A DETAILED PROOF THAT IP=PSPACE

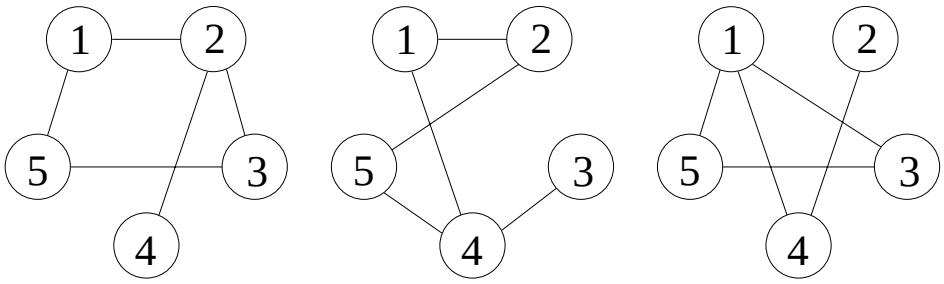
B.J. MARES

Abstract. I will define IP, the class of interactive proofs, and present a thorough proof that IP=PSPACE, assuming minimal prior knowledge. Such a presentation is appropriate for undergraduates in an introductory course on computational theory such as the one for which this paper was written.

1. Graph Non-isomorphism

Suppose we have two graphs G_1 and G_2, each with n nodes labeled $1 \dots n$. We say that G_1 and G_2 are *isomorphic* if there exists a permutation $\sigma : \{1, \dots, n\} \to \{1, \dots, n\}$ of the node labels such that $\sigma(G_1) = G_2$. This is to say that two graphs are isomorphic when they are the same up to how the nodes are labeled. Note that isomorphism is an equivalence relation.

Consider the following example graphs with $n = 5$:



The first two graphs are the isomorphic to each other, since we can relabel the vertices in either graph until they look the same. The third graph contains a triangle, but the other two do not. Therefore, the third graph is not isomorphic to either of the first two.

In general, it is difficult to tell whether or not two graphs are isomorphic. A Turing machine is not so visual. It represents graphs with adjacency matrices. To a Turing machine, the above graphs look like:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

We define the language $ISO := \{\langle G_1, G_2\rangle : G_1 \text{ and } G_2 \text{ are isomorphic graphs}\}$. It is not known to be in P. It is in NP since the permutation σ such that $\sigma(G_1) = G_2$ acts as a

certificate. Permuting an adjacency matrix amounts to permuting both its rows and columns by σ, each of which can be done with a single matrix multiplication in polynomial time.

Consider the compliment language $NONISO$. It is not known to be in NP or BPP. Suppose that we have two non-isomorphic graphs G_1 and G_2, but we are not yet convinced that they are non-isomorphic. Also suppose that we have a friend with unlimited computational resources trying to convince us that G_1 and G_2 are non-isomorphic. Can our friend succeed in polynomial time?

Consider the following algorithm: we flip a coin to choose $i \in \{1, 2\}$. We then devise a random permutation σ and apply it to G_i. We then send our friend G_1, G_2, and $\sigma(G_i)$ and ask for i. If G_1 and G_2 are truly non-isomorphic, our friend will have no trouble determining i by figuring out which of G_1 and G_2 is isomorphic to $\sigma(G_i)$.

Now suppose that G_1 and G_2 are actually isomorphic and our friend is trying to trick us into believing that they aren't. Then $\sigma(G_i)$ is isomorphic to both G_1 and G_2. Moreover, it should be intuitive that the number of permutations that take G_1 to $\sigma(G_i)$ equals the number of permutations that take G_2 to $\sigma(G_i)$. This is formally proven in elementary group theory. Therefore, when we send $\sigma(G_i)$, our friend has no information about i, and hence a 50-50 chance of correctly guessing it.

Suppose we repeat the process twice. Then if $\langle G_1, G_2\rangle \in NONISO$, our friend correctly guesses i both times with probability one. If $\langle G_1, G_2\rangle \notin NONISO$, the probability that our friend will guess i correctly both times is only $\frac{1}{4}$.

Motivated by this example of $NONISO$, we define the class IP.

2. Interactive Proofs

NP is the class of languages decidable by a nondeterministic Turing machine in polynomial time. It is also equivalent to the class of languages verifiable in polynomial time. For the rest of this discussion, it will be useful to think of NP languages not in terms of nondeterminism, but in terms of verification.

We can view NP as the set of languages for which a supercomputer can provide a certificate to a deterministic polynomial time machine. No languages are added to NP if we extend this definition to allow the deterministic polynomial time machine to talk back and forth. This is because the supercomputer is capable of simulating the polynomial machine, anticipating its questions, and providing its answers all at the beginning. Thus NP is the language obtained by permitting interaction with a supercomputer whose responses must be verifiable.

Similarly, we may view BPP as the result of augmenting a deterministic polynomial time machine with an additional capability: randomness.

Consider the following diagram, where right arrows denote the result of introducing randomness, and up arrows denote the result of introducing interaction:

$$\begin{array}{ccc} NP & \rightarrow & IP \\ \uparrow & & \uparrow \\ P & \rightarrow & BPP \end{array}$$

Thus NP results from adding interaction to P, and similarly, BPP results from adding randomness to P. What happens when we add both interaction and randomness to P?

We will define the result to be IP, the class of languages decidable by an interactive proof system.

To be more precise, we must formally define what is meant by an interactive proof system. A *message history* is a string of the form $m_1\#m_2\#\ldots\#m_i$, where the m_i denote successive messages in a dialogue. A *verifier* V is a probabilistic, polynomial-time computable function such that given an input string w and a message history $m_1\#m_2\#\ldots\#m_i$, it computes $V(w, m_1\#m_2\#\ldots\#m_i) = m_{i+1}$ for i even. Also, V must accept or reject after a polynomial number of interactions. Note that m_{i+1} is not deterministic, but determined only up to a probability distribution. A *prover* P is an arbitrary function that computes $P(w, m_1\#m_2\#\ldots\#m_i) = m_{i+1}$ for i odd, with the only restriction that m_{i+1} is polynomial length.

Given a prover P and a verifier V, they interact by alternately augmenting the message history until V accepts or rejects. We denote this interaction by $V \leftrightarrow P$. Note that any interaction requires a polynomial number of steps of the verifying machine. Given an input string w, there is a probability associated with $V \leftrightarrow P$ accepting w. We denote this probability by $\Pr[V \leftrightarrow P \text{ accepts } w]$.

Formally, a language $L \in$IP if there exists a verifier V such that:

(1) $\exists P : w \in L \Rightarrow \Pr[V \leftrightarrow P \text{ accepts } w] \geq \frac{2}{3}$,

(2) $\forall \tilde{P} : w \notin L \Rightarrow \Pr\left[V \leftrightarrow \tilde{P} \text{ accepts } w\right] \leq \frac{1}{3}$.

This is to say that there's a prover P capable of convincing the verifier that $w \in L$ to a high probability whenever a string is in the language. If a string is not in the language, it's impossible for any deceitful prover $\tilde{P}$ to fool the verifier into believing that $w \in L$ to a high probability.

