UCT

Interactive Proofs

Klaus Sutner

Carnegie Mellon University

Spring 2024

1 **Interactive Proofs**

2 IP $\subseteq$ **PSPACE**

3 **Warmup Exercise**

4 $\ast$ **PSPACE** $\subseteq$ IP

Where Are We?

We have a number of examples of randomized algorithms, including critically important ones such as primality testing.

There are fairly natural probabilistic complexity classes that capture these algorithms.

The probabilistic classes fit nicely into that the standard complexity landscape.

Are there are other interesting randomized classes?

Recall: Guess & Verify

One popular model of $\mathbb{NP}$ is to break up the computation into two phases. On input x:

Guess the required witness w, then

Verify that w works for x.

Of course, we are really dealing with projections and nondeterministic Turing machines. Still, for intuition the guess & verify model is perfect.

Wild Idea: How about trying to understand the witnesses in greater detail? Just saying "it exists" is a bit feeble.

Witnesses as Proofs

Witnesses may carry a lot of information, it is not much of a stretch to think of them as a proof of the fact that some instance is a Yes-instance.

For example, a witness of SAT is a satisfying truth assignment α. Given α and the formula $\Phi(\boldsymbol{x})$, one can simply evaluate the formula in polynomial time and show that its value is indeed true. The computation constitutes the needed proof.

More precisely, it might be useful to think of a witness w as the core of a **proof** of the assertion "$x \in L$". The actual proof can then be trivially constructed in polynomial time; in particular the proof has to have polynomial length.

It is important that we don't really care how intrinsically complicated the core of the proof is: it might take a huge effort to find it and the reasoning may be enormously clever, but checking its correctness must be fairly simple.

Pratt's Theorem

If this sounds too weak, recall Pratt's result that primality is in $\mathbb{NP}$. The following two tables literally give a proof of the primality of 733, given some background results in basic number theory.

$n = 733$
$n - 1 = 2^2 \cdot 3 \cdot 61,\ g = 6$
$6^{732} \equiv 1 \pmod{733}$
$6^{366} \equiv -1 \pmod{733}$
$6^{244} \equiv 425 \pmod{733}$
$6^{12} \equiv 299 \pmod{733}$

$n = 61$
$n - 1 = 2^2 \cdot 3 \cdot 5,\ g = 2$
$2^{60} \equiv 1 \pmod{61}$
$2^{30} \equiv -1 \pmod{61}$
$6^{20} \equiv 47 \pmod{61}$
$6^{12} \equiv 9 \pmod{61}$

Lonely Verifier

In the usual math scenario, someone spends a lot of effort to build a proof, submits it, it's peer-reviewed, published, and then anyone interested can read the proof and verify without too much effort that it is correct[†].

In principle, the verification should not take much more then linear time: if we insist that the proof is presented is a strict Hilbert-style system, checking correctness comes down to so much wordprocessing.

OK, but in the RealWorld$^{\text{TM}}$ proofs are not formal (though a few have indeed been formalized, a horribly labor-intensive process). Instead, the reader has to work a bit harder, find typos, resolve ambiguities, fill in gaps, and so on.

For this process of understanding and verification it is immeasurably useful for the reader to be able to pose questions, to the author or some other authority.

[†]If only. A lot of published material is quite poorly presented. And, some of it is plain wrong.

So?

Burning Question: Can we get more mileage out of the idea of some kind of protocol between a “prover” and a “verifier”?

The idea is that the verifier can ask a sequence of questions, and the prover provides the requested answers.

In the end, if the verifier is convinced of the correctness of the claim, they accept; otherwise they reject.

The verifier is limited in compute power, say, to polynomial time. The prover, however, can use arbitrarily much compute power.

Prover and Verifier

So, we have two entities communicating with each other:

Prover	answers requests for evidence by the verifier;
Verifier	checks the information provided by the prover, and can request more input.

After multiple question/answer rounds, the verifier announces a decision.

If the given instance is a Yes-instance, the prover should have a way to convince the verifier by sending back the right answers.

On the other hand, if we have a No-instance, then nothing the prover does should fool the verifier into acceptance.

More Formally . . .

Suppose we have two functions $f, g : \mathbf{2}^\star \to \mathbf{2}^\star$ and $k \geq 0$. Define the k-round interaction btw f and g on input $x \in \mathbf{2}^\star$ as a sequence of strings $a_i \in \mathbf{2}^\star$:

$$\begin{array}{lr@{\;}l}
\mathcal{V}: & a_1 &= f(x) \\
\mathcal{P}: & a_2 &= g(x, a_1) \\
\mathcal{V}: & a_3 &= f(x, a_1, a_2) \\
 & \vdots & \\
\mathcal{V}: & a_{2i+1} &= f(x, a_1, a_2, \ldots, a_{2i}) \\
\mathcal{P}: & a_{2i+2} &= g(x, a_1, a_2, \ldots, a_{2i+1}) \\
 & \vdots & \\
\mathcal{V}: & a_k &= f(x, a_1, a_2, \ldots, a_{k-1})
\end{array}$$

We call $(a_1, a_2, \ldots, a_k) \in (\mathbf{2}^\star)^k$ the transcript of the interaction.

