates back to Fortnow, Rompel,

and Sipser, who introduced the PCP model, albeit under a different name.

Lemma 9.2. Suppose $\mathcal{L} \subseteq \{0,1\}^*$ has a k-prover MIP in which $\mathcal{V}$ sends exactly one message to each prover, with each message consisting of at most r_Q bits, and each prover sends at most r_A bits in response to the verifier. Then $\mathcal{L}$ has a k-query PCP system over an alphabet Σ of size 2^{r_A}, where the proof length is $k \cdot 2^{r_Q}$, with the same verifier runtime and soundness and completeness errors as the MIP.

Sketch. For every prover $\mathcal{P}_i$ in the MIP, the PCP proof has an entry for every possible message that $\mathcal{V}$ might send to $\mathcal{P}_i$. The entry is equal to the prover's response to that message from $\mathcal{V}$. The PCP verifier simulates the MIP verifier, treating the proof entries as prover answers in the MIP. □

Remark 9.3. It is also straightforward to obtain a PCP from a k-prover MIP in which $\mathcal{V}$ sends multiple messages to each prover. If each prover $\mathcal{P}_i$ is sent z messages $m_{i,1}, \ldots, m_{i,z}$ in the MIP, obtain a new MIP by replacing $\mathcal{P}_i$ with z provers $\mathcal{P}_{i,1}, \ldots, \mathcal{P}_{i,z}$ who are each sent one message (the message to $\mathcal{P}_{i,j}$ being the concatenation of $m_{i,1}, \ldots, m_{i,j}$).[1] The verifier in the $(z \cdot k)$-prover MIP simulates the verifier in the k-prover MIP, treating $\mathcal{P}_{i,j}$'s answer as if it were $\mathcal{P}_i$'s answer to $m_{i,j}$. Completeness of the resulting $(z \cdot k)$-prover MIP follows from completeness of the original k-prover MIP, by having prover $\mathcal{P}_{i,j}$ answer the same as $\mathcal{P}_i$ would upon receiving message $m_{i,j}$. Soundness of the resulting $(z \cdot k)$-prover MIP is implied by soundness of the original k-prover MIP.

Finally, apply Lemma 9.2 to the resulting $(z \cdot k)$-prover MIP.

Lemma 9.2 highlights a fundamental difference between MIPs and PCPs: in a PCP, the prover must pre-compute a response for every possible query of the verifier, which will result in a very large prover runtime unless the number of possible queries from the verifier is small.

[1] The reason $\mathcal{P}_{i,j}$ must be sent the concatenation of the first j messages to $\mathcal{P}_i$ rather than just $m_{i,j}$ is to ensure completeness of the resulting $(z \cdot k)$-prover MIP. $\mathcal{P}_i$'s answer to $m_{i,j}$ is allowed to depend on all preceding messages $m_{i,1}, \ldots, m_{i,j-1}$. So in order for $\mathcal{P}_{i,j}$ to be able to determine $\mathcal{P}_i$'s answer to $m_{i,j}$, it may be necessary for $\mathcal{P}_{i,j}$ to know $m_{i,1}, \ldots, m_{i,j-1}$.

Whereas in an MIP, the provers only need to compute responses "on demand", ignoring any queries that the verifier *might* have asked, but did not. Hence, the MIP $\Longrightarrow$ PCP transformation of Lemma 9.2 may cause a huge blowup in prover runtime.

Lemma 8.2 gave a transformation from a PCP to a 2-prover MIP, but this transformation was also expensive. In summary, the tasks of constructing efficient MIPs and PCPs are incomparable. On the one hand, PCP provers are inherently non-adaptive, but they must pre-compute the answers to all possible queries of the verifier. MIP provers only need to compute answers "on demand", but they can behave adaptively, and while there are generic techniques to force them to behave non-adaptively, these techniques are expensive.

9.2 Compiling a PCP Into a Succinct Argument

We saw in Section 7 that one can turn the GKR interactive proof for arithmetic circuit *evaluation* into a succinct argument for arithmetic circuit *satisfiability* (recall that the goal of a circuit satisfiability instance $\{C, x, y\}$ is to determine whether there exists a witness w such that $\mathcal{C}(x, w) = y$). At the start of the argument, the prover sends a cryptographic commitment to the multilinear extension $\tilde{w}$ of a witness w. The prover and verifier then run the GKR protocol to check that $\mathcal{C}(x, w) = y$. At the end of the GKR protocol, the prover is forced to make a claim about the value of $\tilde{w}(r)$ for a random point r. The argument system verifier confirms that this claim is consistent with the corresponding claim derived from the cryptographic commitment to $\tilde{w}$.

The polynomial commitment scheme described in Section 7.3.2.2 consisted of two pieces; a *string-commitment* scheme using a Merkle tree, which allowed the prover to commit to *some* fixed function claim to equal $\tilde{w}$, and a low-degree test, which allowed the verifier to check that the function committed to was indeed (close to) a low-degree polynomial.

If our goal is to transform a PCP rather than an interactive proof into a succinct argument, we can use a similar approach, but omit the low-degree test. Specifically, as explained below, Kilian [168] famously showed that any PCP can be combined with Merkle-hashing to yield

four-message argument systems for all of **NP**, assuming that collision-resistant hash functions exist. The prover and verifier runtimes are the same as in the underlying PCP, up to low-order factors, and the total communication cost is $O(\log n)$ cryptographic hash values per PCP query. Micali [192] showed that applying the Fiat-Shamir transformation to the resulting four-message argument system yields a *non-interactive* succinct argument in the random oracle model.[2]

The idea is the following. The argument system consists of two phases: commit and reveal. In the commit phase, the prover writes down the PCP π, but doesn't send it to the verifier. Instead, the prover builds a Merkle tree, with the symbols of the PCP as the leaves, and sends the root hash of the tree to the verifier. This binds the prover to the string π. In the reveal phase, the argument system verifier simulates the PCP verifier to determine which symbols of π need to be examined (call the locations that the PCP verifier queries $q_1, \ldots, q_k$). The verifier sends $q_1, \ldots, q_k$ to the prover to $\mathcal{P}$, and the prover sends back the answers $\pi(q_1), \ldots, \pi(q_k)$, along with their authentication paths.

