Lecture Notes, Week 8

Math 480A2: Mathematics of Blockchain Protocols, Fall 2022

Lecturer: Bryan Gillespie

Interactive Proof Protocols

Definition 1. An **algorithm** $\mathcal{A}$ is a *computer program* which operates on an *input* in a set X, and produces an *output* in a set Y. Execution of an algorithm uses *time* and *space*, with usage measured in units dependent on the application and computational model.

An algorithm is often modeled by the theoretical construction of a *Turing machine*, but in practice a pseudo-code computer program with features similar to the C family of programming languages is a good way to express algorithms with the same theoretical capabilities.

Algorithm 1 Example algorithm finding the maximum in a list of rational numbers

```
1 function MAX(L: ℚ[]) → ℚ ∪ {−∞}
2     LARGEST ← −∞
3     for VAL ∈ L do
4         if VAL > LARGEST then
5             LARGEST ← VAL
6     return LARGEST
```

Algorithm 1 uses two rational numbers worth of space in its execution (the variables LARGEST and VAL), which can be written as $O(1)$, and its execution uses a constant amount of time for each iteration of the for loop, to compare VAL with LARGEST and to possibly save VAL as the new largest value. Thus if the input size is given by n, the length of the input list L, then the algorithm runs in time $O(n)$. Note that since comparison of rational numbers and assignment of a variable don't depend on the length of the input list, we can reasonably assume that these computational costs don't grow with the length of the input list.

However, notice an interesting quandary: the computational cost of comparing two rational numbers could concretely depend on the representation of the numbers, and how large their numerators and denominators are, so this analysis does not capture some subtleties of the algorithm's performance which are hidden behind the assumption that "comparison of two rational numbers takes a constant amount of time". It is important in applications to make assumptions which properly capture the most significant factors influencing performance. This is also a good example of why finite fields are often used in cryptographic settings: common operations have well-controlled computational costs.

An algorithm may make use of *randomness* in its computation, in which case it is called a **probabilistic** algorithm. Such an algorithm can be thought of as having access to a sequence of independent random bits (think of these as random coin flips), from which it can take values for use in its computations.

Definition 2. A **probabilistic algorithm** is a family of deterministic algorithms $\mathcal{A}_r$, where the subscript $r = (r_i)_{i=1}^{\infty}$ is a sequence of bits, $r_i \in \{0, 1\}$. A probabilistic algorithm on a fixed input can be interpreted as a random variable on the space of random values of r:

$$\mathcal{A}(x) : R_1 \times R_2 \times \cdots \to Y$$

where $\mathcal{A}(x) : r \mapsto \mathcal{A}_r(x)$.

A probabilistic algorithm is said to satisfy a given bound on time or space usage (e.g. *polynomial time* or $O(n \log n)$) if the maximum time or space usage over all possible random sequences satisfies the bound.

In the following, let $f : X \to Y$ be a function, where X and Y are finite sets. We want to formulate a method of "interactive proof protocols" between a prover and a verifier for statements of the form "I claim that $f(x)$ is equal to y". In such a proof protocol, we want two assurances:

- If $f(x) = y$, then the prover is able to convince the verifier of this fact with high probability
- If $f(x) \neq y$, then no matter how the prover proceeds, the verifier is able to correctly reject the claim with high probability

An interactive proof protocol is defined using two algorithms, a deterministic **prover** algorithm $\mathcal{P}$, and a probabilistic polynomial time **verifier** algorithm $\mathcal{V}$. (Here "polynomial time" must be with respect to some parameter for a space of functions for which the prover and verifier algorithms apply. Usually if f has domain X and codomain Y, then the "input size" of the claim $f(x) = y$ is $\log |X| + \log |Y|$.) The prover and verifier algorithms start by each receiving as inputs the two values x and y constituting the claim "$f(x)$ is equal to y", and exchange a sequence of messages $m_1, m_2, \ldots, m_k$ in the following way. The protocol designates either the prover or the verifier to send the first message, and then prover and verifier take turns sending messages to each other, with the last message sent by the prover. At the end, after the last message has been sent by the prover, the verifier algorithm $\mathcal{V}$ chooses to either ACCEPT or REJECT the claim.