Output and Step 1

The output is $\text{res}(\mathcal{V}, \mathcal{P})(x) = a_k$, which we assume to be a single bit (so k must be odd), the yes-or-no answer.

Objection:

The prover has unlimited power, so it can easily compute $a_1 = f(x)$. So would it not make more sense to start with the prover computing a_2?

Yes, but later we will add randomness to the mix, and then the prover cannot determine a_1, it needs the verifier to speak first.

Completeness and Soundness

A language L has a k-round deterministic interactive proof system (dIP) if there is a polynomial time Turing machine $\mathcal{V}$ that runs on inputs $x, a_1, \ldots, a_i \in (\mathbf{2}^\star)^{i+1}$ and has a k-round, k polynomially bounded by $|x|$, interaction with a similar prover function $\mathcal{P}$ such that

Completeness $x \in L$ implies $\mathsf{res}(\mathcal{V}, \mathcal{P})(x) = 1$

Soundness $x \notin L$ implies $\mathsf{res}(\mathcal{V}, \widetilde{\mathcal{P}})(x) = 0$ for any $\widetilde{\mathcal{P}}$

Note the universal quantifier in soundness: the verifier cannot be fooled by any malicious prover $\widetilde{\mathcal{P}}$, no matter how powerful or devious.

Comment: It would also work to have a good prover for every Yes-instance, rather than one universal good prover for all Yes-instances (though the uniform prover seems a lot more natural).

Proof Systems

Again, this is really very similar to the problem of trying to construct proofs in some nice formal systems (Hilbert's doomed dream).

Given some assertion Φ,

- if Φ is true, then there should be a proof of Φ in our system;
- if Φ is false, then any attempt to cobble together a proof will fail.

In our setting, the proof for a true statement cannot be too complicated or long, it must be easy to check for correctness by a polynomial verifier.

The attempt to prove a false statement may be enormously complicated (ask any quack about squaring the circle), but in the end it will always be flawed. In principle, it is easy to check a proof for correctness (true for formal proofs, quite wrong for the usual informal arguments).

Arbitrary Compute Power?

One might feel somewhat uneasy about the idea that the prover can have arbitrary power, can be omniscient. For examples, the prover might be able to use the Halting set as an oracle.

True, but recall that the verifier has to work in polynomial time. As a consequence, there is no way to

- read exponentially long messages from the prover, and
- it becomes easily impossible to verify alleged claims by the prover.

Still worried? In the end it turns out that only provers in PSPACE are relevant. A polynomially stupid verifier ruins even the Halting oracle[†].

[†] Against stupidity the gods themselves contend in vain.
F. von Schiller, *The Maid of Orleans*, 1801. Nothing has changed in 200 years.

Oracle Prover

Suppose the prover has access to the Halting Set, and we would like to exploit this to define an interaction to answer hugely difficult questions about the Busy Beaver function.

What is the parity of BB(n)?

Here goes: on input n, the protocol could look like so:

$\mathcal{V}$: says hello

$\mathcal{P}$: computes $\alpha =$ BB(n), sends $a = \alpha \bmod 2$

$\mathcal{V}$: asks for the corresponding busy beaver

$\mathcal{P}$: sends back $\mathcal{M}$

$\mathcal{V}$: why is this the busy beaver?

$\mathcal{P}$: you'll never understand ...

Total Failure

The verifier has life span $O(\log^c n)$, so it cannot even read the transition table for $\mathcal{M}$.

Even if they could, there is no way to actually run the machine to completion.

Even if that worked, there is no way to ensure that $\mathcal{M}$ really is the busy beaver for n.

The prover can cheat to his heart's content.

Exercise

Convince yourself that there is no easy fix to this problem.
Then show that the closely related Halting problem is indeed not in dIP.

Lipstick an a Pig

Claim: k-round deterministic interactive proof systems produce exactly $\mathbb{NP}$.

It is clear that $\mathbb{NP}$ can be expressed in terms of a dIP: all the verifier needs from the prover is a witness.

$\mathcal{V}$: $a_1 = f(x) =$ "howarya"

$\mathcal{P}$: $a_2 = g(x, a_1)$ where a_2 is a witness for instance x, 0 otherwise.

$\mathcal{V}$: Checks if a_2 really is a witness, accepts/rejects accordingly.

Three rounds are plenty.

Opposite Direction

For the opposite direction, assume we have a dIP.

Case 1: $x \in L$.

Let $(a_1, \dots, a_k)$ be the transcript of the interaction with the good prover $\mathcal{P}$, as per completeness. We think of the transcript as a witness that we can guess, and we can verify in polynomial time that $\mathcal{V}(x, a_1, \dots, a_{2i}) = a_{2i+1}$.

Case 2: $x \notin L$.