Completeness can be argued as follows. If the PCP satisfies perfect completeness, then whenever there exists a w such that $\mathcal{C}(x, w) = y$, there is always some proof π that would convince the PCP verifier to accept. Hence, if the prover commits to π in the argument system, and executes the reveal phase as prescribed, the argument system verifier will also be convinced to accept.

Soundness can be argued roughly as follows. The analysis of Section 7.3.2.2 showed that the use of the Merkle tree binds the prover to a fixed string π', in the sense that after the commit phase, for each possible query q_i, there is at most one value $\pi'(q_i)$ that the prover can successfully reveal without finding a collision under the hash function used to build the Merkle tree (and collision-finding is assumed to be intractable). Hence, if the argument system prover convinces the argument system verifier to accept, π' would convince the PCP verifier

[2] In the non-interactive argument obtained by applying the Fiat-Shamir transformation to Kilian's 4-message argument, the honest prover uses the random oracle in place of a collision-resistant hash function to build the Merkle tree over the PCP proof, and the PCP verifier's random coins are chosen by querying the random oracle at the root hash of the Merkle tree.

to accept. Soundness of the argument system is then immediate from soundness of the PCP system.

Remark 9.4. In order to turn a PCP into a succinct argument, we used a Merkle tree, and did not need to use a low-degree test. This is in contrast to Section 7.3, where we turned an interactive proof into a succinct argument by using a polynomial commitment scheme; the polynomial commitment scheme given in Section 7.3 combined a Merkle tree and a low-degree test.

However, the PCP approach to building succinct arguments has not "really" gotten rid of the low-degree test. It has just pushed it out of the commitment scheme and "into" the PCP. That is, short PCPs are themselves typically based on low-degree polynomials, and the PCP itself typically makes use of a low-degree test.

A difference between the low-degree tests that normally go into short PCPs and the low-degree tests we've already seen is that short PCPs are usually based on low-degree *univariate* polynomials (see Section 9.4 for details). So the low-degree tests that go into short PCPs are targeted at univariate rather than multi-variate polynomials. Low-degree univariate polynomials are codewords in the Reed-Solomon error-correcting code, which is why many papers on PCPs refer to "Reed-Solomon PCPs" and "Reed-Solomon testing". In contrast, efficient interactive proofs and MIPs are typically based on low-degree multivariate polynomials (also known as Reed-Muller codes), and hence use low-degree tests that are tailored to the multivariate setting.

9.2.1 Knowledge-Soundness of Kilian and Micali's Arguments

Recall (see Section 7.4) that an argument system satisfies knowledge-soundness if, for any efficient prover $\mathcal{P}$ that convinces the argument system verifier to accept with non-negligible probability, $\mathcal{P}$ must *know* a witness w to the claim being proven. This is formalized by demanding that there is an efficient algorithm $\mathcal{E}$ that is capable of outputting a valid witness if given the ability to repeatedly "run" $\mathcal{P}$.

Barak and Goldreich [19] showed that Kilian's argument system is not only sound, but in fact knowledge-sound. This assertion assumes that the underlying PCP that the argument system is based on also

satisfies an analogous knowledge-soundness property, meaning that given a convincing PCP proof π, one can efficiently compute a witness. All of the PCPs that we cover in this survey have this knowledge-soundness property.

Valiant [239] furthermore showed that applying the Fiat-Shamir transformation to render Kilian's argument system non-interactive (as per Micali [192]) yields a *knowledge-sound* argument in the random oracle model. Recall that the Fiat-Shamir transformation "removes" from Kilian's argument system the verifier's message specifying the symbols of the committed PCP proof that it wishes to query. This message is chosen to equal the evaluation of the random oracle at the argument-system prover's first message, which specifies the Merkle-hash of the committed PCP proof.

The rough idea of Valiant's analysis is to show that, if a prover $\mathcal{P}$ in the Fiat-Shamir-ed protocol produces an accepting transcript for Kilian's interactive protocol, then one of following three events must have occurred: either (1) $\mathcal{P}$ found a "hash collision" enabling it to break binding of the Merkle tree, or (2) $\mathcal{P}$ built Merkle trees over one or more "unconvincing" PCP proofs π, yet applying the Fiat-Shamir transformation to determine which symbols of π are queried caused the PCP verifier to accept π anyway, or (3) $\mathcal{P}$ built a Merkle tree over a "convincing" PCP proof π, and the first message of the transcript produced by $\mathcal{P}$ is the root hash of this Merkle tree.

The first event is unlikely to occur unless the prover makes a huge number of queries to the random oracle. This is because the probability of finding a collision after T queries to the random oracle is at most $T^2/2^\lambda$ where 2^λ is the output length of the random oracle. The second event is also unlikely to occur, assuming the soundness error ε of the PCP is negligible. Specifically, if the prover makes T queries to the random oracle, the probability event (2) occurs is at most $T \cdot \varepsilon$.

This means that (3) must hold (unless $\mathcal{P}$ makes super-polynomially many queries to the random oracle). That is, any prover $\mathcal{P}$ for the non-interactive argument that produces accepting transcripts with non-negligible probability must build a Merkle tree over a convincing PCP proof π and produce a transcript whose first message is the root hash of the Merkle tree. In this case, one can identify the entire Merkle tree by

observing $\mathcal{P}$'s queries to the random oracle. For example, if v_0 denotes the root hash provided in the transcript, then one can learn the values v_1, v_2 of the children of the root in the Merkle tree by looking for the (unique) query (v_1, v_2) made by $\mathcal{P}$ to the random oracle R satisfying $R(v_1, v_2) = v_0$. Then one can learn the values of the grandchildren of the root by looking for the (unique) random oracle queries (v_3, v_4) and (v_5, v_6) made by $\mathcal{P}$ such that $R(v_3, v_4) = v_1$ and $R(v_5, v_6) = v_2$. And so on.