We have outlined an algorithm for a $NONISO$ verifier and shown that

(1) $\exists P : \langle G_1, G_2\rangle \in NONISO \Rightarrow \Pr[V \leftrightarrow P \text{ accepts } \langle G_1, G_2\rangle] = 1$,

(2) $\forall \tilde{P} : \langle G_1, G_2\rangle \notin NONISO \Rightarrow \Pr\left[V \leftrightarrow \tilde{P} \text{ accepts } \langle G_1, G_2\rangle\right] \leq \frac{1}{4}$.

Therefore $NONISO \in IP$.

Several variations on this definition of IP lead to the same class. The following are equivalent:

(1) IP.

(2) IP defined with one-sided error: $\exists P : w \in L \Rightarrow \Pr[V \leftrightarrow P \text{ accepts } w] = 1$.

(3) IP defined so that the prover knows the random numbers that the verifier has generated.

(4) IP defined so that the prover is a $PSPACE$ computable function.

(5) $PSPACE$.

I will prove only the equivalence of IP and $PSPACE$.

3. $IP \subset PSPACE$

It's easy to show that $IP \subset PSPACE$. Suppose $L \in IP$. Then we have some verifier V for L. The $PSPACE$ algorithm we use to decide L is to compute

$$z = \max_P \Pr[V \leftrightarrow P \text{ accepts } w].$$

By the definition of IP, if $z \geq \frac{2}{3}$, then we know that $w \in L$. If $z \leq \frac{1}{3}$, then we know that $w \notin L$. Note that if V is a verifier for L, $\frac{1}{3} < z < \frac{2}{3}$ will never occur.

Now we need to show that it's possible to compute z in PSPACE. Suppose V runs in $p(n)$ steps, where $n = |w|$. Then any given response by P is no longer than $p(n)$. Also, V chooses at most $p(n)$ random numbers. We may recursively simulate V while branching for each random number, and each possible response by P. Therefore, the recursion depth is polynomial, so we can perform such a recursion in $PSPACE$. We keep a count of the accepting branches produced by P's optimal responses, as well as the total number of branches. These numbers will be exponential in n, but the length of the numerical representation is polynomial in n. The ratio computed by this $PSPACE$ algorithm is z, therefore $IP \subset PSPACE$.

Recall that $TQBF$, the language of true quantified boolean formulas, is $PSPACE$-complete. Therefore, $IP = PSPACE \Leftrightarrow TQBF \in IP$. The goal of this paper will be to show $TQBF \in IP$, hence $IP = PSPACE$.

4. Primes and Polynomials

The central idea of these proofs is to extend boolean arithmetic from a binary number system to a larger finite field modulo a prime. This process is called arithmetization. It will require some facts about finite fields which we will review.

Theorem. *Let p be a prime. For all $a \not\equiv 0 \pmod{p}$ there exists a unique number a^{-1} such that $aa^{-1} \equiv 1 \pmod{p}$.*

Proof. Suppose not. Then we have a number $a \not\equiv 0 \pmod{p}$ such that $ab \not\equiv 1 \pmod{p}$ for all b. Consider the function $f(b) = ab \pmod{p}$. Then f maps all p numbers modulo p to the $p-1$ numbers modulo p excluding $1 \pmod{p}$. By the pigeonhole principle, there must exist numbers c and d such that both $c \not\equiv d \pmod{p}$ and $ac \equiv ad \pmod{p}$. Therefore, $a(c-d) \equiv 0 \pmod{p}$, so $p \mid a(c-d)$. By hypothesis, $p \nmid a$, so $p \mid (c-d)$. Therefore, $c \equiv d \pmod{p}$, contradicting the pigeonhole principle, so a^{-1} must exist.

Now for uniqueness, suppose that $a \not\equiv 0 \pmod{p}$ and we have two inverses c and d so that $ac \equiv ad \equiv 1 \pmod{p}$. Then similarly $p \mid a(c-d)$, but $p \nmid a$, so $p \mid (c-d)$, and $c \equiv d \pmod{p}$. Thus any two inverses of the same number are equivalent. □

Definition. A polynomial modulo p is a polynomial with coefficients modulo p that evaluates to a value modulo p. Two polynomials are equivalent modulo p if their coefficients are equivalent modulo p.

Theorem. *Let $q(x)$ be a nonzero polynomial of degree n modulo p such that $q(a) \equiv 0 \pmod{p}$. Then $q(x) \equiv (x-a)q'(x) \pmod{p}$ for some polynomial $q'(x)$ of degree $n-1$.*

Proof. This follows as a consequence of the polynomial division algorithm one traditionally learns in precalculus. By the existence and uniqueness of multiplicative inverses, we can always find unique coefficients to ensure the proper cancellations. For example, $2 \pmod{5}$ is a root of $x^3 - 2x^2 + 2x + 1 \pmod{5}$, and

$$\begin{array}{rl} & x^2 + 2 \\ x - 2 & \overline{|x^3 - 2x^2 + 2x + 1} \end{array} \pmod{5}.$$

Therefore we may write $x^3 - 2x^2 + 2x + 1 \equiv (x-2)(x^2+2) \pmod{5}$.

In order for this algorithm to work, we must ensure that we always get a remainder of zero so that $(x-a)$ divides $q(x)$ evenly. The remainder will be a number $r \pmod p$. The division algorithm guarantees that $q(x) \equiv (x-a)q'(x) + r \pmod p$. Clearly $r \equiv 0 \pmod p$ since $0 \equiv q(a) \equiv (a-a)q'(a) + r \equiv r \pmod p$. Therefore $q(x) \equiv (x-a)q'(x)$. □

Corollary. *Suppose $s(x)$ and $t(x)$ are distinct polynomials modulo p, each of degree at most n. Then the equation $s(x) \equiv t(x)$ has at most n solutions modulo p.*

Conclusion. Let $q(x) = s(x) - t(x)$. Then every solution to $s(x) \equiv t(x)$ is a root of $q(x)$. Since $s(x)$ and $t(x)$ are of degree of at most n, $q(x)$ has degree at most n. We may iterate the division algorithm on $q(x)$ until we have $q(x) \equiv (x-a_1)(x-a_2)\cdots(x-a_i)q'(x) \pmod p$, where q' has no roots. In this form, we know that the roots of q are $a_1, \ldots, a_i$. Computing the degrees of both sides, we have

$$\begin{aligned} & n \\ \geq\ & \deg q(x) \\ =\ & i + \deg q'(x) \\ \geq\ & i \\ \geq\ & \#\text{ of roots of } q(x) \\ =\ & \#\text{ of solutions to } s(x) \equiv t(x) \pmod p. \end{aligned}$$

This establishes the theorem.

