CS294: Probabilistically Checkable and Interactive Proofs February 7, 2017

Interactive Proofs 5

Instructor: Alessandro Chiesa & Igor Shinkar *Scribe: Patrick Lutz*

1 Introduction

In this lecture, we will see how to complete the theorem stated in the previous lecture, which says that certain classes of languages have interactive proof protocols with efficient provers and extremely efficient verifiers. More formally, we will prove the following theorem.

Theorem 1 *([GKR15]) Let $\mathcal{L}$ be a language computable using $O(\log S)$-space uniform circuits of size S and depth D. Then $\mathcal{L}$ has a public coin interactive proof protocol such that:*

- *The prover runs in* $\operatorname{poly}(S)$ *time.*
- *The verifier runs in* $n \operatorname{poly}(D, \log S)$ *time and* $O(\log S)$ *space.*
- *The communication complexity is* $D \operatorname{poly}(\log S)$.

In the previous lecture, we saw how to construct this protocol when the verifier has access to oracles for low degree extensions of the wiring predicates add_i and mul_i of the circuit family that computes $\mathcal{L}$. In this lecture, we will see how to remove that restriction. That is, we will see how the verifier can compute these low degree extensions without any oracles, using only a little interaction with the prover.

2 Preliminaries

In this section we recall some useful definitions and conventions. First, whenever we write time or space bounds such as S or D, we implicitly mean functions $S\colon \mathbb{N} \to \mathbb{N}$ or $D\colon \mathbb{N} \to \mathbb{N}$. For example, when we write that a computation runs in time $n \operatorname{poly}(\log S)$, we implicitly mean $n \operatorname{poly}(\log S(n))$ (where n, as usual, is the input length).

We will also need to use some conventions about arithmetic circuits, introduced in the last lecture.

Definition 2 *Let C be an arithmetic circuit of size S and depth D. The wiring predicate add_i is a function on triples of gates in C, defined as follows:*

$$add_i(a,b,c) = \begin{cases} 1 & \text{if } a \text{ is an add gate on level } i-1 \text{ of } C \text{ and } b \le c \text{ are gates in layer } i \text{ that are the inputs of } a \\ 0 & \text{otherwise.} \end{cases}$$

The wiring predicate mul_i is defined similarly for multiplication gates.

Definition 3 *If $C = \{C_n\}$ is a family of arithmetic circuits, then C is $O(s)$-space uniform if the wiring predicates add_i and mul_i can be computed in $O(s)$ space.*

Finally, we need the idea of a low degree extension, also introduced in the last lecture.

Definition 4 *Suppose we have field extensions $\mathbb{F}_2 \subset \mathbb{H} \subset \mathbb{F}$. Then for any function $f\colon \mathbb{H}^m \to \mathbb{F}$, a low degree extension of f to $\mathbb{F}$ is an m-variable polynomial $\hat{f}$ over $\mathbb{F}$ such that the degree of $\hat{f}$ is not too large and $\hat{f}|_{\mathbb{H}^m} = f$. Often, not too large will mean that the individual degree is less than $|\mathbb{H}|$, in which case $\hat{f}$ is unique.*

In this lecture, low degree extensions will typically be applied as follows. We start with some function $f\colon \{0,1\}^N \to \{0,1\}$. Fix a bijection $\alpha\colon |\mathbb{H}|^m \to \{0,1\}^N$, where $m = \frac{N}{\log|H|}$, and then use $\hat{f}$ to denote a low degree extension of $f \circ \alpha$. Generally we will identify f and $f \circ \alpha$. In this lecture, we will apply this procedure to the wiring predicates add_i and mul_i of various arithmetic circuits.

3 Using uniformity

As mentioned in the introduction, in the last lecture we saw how to prove theorem 1 when we had access to oracles for $\hat{add}_i$ and $\hat{mul}_i$, using the "bare-bones protocol." In this section, we will see how to replace calls to those oracles with computations using a single log-space machine. More formally, we have the claim below. For the rest of these notes, s should be thought of as $\log(S)$, although the propositions that we prove are true for any s.

Claim 5 *([GKR15]) Let $C = \{C_n\}$ be a family of s-space uniform circuits. Let $\mathbb{F}_2 \subset \mathbb{H} \subset \mathbb{F}$ be field extensions such that $|\mathbb{H}| = O(s)$ and $|\mathbb{F}| = \mathrm{poly}(s)$. Let $m = O(s/\log s)$ and $m' = O(n/\log s)$. Then there is a Turing machine that computes $\hat{add}_i$ and $\hat{mul}_i$ for C in $O(s)$ space (where $\hat{add}_i$ and $\hat{mul}_i$ are the unique low degree extensions of add_i and mul_i of individual degree less than $|\mathbb{H}|$).*

Proof: We will only deal with wiring predicates that don't correspond to the input layer, but the cases are identical (up to replacing m with m'). As mentioned in the previous section, let α be a bijection from $|\mathbb{H}|^m$ to $\{0,1\}^{O(s)}$, for instance the function that computes the boolean lexicographic order on $|\mathbb{H}|^m$. It is not hard to see that α can be computed in space $O(\log|\mathbb{F}| + \log m) = O(s)$.

