**CS294: Probabilistically Checkable and Interactive Proofs** January 19, 2017

# Introduction to Interactive Proofs & The Sumcheck Protocol

*Instructor: Alessandro Chiesa & Igor Shinkar* *Scribe: Pratyush Mishra*

## 1 Introduction

Traditional mathematical proofs are static objects: a prover $\mathcal{P}$ writes down a sequence of mathematical statements, and then at some later time a verifier $\mathcal{V}$ checks that these statements are consistent and correct. Over the years, computer science has changed the notion of a mathematical proof. The first such change was the observation that for all practical purposes, the verification procedure should be efficient; $\mathcal{V}$ should not have to expend large amounts of effort to verify the proof of a claim (at least much less than $\mathcal{P}$ expended to find the proof). This notion of "efficient verification" corresponds to the complexity class $\mathsf{NP}$:

**Definition 1** *A language $\mathcal{L}$ belongs to the class $\mathsf{NP}$ if and only if there exists an efficient algorithm $\mathcal{V}$ such that the following conditions hold.*

COMPLETENESS:

*For all $\mathbb{x} \in \mathcal{L}$, there exists a proof $\pi$ that makes $\mathcal{V}$ accept: $\mathcal{V}(\mathbb{x}, \pi) = 1$.*

SOUNDNESS:

*For all $\mathbb{x} \notin \mathcal{L}$, for all claimed proofs $\pi^*$, $\mathcal{V}$ rejects: $\mathcal{V}(\mathbb{x}, \pi^*) = 0$.*

Even though the verification procedure is now efficient, the proof is still a static object. Computer scientists in the 80's and 90's changed this view by introducing *interaction* and *randomness* into the mix: the prover and verifier were no longer required to be deterministic, and could now talk to each other. How did this change things? As we shall see, introducing interaction allows a computationally bounded (but randomized) verifier to check extraordinary claims efficiently. For instance, Lund, Fortnow, Karloff and Lund (LFKN) showed that an unbounded prover interacting with an efficient verifier can convince the verifier that a boolean formula is unsatisfiable [LFKN92]. Assuming that $\mathsf{CoNP} = \mathsf{NP}$, an efficient verifier cannot verify such a claim in the static proof model described above.

Now we formalize the "interactive proof" model, and then describe (and prove correct) the LFKN protocol, thus showing that all of $\mathsf{CoNP}$ has an interactive proof.

**Definition 2** *A language $\mathcal{L}$ has an interactive proof (and belongs to the class $\mathsf{IP}$) if there exists an efficient, randomized, interactive algorithm $\mathcal{V}$ that satisfies the following conditions:*

COMPLETENESS:

*For all $\mathbb{x} \in \mathcal{L}$, there exists an unbounded interactive 'prover' algorithm $\mathcal{P}$ such that $\mathcal{V}$ interacts with $\mathcal{P}$ and accepts with high probability:*

$$\Pr\left[\langle \mathcal{P}, \mathcal{V} \rangle(\mathbb{x}) = 1\right] \geq \frac{2}{3}$$

SOUNDNESS:
*For all $\mathbb{x} \notin \mathcal{L}$, for all algorithms $\mathcal{P}^*$, $\mathcal{V}$ interacts with $\mathcal{P}^*$ and rejects with high probability:*

$$\Pr\left[\langle \mathcal{P}^*, \mathcal{V}\rangle(\mathbb{x}) = 1\right] \leq \frac{1}{3}$$

*Here $\langle \cdot, \cdot \rangle$ denotes interaction.*

## 2 The Sumcheck protocol

Which languages have interactive proofs? The discussion above indicates that $\mathsf{CoNP} \subseteq \mathsf{IP}$, but do we have a tight characterization of the power of interactive proof systems? Yes. In fact, we'll see in the next lecture that interactive proof systems are extraordinarily powerful: Shamir, building on the LFKN protocol, proved that every language in $\mathsf{PSPACE}$ has an interactive proof system [Sha92]!

For now, we will describe the core ingredient of the LFKN protocol, called the *Sumcheck Protocol*.

**Theorem 3 ([LFKN92])**

$$\mathsf{CoNP} \subseteq \mathsf{IP}.$$

**Proof:** We proceed by demonstrating an interactive proof system for the $\mathsf{CoNP}$-complete language $\mathsf{UNSAT}$, which is the language of all unsatisfiable 3CNF boolean formulae.