We can define a prover arbitrarily by $\widetilde{\mathcal{P}}(x, a_1, \dots, a_{2i+1}) = a_{2i+2}$ where a_{2i+2} is some polynomial length string, say, "Imastablegenius". By soundness, the verifier must reject.

Probabilistic Verifiers

To get real mileage out of the idea of interaction, we need another ingredient: randomness. We allow the prover and verifier to be probabilistic with performance guarantees.

Completeness $x \in L$ implies $\Pr[\text{res}(\mathcal{V}, \mathcal{P})(x) = 1] \geq 2/3$

Soundness $x \notin L$ implies $\Pr[\text{res}(\mathcal{V}, \widetilde{\mathcal{P}})(x) = 1] \leq 1/3$ for all $\widetilde{\mathcal{P}}$

This class is called $\text{IP}[k]$, k-round interactive proofs.

Set

$$\text{IP} = \bigcup \text{IP}[n^c],$$

polynomial interactive proofs. So we allow the number of interactions to depend on the size of the input.

So What?

We know $\mathbb{NP} \subseteq \mathrm{IP}$, but it is not so clear where in the complexity landscape IP lives. For example, what is the relationship between IP and the polynomial hierarchy?

In fact, one might be slightly pessimistic about this whole project:

- If indeed $\mathbb{P} = \mathrm{BPP}$, then randomness does not help much.
- Allowing for, say, quadratically many rounds may sound impressive, but it's not so clear how to exploit this ability. Many natural protocols have a finite number of rounds.

As we will see, things work out just fine, but that was a bit of a surprise in the 1990s.

Amplifying Truth

As usual, the constants $2/3$ and $1/3$ in the definition of IP matter little, one can apply the same boosting techniques used in BPP (Chernoff bounds).

One can even get the completeness bound $2/3$ up to 1, but that is hard. By contrast, lowering the soundness bound $1/3$ to 0 pushes us into $\mathbb{NP}$.

Exercise

Show how to replace the constants by $1 - 2^{-n^d}$ *and* 2^{-n^d}*, respectively.*

IP **and Graph-Non-Isomorphism**

Recall the old chestnut: given two ugraphs, determine whether they are isomorphic. Here we want to use the negation.

Problem:	**Graph Non-Isomorphism (GNI)**
Instance:	Two ugraphs G_1 and G_2.
Question:	Are G_1 and G_2 non-isomorphic?

The Graph Isomorphism problem is clearly in $\mathbb{NP}$, and currently neither known to be in $\mathbb{P}$ nor known to be $\mathbb{NP}$-complete. On the other hand, GNI is in co-$\mathbb{NP}$.

Lemma

Graph Non-Isomorphism is in IP.

Proof

Here is the protocol, presented informally:

$\mathcal{V}$: Pick $i \in \{1, 2\}$ uniformly at random.
Randomly vertex-permute G_i to produce H; send H to $\mathcal{P}$.

$\mathcal{P}$: Check which of G_1 or G_2 produced H, say, G_j; send j to $\mathcal{V}$.

$\mathcal{V}$: Accept if $i = j$, reject otherwise.

This works as advertised: if the graphs are non-isomorphic, the honest prover can search through all permutations and always produce the correct answer. But if they are isomorphic, even the most powerful prover can only flip a coin.

Note that it is critical here that the prover does not know the random bit i.

Also note the shortness of the "proof": $\mathcal{P}$ just sends a single bit.

IP and Quadratic Non-Residues

Recall from our discussion of Solovay-Strassen:

$\mathbb{Z}_m^\star$ is the multiplicative subgroup of $\mathbb{Z}_m$. A modular number $a \in \mathbb{Z}_m^\star$ is a quadratic residue (mod m) if $a = x^2 \pmod{m}$ for some x, and a quadratic non-residue otherwise.

In particular when $m = p$ is an odd prime, there is an easy characterization of the quadratic residues. Let g be a generator of $\mathbb{Z}_m^\star = \{1, 2, \ldots, m-1\}$. Then the residues are all the even powers of g (which is why we can compute the Legendre symbol nicely by exponentiation):

$$g^2, g^4, \ldots, g^{m-3}, g^{m-1}$$

For example, for $m = 11$ and $g = 2$ we get quadratic residues $4, 5, 9, 3, 1$ with "square roots" $2, 4, 3, 5, 1$.

Protocol

To check whether $a \in \mathbb{Z}_p^\star$ fails to be a quadratic residue, do the following:

$\mathcal{V}$: Pick $b \in \mathbf{2}$ and $r \in \{1, 2, \ldots, p-1\}$ uniformly at random.
If $b = 0$, send $r^2 \pmod{p}$, else send $ar^2 \pmod{p}$.

$\mathcal{P}$: Try to determine b, send b' accordingly.

$\mathcal{V}$: Accept if $b = b'$, reject otherwise.

Correctness

Soundness Whenever a is a quadratic residue, both $r^2 \pmod m$ and $ar^2 \pmod m$ vary uniformly over all quadratic residues, so the prover has no way of distinguishing between the two cases; essentially, it can do no better than to flip a coin—huge compute power is of no use.

