**CS294: Probabilistically Checkable and Interactive Proofs** January 24, 2017

# Interactive Proofs 1

*Instructor: Alessandro Chiesa & Igor Shinkar* *Scribe: Mariel Supina*

## 1 PSPACE $\subseteq$ IP

The first proof that PSPACE $\subseteq$ IP is due to Shamir, and a simplified proof was given by Shen. These notes discuss the simplified version in [She92], though most of the ideas are the same as those in [Sha92]. Notes by Katz also served as a reference [Kat11].

**Theorem 1 ([Sha92])** PSPACE $\subseteq$ IP.

To show the inclusion of PSPACE in IP, we need to begin with a PSPACE-complete language.

### 1.1 True Quantified Boolean Formulas (TQBF)

**Definition 2** *A* quantified boolean formula (QBF) *is an expression of the form*

$$\forall x_1 \exists x_2 \forall x_3 \dots \exists x_n \phi(x_1, \dots, x_n), \tag{1}$$

*where $\phi$ is a boolean formula on $n$ variables.*

Note that since each variable in a QBF is quantified, a QBF is either true or false.

**Definition 3** TQBF *is the language of all boolean formulas $\phi$ such that if $\phi$ is a formula on $n$ variables, then the corresponding QBF* (1) *is true.*

**Fact 4** TQBF *is* PSPACE*-complete (see section 2 for a proof).*

Hence to show that PSPACE $\subseteq$ IP, it suffices to show that TQBF $\in$ IP.

**Claim 5** TQBF $\in$ IP.

In order prove claim 5, we will need to present a complete and sound interactive protocol that decides whether a given QBF is true. In the sum-check protocol we used an arithmetization of a 3-CNF boolean formula. Likewise, here we will need a way to arithmetize a QBF.

### 1.2 Arithmetization of a QBF

We begin with a boolean formula $\phi$, and we let $n$ be the number of variables and $m$ the number of clauses of $\phi$. We may assume that $\phi$ is in 3-CNF, since if not, we can find an equivalent 3-CNF boolean formula $\phi'$ and reassign values to $n$ and $m$ accordingly.

### 1.2.1 Arithmetization of a 3-CNF Boolean Formula

To arithmetize $\phi$, we use the arithmetization technique wherein we map $\phi(x_1, \ldots, x_n)$ to a polynomial $p(x_1, \ldots, x_n)$ such that for any assignment of boolean values $v_i \in \{0, 1\}$ to the $x_i$,

$$\phi(v_1, \ldots, v_n) = p(v_1, \ldots, v_n).$$

We define the arithmetization map $A$ as follows for variables $x_i$, single-variable expressions $x$, $y$, and $z$, and clauses $c_1, \ldots, c_m$:

$$\begin{array}{rcl} x_i & \mapsto & x_i \\ \neg x_i & \mapsto & 1 - x_i \\ x \vee y \vee z & \mapsto & 1 - (1 - A(x))(1 - A(y))(1 - A(z)) \\ c_1 \wedge \cdots \wedge c_m & \mapsto & \prod_{i=1}^{m} A(c_i) \end{array}$$

Applying the map $A$ to $\phi$, we obtain an arithmetization $p(x_1, \ldots, x_n)$ with the desired property.

### 1.2.2 Arithmetization of $\forall$ and $\exists$

It remains to find an arithmetization of the quantifiers $\forall$ and $\exists$. For the purposes of arithmetization, $\forall$ is similar $\wedge$ in that the statement $\forall x\, \psi(x)$ is true if and only if the statement $\psi(0) \wedge \psi(1)$ is true. Hence if $\psi$ is the part of the QBF following $\forall x_i$ and $A(\psi) = p$, then our arithmetization map $A$ should be

$$\begin{aligned} \forall x_i\, \psi(x_1, \ldots, x_n) \mapsto & \prod_{x_i} p(x_1, \ldots, x_n) \\ & = p(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) p(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n). \end{aligned}$$

Likewise, $\exists$ is similar to $\vee$ since $\exists x\, \psi(x) \iff \psi(0) \vee \psi(1)$. This gives the arithmetization

$$\begin{aligned} \exists x_i\, \psi(x_1, \ldots, x_n) \mapsto & \coprod_{x_i} p(x_1, \ldots, x_n) \\ & = 1 - (1 - p(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n))(1 - p(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)). \end{aligned}$$

It would simplify the arithmetic to take all computations over a finite field $\mathbb{F}_q$ for some prime number $q$. Since our arithmetization can only evaluate to values in $\{0, 1\}$, any prime number will suffice without affecting our computations.