Suppose that we have a polynomial $s(x)$ modulo p of degree n, where n is much smaller than p. Is it ever possible to construct a different polynomial $t(x)$ such that $s(x) \equiv t(x)$ for most values of x? This corollary surprisingly proves that the answer is no. Any two polynomials of low degree modulo p are either identical or they rarely agree. We can exploit this as an algorithm to determine to a high probability whether or not two polynomials are the same. More specifically, if $s(x) \equiv t(x) \pmod p$, then

$$Pr[s(y) \equiv t(y) \pmod p \text{ for a randomly chosen } y] = 1.$$

If $s(x) \not\equiv t(x) \pmod p$, then

$$Pr[s(y) \equiv t(y) \pmod p \text{ for a randomly chosen } y \bmod p] \leq \frac{n}{p}.$$

One may be wondering at this point why such an algorithm is of any use when we can determine whether or not two polynomials are equal by comparing their coefficients. The answer is that we may not be able to explicitly compute the coefficients of one of the polynomials. Therefore, a verifier can check to a high probability whether or not a particular polynomial is an expansion of another without expanding the polynomial itself.

To implement our future algorithm, we will need to discover such primes p. It will be useful to know that the primes are distributed well enough so that we can find one whenever we need one. Let $\pi(m)$ denote the number of primes $\leq m$.

Theorem. *For $m \geq 3$, $\sqrt{m} \leq \pi(m) \leq m$.*

Proof. The upper bound follows directly from the definition of $\pi(m)$. The lower bound is more complicated.

A number n is prime if and only if all primes $p < \sqrt{n}$ do not divide it. Most everyone is familiar with the sieve of Eratosthenes in which one lists all the numbers from $1 \dots m$, circles the lowest number greater than one, which will be prime, crosses out all multiples of that prime, and then repeats this process for all primes $\leq \sqrt{m}$. After this is done, all numbers not crossed off are primes. A careful analysis of how many numbers can get crossed off yields the desired bound.

Suppose $m = 51$, so we start off with a list of 51 numbers. For $p = 2$, we cross off $\frac{24}{51}$ of these numbers: $4, 6, 8, \dots, 50$. This is less than $\frac{1}{2}$ of the m numbers in our list. Now when $p = 3$, how many of the remaining 27 numbers do we cross off? We can list the numbers we attempt to cross off: $3 \cdot 2$, $3 \cdot 3$, $3 \cdot 4$, $\dots$, $3 \cdot 17$, however, many of these numbers are already marked off: the even multiples of three. In fact, $\frac{8}{16}$ of these numbers are already marked off, so on this step we mark off $\frac{8}{27} < \frac{1}{3}$ of the remaining numbers. This is less than $\frac{1}{3}$ of the $27 > \frac{1}{2}m$ numbers not crossed off by 2. Now for $p = 5$, how many of the remaining 19 numbers do we cross off? We attempt to cross off $5 \cdot 2$, $5 \cdot 3$, $\dots$, $5 \cdot 10$. Now the pattern begins to emerge: the multiples of 5 already crossed out are the multiples whose quotient is already crossed out. This time we can cross off 2 numbers: $5 \cdot 5$ and $5 \cdot 7$. This is less than $\frac{1}{5}$ of the $19 > \frac{2}{3} \cdot \frac{1}{2}m$ numbers not crossed off by $p = 2$ or $p = 3$. Now $17 > \frac{4}{5} \cdot \frac{2}{3} \cdot \frac{1}{2}m$ numbers remain. For $p = 7$, we only cross out $7 \cdot 7$. Thus $16 > \frac{6}{7} \cdot \frac{4}{5} \cdot \frac{2}{3} \cdot \frac{1}{2}m$ numbers remain. $p = 7$ is the last prime $< \sqrt{51}$. The number one is not prime. Therefore,

$$\pi(51) = 15 > \frac{6}{7} \cdot \frac{4}{5} \cdot \frac{2}{3} \cdot \frac{1}{2}m = m \prod_{p<\sqrt{m}} \frac{p-1}{p}.$$

It is somewhat tricky, although elementary, to prove that in general,

$$\pi(m) \geq m \prod_{p<\sqrt{m}} \frac{p-1}{p}.$$

This product over primes is not in general easy to compute. We can approximate it by the following:

$$m \prod_{p<\sqrt{m}} \frac{p-1}{p} \geq m \prod_{i=2}^{\lfloor\sqrt{m}\rfloor} \frac{i-1}{i}.$$

This new product collapses, so we see

$$\pi(m) \geq \frac{m}{\lfloor\sqrt{m}\rfloor} \geq \sqrt{m}.$$

□

Corollary. *There is always a prime p such that $m < p \leq (m+1)^2$.*

Proof. To prove this, we know there is such a prime if and only if $\pi\left((m+1)^2\right) - \pi(m) \geq 1$ by the definition of $\pi(m)$. Using the bounds of the previous theorem,

$$\pi\left((m+1)^2\right) - \pi(m) \geq \sqrt{(m+1)^2} - m = 1.$$

□

This result is extremely weak. There are much tighter bounds on $\pi(m)$ that produce better results. The Prime Number Theorem states that $\pi(m)$ is asymptotic to $m/\ln m$. The proof involves advanced analytic number theory beyond the scope of this paper.

Corollary. *For all $c > 1$, there is always a prime p such that $m < p \leq cm$ for sufficiently large x.*

Proof. The Prime Number Theorem says that for all $\epsilon > 0$, $(1-\epsilon)\frac{m}{\ln m} < \pi(m) < (1+\epsilon)\frac{m}{\ln m}$ for sufficiently large m. Therefore, for large m, we have the bound $\pi(cm) - \pi(m) > (1-\epsilon)\frac{cm}{\ln(cm)} - (1+\epsilon)\frac{m}{\ln m}$. Choosing $\epsilon = \frac{c-1}{c+3}$, this bound proves that $\pi(cm) - \pi(m) > 1$ for large m. □

5. *COUNTSAT* SIMULATIONS

Suppose we are given the string $w = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3), 6$, and we wish to show that it is in *COUNTSAT*, that is to say that $\phi(x_1, x_2, x_3) := (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$ has 6 satisfying assignments. Identifying "true" with 0 and "false" with 1, the truth table is as follows:

	$x_3 = 0$		$x_3 = 1$	
	$x_2 = 0$	$x_2 = 1$	$x_2 = 0$	$x_2 = 1$
$x_1 = 0$	1	0	1	1
$x_1 = 1$	0	1	1	1

We can express the number of satisfying assignments of ϕ as follows:

$$\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \sum_{x_3=0}^{1} \phi(x_1, x_2, x_3).$$

The problem then reduces to verifying that this sum is in fact equal to 6. It is useful to break this problem up into parts. For the general case with m boolean variables, we define functions

$$f_i(x_1, \ldots, x_i) = \sum_{x_{i+1}=0}^{1} \sum_{x_{i+2}=0}^{1} \cdots \sum_{x_m=0}^{1} \phi(x_1, \ldots, x_m).$$

For our example where $m = 3$, this defines four functions:

$$\begin{aligned} f_3(x_1, x_2, x_3) &= \phi(x_1, x_2, x_3) \\ f_2(x_1, x_2) &= \textstyle\sum_{x_3=0}^{1} \phi(x_1, x_2, x_3) \\ f_1(x_1) &= \textstyle\sum_{x_2=0}^{1} \sum_{x_3=0}^{1} \phi(x_1, x_2, x_3) \\ f_0() &= \textstyle\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \sum_{x_3=0}^{1} \phi(x_1, x_2, x_3) \end{aligned}$$

Notice that we have the recurrence $f_i(x_1, \ldots, x_i) = f_{i+1}(x_1, \ldots, x_i, 0) + f_{i+1}(x_1, \ldots, x_i, 1)$.