Completeness On the other hand, if a is a non-residue, then $r^2 \pmod m$ and $ar^2 \pmod m$ are two disjoint distributions and it is trivial for the obvious honest prover to determine the actual b.

1 **Interactive Proofs**

2 IP $\subseteq$ **PSPACE**

3 **Warmup Exercise**

4 ∗ **PSPACE** $\subseteq$ IP

Connections

It is clear that IP contains $\mathbb{NP}$ and BPP.

Graph Non-Isomorphism provides an example of a problem in IP not known to be in $\mathbb{NP}$ or BPP.

So the question still is: where in the classical complexity landscape does IP fit, if at all? The surprising answer is this:

Theorem (Shamir, Lund-Fortnow-Karloff-Nisan 1990)

IP $= \mathrm{PSPACE}$.

As it turns out, both directions require effort, and the second one is a bit of a nightmare.

Handling Random Bits

There is a technical subtlety wrto the random bits used in the protocol. In our model, the random bits of the verifier must be kept secret at all costs (the random bits of the prover are less important).

One often indicates this by writing the verifier function with an additional argument r representing a vector of random bits. The prover does **not** have access to these bits:

$$\mathcal{V}: \qquad a_{2i+1} = f(x, r, a_1, a_2, \ldots, a_{2i})$$

This is referred to as the private coin model. There is also a public coin model where the random bits are shared.

The Problem

We are supposed to show that IP $\subseteq \mathrm{PSPACE}$.

Since the number of bits involved in the protocol is polynomially bounded, a PSPACE machine can perform an explicit search over all possibilities, and certainly check the verifier's work.

But there is a glitch (a potentially fatal one): the prover may well be far outside of PSPACE, just think about a universal Turing machine. Any direct simulation is bound to fail.

A Workaround

Instead of attempting a direct simulation, we compute the maximum probability with which a prover can persuade the verifier to accept.

Think of a tree that describes all possible sequences of messages a_i sent during execution of the protocol (partial transcripts). The tree has polynomial depth and branching factor of at most $2^{O(n^c)}$.

We label the nodes in the tree by the appropriate probabilities, starting at the leaves and working backwards to the root. If the value at the root is at least $2/3$, we accept and reject otherwise.

While the tree itself cannot be built in PSPACE, the labeling algorithm can be implemented in PSPACE.

Details

We need to take a closer look partial transcripts $(a_1, a_2, \ldots, a_k) \in (\mathbf{2}^\star)^k$.

To this end, let $\boldsymbol{t} = (a_1, a_2, \ldots, a_j)$. We are interested in extensions to a full transcript such that for all i, $0 \leq i < k$:

$$\begin{aligned} a_{i+1} &= f(x, r, a_1, \ldots, a_i) \qquad & i \text{ even} \\ a_{i+1} &= g(x, a_1, \ldots, a_i) \qquad & i \text{ odd} \\ a_k &= \text{Yes} \end{aligned}$$

Let's say that a partial transcript $\boldsymbol{t}$ is extensible if there exists a full extension $\boldsymbol{t}' = \boldsymbol{t} \oplus (a_{j+1}, \ldots, a_k)$ with these properties. Here $\oplus$ means join the two lists. Note that $\boldsymbol{t}$ extensible ensures that $\boldsymbol{t}$ conforms to the protocol.

Acceptance Values

Next we define a rational number $N(\boldsymbol{t})$, the acceptance value of $\boldsymbol{t}$, for any partial transcript $\boldsymbol{t}$ as follows.

Case $|\boldsymbol{t}| = k$:

Then $N(\boldsymbol{t}) = 1$ if t is consistent with some random string r and accepting; 0 otherwise.

Case $|\boldsymbol{t}| = i < k$:

$$N(\boldsymbol{t}) = \begin{cases} \mathrm{avrg}(\, N(\boldsymbol{t} \oplus a) \mid a\,) & \text{if } i \text{ even,} \\ \max\big(\, N(\boldsymbol{t} \oplus a) \mid a\,\big) & \text{if } i \text{ odd.} \end{cases}$$

The weighted average in the first case is supposed to be

$$\sum_a \Pr_r[f(x, r, \boldsymbol{t}) = a] \cdot N(\boldsymbol{t} \oplus a)$$

Quoi?

The purpose of this definition is the following.

First, we can verify that the whole computation of the $N(\boldsymbol{t})$ is in polynomial space (though presumably not polynomial time): the strings r and a are polynomial bounded in length, so in polynomial space we can try them all and simply count the good outcomes.

Second, the reason we care about $N(\boldsymbol{t})$ is this: the probability that the verifier accepts instance x is $\Pr[\mathcal{V} \text{ accepts } x] = \max\big(\Pr[(\mathcal{V},\mathcal{P}) \text{ accepts } x] \mid \mathcal{P}\,\big)$.