Thus, if we are given the QBF

$$\forall x_1 \exists x_2 \forall x_3 \ldots \exists x_n \phi(x_1, \ldots, x_n), \tag{2}$$

our arithmetization maps it to

$$\prod_{x_1} \coprod_{x_2} \cdots \coprod_{x_n} p(x_1, \ldots, x_n) \mod q. \tag{3}$$

### 1.2.3 Degree Reduction

Our goal is to develop an interactive protocol that determines whether a given QBF is true. Now that we have a way to arithmetize a QBF, a natural idea is to adapt the sum-check protocol for this problem, i.e. to strip off the operators $\prod$ and $\coprod$ one at a time, from the outside in. Proceeding in this way, we will quickly encounter a problem.

Recall that in each round of the sum-check protocol, the prover sends the polynomial $p_i$ to the verifier, where $p_i(x_1, \ldots, x_n) = \sum_{a_{i+1},\ldots,a_n} p(r_1, \ldots, r_{i-1}, x, a_{i+1}, \ldots, a_n)$. This polynomial has degree at most $m$ if we arithmetize $\vee$ to addition, or $3m$ if we use the multiplicative arithmetization of $\vee$ described in section 1.2.1. However, if in the protocol for TQBF we likewise define $p_i$ to be our arithmetization with the first $i$ operators stripped off and with $x_j$ evaluated at a random $r_j$ for $j < i$, we get

$$p_i(x) = \prod_{x_{i+1}} \cdots \coprod_{x_n} p(r_1, \ldots, r_{i-1}, x, x_{i+1}, \ldots, x_n). \tag{4}$$

This polynomial could have degree up to $2^{n-i}3m$, since each $\prod$ or $\coprod$ doubles the degree. The exponential degree of this polynomial is too large for the verifier to handle.

The solution to this problem is due to Shen [She92]. Observe that we plan to evaluate all polynomials in this protocol only on the boolean hypercube, that is, $x_i \in \{0, 1\}$ for all $i$. Since for any $k > 0$ and $a \in \{0, 1\}$ we have $a^k = a$, we can reduce all exponents greater than 1 appearing in these polynomials to 1. For example, we could reduce the polynomial $x_1^3 x_3^2 + x_2^5 x_4^4$ to $x_1 x_3 + x_2 x_4$ for the purposes of this protocol, since these two polynomials agree on the boolean hypercube.

Define the degree reduction operator $R_{x_i}$, which reduces all positive exponents of $x_i$ appearing in an expression to 1. Then in order to ensure that the degree of the arithmetization remains small, we need to perform degree reduction on all variables prior to each $\prod$ or $\coprod$. This gives the final improvement to our arithmetization, so the QBF $\forall x_1 \exists x_2 \forall x_3 \ldots \exists x_n \phi(x_1, \ldots, x_n)$ is arithmetized to

$$\prod_{x_1} R_{x_1} \coprod_{x_2} R_{x_1} R_{x_2} \prod_{x_3} \cdots \prod_{x_{n-1}} R_{x_1} \ldots R_{x_{n-1}} \coprod_{x_n} R_{x_1} \ldots R_{x_n} p(x_1, \ldots, x_n) \mod q. \tag{5}$$

## 1.3 Interactive Protocol for TQBF

Let $l = \frac{1}{2}(n^2 + 3n)$ and write the arithmetization of a given QBF as

$$\mathcal{O}_1 \ldots \mathcal{O}_l \, p(x_1, \ldots, x_n) \mod q, \qquad \mathcal{O}_k \in \Big\{ \prod_{x_i}, \coprod_{x_i}, R_{x_i} : 1 \le i \le n \Big\}. \tag{6}$$