Since C is s-space uniform, there is a Turing machine that can output any bit of add_i or mul_i on any input, using $O(s)$ space. So $add_i \circ \alpha$ and $mul_i \circ \alpha$ can be computed in $O(s)$ space.

Now recall that, as we saw in the last lecture, for any function $f\colon \mathbb{H}^m \to \mathbb{F}$ we can compute $\hat{f}$ as follows

$$\hat{f}(\bar{z}) = \sum_{\bar{p} \in \mathbb{H}^m} EQ(\bar{z}, \bar{p}) f(\bar{p})$$

where

$$EQ(\bar{x}, \bar{y}) = \prod_{i=1}^{m} \sum_{\omega \in \mathbb{H}} \prod_{\gamma \in \mathbb{H}\colon\, \gamma \neq \omega} \frac{(x_i - \gamma)(y_i - \gamma)}{(\omega - \gamma)^2}.$$

So to compute $\hat{add}_i$ or $\hat{mul}_i$ we just need to keep track of four values in $\mathbb{F}$ (one for each sum and product in the above expression for $\hat{f}$) as well as a finite number of counters that range over $\mathbb{H}$, $[m]$ or $|\mathbb{H}|^m$. Since a counter on $|\mathbb{H}|^m$ takes $m \log|\mathbb{H}| = O(s)$ space and a value in $\mathbb{F}$ takes $O(\log s)$ space, we will not violate the memory restrictions with this data. And we only need to be able to evaluate add_i and mul_i as well as perform addition and multiplication in $\mathbb{F}$. So the entire computation can be done in $O(s)$ space. □

For more details, see claim 4.6 of [GKR15].

4 Doubly-Efficient Interactive Proofs for NL

In the previous section we showed that we can replace the oracle calls in the bare-bones protocol with computations that take $O(\log S)$ space. But we wanted a verifier with extremely low *time* complexity, and so far we only have low space complexity. In this section we will see how to replace these log-space computations with interactive protocols. In particular, we show that any language computable in log-space has a doubly efficient interactive proof protocol. We construct this protocol by using the bare-bones protocol and implementing the $\hat{add}_i$ and $\hat{mul}_i$ oracles directly. It turns out that the proof we give works just as well for nondeterministic machines, so we state that as part of the theorem.

Theorem 6 *([GKR15]) If $\mathcal{L}$ is computable by a nondeterministic machine with space s (where $s \geq \log n$) and time t then $\mathcal{L}$ has a public coin interactive proof protocol such that*

- *The prover runs in* $\text{poly}(2^s)$ *time.*
- *The verifier runs in* $n\,\text{poly}(s)$ *time and* $O(s)$ *space.*
- *The communication complexity is* $\text{poly}(s)$.

Proof: We will use the bare-bones protocol. To do this, we will show that there is a family of arithmetic circuits computing $\mathcal{L}$ for which $\hat{add}_i$ and $\hat{mul}_i$ can themselves be computed by small arithmetic circuits. As usual, let $\mathbb{F}_2 \subset \mathbb{H} \subset \mathbb{F}$ be field extensions of sizes $|\mathbb{H}| = O(s)$ and $|\mathbb{F}| = \text{poly}(s)$. We will show that:

1. $\mathcal{L}$ can be computed by a family of arithmetic circuits over $\mathbb{F}_2$ of size $\text{poly}(2^s)$ and depth $O(s \log t)$.
2. The functions $\hat{add}_i$ and $\hat{mul}_i$ for this family of circuits can be computed by arithmetic circuits of degree $|\mathbb{H}|\,\text{poly}(s)$ and size $\text{poly}(s, m)$, where m is large enough that $|\mathbb{H}^m|$ is larger than the size of the circuit computing $\mathcal{L}$ (i.e. m is large enough that we can think of the labels of gates as elements of $\mathbb{H}^m$).

Since there are many details to check, we will only give a sketch of the argument for these two claims.

Let M be a Turing machine (deterministic or nondeterministic) that computes $\mathcal{L}$ using space s and time t. For any input x, let G_x be the configuration graph of M on input x, with self-loops added to all vertices. In other words, the vertices of G_x are the configurations of M and the work tape and there is an edge from one configuration to another if M can move from one to the other when the input is x, or if the edge is a self-loop. Notice that G_x has $O(2^s)$ vertices. Let u_{accept} be the vertex in G_x corresponding to the accepting configuration and let u_{start} be the vertex corresponding to the starting configuration. Let B_x be the adjacency matrix of G_x (so B_x has size $O(2^s) \times O(2^s)$). Then M accepts on input x if and only if there is a path of length at most t in G_x from b to a. Equivalently, $x \in \mathcal{L}$ if and only if the entry (u_{start}, u_{accept}) in B_x^t is nonzero.