Proposition

$N(\text{nil}) = \Pr[\mathcal{V}$ *accepts* $x]$

Step

We use (backwards) induction to show $N(\boldsymbol{t}) = \Pr[\mathcal{V}$ accepts x extending $\boldsymbol{t}]$.

The case $|\boldsymbol{t}| = k$ is by definition.

Suppose $i = |\boldsymbol{t}| < k$ is even.

In this case, the query a is sent from $\mathcal{V}$ to $\mathcal{P}$. Then

$$\begin{aligned} N(\boldsymbol{t}) &= \sum_a \Pr_r[f(x, r, \boldsymbol{t}) = a] \cdot N(\boldsymbol{t} \oplus a) \\ &= \sum_a \Pr_r[f(x, r, \boldsymbol{t}) = a] \cdot \Pr[\mathcal{V} \text{ accepts } x \text{ extending } \boldsymbol{t} \oplus a] \\ &= \Pr[\mathcal{V} \text{ accepts } x \text{ extending } \boldsymbol{t}] \end{aligned}$$

Step

If i is odd, the proof a is sent from $\mathcal{P}$ to $\mathcal{V}$. Then

$$\begin{aligned} N(\boldsymbol{t}) &= \max\Big(\, N(\boldsymbol{t} \oplus a) \mid a \,\Big) \\ &= \max\Big(\, \Pr[\mathcal{V} \text{ accepts } x \text{ extending } \boldsymbol{t} \oplus a] \mid a \,\Big) \\ &= \Pr[\mathcal{V} \text{ accepts } x \text{ extending } \boldsymbol{t}] \end{aligned}$$

This proves the proposition. □

It follows that IP $\subseteq \mathrm{PSPACE}$.

1 Interactive Proofs

2 IP $\subseteq$ **PSPACE**

3 **Warmup Exercise**

4 ∗ **PSPACE** $\subseteq$ IP

The Opposite Direction

We still need to show that $\mathrm{PSPACE} \subseteq$ IP.

Alas, PSPACE is a rather huge class, we have polynomial space bounds but the running time can be wildly exponential.

We are supposed to fit any such computation into a polynomial depth protocol, with a polynomial amount of computation on the verifier's side. It's not at all clear that this is possible. In fact, on the face of it, it sounds plain wrong.

Warmup

Rather than tackling $\mathrm{PSPACE} \subseteq$ IP directly, here is a little warmup exercise.

Consider $\#_D$SAT, the counting version of SAT, dressed up as a decision problem:

$$\#_D\text{SAT} = \{\, \varphi\#k \mid \varphi \text{ has exactly } k \text{ satisfying assignments} \,\}$$

Clearly, $\varphi\#k, \varphi\#\ell \in \#_D$SAT implies $k = \ell$, so we are just expressing a function as a language. Also, φ fails to be satisfiable iff $\varphi\#0 \in \#_D$SAT, so this language is hard, at least co-$\mathbb{NP}$-hard.

We would like a nice protocol to convince the verifier that a formula $\varphi(\boldsymbol{x})$ has some number k of satisfying assignments.

Arithmetization

As usual, identify true and false with $\{0, 1\} \subseteq \mathbb{Z}$. Here is a way to translate Boolean formulae into polynomials, thus opening the door to algebraic attack.

Definition

A multivariate polynomial $p \in \mathbb{Z}[\boldsymbol{x}]$ with integer coefficients is called a Boolean polynomial if $p(\mathbf{2}, \mathbf{2}, \ldots, \mathbf{2}) \subseteq \mathbf{2}$.

Thus, p assumes only values 0 and 1 when the arguments are constrained to be only 0 and 1. Hence p duly represents a Boolean function/formula.

In the literature and in computer algebra systems, a Boolean polynomial is sometimes defined as a polynomial over $\mathbb{Z}_2$ (aka Zhegalkin polynomial). We will stick with our definition.

Direct Conversion

Let $\boldsymbol{x} = (x_1, \ldots, x_n)$. For $\alpha \in \mathbf{2}^n$, define the monomial

$$\boldsymbol{x}^\alpha = z_1 z_2 \ldots z_n \qquad z_i = \begin{cases} x_i & \text{if } \alpha_i = 1 \\ 1 - x_i & \text{otherwise.} \end{cases}$$

Then any Boolean function $f : \mathbf{2}^n \to \mathbf{2}$ can be written as the Boolean polynomial

$$\mathsf{B}_f(\boldsymbol{x}) = \sum_{\alpha \in \mathbf{2}^n} f(\alpha)\, \boldsymbol{x}^\alpha$$

a kind of disjunctive normal form (with mutually exclusive conjuncts).

The monomials are all flat in the sense that no variable has exponent higher than 1, so the total degree of the polynomial is at most n. Alas, there are potentially exponentially many terms.

Problem Solved . . .

Proposition

The number of satisfying assignments for f is

$$\#\{\,\alpha \mid f(\alpha) = 1\,\} = 2^n \mathsf{B}_f(1/2, \ldots, 1/2)$$

Right?

Wrong! B_f has size up $O(n\,2^n)$ in the uniform model.