The overall goal of the interactive protocol is for the prover to convinve the verifier that

$$\mathcal{O}_1 \ldots \mathcal{O}_l \, p(x_1, \ldots, x_n) \mod q = 1. \tag{7}$$

In round $k$ of the protocol, the verifier stores some value $v_k$. The prover then sends the verifier a proof that $v_k = \mathcal{O}_{k+1} \ldots \mathcal{O}_l \, p_k \mod q$ for some polynomial $p_k$. The verifier calls a sub-protocol to check this and then computes and stores a value $v_{k+1}$ to send to the prover. Next, the prover sends the verifier a proof that $v_{k+1} = \mathcal{O}_{k+1} \ldots \mathcal{O}_l \, p_{k+1} \mod q$ for some polynomial $p_{k+1}$, and the process repeats until $l$ rounds have been completed. Initially, $p_0 = p$ and the verifier stores the value $v_0 = 1$, so that the claim to be checked is the claim in (7).

The sub-protocol that the verifier calls in round $k$ must check the type of the operator $\mathcal{O}_k$ ($\prod$, $\coprod$, or $R$) and act accordingly.

### 1.3.1 Case 1: $\mathcal{O}_{k+1} = \prod_{x_i}$ for some $i$

In this case, the verifier has already stored the value $v_k$, which the prover claims is equal to

$$\prod_{x_i} R_{x_1} \dots R_{x_i} \coprod_{x_{i+1}} \dots \coprod_{x_n} R_{x_1} \dots R_{x_n}\, p(r_1, \dots, r_{i-1}, x_i, \dots, x_n) \tag{8}$$

for some random $r_1, \dots, r_{i-1} \in \mathbb{F}_q$ previously generated by the verifier. The claim that $v_k$ is equal to the value in (8) is the *input claim*. The prover sends the verifier some polynomial $\tilde{p}_k(x_i)$. If the proof is correct, then $\tilde{p}_k(x_i)$ should be the expression in (8) with the operator $\prod_{x_i}$ removed. Hence, to check the prover's claim, the verifier must check whether $v_k = \prod_{x_i} \tilde{p}_k(x_i) = \tilde{p}_k(0)\tilde{p}_k(1)$. If $\tilde{p}_k$ fails this check, the verifier rejects the proof and the protocol halts. Otherwise, the verifier randomly samples $r_i$ from $\mathbb{F}_q$, stores the value $v_{k+1} = \tilde{p}_k(r_i)$, and sends $r_i$ to the prover. This ends round $k$ of the sub-protocol. In the next round, the prover tries to convince the verifier that

$$v_{k+1} = R_{x_1} \dots R_{x_i} \coprod_{x_{i+1}} \dots \coprod_{x_n} R_{x_1} \dots R_{x_n}\, p(r_1, \dots, r_i, x_{i+1}, \dots, x_n), \tag{9}$$

which is called the *output claim*.

To determine completeness of this sub-protocol, consider the prover that always sends the polynomial

$$p_k(x_i) = R_{x_1} \dots R_{x_i} \coprod_{x_{i+1}} \dots \coprod_{x_n} R_{x_1} \dots R_{x_n}\, p(r_1, \dots, r_{i-1}, x_i, \dots, x_n). \tag{10}$$

Then if the input claim is true, we will always have $p_k(0)p_k(1) = v_k$, so the verifier will continue with the protocol. After sampling $r_i$, the verifier will store $v_{k+1} = p_k(r_i)$, and it follows from (10) that the output claim in (9) will be true with probability 1.