As algorithms, this can be thought of as "next-message-computing algorithms", so that if $\mathcal{P}$ starts, then m_1 is computed by $\mathcal{P}(x, y)$, m_2 is computed by $\mathcal{V}(x, y, m_1)$, m_3 is computed by $\mathcal{P}(x, y, m_1, m_2)$, and so on. Finally, $\mathcal{V}(x, y, m_1, \ldots, m_k)$ computes as output one of either ACCEPT or REJECT to represent its conclusion from the protocol. Each time that $\mathcal{P}$ and $\mathcal{V}$ take turns sending messages is called a **round** of the protocol, so the number of rounds for a protocol with k messages is $\lceil k/2 \rceil$.

The entire sequence of messages $\tau = (m_1, \ldots, m_k)$ is called a **transcript** of the interactive protocol. We will write $\text{transcript}_r(\mathcal{P}, \mathcal{V}, x, y)$ for the transcript of the interactive protocol with specified prover and verifier algorithms, inputs x and y, and verifier randomness r. We will similarly write $\text{out}_r(\mathcal{P}, \mathcal{V}, x, y)$ for the output of the protocol in this setting, either ACCEPT or REJECT. These are deterministic functions of the inputs, but if r is omitted from the subscripts, then we interpret them as random variables over the space of possible sequences r.

With all of the above in mind, the following definition gives the two assurances required for an interactive proof system to be effective at allowing a prover to convince a verifier of the validity of a statement, the so-called *completeness* and *soundness* properties.

Definition 3. Let $\mathcal{P}$, $\mathcal{V}$ be the prover and verifier algorithms for an interactive proof system as discussed above. Then $(\mathcal{P}, \mathcal{V})$ is said to have **completeness error** δ_C and **soundness error** δ_S for f if the following two properties hold.

- *(Completeness)* For every $x \in X$,

$$\Pr[\text{out}(\mathcal{P}, \mathcal{V}, x, f(x)) = \textsc{Accept}] \geq 1 - \delta_C$$

- *(Soundness)* If $\mathcal{P}'$ is *any* deterministic prover algorithm, then for every input $x \in X$ and every output $y \in Y$ different than $f(x)$,

$$\Pr[\text{out}(\mathcal{P}', \mathcal{V}, x, y) = \textsc{Reject}] \geq 1 - \delta_S$$

The interactive proof system is called **valid** for f if it has completeness and soundness error at most $1/3$.

A number of performance measures are significant for ineractive protocols:

- The prover algorithm's time and space complexity
- The verifier algorithm's time and space complexity
- The **round complexity** of the protocol, given by the maximum value of $\lceil k/2 \rceil$ where k is the number of messages exchanged
- The total number of bits communicated, i.e. $\sum \ell(m_i)$ where ℓ denotes the length of the message, in bits

The latter two parameters may vary depending on the choice of input x and output y, and on the choice of the randomness r used by the verifier. To quantify these values when this happens, either a maximum or an average of the numbers of rounds may be used.

Remark 4. One might ask the (justified) question of what is gained by introducing the formalism of an interactive proof. The answer, roughly, is that interactive proof protocols have the ability to provide a practical method of proof for statements which are more computationally difficult than what is possible to prove without the added features of interaction and randomness.

In the theory of computation, a **language** $\mathcal{L}$ is a subset of a space U of possible outcomes — sometimes defined as $U = \{0,1\}^n$ for some integer n, but more generally any finite or countably infinite set is allowable. The language $\mathcal{L}$ describes the elements of U which are the members of interest, and a computation solves the **decision problem** for a language if, given an element $x \in U$, it can determine whether x is a member of $\mathcal{L}$. An interactive protocol is said to solve the decision problem for a language L with soundness error δ_C and completeness error δ_S if for any $x \in \mathcal{L}$, the probability that $\text{out}(\mathcal{P}, \mathcal{V}, x) = \textsc{Accept}$ is

bounded below by $1 - \delta_C$, and for any $x \notin \mathcal{L}$ and any prover strategy $\mathcal{P}'$, the probability that $\text{out}(\mathcal{P}', \mathcal{V}, x) = \textsc{Reject}$ is bounded below by $1 - \delta_S$. The function definition of an interactive proof protocol is thus the definition just given, for the language $\mathcal{L} = \{(x, f(x)) : x \in X\} \subseteq X \times Y = U$.

The space of languages which are decidable by a deterministic polynomial time algorithm with the help of a pre-computed proof (sometimes called a "witness" or "certificate") is the well-known class of NP problems. The languages decidable by an interactive proof protocol in the above way can be shown to be exactly the class PSPACE of languages which are decidable by a deterministic algorithm using at most polynomial space. This class includes NP as a subset, but is believed (but not proven!) to be larger than NP.