The values of the leaves of the Merkle tree are just the symbols of the convincing PCP proof π. By assumption that the PCP system satisfies knowledge-soundness, one can efficiently extract a witness from π.

The next section (Section 10) covers IOPs, an interactive generalization of PCPs. Ben-Sasson *et al.* [44] generalized Micali's PCP-to-SNARK transformation to an IOP-to-SNARK transformation, and via an analysis similar to Valiant's, established that the transformation preserves knowledge-soundness of the IOP.[3] See Section 10.1 for details of the IOP-to-SNARK transformation.

9.3 A First Polynomial Length PCP, From an MIP

In light of Lemma 9.2, it is reasonable to ask whether the MIP of Section 8.2 can be transformed into a PCP for arithmetic circuit satisfiability, of length polynomial in the circuit size S. The answer is yes, though the polynomial is quite large—at least S^3.

Suppose we are given an instance $(\mathcal{C}, x, y)$ of arithmetic circuit satisfiability, where $\mathcal{C}$ is defined over field $\mathbb{F}$. Recall that in the MIP of Section 8.2, the verifier used the first prover to apply the sum-check protocol to a certain $(3 \log S)$-variate polynomial $h_{x,y,Z}$ over $\mathbb{F}$, where S is the size of $\mathcal{C}$. This polynomial was itself derived from a polynomial Z, claimed to equal the multilinear extension of a correct transcript for $(\mathcal{C}, x, y)$. The MIP verifier used the second prover to apply the point-vs-line low-degree test to the $O(\log S)$-variate polynomial Z, which required the verifier to send $\mathcal{P}_2$ a random line in $\mathbb{F}^{\log S}$ (such a line can be specified

[3] More precisely, knowledge-soundness of the resulting SNARK is characterized by knowledge-soundness of the IOP against a class of attacks called state-restoration attacks, discussed in Section 5.2.

with $2 \log S$ field elements). In order to achieve a soundness error of, say, $1/\log(n)$, it was sufficient to work over a field $\mathbb{F}$ of size at least $\log(S)^{c_0}$ for a sufficiently large constant $c_0 > 0$.[4]

The total number of bits that the verifier sent to each prover in this MIP was $r_Q = \Theta(\log(S) \log |\mathbb{F}|)$, since the verifier had to send a field element for each variable over which $h_{x,y,Z}$ was defined. If $|\mathbb{F}| = \Theta(\log(S)^c)$, then $r_Q = \Theta(\log(S) \log\log(S))$. Applying Lemma 9.2 and Remark 9.3 to transform this MIP into a PCP, we obtain a PCP of length $\tilde{O}\left(2^{r_Q}\right) = S^{O(\log\log S)}$. This is slightly superpolynomial in S. On the plus side, the verifier runs in time $O(n + \log S)$, which is linear in the size n of the input assuming $S < 2^n$.

However, by tweaking the parameters used within the MIP itself, we can reduce r_Q from $O(\log(S) \log\log(S))$ to $O(\log S)$. Recall that within the MIP, each gate in $\mathcal{C}$ was assigned a *binary* label, and the MIP made use of functions add_i, mult_i, io, I, and W that take as input $O(\log S)$ binary variables representing the labels of one or more gates. The polynomial $h_{x,y,Z}$ was then defined in terms of the *multilinear* extensions of these functions. This led to an efficient MIP, in which the provers' runtime was $O(S \log S)$. But by defining the polynomials to be over $\Omega(\log S)$ many variables, r_Q becomes slightly super logarithmic, resulting in a PCP of length superpolynomial in S. To rectify this, we must find a way to redefine the polynomials, such that they involve fewer than $\log S$ variables.

To this end, suppose we assign each gate in $\mathcal{C}$ a label in base b instead of base 2. That is, each gate label will consist of $\log_b(S)$ digits, each in $\{0, 1, \ldots, b-1\}$. Then we can redefine the functions add_i, mult_i, io, I, and W to take as input $O(\log_b(S))$ variables representing the b-ary labels of one or more gates. Observe that, the larger b is, the smaller the number of variables these functions are defined over.

[4] In cryptographic applications, one would want soundness error $n^{-\omega(1)}$ rather than $1/\log n$. The soundness error of the PCP in this section could be improved to $n^{-\omega(1)}$ by repeating the PCP $O(\log n)$ times independently, and rejecting if the PCP verifier rejects in any one of the runs. Such repetition is expensive in practice, but the PCP of this section is presented for didactic reasons and not meant to be practical.

We can then define $h_{x,y,Z}$ exactly as in Section 8.2, except if $b > 2$ then higher-degree extensions of add_i, mult_i, io, I, and W must be used in the definition, rather than multilinear extensions. Specifically, these functions, when defined over domain $\{0, 1, \ldots, b-1\}^v$ for the relevant value of v, each have a suitable extension of degree at most b in each variable.

Compared to the MIP of Section 8.2, the use of the higher-degree extensions increases the degrees of all of the polynomials exchanged in the sum-check protocol and in the low-degree test by an $O(b)$ factor. Nonetheless, the soundness error remains at most $O\left(b \cdot \log_b(S)/|\mathbb{F}|^c\right)$ for some constant $c > 0$. Recall that we would like to take b as large as possible, but this is in tension with the requirement to keep the soundness error $o(1)$ when working over a field of size polylogarithmic in S. Fortunately, it can be checked that if $b \leq O\left(\log(S)/\log\log(S)\right)$, then $b \cdot \log_b(S) \leq \text{polylog}(S)$, and hence the soundness error is still at most $\text{polylog}(S)/|\mathbb{F}|^c$. In conclusion, if we set b to be on the order of $\log(S)/\log\log(S)$, then as long as the MIP works over a field $\mathbb{F}$ of size that is a sufficiently large polynomial in $\log(S)$, the soundness error of the MIP is still at most, say, $1/\log n$.