I will now present a potential verification algorithm for *COUNTSAT* with this example that doesn't quite work. Suppose we have the following interaction between the prover and the verifier:

- P sends $f_0() = 6$.

- V checks that 6 is the answer claimed in the string w.
- P sends $f_1(0) = 3, \quad f_1(1) = 3.$
- V checks the recurrence that
$$\begin{array}{ccccc} f_1(0) & + & f_1(1) & = & f_0(). \\ 3 & + & 3 & = & 6. \end{array}$$
- P sends
$$\begin{array}{ll} f_2(0,0) = 2, & f_2(0,1) = 1. \\ f_2(1,0) = 1, & f_2(1,1) = 2. \end{array}$$
- V checks that
$$\begin{array}{ccccc} f_2(0,0) & + & f_2(0,1) & = & f_1(0). \\ 2 & + & 1 & = & 3. \\ f_2(1,0) & + & f_2(1,1) & = & f_1(1). \\ 1 & + & 2 & = & 3. \end{array}$$
- P sends
$$\begin{array}{ll} f_3(0,0,0) = 1, & f_3(0,0,1) = 1. \\ f_3(0,1,0) = 0, & f_3(0,1,1) = 1. \\ f_3(1,0,0) = 0, & f_3(1,0,1) = 1. \\ f_3(1,1,0) = 1, & f_3(1,1,1) = 1. \end{array}$$
- V checks that
$$\begin{array}{ccccc} f_3(0,0,0) & + & f_3(0,0,1) & = & f_2(0,0). \\ 1 & + & 1 & = & 2. \\ f_3(0,1,0) & + & f_3(0,1,1) & = & f_2(0,1). \\ 0 & + & 1 & = & 1. \\ f_3(1,0,0) & + & f_3(1,0,1) & = & f_2(1,0). \\ 0 & + & 1 & = & 1. \\ f_3(1,1,0) & + & f_3(1,1,1) & = & f_2(1,1). \\ 1 & + & 1 & = & 2. \end{array}$$
- V checks ϕ on all (eight in our example) possible values of f_m.

This clearly doesn't work since it is exponential, but it exemplifies the strategy we will use, so we will analyze it anyway. Suppose $w \in COUNTSAT$. Then if P simply tells the truth, P will convince V.

Now suppose that $w \notin COUNTSAT$, and a prover $\tilde{P}$ is trying to convince V otherwise. For example, suppose $w = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3), 5$. Then $\tilde{P}$ will have to lie at the first step and say $f_0() = 5$. It could then proceed by lying that $f_1(0) = 2$ and $f_1(1) = 3$ so that it satisfies the recurrence $f_1(0) + f_1(1) = f_0()$. Similarly, it could lie about values for f_2 and f_3. It would be forced to continue lying, but it wouldn't get caught until the end when V discovers that ϕ doesn't match f_3.

There is no reason to restrict the x_i to the values 0 and 1. Without changing the value of $\phi(x_1, x_2, x_3)$ on boolean values, we may arithmetize the expression by the following rules:

$$\begin{array}{rcl} \alpha \wedge \beta & = & \alpha \cdot \beta, \\ \neg\alpha & = & (1 - \alpha), \\ \alpha \vee \beta & = & 1 - (1 - \alpha)(1 - \beta). \end{array}$$

Notice that there is a one-to-one correspondence between the variables on the left hand side of the equality and the right hand side of the equality. Therefore, we may arithmetize a boolean formula in polynomial time. Our example arithmetizes to

$$\begin{array}{l} (1 - (1 - (1 - (1 - (1 - x_1))(1 - x_2)))(1 - x_3)) \\ \cdot \quad (1 - (1 - (1 - (1 - x_1)(1 - (1 - x_2))))(1 - x_3)) \end{array}.$$

Now we can evaluate ϕ on integer values. The maximum number of satisfying assignments is 2^m. If we choose a prime modulus p larger than this, then we can sum ϕ modulo p over all boolean values, and the resulting sum will still be the exact number of satisfying assignments. To simplify future analysis, $m < n$, so we will work modulo a prime slightly larger than 2^n.

In order to work modulo a prime larger than 2^n we need to be able to find such a number and verify that it's prime. It's conceivable that there might be huge gaps between primes and that the size of the next prime after 2^n is not always polynomial in n. Fortunately this is not the case.

By the theorems proven earlier, P can easily find a prime p between 2^n and 2^{3n} and send it to V. The binary representation of p will have no more than $3n$ digits, so V can perform standard modular arithmetic modulo p in polynomial time. Recall that $PRIMES \in NP$, so P can also send a certificate so V can verify primality.

Working modulo p, we can modify the current $COUNTSAT$ algorithm to probabilistically check polynomials modulo p to verify the result.

Suppose $w = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3), 6$. Then depending on our representation, $n \approx 25$, so we wish to find a prime p such that $2^{25} < p < 2^{75}$, or

$$33554432 < p < 37778931862957161709568.$$

We don't have to look far to see that, $p = 33554467$ suffices. The verification procedure would proceed as follows:

- P calculates that $n = 25$, finds the prime $p = 33554467$, and sends it to V with a primality certificate. P also calculates $f_0() = 6$ and sends it to V.
- V checks that $p = 33554467$ is prime, $2^{25} < p < 2^{75}$, and that 6 is in fact the answer given in w.
- P sends $f_1(x)$ as a polynomial modulo p in x. In this case, $f_1(x) \equiv 3 \pmod p$, a constant polynomial. (In general this will be a higher degree polynomial in x.)
- V confirms that

$$\begin{array}{ccccccc} f_1(0) & + & f_1(1) & \equiv & f_0() & \pmod p. \\ 3 & + & 3 & \equiv & 6 & \pmod p. \end{array}$$

  It then selects at random $r_1 \equiv 20458155 \pmod p$ and sends this to P.
- P calculates $f_2(r_1, x)$ as a polynomial modulo p in x and sends it to V. In this case, $f_2(r_1, x) \equiv 11462892x^2 + 29453417x + 13096314 \pmod p$.
- V confirms that

$$\begin{array}{ccccccc} f_2(r_1,0) & + & f_2(r_1,1) & \equiv & f_1(r_1) & \pmod p. \\ 13096314 & + & 20458156 & \equiv & 3 & \pmod p. \end{array}$$

  Then V randomly chooses $r_2 \equiv 30623655 \pmod p$ and sends it to P.
- P calculates $f_3(r_1, r_2, x) \equiv 16373423x^2 + 2771140x + 14409905 \pmod p$ and sends it to V.
- V confirms that

$$\begin{array}{ccccccc} f_3(r_1,r_2,0) & + & f_3(r_1,r_2,1) & \equiv & f_2(r_1,r_2) & \pmod p. \\ 14409905 & + & 1 & \equiv & 14409906 & \pmod p. \end{array}$$

- V selects $r_3 \equiv 11829775 \pmod{p}$. Then V checks that

$$\begin{array}{rcl} f_3(r_1, r_2, r_3) & \equiv & \phi(r_1, r_2, r_3) \pmod{p}. \\ 20783835 & \equiv & 20783835 \pmod{p}. \end{array}$$

  V accepts if all the checks have passed and rejects otherwise.