A further theoretical gap is obtained from a further generalization of interactive proof protocols, called *interactive oracle proofs* or *IOPs*. We will define these protocols later, but it can be shown that IOPs can decide languages in the (probably) even larger class NEXP of languages decideable by a "non-deterministic" algorithm in exponential time (here, 2 raised to some polynomial in the input size). These complexity classes satisfy an inclusion hierarchy:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXP} \subseteq \text{NEXP}$$

Unfortunately, there is no proof that any of these inclusions are strict inclusions — however, it is expected that all of them are. One conclusion that can be drawn from known results (specifically, the *time hierarchy theorem*) is that there is a strict inclusion in at least one of the steps between P and EXP. Besides this, no further relations are known.

Sum-check Protocol

We have previously seen one interactive protocol, giving a proof system for the "graph non-isomorphism" problem. We will return to this protocol again later when we discuss zero-knowledge proofs. We now introduce a protocol that gets significant mileage either by specializing it to solve useful problems, or as a sub-component of more complicated protocols: the *sum-check protocol*.

Let $f \in K[x_1, \ldots, x_v]$ be an n-variate polynomial over a finite field K with degree at most d_i in each variable x_i, and let A be a small finite subset of K. Then the sum-check protocol provides an interactive proof system for the function

$$S : f \mapsto \sum_{x_1, \ldots, x_n \in A} f(x_1, \ldots, x_n)$$

We will focus on the case of $A = \{0, 1\}$, wherein the polynomial is summed over all binary inputs, but note that the algorithm generalizes to different sets A without significant modification. In the binary case, we are trying to prove interactively that the summation

$$\sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} f(x_1, \ldots, x_n)$$

has a particular value. The naive approach in which a verifier just checks the sum themselves requires evaluating f at 2^n separate inputs and computing the sum of all of these values. We

will see that the interactive protocol below will allow the verifier to confirm the value of this sum with high probability in time $O(v + \tau)$, where τ is the time it takes to evaluate f at a single input in K^n. The prover will have a little more work to do than the original summation in order to produce this proof, but the prover's runtime is still $O(2^n\tau)$ for polynomials of uniformly bounded degree, which is only a (usually small) constant factor more than just computing the sum without a proof. The protocol is given by the following.

- $\mathcal{P}$ and $\mathcal{V}$ are given access to the polynomial f and the claimed sum T as inputs to the protocol
- For each round $i = 1, \ldots, n$, a tuple of previous random challenges $(r_1, \ldots, r_{i-1})$ has been generated, and the following steps are executed:
  - $\mathcal{P}$ sends the univariate polynomial $g_i(y)$ claimed to be equal to the polynomial
$$f_i(y) = \sum_{x_{i+1},\ldots,x_n \in \{0,1\}} f(r_1, \ldots, r_{i-1}, y, x_{i+1}, \ldots, x_n)$$
  - $\mathcal{V}$ checks that g_i has degree at most d_i, and that $g_{i-1}(r_{i-1}) = g_i(0) + g_i(1)$, and rejects if either of these conditions fails
  - $\mathcal{V}$ chooses a random challenge $r_i \in K$ uniformly at random, and sends it to $\mathcal{P}$
- $\mathcal{V}$ returns Accept if $g_n(r_n) = f(r_1, \ldots, r_n)$, and otherwise returns Reject.

In the above, for round 1, the tuple $(r_1, \ldots, r_{i-1})$ is interpreted as an empty tuple, and the polynomial $g_{i-1}(r_{i-1})$ is interpreted as the claimed sum T. For round n, it is not necessary for $\mathcal{V}$ to send r_n back to $\mathcal{P}$, as the role of $\mathcal{P}$ in the protocol has concluded by this point.

Example 5. Let $f(x_1, x_2, x_3) = x_1^2 + x_1x_2x_3 + 3x_1x_3 + x_2^2$. Then the sum of f over all binary inputs is $S(f) = 4 + 1 + 6 + 4 = 15$. The sum-check protocol with inputs f and $T = 15$ then proceeds as follows. In round 1, $\mathcal{P}$ computes the sum

$$\begin{aligned} g_1(y) = f_1(y) &= \sum_{x_2,x_3 \in \{0,1\}} f(y, x_2, x_3) \\ &= f(y, 0, 0) + f(y, 0, 1) + f(y, 1, 0) + f(y, 1, 1) \\ &= (y^2) + (y^2 + 3y) + (y^2 + 1) + (y^2 + y + 3y + 1) \\ &= 4y^2 + 7y + 2 \end{aligned}$$