For simplicity, let us choose b such that $b^b = S$. This choice of b is in the interval $[b_1, 2b_1]$ where $b_1 = \log(S)/\log\log(S)$.[5] In this modified MIP, the total number of bits sent from the verifier to the provers is $r_Q = O\left(b \cdot \log|\mathbb{F}|\right) = O\left(\left(\log(S)/\log\log(S)\right) \cdot \log\log S\right) = O(\log S)$. If we apply Lemma 9.2 and Remark 9.3 to this MIP, the resulting PCP length is $\tilde{O}\left(2^{r_Q}\right) \leq \text{poly}(S)$.

Unfortunately, when we write $r_Q = O(\log S)$, the constant hidden by the Big-Oh notation is at least 3. This is because $h_{x,y,Z}$ is defined over $3\log_b(S)$ variables, which is at least $3b$ when $b^b = S$, and applying the sum-check protocol to $h_{x,y,Z}$ requires $\mathcal{V}$ to send at least one field element per variable. Meanwhile, the field size must be at least $3\log_b(S) \geq 3b$ to ensure non-trivial soundness. Hence, $2^{r_Q} \geq (3b)^{3b} \geq S^{3-o(1)}$. So while the proof length of the PCP is polynomial in S, it is a large polynomial in S.

[5] Indeed, $b_1^{b_1} \leq S$, while $(2b_1)^{2b_1} \geq S^{2-o(1)}$.

Table 9.1: Costs of PCP of Section 9.3 for arithmetic circuit satisfiability (obtained from the MIP of Section 8.2), when run on a circuit $\mathcal{C}$ of size S. The stated bound on $\mathcal{P}$'s time assumes $\mathcal{P}$ knows a witness w for $\mathcal{C}$.

Communication	Queries	$\mathcal{V}$ Time	$\mathcal{P}$ Time
$\text{polylog}(S)$ bits	$O(\log S / \log\log S)$	$O(n + \text{polylog}(S))$	$\text{poly}(S)$

Nonetheless, this yields a non-trivial result: a PCP for arithmetic circuit satisfiability in which the prover's runtime is $\text{poly}(S)$, the verifier's is $O(n)$, and the number of queries the verifier makes to the proof oracle is $O(\log(S)/\log\log(S))$. As the total communication cost of the MIP is at most $\text{polylog}(S)$, all of the answers to the verifier's queries can be communicated in $\text{polylog}(S)$ bits in total (i.e., the alphabet size of the PCP is $|\Sigma| \leq 2^{\text{polylog}(S)}$). The costs of the PCP are summarized in Table 9.1. Applying the PCP-to-argument compiler of Section 9.2 yields a succinct argument for arithmetic circuit satisfiability with a verifier that runs in time $O(n)$ and a prover that runs in time $\text{poly}(S)$.

Remark 9.5. To clarify, the use of labels in base $b = 2$ rather than base $b = \Theta(\log(S)/\log\log(S))$ is the superior choice in interactive settings such as IPs and MIPs, if the goal is to minimize total communication. The reason is that binary labels allow the IP or MIP prover(s) to send polynomials of degree $O(1)$ in each round, and this keeps the communication costs low.

To recap, we have obtained a PCP for arithmetic circuit satisfiability with a linear time verifier, and a prover who can generate the proof in time *polynomial* in the size of the circuit. But to get a PCP that has any hope of being practical, we really need the prover time to be very close to *linear* in the size of the circuit. Obtaining such PCPs is quite complicated and challenging. Indeed, researchers have not had success in building plausibly practical VC protocols based on "short" PCPs, by which we mean PCPs for circuit satisfiability whose length is close to linear in the size of the circuit. To mitigate the bottlenecks in known short PCP constructions, researchers have turned to the more general *interactive oracle proof* (IOP) model. The following section and section cover highlights from this line of work. Specifically, Section 9.4

sketches the construction of PCPs for arithmetic circuit satisfiability where the PCP can be generated in time *quasilinear* in the size of the circuit. This construction remains impractical and is included in this survey primarily for historical context. Section 10 describes IOPs that come closer to practicality.

9.4 A PCP of Quasilinear Length for Arithmetic Circuit Satisfiability

We have just seen (Sections 9.1 and 9.3) that known MIPs can fairly directly yield a PCP of polynomial size for simulating a (non-deterministic) Random Access Machine (RAM) M, in which the verifier runs in time linear in the size of the input x to M. But the proof length is a (possibly quite large) polynomial in the runtime T of M, and the length of a proof is of course a lower bound on the time required to generate it. This section describes how to use techniques tailored specifically to the PCP model to reduce the PCP length to $T \cdot \text{polylog}(T)$, while maintaining a verifier runtime of $n \cdot \text{polylog}(T)$.

The PCP described here originates in work of Ben-Sasson and Sudan [50]. Their work gave a PCP of size $\tilde{O}(T)$ in which the verifier runs in time $\text{poly}(n)$ and makes only a polylogarithmic number of queries to the proof oracle. Subsequent work by Ben-Sasson *et al.* [48] reduced the verifier's time to $n \cdot \text{polylog}(T)$. Finally, Ben-Sasson *et al.* [41] showed how the prover can actually generate the PCP in $T \cdot \text{polylog}(T)$ time using FFT techniques, and provided various concrete optimizations and improved soundness analysis. This PCP system is fairly involved, so we elide some details in this survey, seeking only to convey the main ideas. The PCP's costs are summarized in Table 9.2.