The exact rationale of this algorithm will not become clear until we conduct the worst-case analysis. The basic idea is that V cannot compute the f_i, so P computes them instead, and V checks a random value to make sure that P is telling the truth. Recall that $f_0()$ is the answer we want to verify and $f_m(x_1, \ldots, x_m)$ is simply $\phi(x_1, \ldots, x_m)$. As i gets larger, $f_i(x_1, \ldots, x_i)$ becomes easier to compute. V can compute $\phi(r_1, \ldots, r_m)$ in polynomial time, so that's what this algorithm works towards.

Clearly $w \in COUNTSAT \Rightarrow \Pr[V \leftrightarrow P \text{ accepts } w] = 1$. As long as P performs the appropriate computations as described above, V will always accept. It remains to show that $\forall \tilde{P} : w \notin COUNTSAT \Rightarrow \Pr\left[V \leftrightarrow \tilde{P} \text{ accepts } w\right] \leq \frac{1}{3}$.

To prove the latter statement, must now take an adversarial approach and assume that $\tilde{P}$ is attempting to trick V into accepting a string with the wrong count. Suppose we were instead given the string $w = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3), 5$, a string not in $COUNTSAT$ since that formula has six satisfying assignments, not five. An interaction with a clever deceitful prover $\tilde{P}$ might proceed as follows:

- $\tilde{P}$ calculates that $n = 25$, finds the prime $p = 33554467$, and sends it to V with a primality certificate. $\tilde{P}$ also lies that $\tilde{f}_0() = 5$.
- V checks that $p = 33554467$ is prime, $p > 2^{25}$, and that 5 is in fact the answer given in w.
- $\tilde{P}$ must devise a polynomial $\tilde{f}_1(x)$ to send to $\tilde{P}$. To prevent V from rejecting, $\tilde{f}_1(x)$ must satisfy $\tilde{f}_1(0) + \tilde{f}_1(1) \equiv \tilde{f}_0() \pmod{p}$. Suppose that $\tilde{P}$ arbitrarily makes up the polynomial $\tilde{f}_1(x) = x + 2$.
- V confirms that

$$\begin{array}{rcrcl} \tilde{f}_1(0) & + & \tilde{f}_1(1) & \equiv & \tilde{f}_0() \pmod{p}. \\ 2 & + & 3 & \equiv & 5 \pmod{p}. \end{array}$$

  It then selects at random $r_1 \equiv 20458155 \pmod{p}$ and sends this to $\tilde{P}$.
- $\tilde{P}$ must devise a polynomial $\tilde{f}_2(x)$ to send to $\tilde{P}$. To prevent V from rejecting, $\tilde{f}_2(x)$ must satisfy $\tilde{f}_2(0) + \tilde{f}_2(1) \equiv \tilde{f}_1(r_1) \pmod{p}$. Suppose that $\tilde{P}$ sends $\tilde{f}_2(x) = x + 10229077$
- V confirms that

$$\begin{array}{rcrcl} \tilde{f}_2(0) & + & \tilde{f}_2(1) & \equiv & \tilde{f}_1(r_1) \pmod{p}. \\ 10229078 & + & 10229079 & \equiv & 20458157 \pmod{p}. \end{array}$$

  Then V randomly chooses $r_2 \equiv 30623655 \pmod{p}$ and sends it to P.
- $\tilde{P}$ must devise a polynomial $\tilde{f}_3(x)$ to send to $\tilde{P}$. This polynomial $\tilde{f}_3(x)$ must satisfy $\tilde{f}_3(0) + \tilde{f}_3(1) \equiv \tilde{f}_2(r_2) \pmod{p}$. Since on the final step, V will check this polynomial against $\phi(r_1, r_2, x) \equiv 16373423x^2 + 2771140x + 14409905 \pmod{p}$ on a random x, it will be desirable to send this exact polynomial for $\tilde{f}_3(x)$. Unfortunately for $\tilde{P}$,

$\phi(r_1, r_2, 0) + \phi(r_1, r_2, 1) \equiv 20783835 \not\equiv \tilde{f}_2(r_2) \equiv 7298265 \pmod{p}$, so $\tilde{P}$ cannot send $\phi(r_1, r_2, x)$ or it will be immediately rejected. Suppose that instead $\tilde{P}$ sends $\tilde{f}_3(x) = x + 3649132$.

- V confirms that
$$\begin{array}{rcrcl} \tilde{f}_3(0) & + & \tilde{f}_3(1) & \equiv & \tilde{f}_2(r_2) \pmod{p}. \\ 3649132 & + & 3649133 & \equiv & 7298265 \pmod{p}. \end{array}$$
- V selects $r_3 \equiv 11829775 \pmod{p}$. Then V checks that
$$\begin{array}{rcl} \tilde{f}_3(r_3) & \equiv & \phi(r_1, r_2, r_3) \pmod{p}. \\ 15478907 & \not\equiv & 20783835 \pmod{p}. \end{array}$$
V has caught $\tilde{P}$ in a lie!

In the last step, V chose a random number r_3 and checked $\tilde{f}_3(x)$ on $x = r_3$ to make sure it was $\phi(r_1, r_2, r_3)$. Recall that $\tilde{f}_3(x)$ is the polynomial $\tilde{P}$ claims to be $f_3(r_1, r_2, x) \equiv \phi(r_1, r_2, x) \pmod{p}$. If $\tilde{P}$ is so clever that it is actually able to send $\tilde{f}_3(x) \equiv f_3(r_1, r_2, x) \equiv \phi(r_1, r_2, x) \pmod{p}$, then V will always pass the final check. However if $\tilde{f}_3(x) \not\equiv f_3(r_1, r_2, x) \pmod{p}$, for how many values of x does $\tilde{f}_3(x) \equiv f_3(r_1, r_2, x) \pmod{p}$? This depends on the degrees of these two polynomials.

In general, V will choose a random number r_m and check that $\tilde{f}_m(r_m) \equiv \phi(r_1, \ldots, r_m) \pmod{p}$. In any single variable, the degree of ϕ, and hence f_i, is at most one more than the number of multiplications in the arithmetization of ϕ. There are less than n multiplications in ϕ, so therefore all polynomials sent by P should have degree at most n. In fact for our earlier example, the degree of each polynomial never exceeded two. We should modify V to reject any polynomials of degree greater than n.