For soundness, suppose the input claim is false. Then any prover that sends the polynomial in (10) will always fail to convince the verifier, since if $v_k$ is not equal to the value in (8), then it follows immediately that $v_k \neq \prod_{x_i} p_k(x_i)$. Now consider any prover that sends a polynomial $\tilde{p}_k(x_i) \neq p_k(x_i)$. From (9) and (10), we see that we can write the output claim as $v_{k+1} = p_k(r_i)$. Since $\tilde{p}_k(x_i)$ and $p_k(x_i)$ are both polynomials of degree 1 and $\tilde{p}_k \neq p_k$, they may agree at at most one point in $\mathbb{F}_q$. The probability that $r_i$ is this point is $\frac{1}{q}$. Hence the output claim will be true with probability $\frac{1}{q}$ and false with probability $1 - \frac{1}{q}$. Since $\frac{1}{q}$ is small, the sub-protocol is sound.

### 1.3.2 Case 2: $\mathcal{O}_{k+1} = \coprod_{x_i}$ for some $i$

This case is almost identical to Case 1 (see section 1.3.1). The only difference is that the verifier must check whether $v_k = \coprod_{x_i} \tilde{p}_k(x_i) = 1 - (1 - \tilde{p}_k(0))(1 - \tilde{p}_k(1))$. The analyses of completeness and soundness for this sub-protocol are the same as for $\prod$.

### 1.3.3 Case 3: $\mathcal{O}_{k+1} = R_{x_i}$ for some $i$

In this case, the verifier has the value $v_k$ stored. The input claim is that

$$v_k = R_{x_i} \mathcal{O}_{k+2} \dots \coprod_{x_n} R_{x_1} \dots R_{x_n}\, p(r_1, \dots r_j, x_{j+1}, \dots, x_n), \qquad j \geq i. \tag{11}$$

Next, the prover sends the verifier some polynomial $\tilde{p}_k(x_i)$. The verifier should check that $(R_{x_i}\tilde{p}_k(x_i))(r_i) = v_k$, that is, that $v_k$ is equal to the value obtained after applying degree reduction to $\tilde{p}_k(x_i)$ and then

evaluating at $x_i = r_i$. The verifier then randomly samples $r_i^{new}$ from $\mathbb{F}_q$, stores $v_{k+1} = \tilde{p}_k(r_i^{new})$, and sends $r_i^{new}$ to the prover. The output claim is that

$$v_{k+1} = \mathcal{O}_{k+2} \dots \coprod_{x_n} R_{x_1} \dots R_{x_n} \, p(r_1, \dots, r_{i-1}, r_i^{new}, r_{i+1}, \dots, r_j, x_{j+1}, \dots, x_n). \tag{12}$$

To argue completeness of this sub-protocol, consider the prover that always sends

$$p_k(x_i) = \mathcal{O}_{k+2} \dots \coprod_{x_n} R_{x_1} \dots R_{x_n} \, p(r_1, \dots, r_{i-1}, x_i, r_{i+1}, \dots, r_j, x_{j+1}, \dots, x_n). \tag{13}$$

If the input claim in (11) is true, then it follows that $(R_{x_i} p_k(x_i))(r_i) = v_k$, so the output claim in (12) will be true with probability 1.

For soundness, suppose (11) is false. Then any prover that sends the polynomial in (13) will fail to convince the verifier, so assume that the prover sends some polynomial $\tilde{p}_k(x_i) \neq p_k(x_i)$. If the reduction operator $R_{x_i}$ that we removed in this round is among the innermost $n$ reduction operators, then the polynomial prior to reduction has degree at most $3m$. Otherwise, the polynomial has degree 2. Since two polynomials of degree $d$ over $\mathbb{F}_q$ may agree at at most $d$ points in $\mathbb{F}_q$, the output claim will be true with probability either $\frac{2}{q}$ or $\frac{3m}{q}$ (depending on the location of $R_{x_i}$ in the expression. Both of these probabilities are small, so the sub-protocol is sound.