9.4.1 Step 1: Reduce to Checking That a Polynomial Vanishes on a Designated Subspace

In Ben-Sasson and Sudan's PCP, the claim that $M(x) = y$ is first turned into an equivalent circuit satisfiability instance $\{\mathcal{C}, x, y\}$, and the prover (or more precisely, the proof string π) claims to be holding a low-degree extension Z of a correct transcript W for $\{\mathcal{C}, x, y\}$, just like in the MIP of Section 8.2. And just as in the MIP, the first step of Ben-Sasson and

Table 9.2: Costs of PCP from Section 9.4 when run on a non-deterministic circuit $\mathcal{C}$ of size S. The PCP is due to Ben-Sasson and Sudan [50], as refined by Ben-Sasson *et al.* [48] and [41]. The stated bound on $\mathcal{P}$'s time assumes $\mathcal{P}$ knows a witness w for $\mathcal{C}$

Communication	**Queries**	$\mathcal{V}$ **Time**	$\mathcal{P}$ **Time**
polylog(S) bits	polylog(S)	$O(n \cdot \text{polylog}(S))$	$O(S \cdot \text{polylog}(S))$

Sudan's PCP is to construct a polynomial $g_{x,y,Z}$ such that Z extends a correct transcript for $\{\mathcal{C}, x, y\}$ if and only if $g_{x,y,Z}(a) = 0$ for all a in a certain set H.

The details, however, are different and somewhat more involved than the construction in the MIP. We elide several of these details here, and focus on highlighting the primary simlarities and differences between the constructions in the PCP and the MIP of Section 8.2.

Most importantly, in the PCP, $g_{x,y,Z}$ ***is a univariate polynomial.*** The PCP views a correct transcript as a univariate function $W : [S] \to \mathbb{F}$ rather than as a v-variate function (for $v = \log S$) mapping $\{0,1\}^v$ to $\mathbb{F}$ as in the MIP. Hence, any extension Z of W is a univariate polynomial, and $g_{x,y,Z}$ is defined to be a univariate polynomial too. (The reason for using univariate polynomials is that it allows the PCP to utilize low-degree testing techniques in Steps 2 and 3 below that are tailored to univariate rather than multivariate polynomials. It is not currently known how to obtain PCPs of quasilinear length based on multivariate techniques, where by quasilinear length, we mean quasilinear in T, the runtime of the RAM that the prover is supposed to execute). Note that even the lowest-degree extension Z of W may have degree $|S| - 1$, which is much larger than the degrees of the multivariate polynomials that we've used in previous sections, and $g_{x,y,Z}$ will inherit this degree.

The univariate nature of $g_{x,y,Z}$ forces several additional differences in its construction, compared to the $O(\log S)$-variate polynomial used in the MIP. In particular, in the univariate setting, $g_{x,y,Z}$ is specifically

defined over a field of characteristic 2.[6] The structure of fields of characteristic 2 are exploited multiple times in the construction of $g_{x,y,Z}$ and in the PCP as a whole. For example:

- Let us briefly recall a key aspect of the transformation from Section 6.5 that turned a RAM M into an equivalent circuit satisfiability instance $\{\mathcal{C}, x, y\}$. De Bruijn graphs played a role in the construction of $\mathcal{C}$, where they were used to "re-sort" a purported trace of the execution of M from time order into memory order.

  To ensure that the MIP or PCP verifier does not have to fully materialize $\mathcal{C}$ (which is of size at least T, far larger than the verifier's allowed runtime of $n \cdot \text{polylog}(T)$), it is essential that $\mathcal{C}$ have an "algebraically regular" wiring pattern. In particular, in both the PCP of this section and the MIP of Section 8.2, it is important that $\mathcal{C}$'s wiring pattern be "capturable" by a low-degree polynomial that the verifier can quickly evaluate. This is essential for ensuring that the polynomial $g_{x,y,Z}$ used within the MIP or PCP satisfies the following two essential qualities: (1) the degree of $g_{x,y,Z}$ is not much larger than that of Z (2) the verifier can efficiently evaluate $g_{x,y,Z}(r)$ at any point r, if given Z's values at a handful of points derived from r.

  In Ben-Sasson and Sudan's PCP, the construction of $g_{x,y,Z}$ exploits the fact that there is a way to assign labels from $\mathbb{F} = \mathbb{F}_{2^\ell}$ to nodes in a De Bruijn graph such that, for each node v, the labels of the neighbors of v are *affine* (i.e., degree 1) functions of the label of v. (Similar to Section 6.5, the reason this holds boils down to the fact that the neighbors of a node with label v are simple bit-shifts of v. When v is an element of $\mathbb{F}_{2^\ell}$, a bit-shift of v is an affine function of v.).

[6] The characteristic of a field $\mathbb{F}$ is the smallest number n such that $\underbrace{1+1+\cdots+1}_{n \text{ times}} = 0$. If a $\mathbb{F}$ has size p^k for prime p and integer $k > 0$, then its characteristic is p. In particular, any field of size equal to a power of 2 has characteristic 2. We denote the field of size 2^k as $\mathbb{F}_{2^k}$.

This is crucial for ensuring that the degree of $g_{x,y,Z}$ is not much larger than the degree of Z itself. In particular, the univariate polynomial $g_{x,y,Z}$ over $\mathbb{F}$ used in the PCP has the form

$$g_{x,y,Z}(z) = A(z, Z(N_1(z)), \ldots, Z(N_k(z))), \tag{9.1}$$

where $(N_1(z), \ldots, N_k(z))$ denotes the neighbors of node z in the De Bruijn graph, and A is a certain "constraint polynomial" of polylogarithmic degree. Since $N_1, \ldots, N_k$ are affine over $\mathbb{F}_{2^\ell}$, $\deg(g_{x,y,Z})$ is at most a polylogarithmic factor larger than the degree of Z itself. Moreover, the verifier can efficiently evaluate each affine function $N_1, \ldots, N_k$ at a specified input r [48].