Then $\mathcal{V}$ checks that g_1 has degree at most 2, and that

$$g_1(0) + g_1(1) = 2 + (4 + 7 + 2) = 15 = T$$

The verifier then chooses a random number, say $r_1 = 4$, and sends it back to $\mathcal{P}$. In round 2, $\mathcal{P}$ computes the sum

$$\begin{aligned} g_2(y) = f_2(y) &= \sum_{x_3 \in \{0,1\}} f(r_1, y, x_3) \\ &= f(4, y, 0) + f(4, y, 1) \\ &= (16 + y^2) + (16 + 4y + 12 + y^2) \\ &= 2y^2 + 4y + 44 \end{aligned}$$

Then $\mathcal{V}$ checks that g_2 has degree at most 2, and equality between

$$g_2(0) + g_2(1) = 44 + (2 + 4 + 44) = 94$$

and

$$g_1(r_1) = 4(4)^2 + 7(4) + 2 = 64 + 28 + 2 = 94$$

The verifier chooses another random number, say $r_2 = -2$, and sends it to $\mathcal{P}$. Finally, in round 3, $\mathcal{P}$ computes the sum

$$\begin{aligned} g_3(y) = f_3(y) &= f(r_1, r_2, y) \\ &= f(4, -2, y) \\ &= 16 - 8y + 12y + 4 \\ &= 4y + 20 \end{aligned}$$

Then $\mathcal{V}$ checks that g_3 has degree at most 1, and equality between

$$g_3(0) + g_3(1) = 20 + (4 + 20) = 44$$

and

$$g_2(r_2) = 2(-2)^2 + 4(-2) + 44 = 8 - 8 + 44 = 44$$

Last, $\mathcal{V}$ picks a final random number, say $r_3 = 5$, and checks equality between

$$g_3(r_3) = 4(5) + 20 = 40$$

and

$$f(r_1, r_2, r_3) = f(4, -2, 5) = (4)^2 + (4)(-2)(5) + 3(4)(5) + (-2)^2 = 16 - 40 + 60 + 4 = 40$$

The last check satisfied, the verifier now returns ACCEPT.

In the following, let K be a finite field, let $\mathbf{d} = (d_1, \ldots, d_n)$ be a vector of nonnegative integers, and let $\mathcal{P}_\mathbf{d}$ be the set of polynomials in $K[x_1, \ldots, x_n]$ having degree at most d_i in each variable x_i.

Proposition 6. *Let $S : \mathcal{P}_\mathbf{d} \to K$ be the binary summation function, defined by*

$$S : f \mapsto \sum_{x_1, \ldots, x_n \in \{0,1\}} f(x_1, \ldots, x_n)$$

Then the sum-check protocol is an interactive proof system for S with completeness error $\delta_C = 0$, and soundness error $\delta_S = (d_1 + \ldots + d_n)/|K|$.

Proof. If the claimed sum T coincides with the actual sum $S(f)$ and $\mathcal{P}$ is the prescribed sum-check prover, then all of the checks made by $\mathcal{V}$ will succeed. This is because of the degree restrictions on f, and the fact that

$$\begin{aligned} f_{i-1}(r_{i-1}) &= \sum_{x_i, \ldots, x_n \in \{0,1\}} f(r_1, \ldots, r_{i-1}, x_i, x_{i+1}, \ldots, x_n) \\ &= \sum_{y \in \{0,1\}} \sum_{x_{i+1}, \ldots, x_n \in \{0,1\}} f(r_1, \ldots, r_{i-1}, y, x_{i+1}, \ldots, x_n) = f_i(0) + f_i(1) \end{aligned}$$

In the end, $f_n(r_n)$ is equal to $f(r_1, \ldots, r_n)$ by definition, so in this case, $\mathcal{V}$ returns ACCEPT.