### 1.3.4 Total Soundness Error

Together, the completeness analysis for the three cases describes a prover that convinces the verifier with probability 1, so it only remains to calculate the soundness error of the entire protocol. There are $n$ total $\prod$ and $\coprod$ operators, and the sub-protocol for each of these has soundness error $\frac{1}{q}$ (see sections 1.3.1 and 1.3.2), giving soundness error $\frac{n}{q}$ for the $\prod$ and $\coprod$ parts of the protocol. The sub-protocol for the innermost $n$ $R_{x_i}$ operators has soundness error $\frac{3m}{q}$ for each operator (section 1.3.3), for a total of $\frac{3mn}{q}$. The sub-protocol for the remaining reduction operators has soundness error $\frac{2}{q}$ for each operator (section 1.3.3), giving a total of $\frac{2}{q}\sum_{i=1}^{n-1} i$. Hence, the entire protocol has soundness error

$$\frac{n}{q} + \frac{3mn}{q} + \frac{2}{q}\sum_{i=1}^{n-1} i = \frac{3mn + n^2}{q}. \tag{14}$$

By choosing $q$ to be sufficiently large, we can make this error value as small as we desire. Hence the overall protocol is sound.

We have demonstrated a complete and sound interactive protocol for TQBF. Thus TQBF $\in$ IP, and since TQBF is PSPACE-complete, we have proved theorem 1 and we may conclude that IP = PSPACE.

## 2 TQBF is PSPACE-complete

Our proof that IP = PSPACE relies on fact 4, which says that TQBF is PSPACE-complete. It remains to prove this. To show that a language is PSPACE-complete, we must show that the language is in PSPACE and that the language is PSPACE-hard. Feigenbaum [Fei12] provides very clear notes on this subject, which served as a reference for sections 2.1 and 2.2.

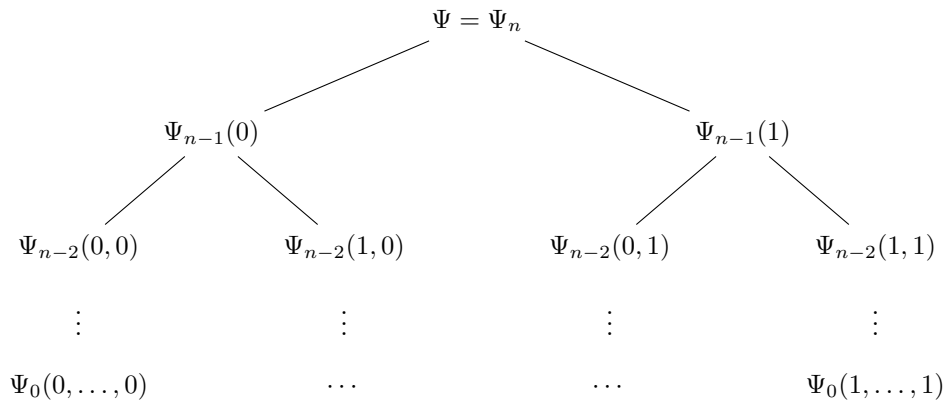


Figure 1: Recursively evaluating a QBF $\Psi$

## 2.1 TQBF $\in$ PSPACE

Suppose we are given a QBF $\Psi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \, \phi(x_1, \ldots, x_n)$ for some quantifiers $Q_i \in \{\forall, \exists\}$ and some boolean formula $\phi$ on $n$ variables with $m$ clauses. For each $i$, define

$$\Psi_i(a_1, \ldots, a_{n-i}) = Q_{n-i+1} x_{n-i+1} \ldots Q_n x_n \, \phi(a_1, \ldots, a_{n-1}, x_{n-i+1}, \ldots, x_n) \tag{15}$$

and let $S(\phi, i)$ be the amount of space necessary to evaluate $\Psi_i$ at the point $(a_1, \ldots, a_{n-i}) \in \{0,1\}^{n-i}$. Since $\Psi = \Psi_n$, the total space complexity is $S(\phi, n)$.

We can recursively evaluate $\Psi$ as shown in Figure 1, beginning with $\Psi_n$. At level $i$ of the tree, we strip off $Q_i$ from $\Psi_{n-i+1}$ and evaluate the resulting boolean statement at $x_i = 0$ and $x_i = 1$. If $Q_i = \forall$, we accept only if both values are 1; if $Q_i = \exists$, we accept if at least one value is 1. The key is that after we evaluate the statement at $x_i = 0$, we can reuse the same space to evaluate the statement at $x_i = 1$.