- The set H on which $g_{x,y,Z}$ should vanish if Z extends a correct transcript is chosen to ensure that the polynomial $\mathbb{Z}_H(z) = \prod_{\alpha \in H}(z - \alpha)$ is *sparse* (having $O(\text{polylog}(S))$ nonzero coefficients). The polynomial $\mathbb{Z}_H$ is referred to as the *vanishing polynomial* for H, and via Lemma 9.3 in the next section, it plays a central role in the PCPs, IOPs (Section 10), and linear PCPs (Section 17.4) described hereafter in this survey. The sparsity of $\mathbb{Z}_H$ ensures that it can be evaluated an any point in polylogarithmic time, even though H is a very large set (of size $\Omega(S)$). This will be crucial to allowing the verifier to run in polylogarithmic time in Step 2 of the PCP, discussed below. It turns out that if $\mathbb{F}$ has characteristic $O(1)$ and H is a linear subspace of $\mathbb{F}$, then $\mathbb{Z}_H(z)$ has sparsity $O(\log S)$ as desired. Later in this monograph (e.g., in the IOP of Section 10.3.2), H will instead consist of all n'th roots of unity in $\mathbb{F}$, in which case $\mathbb{Z}_H(z) = z^n - 1$ is clearly sparse.

The final difference worth highlighting is that the field $\mathbb{F}_{2^\ell}$ over which $g_{x,y,Z}$ is defined must be small in the PCP (or, at least, the set of inputs at which the verifier might query $g_{x,y,Z}$ must be a small). In particular, the set must be of size $O(S \cdot \text{polylog}(S))$, since the proof length is lower bounded by the size of the set of inputs at which the verifier might ask for any evaluation of $g_{x,y,Z}$. This is in contrast to the MIP setting, where we were happy to work a very large field size (of size, say, 2^{128} or larger) to ensure negligible soundness error. This is a

manifestation of the fact (mentioned in Section 9.1) that in an MIP the prover only has to "think up" answers to queries that the verifier actually asks, while in a PCP, the prover has to write down the answer to every possible query that the verifier *might* ask.

9.4.2 Step 2: Reduce to Checking That a Related Polynomial is Low-Degree

Note that checking whether a low-degree polynomial $g_{x,y,W}$ vanishes on H is very similar to the core statement checked in our MIP from Section 8.2. There, we checked that a *multilinear* polynomial derived from x, y, and W vanished on all Boolean inputs. Here, we are checking whether a *univariate* polynomial $g_{x,y,W}$ vanishes on all inputs in a pre-specified set H. We will rely on the following simple but essential lemma, which will arise several other times in this survey (including when we cover linear PCPs in Section 17).

Lemma 9.3. (Ben-Sasson and Sudan [50]) Let $\mathbb{F}$ be a field and $H \subseteq \mathbb{F}$. For $d \geq |H|$, a degree-d univariate polynomial g over $\mathbb{F}$ vanishes on H if and only if the polynomial $\mathbb{Z}_H(t) \coloneqq \prod_{\alpha \in H}(t - \alpha)$ divides g, i.e., if and only if there exists a polynomial h^* with $\deg(h^*) \leq d - |H|$ such that $g = \mathbb{Z}_H \cdot h^*$.

Proof. If $g = \mathbb{Z}_H \cdot h^*$, then for any $\alpha \in H$, it holds that $g(\alpha) = \mathbb{Z}_H(\alpha) \cdot h^*(\alpha) = 0 \cdot \alpha = 0$, so g indeed vanishes on H.

For the other direction, observe that if $g(\alpha) = 0$, then the polynomial $(t - \alpha)$ divides $g(t)$. It follows immediately that if g vanishes on H, then g is divisible by $\mathbb{Z}_H$. □

So to convince $\mathcal{V}$ that $g_{x,y,Z}$ vanishes on H, the proof merely needs to convince $\mathcal{V}$ that $g_{x,y,Z}(z) = \mathbb{Z}_H(z) \cdot h^*(z)$ for some polynomial h^* of degree $d - |H|$. To be convinced of this, $\mathcal{V}$ can pick a random point $r \in \mathbb{F}$ and check that

$$g_{x,y,Z}(r) = \mathbb{Z}_H(r) \cdot h^*(r). \tag{9.2}$$

Indeed, if $g_{x,y,Z} \neq \mathbb{Z}_H \cdot h^*$, then this equality will fail with probability $\frac{999}{1000}$ as long as $|\mathbb{F}|$ is at least 1000 times larger than the degrees of $g_{x,y,Z}$ and $\mathbb{Z}_H \cdot h^*$.

A PCP convincing $\mathcal{V}$ that Equation (9.2) holds consists of four parts. The first part contains the evaluations of $Z(z)$ for all $z \in \mathbb{F}$. The second part contains a proof π_Z that Z has degree at most $|H| - 1$, and hence that $g_{x,y,Z}$ has degree at most $d = |H| \cdot \text{polylog}(S)$. The third part contains the evaluation of $h^*(z)$ for all $z \in \mathbb{F}$. The fourth part purportedly contains a proof π_{h^*} that $h^*(z)$ has degree at most $d - |H|$, and hence that $\mathbb{Z}_H \cdot h^*$ has degree at most d.

Let us assume that the verifier can efficiently check π_Z and π_{h^*} to confirm that Z and $h^*(z)$ have the claimed degrees (this will be the purpose of Step 3 below). $\mathcal{V}$ can evaluate $g_{x,y,Z}(r)$ in quasilinear time after making a constant number of queries to the first part of the proof specifying Z. $\mathcal{V}$ can compute $h^*(r)$ with a single query to the third part of the proof. Finally, $\mathcal{V}$ can evaluate $\mathbb{Z}_H(r)$ without help in polylogarithmic time as described in Step 1 (Section 9.4.1). The verifier can then check that $g_{x,y,W}(r) = h^*(r) \cdot \mathbb{Z}_H(r)$.