Assume the prover $\mathcal{P}$ and verifier $\mathcal{V}$ are given a 3CNF boolean formula $\varphi$ having $n$ variables and $m$ clauses. $\mathcal{P}$'s task is to convince $\mathcal{V}$ that $\varphi$ is unsatisfiable. The first step to doing this is to *arithmetize* the formula.

**Arithmetization.** Arithmetization is the technique of 'converting' a boolean formula $\varphi$ (over $\{0,1\}$) to an equivalent polynomial $p$ (over some field $\mathbb{F}$) such that the polynomial has certain properties if and only if the formula is satisfiable. In particular, we want $p$ to be $0$ if and only if $\varphi$ is unsatisfiable. One such transformation is as follows:

- $x \vee y \mapsto x + y$: Convert every disjunction to addition.
- $x \wedge y \mapsto x \times y$: Convert every conjunction to multiplication.
- $\neg x \mapsto (1 - x)$: Convert every negation as indicated.

Thus under this mapping, the clause $(x \vee y \vee \neg z)$ gets transformed to the polynomial $(x + y + (1 - z))$. Notice now that if a formula is satisfied by an assignment, then the corresponding polynomial (evaluated at the same points) takes on only positive values. If it is not satisfied, the polynomial (on the same assignment) evaluates to zero. For example, plugging in $x = 1$, $y = 1$, $z = 1$ into the previous clause and its polynomial, we get values 1 and 2 respectively. However, if we plug in $x = 0$, $y = 0$, $z = 1$, the clause and polynomial both evaluate to $0$.

Let us fix a formula $\varphi$ having $n$ variables and $m$ clauses. Let the arithmetization of this formula result in a polynomial $p$ in $n$ variables. The formula is unsatisfiable if and only if $p$ takes on the value $0$ at all possible evaluations over the Boolean hypercube $\{0,1\}^n$. This means that the summation of all such evaluations should also be $0$, and this gives us a new way to attack the problem: we can reduce the problem of interactively proving that $\varphi$ is unsatisfiable to the problem of interactively proving that evaluations of $p$ over the Boolean hypercube sum up to zero, i.e. that

$$\sum_{x_1,\ldots,x_n\in\{0,1\}} p(x_1,\ldots,x_n)=0 \quad .$$

Notice that the maximum value of the arithmetization of each clause is 3, and thus the maximum value of the above summation is at most $2^n 3^m$. Hence we do not need to work over $\mathbb{Z}$, but can instead work over the field $\mathbb{Z}_q$, where $q$ is a prime number greater than $2^n 3^m$. The above condition is thus equivalent to:

$$\varphi \in \mathsf{UNSAT} \iff \sum_{x_1,\ldots,x_n\in\{0,1\}} p(x_1,\ldots,x_n)=0 \pmod q \quad .$$

The benefit of this is that we do not have to worry about intermediate computations resulting in large numbers (results will wrap around).

Finally, notice that **the degree of $p$ is at most $m$**. This property will come in handy later. We are now ready to present an interactive protocol for the following problem:

**Definition 4** *The **Sumcheck problem** is the problem of proving that evaluations of the arithmetization of a boolean formula over the Boolean hypercube sum up to a value $s$. All arithmetic is performed in a finite field $\mathbb{Z}_q$ that is large enough to represent the result of the summation.*

The **Sumcheck protocol** (see Figure 1) solves the **Sumcheck problem**. Informally, the protocol proceeds as follows:

1. The prover $\mathcal{P}$ and verifier $\mathcal{V}$ are given as input a boolean formula $\varphi$ and a field element $s$. Both arithmetize $\varphi$ to obtain a polynomial $p$ in $n$ variables $x_1,\ldots,x_n$.
2. $\mathcal{V}$ generates a random prime $q$ that is greater than $2^n 3^m$ and sends it to $\mathcal{P}$.
3. $\mathcal{V}$ initializes the 0th *check-value* $v_0 := s$.
4. The following interaction is repeated for all $i=1$ to $n$:
   (a) Leaving $x_i$ free, $\mathcal{P}$ evaluates $p$ at $x_{i+1}\in\{0,1\},\ldots,x_n\in\{0,1\}$ to obtain polynomial $p_i$ in $x_i$:
   $$p_i(x_i) := \sum_{x_{i+1},\ldots,x_n\in\{0,1\}} p(r_1,r_2,\ldots,x_i,\ldots,x_n) \quad .$$
   $\mathcal{P}$ sends $p_i$ over to $\mathcal{V}$.
   (b) $\mathcal{V}$ checks that $p_i(0)+p_i(1)=v_{i-1}$. If so, it samples a random field element $r_i$, computes the next check-value $v_i := p_i(r_i)$, and sends $r_i$ to $\mathcal{P}$.
5. In the final round, instead of sending $r_n$ over to $\mathcal{P}$, $\mathcal{V}$ checks that $p(r_1,\ldots,r_n)=v_n$.