In the base case, we need to evaluate $\Psi_0(a_1, \ldots, a_n) = \phi(a_1, \ldots, a_n)$. Since $\phi$ has $n$ variables and $m$ clauses, the total space $S(\phi, 0)$ required to evaluate $\phi$ is $O(mn)$. At step $i$ we have

$$S(\phi, i) = S(\phi, i-1) + O(mn), \tag{16}$$

since $S(\phi, i-1)$ is the amount of space needed to evaluate the statement at 0 and 1, and $O(mn)$ is the amount of space needed to store the unevaluated statement. Evaluating the recursion in (16), we can conclude that $S(\phi, n) = O(mn^2)$. Thus this algorithm evaluates $\Psi$ in polynomial space, and TQBF $\in$ PSPACE.

## 2.2 TQBF is PSPACE-hard

Let $L \in$ PSPACE, so there exists some polynomial $S$ and $S(n)$-space machine $M$ that decides whether an input $x \in L$ of length $n$ is in PSPACE. Let $G$ be the configuration graph of $M(x)$. Then $G$ has roughly $2^{S(n)}$ vertices representing states of $M$, with a directed edge between two vertices if there exists a transition from one state to the other. Let $C_{start}$ and $C_{accept}$ be vertices corresponding to the starting and accepting states of $M$, respectively. Then $x \in L$ if and only if there exists a path in $G$ from $C_{start}$ to $C_{accept}$.

We will construct a collection of QBFs $\Phi_i$ such that $\Phi_i(C, C') \iff$ there exists a path in $G$ from $C$ to $C'$ of length at most $2^i$. For the base case, $\Phi_0(C, C')$ for distinct vertices $C$ and $C'$ means that $C$ and $C'$ are connected by an edge in $G$, that is, applying the transition function of $M$ to $C$ gives $C'$. We know by the Cook-Levin Theorem that such a formula exists and that it can be constructed in poly$(n)$ time. In general, if we have a path from $C$ to $C'$ of length at most $2^i$, we can choose a vertex $C''$ approximately halfway between $C$ and $C'$ so that there exist paths of length $2^{i-1}$ from $C$ to $C''$ and from $C''$ to $C'$. Hence,

$$\Phi_i(C, C') \iff \exists C'' \, \Phi_{i-1}(C, C'') \wedge \Phi_{i-1}(C'', C'). \tag{17}$$

Unfortunately, with this recursive formulation, we get that $|\Phi_i| \geq 2|\Phi_{i-1}|$, which would cause $|\Phi_i|$ to grow exponentially. To avoid this, we can reformulate the statement in (17) as follows:

$$\Phi(C, C') \iff \Big(\exists C'' \forall D_1, D_2 \Big((D_1 = C \wedge D_2 = C'') \vee (D_1 = C'' \wedge D_2 = C')\Big) \Rightarrow \Phi_{i-1}(D_1, D_2)\Big) \tag{18}$$

With this formulation, we have that $|\Phi_i| = |\Phi_{i-1}| + poly(S(n))$, which is polynomial in $n$. Thus, we have reduced the problem of deciding $L$ to TQBF, which shows that TQBF is PSPACE-hard and is therefore PSPACE-complete.

# References


[Fei12] Joan Feigenbaum, *TQBF is PSPACE-complete*, 2012, Available at `http://zoo.cs.yale.edu/classes/cs468/fall12/TQBF-complete.pdf`.

[Kat11] Jonathan Katz, *Notes on complexity theory: Lecture 19*, 2011, Available at `http://www.cs.umd.edu/~jkatz/complexity/f11/lecture19.pdf`.

[Sha92] Adi Shamir, *IP = PSPACE*, Journal of the ACM **39** (1992), no. 4, 869–877.

[She92] Alexander Shen, *IP = PSPACE: Simplified proof*, Journal of the ACM **39** (1992), no. 4, 878–880.