In actuality, Step 3 will not be able to guarantee that π_Z and π_{h^*} are *exactly* equal to low-degree polynomials, but will be able to guarantee that, if the verifier's checks all pass, then they are each close to some low-degree polynomial Y and h' respectively. One can then argue that $g_{x,y,Y}$ vanishes on H, analogously to the proof of Theorem 8.4 in the context of the MIP from Section 8.2.

9.4.3 Step 3: A PCP for Reed-Solomon Testing

Overview. The meat of the PCP construction is in this third step, which checks that a univariate polynomial has low-degree. This task is referred to in the literature as Reed-Solomon testing, because codewords in the Reed-Solomo code consist of (evaluations of) low-degree univariate polynomials (cf. Remark 9.4).

The construction is recursive. The basic idea is to reduce the problem of checking that a *univariate* polynomial G_1 has degree at most d to the problem of checking that a related *bivariate* polynomial Q over $\mathbb{F}$ has degree at most $\sqrt{d}$ in each variable. It is known (cf. Lemma 9.5 below) how the latter problem can in turn be reduced back to a univariate problem, that is, to checking that a related univariate polynomial G_2 over $\mathbb{F}$ has degree at most $\sqrt{d}$. Recursing $\ell = O(\log \log n)$ times results

in checking that a polynomial G_ℓ has *constant* degree, which can be done with a constant number of queries to the proof. We fill in some of the details of this outline below.

The precise soundness and completeness guarantees of this step are as follows. If G_1 indeed has degree at most d, then there is a proof π that is always accepted. Meanwhile, the soundness guarantee is that there is some universal constant k satisfying the following property: if a proof π is accepted with probability $1-\varepsilon$, then there is a polynomial G of degree at most d such that G_1 agrees with G on at least a $1-\varepsilon\cdot\log^k(S)$ fraction of points in $\mathbb{F}$ (we say that G and G_1 are at most δ-*far*, for $\delta=\varepsilon\cdot\log^k(S)$.)

The claimed polylogarithmic query complexity of the PCP as a whole comes by repeating the base protocol, say, $m=\log^{2k}(S)$ times and rejecting if any run of the protocol ever rejects. If a proof π is accepted by the m-fold repetition with probability $1-\varepsilon$, then it is accepted by the base protocol with probability at least $1-\varepsilon/\log^k m$, implying that G is ε-far from a degree d polynomial G_1.

Reducing Bivariate Low-Degree Testing on Product Sets to Univariate Testing. The bivariate low-degree testing technique described here is due to Polishchuk and Spielman [212]. Assume that Q is a bivariate polynomial defined on a product set $A\times B\subseteq\mathbb{F}\times\mathbb{F}$, claimed to have degree d in each variable. (In all recursive calls of the protocol, A and B will in fact both be subspaces of $\mathbb{F}$). The goal is to reduce this claim to checking that a related univariate polynomial G_2 over $\mathbb{F}$ has degree at most d.

Definition 9.4. For a set $U\subseteq\mathbb{F}\times\mathbb{F}$, partial bivariate function $Q\colon U\to\mathbb{F}$, and nonnegative integers d_1,d_2, define $\delta^{d_1,d_2}(Q)$ to be the relative distance of Q from a polynomial of degree d_1 in its first variable and d_2 in its second variable.[7] Formally,

$$\delta^{d_1,d_2}(Q) := \min_{f(x,y)\colon U\to\mathbb{F},\deg_x(f)\le d_1,\deg_y(f)\le d_2}\delta(Q,f).$$

[7] By relative distance between Q and another polynomial P, we mean the fraction of inputs in $x\in U$ such that $Q(x)\neq P(x)$.

Let $\delta^{d_1,*}(Q)$ and $\delta^{*,d_2}(Q)$ denote the relative distances when the degree in one of the variables is unrestricted.

Lemma 9.5. (Bivariate test on a product set [212]). There exists a universal constant $c_0 \geq 1$ such that the following holds. For every $A, B \subseteq \mathbb{F}$ and integers $d_1 \leq |A|/4$, $d_2 \leq |B|/8$ and function $Q\colon A \times B \to \mathbb{F}$, it is the case that $\delta^{d_1,d_2}(Q) \leq c_0 \cdot \left(\delta^{d_1,*}(Q) + \delta^{*,d_2}(Q)\right)$.

The proof of Lemma 9.5 is not long, but we omit it from the survey for brevity.

Lemma 9.5 implies that, to test if a bivariate polynomial Q defined on a product set has degree at most d in each variable, it is sufficient to pick a variable $i \in \{1, 2\}$, then pick a random value $r \in \mathbb{F}$ and test whether the univariate polynomial $Q(r, \cdot)$ or $Q(\cdot, r)$ obtained by restricting the ith coordinate of Q to r has degree at most d.

To be precise, if the above test passes with probability $1 - \varepsilon$, then $\left(\delta^{d,*}(Q) + \delta^{*,d}(f)\right)/2 = \varepsilon$, and Lemma 9.5 implies that $\delta^{d,d}(Q) \leq 2 \cdot c_0 \cdot \varepsilon$. $Q(r, \cdot)$ and $Q(\cdot, r)$ are typically called a "random row" or "random column" of Q, respectively, and the above procedure is referred to as a "random row or column test".