Since the degrees of $\tilde{f}_m(x)$ and $\phi(r_1, \ldots, r_{m-1}, x)$ are at most n, if they are not the same polynomial, then they agree for at most n values. The probability that these two polynomials agree on a random number is at most $\frac{n}{p} < \frac{n}{2^n}$. Hence

$$\begin{array}{rl} & \Pr\left[V \leftrightarrow \tilde{P} \text{ accepts on the last step}\right] \\ = & \Pr\left[\tilde{f}_m(r) \equiv \phi(r_1, \ldots, r_{m-1}, r) \pmod{p} \text{ for a random } r\right] \\ \leq & \Pr\left[\tilde{f}_m(x) \equiv \phi(r_1, \ldots, r_{m-1}, x) \pmod{p}\right] + \frac{n}{2^n} \Pr\left[\tilde{f}_m(x) \not\equiv \phi(r_1, \ldots, r_{m-1}, x) \pmod{p}\right] \\ = & 1 - \left(1 - \frac{n}{2^n}\right) \Pr\left[\tilde{f}_m(x) \not\equiv \phi(r_1, \ldots, r_{m-1}, x) \pmod{p}\right]. \end{array}$$

The deceitful prover wishes to maximize the probability of acceptance, or equivalently minimize the probability of rejection:

$$\Pr\left[V \leftrightarrow \tilde{P} \text{ rejects on the last step}\right] \geq \left(1 - \frac{n}{2^n}\right) \Pr\left[\tilde{f}_m(x) \not\equiv \phi(r_1, \ldots, r_{m-1}, x) \pmod{p}\right].$$

For any skilled deceitful prover, the last step is the only difficult step because it is the only point at which V checks what P says against ϕ. If V rejects before the last step, it is because P said something inconsistent. Therefore, without loss of generality, we may say that among any decent provers,

$$\Pr\left[V \leftrightarrow \tilde{P} \text{ rejects } w\right] = \Pr\left[V \leftrightarrow \tilde{P} \text{ rejects on the last step}\right].$$

6. $COUNTSAT \in IP$

Recall that to show $COUNTSAT \in IP$, we need to prove that

$$w \notin COUNTSAT \Rightarrow \Pr\left[V \leftrightarrow \tilde{P} \text{ rejects } w\right] \geq \frac{2}{3}.$$

We only need to show this for large n, since V is capable of deciding $COUNTSAT$ up to a finite n. Suppose that instead we show that there is a constant $c > \frac{2}{3}$ such that for all $\epsilon > 0$,

$$\Pr\left[V \leftrightarrow \tilde{P} \text{ rejects } w\right] > c - \epsilon$$

for large enough n. Then if we set

$$\epsilon = \frac{c - \frac{2}{3}}{2},$$

we get

$$\Pr\left[V \leftrightarrow \tilde{P} \text{ rejects } w\right] > \frac{c}{2} + \frac{1}{3} > \frac{2}{3}$$

for sufficiently large n, and hence $COUNTSAT \in IP$.

We denote the largest such c by

$$c = \liminf_{n\to\infty} \Pr\left[V \leftrightarrow \tilde{P} \text{ rejects } w\right].$$

Thus if we show

$$\liminf_{n\to\infty} \Pr\left[V \leftrightarrow \tilde{P} \text{ rejects } w\right] > \frac{2}{3},$$

we get that $COUNTSAT \in IP$.

In general, the $\liminf_{n\to\infty}$ operator is very similar to $\lim_{n\to\infty}$ except for some technicalities. For all convergent sequences they agree. In fact,

$$\begin{aligned}
& \liminf_{n\to\infty} \Pr\left[V \leftrightarrow \tilde{P} \text{ rejects } w\right] \\
\geq\ & \liminf_{n\to\infty} \left(1 - \tfrac{n}{2^n}\right) \Pr\left[\tilde{f}_m(x) \not\equiv \phi(r_1, \ldots, r_{m-1}, x) \pmod p\right] \\
\geq\ & \left(\lim_{n\to\infty} \left(1 - \tfrac{n}{2^n}\right)\right) \left(\liminf_{n\to\infty} \Pr\left[\tilde{f}_m(x) \not\equiv \phi(r_1, \ldots, r_{m-1}, x) \pmod p\right]\right) \\
=\ & \liminf_{n\to\infty} \Pr\left[\tilde{f}_m(x) \not\equiv \phi(r_1, \ldots, r_{m-1}, x) \pmod p\right].
\end{aligned}$$

Therefore, if we prove that

$$\liminf_{n\to\infty} \Pr\left[\tilde{f}_m(x) \not\equiv \phi(r_1, \ldots, r_{m-1}, x)\right] > \frac{2}{3},$$

we imply that $COUNTSAT \in IP$. I will prove that

$$\liminf_{n\to\infty} \Pr\left[\tilde{f}_m(x) \not\equiv f_m(r_1, \ldots, r_m, x) \equiv \phi(r_1, \ldots, r_{m-1}, x) \pmod p\right] = 1.$$

By assumption, $\tilde{f}_0() \not\equiv f_0() \pmod p$ since we assume $w \notin COUNTSAT$. Since

$$\tilde{f}_1(0) + \tilde{f}_1(1) \equiv \tilde{f}_0() \not\equiv f_0() \equiv f_1(0) + f_1(1) \pmod p,$$

we have that $\tilde{f}_1(x) \not\equiv f_1(x) \pmod{p}$, so

$$\Pr\left[\tilde{f}_1(x) \not\equiv f_1(x) \pmod{p}\right] = 1.$$

The deceitful prover $\tilde{P}$ begins with this lie and wants to end with the truth: $\tilde{f}_m(x) \equiv f_m(r_1, \ldots, r_m, x) \pmod{p}$. If $\tilde{P}$ is able to report the correct polynomial $\tilde{f}_i$ for any i, then $\tilde{P}$ can trick the verifier into accepting by reporting the correct $\tilde{f}_i$ for all subsequent i including the correct $\tilde{f}_m$. Thus if $\tilde{P}$ is able to move from a lie to the truth at any step, $\tilde{P}$ will succeed in deceiving V.