Now suppose that an adversarial prover algorithm $\mathcal{P}'$ attempts to produce a false proof for a claimed sum T which is not equal to $S(f)$. In order for $\mathcal{V}$ to accept the claim, it will have to be the case that $g_n(r_n) = f_n(r_n) = f(r_1, \ldots, r_n)$, as this is the last check that must be satisfied in order for $\mathcal{V}$ to return ACCEPT. This outcome is a subset of the event $[g_i(r_i) = f_i(r_i)$ for some index $i]$, so an upper bound on the probability of this event will suffice to prove our soundness error bound. For the following, let $A_i = [g_i(r_i) = f_i(r_i)]$. Then we have the bound

$$\begin{aligned}
\Pr(A_n) &\leq \Pr\left(\cup_{i=1}^{n} A_i\right) \\
&= \sum_{i=1}^{n} \Pr(A_i \cap (A_1 \cup \cdots \cup A_{i-1})^{\mathrm{c}}) \\
&= \sum_{i=1}^{n} \Pr(A_i \mid (A_1 \cup \cdots \cup A_{i-1})^{\mathrm{c}}) \Pr((A_1 \cup \cdots \cup A_{i-1})^{\mathrm{c}}) \\
&\leq \sum_{i=1}^{n} \Pr(A_i \mid (A_1 \cup \cdots \cup A_{i-1})^{\mathrm{c}})
\end{aligned}$$

The first term in this sum is just $\Pr(A_1)$. In the outcomes in this event, the prover strategy $\mathcal{P}'$ produces a polynomial g_1 which must satisfy $g_1(0) + g_1(1) = T$. Since f_1 is a polynomial satisfying $f_1(0) + f_1(1) = S(f) \neq T$, it must be the case that $g_1 \neq f_1$ as polynomials. Thus the Schwartz-Zippel lemma implies that the probability that $g_1(r_1) = f_1(r_1)$ for r_1 chosen uniformly from K and independently of g_1 and f_1 is bounded above by $d_1/|K|$.

For subsequent terms of the sum, we consider the conditional probability

$$\Pr(A_i \mid (A_1 \cup \cdots \cup A_{i-1})^{\mathrm{c}})$$

for $i > 1$. In the restricted probability space of outcomes satisfying $g_j(r_j) \neq f_j(r_j)$ for $j < i$, $\mathcal{P}'$ produces a polynomial g_i which must satisfy $g_i(0) + g_i(1) = g_{i-1}(r_{i-1})$. Since $f_i(0) + f_i(1) = f_{i-1}(r_{i-1}) \neq g_{i-1}(r_{i-1})$, it must be the case that $g_i \neq f_i$ as polynomials in order for this check to pass. Therefore, we can again conclude by the Schwartz-Zippel lemma that for r_i chosen uniformly from K, the probability that $g_i(r_i) = f_i(r_i)$ is bounded above by $d_i/|K|$.

Summing these bounds gives the desired estimate,

$$\Pr[\mathrm{out}(\mathcal{P}', \mathcal{V}, f, T) = \textsc{Accept}] \leq \Pr(A_n) \leq \sum_{i=1}^{n} \frac{d_i}{|K|} = \frac{\sum_{i=1}^{n} d_i}{|K|}$$

□

The costs of the sum-check protocol can be assessed as follows. There are n rounds of interaction, and the total amount of data communicated is $d_i + 1$ coefficients in K for the polynomial $g_i(y)$, plus a single random field element in all but the final round, which totals to $\sum_i d_i + 2n + 1$ field elements in total.

The verifier checks that $g_i(0) + g_i(1) = g_{i-1}(r_{i-1})$ in each round, so by the end of the protocol, each polynomial g_i has been evaluated 3 times. Additionally, the verifier has to

evaluate $f(r_1, \ldots, r_n)$ at the end. Thus the verifier's runtime is $O(\sum_i d_i) + \tau$, where τ is the time it takes to evaluate f at random field elements.

The prover's runtime is a bit more subtle to analyze. One equivalent way to describe the polynomial f_i is to send its evaluations at $d_i + 1$ different points, say $y \in \{0, 1, \ldots, d_i\}$. Computing these values costs $2^{n-i}(d_i + 1)\tau$, where τ again represents the time required to evaluate f. Summing over the various rounds gives a runtime of

$$O(\sum_{i=1}^{n} 2^{n-i}(d_i + 1)\tau)$$

If the degrees d_i are bounded by a constant, then this can be simplified to $O(2^n\tau)$, since the sum of a geometric sequence of terms 2^{n-i} is bounded by 2^n.

Application: IP for #SAT

In the following, we apply the sum-check protocol to give an interactive proof protocol solving the "#SAT" problem.

Definition 7. A **boolean formula** ϕ over variables $x_1, \ldots, x_n$ is a binary tree where each leaf is labeled by either a variable x_i, or its negation $\bar{x}_i$, and each interior vertex is labeled by either AND or OR. The **size** of ϕ is the number of leaf vertices. A boolean formula can be thought of as a function $\phi : \{0,1\}^n \to \{0,1\}$ by interpreting the values 0 and 1 as False and True respectively, and mapping an input $(x_1, \ldots, x_n)$ to the value obtained by evaluating each gate at the values of its children in the binary tree. An input vector satisfying $\phi(x_1, \ldots, x_n) = 1$ is called a **satisfying assignment** of ϕ.