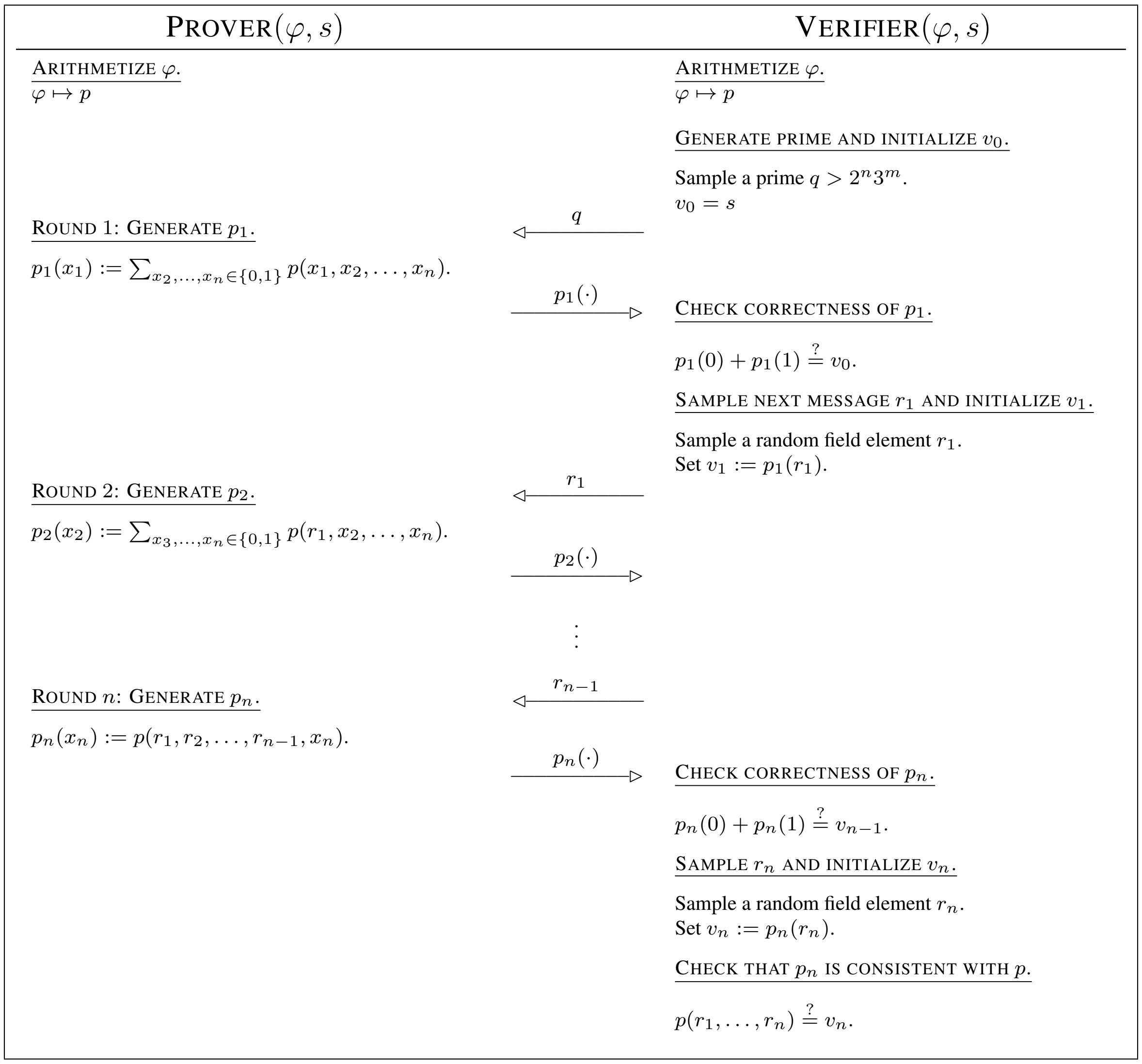
**Theorem 5** *The sumcheck protocol solves the sumcheck problem with completeness* $1$ *and soundness* $nm/q$.