Note that $\delta^{d,d}(f)$ may be larger than the acceptance probability ε by only a constant factor $c_1 = 2c_0$. Ultimately, the PCP will will recursively apply the "Reducing Bivariate Low-Degree Testing to Univariate Testing" technique $O(\log \log n)$ times, and each step may cause $\delta^{d_1,d_2}(Q)$ to blow up, relative to the rejection probability ε, by a factor of c_1. This is why the final soundness guarantee states that, if the recursive test as a whole accepts a proof with probability $1 - \varepsilon$, then the input polynomial G_1 is δ-close to a degree d polynomial, where $\delta = \varepsilon \cdot c_1^{O(\log \log S)} \leq \varepsilon \cdot \text{polylog}(S)$.[8]

Reducing Univariate Low-Degree Testing to Bivariate Testing on a Lower Degree Polynomial. Let G_1 be a univariate polynomial defined

[8] This bound on δ is non-trivial only if ε is smaller than some inverse-polylogarithm in S. That is, the analysis only yields a non-trivial soundness guarantee if the prover convinces the verifier to accept with probability at least $1 - \varepsilon$, which is inverse-polylogarithmically close to 1. Accordingly, to achieve negligible soundness error, the PCP verifier's checks must be repeated polylogarithmically many times, leading to impractical verification costs.

on a linear subspace L of $\mathbb{F}$ (in all recursive calls of the protocol, the domain of G_1 will indeed be a linear subspace L of $\mathbb{F}$). Our goal in this step is to reduce testing that G_1 has degree at most d to testing that a related bivariate polynomial Q has degree at most $\sqrt{d}$ in each variable. It is okay to assume that the number of vectors in L is at most a constant factor larger than d, as this will be the case every time this step is applied.

Lemma 9.6. [50] Given any pair of polynomials $G_1(z)$, $q(z)$, there exists a unique bivariate polynomial $Q(x, y)$ with $\deg_x(Q) < \deg(G_1)$ and $\deg_y(Q) \leq \lfloor \deg(G_1)/\deg(q) \rfloor$ such that $G_1(z) = Q(z, q(z))$.

Proof. Apply polynomial long-division to divide $G_1(z)$ by $(y - q(z))$, where throughout the long-division procedure, terms are ordered first by their degree in z and then by their degree in y.[9] This yields a representation of $G_1(z)$ as:

$$G_1(z) = Q_0(z, y) \cdot (y - q(z)) + Q(z, y). \tag{9.3}$$

By the basic properties of division in this ring, $\deg_y(Q) \leq \lfloor \deg(G_1)/\deg(q) \rfloor$, and $\deg_z(Q) < \deg(q)$. To complete the proof, set $y = q(z)$ and notice that the first summand on the right-hand side of Equation (9.3) vanishes. □

By Lemma 9.6, to establish that G_1 has degree at most d, it suffices for a PCP to establish that $G_1(z) = Q(z, q(z))$, where the degree of Q in each variable is at most $\sqrt{d}$. Thus, as a first (naive) attempt, the proof could specify Q's value on all points in $L \times \mathbb{F}$. Then $\mathcal{V}$ can check that $G_1(z) = Q(z, q(z))$, by picking a random $r \in L$ and checking that $G_1(r) = Q(r, q(r))$. If this check passes, it is safe for $\mathcal{V}$ to believe that

[9] Polynomial long division repeatedly divides the highest-degree term of the remainder polynomial by the highest-degree term of the divisor polynomial to determine a new term to add to the quotient, stopping when the remainder has lower degree than the divisor. See https://en.wikipedia.org/wiki/Polynomial_long_division for details of the univariate case. For division involving multivariate polynomials, the "term of highest degree" is not well-defined until we impose a total ordering on the degree of terms. Ordering terms by their degree in z and breaking ties by their degree in y ensures that the polynomial long division is guaranteed to output a representation satisfying the properties described immediately after Equation (9.3).

$G_1(z) = Q(z, q(z))$, as long as Q is indeed low-degree in each variable, and we have indeed reduced testing that G_1 has degree at most d to testing that Q has degree at most $\sqrt{d}$ in each variable.

The problem with the naive attempt is that the proof has length $|L| \cdot |\mathbb{F}|$, which is far too large; we need a proof of length $\tilde{O}(|L|)$. A second attempt might be to have the proof specify Q's value on all points in the set $\mathcal{T} \coloneqq \{(z, q(z)) \colon z \in L\}$. This would allow $\mathcal{V}$ to check that $G_1(z) = Q(z, q(z))$ by picking a random $r \in L$ and checking that $G_1(r) = Q(r, q(r))$. While this shortens the proof to an appropriate size, the problem is that $\mathcal{T}$ is not a product set, so Lemma 9.5 cannot be applied to check that Q has low-degree in each variable.

To get around this issue, Ben-Sasson and Sudan ingeniously choose the polynomial $q(z)$ in such a way that there is a set $\mathcal{B}$ of points, of size $O(|L|)$, at which it suffices to specify Q's values. Specifically, they choose $q(z) = \prod_{\alpha \in L_0}(z - \alpha)$, where L_0 is a linear subspace of L containing $\sqrt{d}$ vectors. Then $q(z)$ is not just a polynomial of degree $\sqrt{d}$, it is also a *linear map* on L, with kernel equal to L_0. This has the effect of ensuring that $q(z)$ takes on just $|L|/|L_0|$ distinct values, as z ranges over L.

Ben-Sasson and Sudan use this property to show that, although $\mathcal{T}$ is not a product set, it is possible to add $O(L)$ additional points $\mathcal{S}$ to $\mathcal{T}$ to ensure that $\mathcal{B} \coloneqq \mathcal{S} \cup \mathcal{T}$ contains within it a large subset that is product. So $\mathcal{P}$ need only provide Q's evaluation on the points in $\mathcal{B}$: since $\mathcal{T} \subseteq \mathcal{B}$, the verifier can check that $G_1(z) = Q(z, q(z))$ by picking a random $r \in L$ and checking that $G_1(r) = Q(r, q(r))$, and since there is a large product set within $\mathcal{S} \cup \mathcal{T}$, Lemma 9.5 can be applied.