We now want to build a circuit that computes B_x^t for any input x. The straightforward way to do this is by first computing B_x and then using repeated squaring to find B_x^t. However, recall that we ultimately want an arithmetic circuit over $\mathbb{F}_2$. Thus when we square a matrix, we have to be careful about sums that are larger than the characteristic of the field. It is possible to make this work, but the circuit is more complicated and harder to describe. So instead we use a trick based on the fact that we are working in a field of characteristic two. Namely, we define a series of matrices

$B_x = B_{\log t}, B_{\log t-1}, \ldots, B_0$, which are similar to but not quite the same as $B_x, B_x^2, B_x^4, \ldots, B_x^t$. For each p, define

$$B_{p-1}[u,v] = 1 + \prod_{w \in G_x} (1 + B_p[u,w]B_p[w,v]).$$

Since we are doing arithmetic over a field of characteristic two, $B_0[u,v]$ is nonzero if and only if $B^t[u,v]$ is nonzero.

We can now build a circuit using $\log t$ layered subcircuits. The first subcircuit uses the input x to compute $B_{\log t} = B_x$. The i^{th} subcircuit uses $B_{\log t-i+1}$ to compute $B_{\log t-i}$. We can compute the product in the definition of $B_{\log t-i}$ using $O(s)$ layers. There are some additional details required to implement this. For instance, we need to add a gate to each layer that always outputs 0 and another that always outputs 1.

It is tedious, but straightforward, to verify that we can implement the above computations with an arithmetic circuit of the appropriate size and that the wiring predicates for this circuit can be computed by a $O(\log s)$-uniform family of boolean circuits. Now we can convert these boolean circuits for the wiring predicates into arithmetic circuits in the usual way. Then composing these circuits with a circuit that computes the bijection α (i.e. the bijection that converts $\mathbb{H}^m$ into binary strings) gives us the desired circuits for $\hat{add_i}$ and $\hat{mul_i}$. It is this final composition with circuits that compute α that causes the appearance of $|\mathbb{H}|$ and m in the degree and size bounds for the circuits that compute $\hat{add_i}$ and $\hat{mul_i}$.

Now observe that we have everything we need to efficiently implement the bare-bones protocol for $\mathcal{L}$ without recourse to oracles.

$\square$

For a more detailed proof of the above theorem, see section 4.1 of [GKR15].

5 Conclusion

Putting together the results of the previous two sections, we can conclude theorem 1. Given a language $\mathcal{L}$ computable using $O(\log S)$-space uniform circuits of size S and depth D, we obtain a doubly interactive proof protocol as follows. We use the bare-bones protocol, but each time we need to call an oracle for $\hat{mul_i}$ or $\hat{add_i}$ we instead request the value from the prover and then use the interactive proof from theorem 6 to verify that the given value is correct.

If a language $\mathcal{L}$ is computable by a Turing machine in time t and space s then it can also be computed by $O(s)$-space uniform circuits of size $\text{poly}(t \cdot 2^s)$ and depth s^2. So we have the following corollary to theorem 1.

Corollary 7 *([GKR15]) Let $\mathcal{L}$ be a language computable by a Turing machine in time t and space s. Then $\mathcal{L}$ has a public coin interactive proof protocol such that:*

- *The prover runs in* $\text{poly}(t \cdot 2^s)$ *time.*
- *The verifier runs in* $n\,\text{poly}(s)$ *time and* $\text{poly}(s)$ *space.*
- *The communication complexity is* $\text{poly}(s)$.

This improves on the result we get by just using Shamir's protocol (or Shen's), in which case the prover takes $O(2^{s \log t})$ time rather $O(2^{s+\log t})$. On the other hand, this is a little silly given that we already have theorem 6, which is a strict improvement on this corollary (i.e. it allows for the Turing machine to be nondeterministic, the prover has time complexity $O(2^s)$ and the verifier has space complexity $O(s)$). In essence, the above corollary first replaces a Turing machine with a family of circuits, then invokes the bare-bones protocol, whose implementation described above constructs a Turing machine to compute the wiring predicates and then uses the bare-bones protocol again to verify the computations of that Turing machine. But we might as well have just used the implementation of the bare-bones protocol that works for Turing machines—namely theorem 6.

Another observation is that the every time the bare-bones protocol requires an evaluation of the wiring predicates, the input value does not depend on anything the prover is done, only on the verifier's random coins. Thus if we are willing to allow the verifier to do some more expensive offline preprocessing then we can replace calls to the oracles for $\hat{add}_i$ and $\hat{mul}_i$ with cached values for $\hat{add}_i$ and $\hat{mul}_i$ that were computed during the preprocessing phase. See theorem 4.10 of [GKR15] for details.

References

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum, *Delegating computation: Interactive proofs for muggles*, Journal of the ACM **62** (2015), no. 4, 27:1–27:64.