**Proof:** We analyze the completeness and soundness of the sumcheck protocol.

- COMPLETENESS:

  If polynomial $p$ indeed does some up to $s$ over the Boolean hypercube, then at each step $\mathcal{P}$ can evaluate the polynomial correctly, and thus every test of the verifier will pass.

  For each $i=1$ to $n$, let $p_i(x_i) := \sum_{x_{i+1},\ldots,x_n\in\{0,1\}} p(r_1,r_2,\ldots,x_i,\ldots,x_n)$. Then we show that if in the ith round, $\mathcal{P}$ sends $p_i$, $\mathcal{V}$ will always accept.



PROVER$(\varphi, s)$

VERIFIER$(\varphi, s)$

Prover: ARITHMETIZE $\varphi$.

$\varphi \mapsto p$

Verifier: ARITHMETIZE $\varphi$.

$\varphi \mapsto p$

GENERATE PRIME AND INITIALIZE $v_0$.

Sample a prime $q > 2^n 3^m$.
$v_0 = s$

$\xleftarrow{q}$

ROUND 1: GENERATE $p_1$.

$p_1(x_1) := \sum_{x_2,\ldots,x_n \in \{0,1\}} p(x_1, x_2, \ldots, x_n)$.

$\xrightarrow{p_1(\cdot)}$

CHECK CORRECTNESS OF $p_1$.

$p_1(0) + p_1(1) \overset{?}{=} v_0$.

SAMPLE NEXT MESSAGE $r_1$ AND INITIALIZE $v_1$.

Sample a random field element $r_1$.
Set $v_1 := p_1(r_1)$.

$\xleftarrow{r_1}$

ROUND 2: GENERATE $p_2$.

$p_2(x_2) := \sum_{x_3,\ldots,x_n \in \{0,1\}} p(r_1, x_2, \ldots, x_n)$.

$\xrightarrow{p_2(\cdot)}$

$\vdots$

$\xleftarrow{r_{n-1}}$

ROUND $n$: GENERATE $p_n$.

$p_n(x_n) := p(r_1, r_2, \ldots, r_{n-1}, x_n)$.

$\xrightarrow{p_n(\cdot)}$

CHECK CORRECTNESS OF $p_n$.

$p_n(0) + p_n(1) \overset{?}{=} v_{n-1}$.

SAMPLE $r_n$ AND INITIALIZE $v_n$.

Sample a random field element $r_n$.
Set $v_n := p_n(r_n)$.

CHECK THAT $p_n$ IS CONSISTENT WITH $p$.

$p(r_1, \ldots, r_n) \overset{?}{=} v_n$.

**Figure 1:** The Sumcheck protocol.

**Base case.** When $i = 1$, if the prover sends $p_1$, then

$$p_1(0) + p_1(1) = \sum_{x_1 \in \{0,1\}} ( \sum_{x_2,\ldots,x_n \in \{0,1\}} p(x_1, x_2, \ldots, x_n)) = v_0 = s \quad .$$

**Inductive step.** Assume that for all $1 \leq i \leq k$, the prover's $i^{th}$ message was the polynomial $p_i$. Then, we have that

$$p_k(0) + p_k(1) = \sum_{x_i \in \{0,1\}} ( \sum_{x_{i+1},\ldots,x_n \in \{0,1\}} p(r_1, r_2, \ldots, r_{i-1}, x_i, \ldots, x_n)) = p_{i-1}(r_{i-1}) = v_{i-1} \quad .$$

For $k = n$, we additionally have that $p_n(r_n) = p(r_1, \ldots, r_n) = v_n$ (by assumption), and so all checks pass, and the verifier accepts.

- SOUNDNESS.

  Let $E_i$ be the event that the malicious prover $\mathcal{P}^*$ sends a polynomial $\tilde{p}_i = p_i$ in the $i^{th}$ round. Let $W$ be the event that $\mathcal{P}^*$ wins. Notice that $\Pr[W \mid E_1 \wedge \cdots \wedge E_n] = 0$. This is because if $\mathcal{P}$ sends the correct polynomial in the last step, then $p_n(0) + \tilde{p}_n(1) \neq v_{n-1}$.