Conversion by Induction

Is there another way to construct a Boolean polynomial?

Think of f as being given by a Boolean formula φ. We can then exploit induction on the buildup of the formula φ. This will produce an implicit polynomial that can be much shorter than the one in direct conversion.

$$\begin{aligned}
\mathsf{B}_x &= x \\
\mathsf{B}_{\neg\varphi} &= 1 - \mathsf{B}_\varphi \\
\mathsf{B}_{\varphi\wedge\psi} &= \mathsf{B}_\varphi \cdot \mathsf{B}_\psi \\
\mathsf{B}_{\varphi\vee\psi} &= 1 - (1 - \mathsf{B}_\varphi) \cdot (1 - \mathsf{B}_\psi) = \mathsf{B}_\varphi + \mathsf{B}_\psi - \mathsf{B}_\varphi \cdot \mathsf{B}_\psi
\end{aligned}$$

We have written $x \amalg y$ for the last "multiplication" operation (this will be useful later). This operation is potentially dangerous, it roughly doubles the size of the formula.

Example: Counting

Using recursion, the formula "exactly two out of" three variables produces

$$xy(1-z) + x(1-y)z + (1-x)yz - (1-x)x(1-y)yz^2 -$$
$$xy(1-z)(x(1-y)z + (1-x)yz - (1-x)x(1-y)yz^2)$$

After expansion this looks like

$$xy + xz + yz - 3xyz - x^2yz - xy^2z + 2x^2y^2z - xyz^2 + 2x^2yz^2 + 2xy^2z^2 -$$
$$2x^2y^2z^2 - x^3y^2z^2 - x^2y^3z^2 + x^3y^3z^2 - x^2y^2z^3 + x^3y^2z^3 + x^2y^3z^3 - x^3y^3z^3$$

Note than, in a Boolean polynomial, we can replace x^k, $k \geq 1$, by x without affecting the proper representation of a Boolean function. Smashing exponents in this way and canceling we get

$$xy + xz + yz - 3xyz$$

The “Best” Boolean Polynomial

The last form is arguably the canonical one: short and flat.

$$\mathsf{B}'_\varphi = xy + xz + yz - 3xyz$$

This polynomial is the same as the one obtained from direct conversion and simplification. It, too, counts the number of satisfying truth assignments:

$$2^3\,\mathsf{B}'_\varphi(1/2, 1/2, 1/2) = 8\big(3/4 - 3/8\big) = 3$$

But the original B_φ does not, it produces the “count” $169/64$.

Arithmetizing 3-CNF

Suppose we have a formula φ in 3-CNF. Let's fix n to be the number of variables and m the number of clauses:

$$\varphi = (x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \wedge \ldots \wedge (x_{m1} \vee x_{m2} \vee x_{m3})$$

Each conjunct turns into a degree-3 polynomial

$$\mathsf{B}_{x \vee y \vee z} = x + y + z - xy - xz - yz + xyz$$

so that, say,

$$\mathsf{B}_{x \vee \neg y \vee z} = 1 - y + xy + yz - xyz$$

A Small Polynomial

Using recursion, the whole formula turns into a product of m such terms:

$$\mathsf{B}_\varphi = \prod \big(z_{i1} + z_{i2} + z_{i3} - z_{i1}z_{i2} - z_{i1}z_{i3} - z_{i2}z_{i3} + z_{i1}z_{i2}z_{i3} \big)$$

where the z_{ij} are of the form x or $1 - x$.

This size of this polynomial in unexpanded form is linear in m, at least if we use a uniform cost function (otherwise we pick up a $\log n$ factor).

Again, expanding this polynomial out would generally produce an exponential size expression.

The reason the polynomial is small is that we only have to apply the dangerous $\amalg$ operation to formulae of constant size.

Smashing Polynomials

As already mentioned, in a Boolean polynomial, we can smash exponents to obtain a canonical form.

For example we get

$$\mathsf{B}_{x \wedge \neg x} = x(1-x) = x - x^2 \rightsquigarrow 0$$

$$\mathsf{B}_{x \vee \neg x} = 1 - x(1-x) = 1 - x + x^2 \rightsquigarrow 1$$

More generally, we can "linearize" every monomial to the flat form

$$x_1^{d_1} x_2^{d_2} \ldots x_n^{d_n} \rightsquigarrow x_1^{e_1} x_2^{e_2} \ldots x_n^{e_n}$$

where $e_i = \min(d_i, 1) \in \mathbf{2}$.

The degree of these monomials is at most n, the number of variables.

Linearization Operator

If you are suspicious about purely syntactic rewrite operations, here is a more algebraic way of doing the same thing.

Let p be a Boolean polynomial. Set

$$R_x \, p(x, \boldsymbol{y}) = x \, p(1, \boldsymbol{y}) + (1 - x) \, p(0, \boldsymbol{y})$$

So this is the polynomial analogue of the standard Boole-Shannon expansion for Boolean formulae.