If ϕ is a boolean formula, then the **#SAT problem** is the problem of computing the total number of satisfying assignments of ϕ. This number can be written as

$$\sum_{x_1,\ldots,x_n \in \{0,1\}} \phi(x_1, \ldots, x_n)$$

If ϕ were a low-degree polynomial over a finite field, then the sum-check protocol would be an ideal approach to allow a prover to convince a verifier of this value. To accomplish this, we transform ϕ into an equivalent *arithmetic circuit*.

Definition 8. An **arithmetic circuit** ψ over a field K with variables $x_1, \ldots, x_n$ is a directed, acyclic graph with source vertices labeled by variables or constants in K, called the **inputs** of the circuit, non-source vertices labeled by either $+$, $-$, or $\times$ with $+$ and $\times$ vertices having in-degree 2 and $-$ vertices having in-degree 1, and exactly one sink vertex with out-degree 0.

Arithmetic circuits can be thought of as "polynomials with a specified ordering of arithmetic operations". To each arithmetic circuit ψ there is an associated polynomial which is defined recursively by combining the polynomials associated with the inputs ψ_1 and ψ_2 of the sink vertex of ψ using the operation labeling this vertex.

If ϕ is a boolean formula with size m, then an arithmetic circuit ψ can be constructed which extends the boolean function of ϕ to the polynomial of ψ (that is, ψ agrees with ϕ on all boolean inputs). This is accomplished as follows:

- Include one input for each variable x_i of ϕ, and one input for the constant 1
- For each variable x_i which appears negated in ϕ, add a $-$ gate with input x_i and a $+$ gate with inputs 1 and $-x_i$ to give the expression $1 - x_i$.
- For each AND gate with previously constructed arithmetic inputs ψ_1 and ψ_2, add a $\times$ gate representing the expression $\psi_1 \times \psi_2$.
- For each OR gate with previously constructed arithmetic inputs ψ_1 and ψ_2, add gates representing the expression $(\psi_1 + \psi_2) - (\psi_1 \times \psi_2)$

The result of this operation is a polynomial which coincides with ϕ on all boolean inputs, along with an evaluation ordering which allows the evaluation of this polynomial at any input vector using at most m multiplication operations, at most $4m$ addition operations, and at most $2m$ negation operations. Then we can compute

$$\sum_{x_1,\ldots,x_n \in \{0,1\}} \psi(x_1, \ldots, x_n)$$

to solve the #SAT problem. Note that this is a sum in K, so if K has characteristic p then we know from this sum that the number of satisfying assignments is congruent to the sum modulo p. However, as long as $\operatorname{char} K > 2^n$, this identifies the integer sum uniquely. For the analysis of the sum-check protocol, we need an analysis of the variable-wise degrees of the polynomial ψ.

Lemma 9. *If ψ is the arithmetic circuit associated with a boolean formula of size m, then as a polynomial, $\sum_i \deg_i(\psi) \leq m$.*

Proof. This follows by the inductive construction of ψ. If ψ is a variable or its complement then $m = 1$ and the total degree of ψ is 1. Otherwise, if the root gate of the associated boolean formula ϕ is an AND or OR gate, then ψ combines the output of two sub-circuits ψ_1 and ψ_2 corresponding to the inputs of the root gate of ϕ, and whose sizes m_1 and m_2 sum to m. This combination produces a polynomial which is either $\psi_1 \times \psi_2$ or $\psi_1 + \psi_2 - (\psi_1 \times \psi_2)$, and in either case, the degree of this polynomial in a variable x_i is at most the sum of the degrees of ψ_1 and ψ_2 in x_i. In particular, by inductive hypothesis, $\sum_i \deg_i(\psi_j) \leq m_j$, $j = 1, 2$, so

$$\sum_{i=1}^{n} \deg_i(\psi) \leq \sum_{j \in 1,2} \sum_{i=1}^{n} \deg_i(\psi_j) \leq m_1 + m_2 = m$$

□

In particular, the sum-check protocol applied to the associated polynomial of ψ allows a prover to convince a verifier of the number of satisfying assignments of ϕ while requiring only $O(n + \sum_i \deg_i \psi) + O(m) = O(m)$ field operations. The soundness error of the protocol is at most $m/|K|$.