  **Claim 6**

  $$\begin{aligned} \Pr[W] &\leq \frac{nm}{q} + \Pr[W \mid E_1 \wedge \cdots \wedge E_n] \\ &\leq \frac{nm}{q} \end{aligned}$$

  **Proof:** We proceed by induction, proving the following hypothesis:

  $$\Pr[W] \leq \frac{(n-j+1)m}{q} + \Pr[W \mid E_j \wedge \ldots \wedge E_n]$$

  **Base case:** $j = n$.

  $$\begin{aligned} \Pr[W] &\leq \Pr\left[W \mid \overline{E_n}\right] + \Pr[W \mid E_n] \\ &\leq \frac{m}{q} + \Pr[W \mid E_n] \qquad \text{(Schwartz-Zippel Lemma)} \end{aligned}$$

  **Inductive step:**

  $$\begin{aligned} \Pr[W] &\leq \frac{(n-j+1)m}{q} + \Pr[W \mid E_j \wedge \ldots \wedge E_n] \\ \Pr[W] &\leq \frac{(n-j+1)m}{q} + \Pr\left[W \mid \overline{E_{j-1}} \wedge E_j \wedge \ldots \wedge E_n\right] + \Pr[W \mid E_{j-1} \wedge E_j \wedge \ldots \wedge E_n] \\ &\leq \frac{(n-j+1)m}{q} + \frac{m}{q} + \Pr[W \mid E_{j-1} \wedge E_j \wedge \ldots \wedge E_n] \qquad \text{(Schwartz-Zippel Lemma)} \\ &\leq \frac{(n-j)m}{q} + \Pr[W \mid E_{j-1} \wedge E_j \wedge \ldots \wedge E_n] \\ &\leq \frac{(n-j)m}{q} + 0 \qquad (p_{j-1}(0) + p_{j-1}(1) \neq v_{j-2}) \end{aligned}$$

  □

□ □

## 3 Stronger results via different arithmetizations

The arithmetization we used to prove that $\mathsf{CoNP} \in \mathsf{IP}$ was very coarse: if the formula was satisfiable, then its arithmetization was positive; otherwise, it was zero. However, with more precise arithmetizations, we can do better:

- $x \vee y \mapsto (1-(1-x)(1-y))$: Convert every disjunction as indicated.
- $x \wedge y \mapsto x \times y$: Convert every conjunction to multiplication.
- $\neg x \mapsto (1-x)$: Convert every negation as indicated.

Thus a clause $(x \vee y \vee \neg z)$ gets arithmetized to $(1-(1-x)(1-y)(z))$. This arithmetization evaluates to $1$ if a satisfying assignment to the clause is plugged in, and $0$ otherwise. Thus, by summing over the values taken by the arithmetization of a formula when evaluated on the Boolean hypercube, we can get the number of satisfying assignments for the formula, thus putting $\#\mathsf{P}$ inside $\mathsf{IP}$. This is a powerful result, because $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{IP}$.

## 4 Upper bounds on the power of interactive proofs

So far, we have described the classes that have interactive proofs, but what is the limit to the power of interactive proofs? As it turns out, $\mathsf{IP}$ is contained in $\mathsf{PSPACE}$. To see this, consider any language $\mathcal{L}$ in $\mathsf{IP}$. It has a particular proof system where the verifier accepts with probability greater than $2/3$. Our approach is to show that a polynomial-space machine can check that over $2/3$ of the prover-verifier interactions accept if an instance $\mathbb{x} \in \mathcal{L}$, and at most $1/3$ interactions accept if $\mathbb{x} \notin \mathcal{L}$. To do this, we model the prover-verifier interaction as a game. Because each message is of polynomial size, there are at most $2^{\texttt{poly}(n)}$ choices at each node of the game tree. Furthermore, the depth of the tree is also at most $\texttt{poly}(n)$. Thus, to perform depth-first search on the tree, a machine needs at most $\texttt{poly}(n)$ bits. Thus the language can be recognized by a poly-space machine.

## References

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan, *Algebraic methods for interactive proof systems*, Journal of the ACM **39** (1992), no. 4, 859–868.

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