Clearly the x-degree of $R_x \, p$ is at most 1, so $R_x \, p$ is linear in x.

Exercise

Explain why this does not produce an efficient tautology testing algorithm.

Counting Assignments

Time to get serious. We want to show the following:

Lemma

$\#_D$SAT *is in* IP.

Intuitively, think about all possible truth assignments as branches in the complete binary tree $\mathbf{2}^n$.

For the leaves it is easy to check whether a branch satisfies φ.

For the internal nodes, we can count the number of satisfying truth assignments that pass through that node by induction (backwards from the leaves).

The root will wind up with the count of satisfying truth assignments.

Proof

Suppose we have a Boolean formula $\varphi(\boldsymbol{x})$ in 3-CNF, with n variables and m clauses.

Write $S(a_1, \ldots, a_\ell)$ for the number of satisfying assignments extending $a_1, \ldots, a_\ell \in \mathbf{2}^\ell$:

$$S(\boldsymbol{a}) = \#\big(\, \boldsymbol{b} \in \mathbf{2}^{n-\ell} \mid \varphi(\boldsymbol{a}, \boldsymbol{b}) = 1 \,\big)$$

So $S : \mathbf{2}^{\leq n} \to \mathbb{N}$ and $0 \leq S(a_1, \ldots, a_\ell) \leq 2^{n-\ell}$.

Computing S

Clearly

$$\begin{aligned} S(a_1,\ldots,a_n) &= \mathsf{B}_\varphi(a_1,\ldots,a_n) \\ S(a_1,\ldots,a_\ell) &= S(a_1,\ldots,a_\ell,0) + S(a_1,\ldots,a_\ell,1) \\ S(\mathsf{nil}) &= \text{ number of satisfying truth assignments of } \varphi \end{aligned}$$

So $S(a_1,\ldots,a_n)$ is easy to compute. On the other hand, $S(\mathsf{nil})$ seems to require exponential work.

This suggests to work our way backwards inductively from $S(a_1,\ldots,a_n)$ towards $S(\mathsf{nil})$. Then we can simply check whether $\varphi\#k$ is a Yes-instance.

We need to express this backwards induction as a protocol.

Sumcheck Protocol, First Try

We have an instance $x = \varphi\#k$.

1 $\mathcal{P}$ sends $S(\text{nil})$.
$\mathcal{V}$ checks $k = S(\text{nil})$, rejects otherwise.

2 $\mathcal{P}$ sends $S(0)$ and $S(1)$.
$\mathcal{V}$ checks $S(\text{nil}) = S(0) + S(1)$, rejects otherwise.

3 $\mathcal{P}$ sends $S(00)$, $S(01)$, $S(10)$, $S(11)$
$\mathcal{V}$ checks the respective sums.

ℓ $\mathcal{P}$ sends $S(w)$ for $w \in \mathbf{2}^{\ell-1}$.
$\mathcal{V}$ checks additions, rejects if no good.

$n+1$ $\mathcal{V}$ checks all the $S(w)$, $w \in \mathbf{2}^n$, accepts/rejects accordingly.

Of course, the almighty prover has no problem computing the various $S(w)$.

Alas, this scheme fails catastrophically ...

Correctness

First the good news: our sumcheck protocol is correct.

If $\varphi\#k$ is a Yes-instance, then the honest prover that just faithfully carries out all the calculations will convince the verifier.

So suppose $\varphi\#k$ is a No-instance, and $\widetilde{\mathcal{P}}$ is some malicious prover that tries to talk the verifier into accepting.

Since $k \neq S(\text{nil})$, the prover must lie in the first round and send $\widetilde{S}(\text{nil})$.

But then the prover must lie again in the next round with $\widetilde{S}(0)$ and/or $\widetilde{S}(1)$.

And so on, there will be more and more lies, just like the GOP.

The gig will be up in round $n + 1$.

Fiasco

This is a very pretty protocol, but it fails miserably: the prover is sending an exponential amount of information, the verifier cannot even read all this stuff in polynomial time, much less carry out the necessary test.

And there seems to be no easy fix: the verifier is entirely deterministic, so if we could somehow make it polynomial time we would wind up in $\mathbb{NP}$, a rather unlikely proposition.

Time for some outside-of-the-box thinking. We'd like to maintain some sort of sumcheck protocol, but without exponential blowup.

The Trick

Note that the verifier cannot compute the coefficient representation of the polynomial, only the implicit, unexpanded product form.

When it comes to satisfiability, in our setting we more or less have to argue about the roots of Boolean polynomials. That suggests to work in a field rather than just the ring $\mathbb{Z}$.

The sledgehammer approach would be to compute over $\mathbb{C}$, where all our polynomials factor into linear terms. Alas, this won't work: $\mathbb{C}$ is not computable, and certainly no polynomial time computable.

The Trick II

To make things computable, the fallback position is $\mathbb{Q}$, the field of rational numbers.

Alas, we also need to argue probabilistically, so using $\mathbb{Q}$ won't work.

Clever Solution:
We work in a sufficiently large finite field $\mathbb{F}$.