Suppose that $\tilde{P}$ has engineered $\tilde{f}_1(x) \not\equiv f_1(x) \pmod{p}$ in the most clever way possible to maximize the probability that it can send the correct $\tilde{f}_2(x)$. The $\tilde{f}_2(x)$ that $\tilde{P}$ sends must satisfy the constraint $\tilde{f}_2(0) + \tilde{f}_2(1) \equiv \tilde{f}_1(r_1) \pmod{p}$. The real $f_2(r_1, x)$ satisfies $f_2(r_1, 0) + f_2(r_1, 1) \equiv f_1(r_1) \pmod{p}$. Therefore, to be able to send the truthful $\tilde{f}_2(x)$, it must be that $\tilde{f}_1(r_1) \equiv f_1(r_1) \pmod{p}$. V chose the number r_1 after $\tilde{P}$ chose $\tilde{f}_1(x)$. Therefore, $\tilde{P}$ can send the correct $\tilde{f}_2(x)$ only if $\tilde{f}_1(x)$ agrees with $f_1(x)$ for a random x. Hence,

$$\Pr\left[\tilde{f}_2(x) \not\equiv f_2(r_1, x) \pmod{p}\right] \geq 1 - \frac{n}{2^n}.$$

More generally, for $i \geq 1$, if $\tilde{f}_i(x) \not\equiv f_i(r_1, \ldots, r_{i-1}, x) \pmod{p}$, then

$$\Pr\left[\tilde{f}_{i+1}(x) \not\equiv f_{i+1}(r_1, \ldots, r_i, x) \pmod{p}\right] \geq 1 - \frac{n}{2^n}.$$

By induction, for $i \geq 1$,

$$\Pr\left[\tilde{f}_i(x) \not\equiv f_i(r_1, \ldots, r_{i-1}, x) \pmod{p}\right] \geq \left(1 - \frac{n}{2^n}\right)^{i-1}.$$

In particular,

$$\Pr\left[\tilde{f}_m(x) \not\equiv f_m(r_1, \ldots, r_{m-1}, x) \pmod{p}\right] \geq \left(1 - \frac{n}{2^n}\right)^{m-1} \geq \left(1 - \frac{n}{2^n}\right)^{n-1}.$$

Therefore,

$$\liminf_{n\to\infty} \Pr\left[\tilde{f}_m(x) \not\equiv \phi(r_1, \ldots, r_{m-1}, x) \pmod{p}\right] \geq \lim_{n\to\infty} \left(1 - \frac{n}{2^n}\right)^{n-1} = 1$$

by L'Hôpital's Rule. Therefore, for sufficiently large n,

$$w \notin COUNTSAT \Rightarrow \Pr\left[V \leftrightarrow \tilde{P} \text{ rejects } w\right] \geq \frac{2}{3},$$

so $COUNTSAT \in IP$.

7. $TQBF \in IP$

We can adapt this proof that $COUNTSAT \in IP$ to prove that $TQBF$, the language of true fully-quantified boolean formulas, is in IP. Instead of counting the number of satisfying boolean assignments, we wish to have quantifiers. An example of a $TQBF$ string is $w = \forall x_1 \exists x_2 \forall x_3 (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$. We assume that the formula is in prenex normal form, that is the quantifiers appear at the front.

Recall that for $COUNTSAT$, the goal was to compute

$$\sum_{x_1=0}^{1}\sum_{x_2=0}^{1}\cdots\sum_{x_m=0}^{1}\phi(x_1,x_2,\ldots,x_m).$$

We can substitute $\prod_{x_i=0}^{1}$ for $\forall x_i$ and $\biguplus_{x_i=0}^{1}$ for $\exists x_i$, where we define

$$\biguplus_{x_i=0}^{1} f(x_i) := 1-(1-f(0))(1-f(1)).$$

Our example would become

$$\prod_{x_1=0}^{1}\biguplus_{x_2=0}^{1}\prod_{x_3=0}^{1}\phi(x_1,x_2,x_3).$$

With $COUNTSAT$ we had the recurrence

$$f_i(x_1,\ldots,x_i)=f_{i+1}(x_1,\ldots,x_i,0)+f_{i+1}(x_1,\ldots,x_i,1).$$

Here we have the recurrence

$$f_i(x_1,\ldots,x_i)=\begin{cases} f_{i+1}(x_1,\ldots,x_i,0)\cdot f_{i+1}(x_1,\ldots,x_i,1) & \text{if the } i+1\text{st quantifier is } \forall. \\ 1-(1-f_{i+1}(x_1,\ldots,x_i,0))(1-f_{i+1}(x_1,\ldots,x_i,1)) & \text{if the } i+1\text{st quantifier is } \exists. \end{cases}$$

The exact same algorithm carries over, only instead of checking the $COUNTSAT$ recurrence, V checks the $TQBF$ recurrence. Unfortunately, a slight complication arises.

Recall that $\phi(x_1,\ldots,x_m)$ has degree no more than n in any one variable. Adding two polynomials does not increase the degree, so for $COUNTSAT$, f_i has degree no more than n. In the $TQBF$ recurrence, the degree can double at each iteration. Therefore, the degree of the f_i can grow exponentially. This is not acceptable since V must receive these polynomials.

To remedy this situation, we introduce a new operator L called linearization. It is defined by

$$L_{x_i}f(x_i) := (1-x_i)f(0)+x_if(1).$$

Linearization reduces the degree in x_i to one without changing the boolean values. By linearizing each time before a quantifier is applied, the degree never exceeds two after the first quantifier is applied. The desired calculation for our example then becomes

$$\prod_{x_1=0}^{1}L_{x_1}\biguplus_{x_2=0}^{1}L_{x_1}L_{x_2}\prod_{x_3=0}^{1}L_{x_1}L_{x_2}L_{x_3}\phi(x_1,x_2,x_3).$$

As before, we define a new function for each quantifier:

$$\begin{aligned}
f_0() &:= \prod_{x_1=0}^{1} L_{x_1} \biguplus_{x_2=0}^{1} L_{x_1} L_{x_2} \prod_{x_3=0}^{1} L_{x_1} L_{x_2} L_{x_3} \phi(x_1, x_2, x_3). \\
f_1(x_1) &:= L_{x_1} \biguplus_{x_2=0}^{1} L_{x_1} L_{x_2} \prod_{x_3=0}^{1} L_{x_1} L_{x_2} L_{x_3} \phi(x_1, x_2, x_3). \\
f_2(x_1) &:= \biguplus_{x_2=0}^{1} L_{x_1} L_{x_2} \prod_{x_3=0}^{1} L_{x_1} L_{x_2} L_{x_3} \phi(x_1, x_2, x_3). \\
f_3(x_1, x_2) &:= L_{x_1} L_{x_2} \prod_{x_3=0}^{1} L_{x_1} L_{x_2} L_{x_3} \phi(x_1, x_2, x_3). \\
f_4(x_1, x_2) &:= L_{x_2} \prod_{x_3=0}^{1} L_{x_1} L_{x_2} L_{x_3} \phi(x_1, x_2, x_3). \\
f_5(x_1, x_2) &:= \prod_{x_3=0}^{1} L_{x_1} L_{x_2} L_{x_3} \phi(x_1, x_2, x_3). \\
f_6(x_1, x_2, x_3) &:= L_{x_1} L_{x_2} L_{x_3} \phi(x_1, x_2, x_3). \\
f_7(x_1, x_2, x_3) &:= L_{x_2} L_{x_3} \phi(x_1, x_2, x_3). \\
f_8(x_1, x_2, x_3) &:= L_{x_3} \phi(x_1, x_2, x_3). \\
f_9(x_1, x_2, x_3) &:= \phi(x_1, x_2, x_3).
\end{aligned}$$

We get a new recurrence for linearization, and each linearization requires its own verification step. The verification procedure then goes as follows:

- P calculates that $n = 32$, $2^n = 4294967296$, finds the prime $p = 4294967311$, and sends it to V with a primality certificate. P also calculates $f_0() = 1$ and sends it to V.
- V checks that $p = 4294967311$ is prime, $p > 2^{32}$, and that $f_0() = 1$ so that the formula is said to be true.
- P sends $f_1(x)$ as a polynomial modulo p in x. In this case, $f_1(x) \equiv 1 \pmod{p}$.
- V confirms that

$$\begin{array}{ccccc} f_1(0) & \cdot & f_1(1) & \equiv & f_0() \pmod{p}. \\ 1 & \cdot & 1 & \equiv & 1 \pmod{p}. \end{array}$$

  It then selects at random $r_1 \equiv 3009911989 \pmod{p}$ and sends this to P.
- P calculates $f_2(x)$ as a polynomial modulo p in x and sends it to V. In this case, $f_2(x) \equiv x^2 + 4294967310x + 1 \pmod{p}$.
- V confirms that

$$\begin{array}{ccccc} (1 - r_1) f_2(0) & + & r_1 f_2(1) & \equiv & f_1(r_1) \pmod{p}. \\ 1285055323 & + & 3009911989 & \equiv & 1 \pmod{p}. \end{array}$$

  Then V randomly chooses $r_2 \equiv 2557656319 \pmod{p}$ and sends it to P.
- P calculates $f_3(r_2, x) \equiv 820345326x + 1737310993 \pmod{p}$ and sends it to V.
- V confirms that

$$\begin{array}{ccc} 1 - (1 - f_3(r_2, 0))(1 - f_3(r_2, 1)) & \equiv & f_2(r_2) \pmod{p}. \\ 3295805320 & \equiv & 3295805320 \pmod{p}. \end{array}$$

  Then V randomly selects $r_3 \equiv 713657950 \pmod{p}$ and sends it to P.
- P calculates $f_4(x, r_3) \equiv 1427315899x + 3581309362 \pmod{p}$ and sends it to V.
- V confirms that

$$\begin{array}{ccccccc} (1 - r_2) f_4(0, r_3) & + & r_2 f_4(1, r_3) & \equiv & f_3(r_2, r_3) \pmod{p}. \\ 1517975188 & + & 494322145 & \equiv & 2012297333 \pmod{p}. \end{array}$$

  Then V randomly selects $r_4 \equiv 2957361682 \pmod{p}$ and sends it to P.
- P calculates $f_5(r_4, x) \equiv 1619756052x + 1337605630 \pmod{p}$ and sends it to P.

- V confirms that

$$\begin{array}{rcrcrl} (1-r_3)f_5(r_4,0) & + & r_3 f_5(r_4,1) & \equiv & f_4(r_4,r_3) & (\text{mod } p). \\ 2830992477 & + & 2207044797 & \equiv & 743069963 & (\text{mod } p). \end{array}$$

  Then V randomly selects $r_5 \equiv 6744292 \pmod{p}$ and sends it to P.
- P calculates $f_6(r_4, r_5, x) \equiv 2795513735x + 1499453577 \pmod{p}$ and sends it to P.
- V confirms that

$$\begin{array}{rcrcrl} f_6(r_4,r_5,0) & \cdot & f_6(r_4,r_5,1) & \equiv & f_5(r_4,r_5) & (\text{mod } p). \\ 1499453577 & \cdot & 4294967312 & \equiv & 1499453577 & (\text{mod } p). \end{array}$$

  Then V randomly selects $r_6 \equiv 2703152256 \pmod{p}$ and sends it to P.
- P calculates $f_7(x, r_5, r_6) \equiv 1936798735x + 383176761 \pmod{p}$ and sends it to P.
- V confirms that

$$\begin{array}{rcrcrl} (1-r_4)f_7(0,r_5,r_6) & + & r_4 f_7(1,r_5,r_6) & \equiv & f_6(r_4,r_5,r_6) & (\text{mod } p). \\ 1316525994 & + & 1601895337 & \equiv & 2918421331 & (\text{mod } p). \end{array}$$

  Then V randomly selects $r_7 \equiv 3503392733 \pmod{p}$ and sends it to P.
- P calculates $f_8(r_7, x, r_6) \equiv 2421660936x^2 + 1825837894x + 1375310369 \pmod{p}$ and sends it to P.
- V confirms that

$$\begin{array}{rcrcrl} (1-r_5)f_8(r_7,0,r_6) & + & r_5 f_8(r_7,1,r_6) & \equiv & f_7(r_7,r_5,r_6) & (\text{mod } p). \\ 3960328441 & + & 2981683416 & \equiv & 2647044546 & (\text{mod } p). \end{array}$$

  Then V randomly selects $r_8 \equiv 3913141309 \pmod{p}$ and sends it to P.
- P calculates $f_9(r_7, r_8, x) \equiv 3398965994x^2 + 4073427229x + 1117541400 \pmod{p}$ and sends it to P.
- V confirms that

$$\begin{array}{rcrcrl} (1-r_6)f_9(r_7,r_8,0) & + & r_6 f_9(r_7,r_8,1) & \equiv & f_8(r_7,r_8,r_6) & (\text{mod } p). \\ 14121323 & + & 1507924447 & \equiv & 2717273579 & (\text{mod } p). \end{array}$$

- V selects $r_9 \equiv 309850783 \pmod{p}$. Then V checks that

$$\begin{array}{rcrl} f_9(r_7,r_8,r_9) & \equiv & \phi(r_7,r_8,r_9) & (\text{mod } p). \\ 3822769522 & \equiv & 3822769522 & (\text{mod } p). \end{array}$$

  V accepts if all the checks have passed and rejects otherwise.

Analysis of this algorithm is virtually the same as for $COUNTSAT$. If $w \in COUNTSAT$, V will always accept. If $w \notin COUNTSAT$, $\tilde{f}_1(x)$ will always be a lie. The criteria for being able to tell the truth for each subsequent $\tilde{f}_i(x)$ given by $\tilde{P}$ is that $\tilde{f}_{i-1}(x)$ agrees with the true $f_{i-1}(\dots x \dots)$ for a random x. Therefore, there is at most an $\frac{n}{2^n}$ probability that it can start telling the truth on that step. The only real difference is that there are $\frac{m^2+3m}{2} < n^2$ f_i instead of m. Therefore, we get the limit

$$\lim_{n\to\infty} \left(1 - \frac{n}{2^n}\right)^{n^2} = 1$$

again by L'Hôpital's Rule. Therefore, $IP = PSPACE$.

8. Concluding Remarks

There are many subtleties in the proof of $IP = PSPACE$ that most texts gloss over. I mainly followed Sipser's approach because his seemed the most polished. Unfortunately, his was also the most lacking in details. Hopefully I successfully filled them all in and acknowledged all the subtleties.

One subtlety I might have missed is the requirement that the boolean formulas be in CNF. Both Sipser and Goldreich required this, yet they did not explain where in their proofs it was necessary. I see no reason why this should matter.

9. References

- Goldreich, Oded. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.*
- Papadimitriou, Christos H. *Computational Complexity.*
- Sipser, Michael. *Introduction to the Theory of Computation.*