Note well: this change of domain will not affect the meaning of the sum, as long as the finite field has sufficiently large characteristic.

Come Again

Since we only need a few integers, we can use algebra over a finite field rather than the actual integers: the prime field $\mathbb{Z}_p$ contains the integers $\{0, 1, \ldots, p-1\}$ where p is the characteristic of $\mathbb{F}$. Even better: for sufficiently small such integers, the arithmetic is the same as over $\mathbb{Z}$. For example, $3 + 5 = 8$ and $3 \cdot 5 = 15$ over $\mathbb{F} = \mathbb{Z}_{31}$.

We will use a finite field $\mathbb{F}$ of size at least $2^{|\varphi|}$ to make sure the arithmetic is suitable. For simplicity, just think about $\mathbb{Z}_p$, p prime and large enough. We won't worry about how we can get our hands on such a prime, though you might want to think about it.

The prover and the verifier agree on p, once and for all. From now on, all arithmetic is in $\mathbb{F} = \mathbb{Z}_p$.

Counting, Again

Let's return to our satisfying assignment counting function S from above: for $\boldsymbol{a} \in \mathbf{2}^\star$ define

$$S(\boldsymbol{a}) = \sum_{\boldsymbol{b} \in \mathbf{2}^\star} S(\boldsymbol{a}, \boldsymbol{b})$$

where $|\boldsymbol{b}| = n - |\boldsymbol{a}|$ and the arithmetic is over $\mathbb{F}$, and the same as over $\mathbb{N}$ for $\mathbb{F}$ large enough.

So $S(\text{nil})$ is what we are after, but only $S(a_1, \ldots, a_n) = \mathsf{B}_\varphi(a_1, \ldots, a_n)$ is easy to compute.

We need to exploit the prover to help with the backward induction, but bear in mind that it can only send a polynomial amount of information, and everything must be polynomially checkable.

This is where evaluation over $\mathbb{F}$ comes in handy.

Cheesy Example

Let

$$\varphi = x \wedge (y \vee \overline{z})$$

so that

$$\begin{aligned} S(x,y,z) &= x \cdot (y \amalg \overline{z}) \\ &= x(y + (1{-}z) - y(1{-}z)) \\ &= x - xz + xyz \end{aligned}$$

By repeated Boole-Shannon expansion we get

$$\begin{aligned} S(x,y) &= S(x,y,0) + S(x,y,1) = x + xy \\ S(x) &= S(x,0) + S(x,1) = 3x \\ S(\mathsf{nil}) &= S(0) + S(1) = 3 \end{aligned}$$

Sumcheck Protocol

On input $\varphi\#k$, both prover and verifier compute B_φ (but the verifier cannot expand the polynomial).

The prover computes a prime $q > 2^{n+m}$ and sends it to the verifier, who checks that the number is indeed prime and large enough.

From now on, all the arithmetic takes place in $\mathbb{F} = \mathbb{Z}_q$. In particular we use the map S over $\mathbb{F}$, $S : \mathbb{F}^{\leq n} \to \mathbb{F}$.

The verifier sets $v_0 = k$ and the prover needs to establish the correctness of this value.

Sumcheck Protocol II

Suppose in previous rounds the verifier has already chosen $\ell - 1$ random elements $\boldsymbol{r} = r_1, r_2, \ldots, r_{\ell-1}$ of $\mathbb{F}$.

Here is the key part of round ℓ: $\mathcal{P}$ now sends polynomials, rather than values.

$\mathcal{P}$ sends a univariate polynomial $P_\ell(z)$ of degree at most m.

The honest prover will send the actual counting polynomial, but we have to guard against a malicious prover trying to cheat.

Sumcheck Protocol III

$\mathcal{V}$ checks

- $v_{l-1} = P_\ell(0) + P_\ell(1)$ and
- the degree condition.

$\mathcal{V}$ rejects if anything fails.

Then $\mathcal{V}$ throws out another challenge to the prover:

- picks $r = r_\ell$ uniformly at random from $\mathbb{F}$,
- sends $v_\ell = P_\ell(r)$ and r to $\mathcal{P}$.

Last round: $\mathcal{V}$ checks

- $v_n = \mathsf{B}_\varphi(r_1, \ldots, r_n)$; accepts/rejects accordingly.

Correctness

If we have a Yes-instance, the honest prover uses the actual counting function.

$$P_\ell(z) = \sum_{\boldsymbol{b}} \mathsf{B}_\varphi(r_1, \ldots, r_{\ell-1}, z, \boldsymbol{b})$$

and the verifier accepts, always (no matter the random r_i). This works since q is large enough not to disturb the arithmetic.

Now consider a No-instance $\varphi\#k$. A malicious prover could send a fake polynomial $\widetilde{\mathcal{P}}_1$ to justify the wrong value k. Then another fake polynomial $\widetilde{\mathcal{P}}_2$ to justify $\widetilde{\mathcal{P}}_1$, and so on.

Here is the critical fact: the verifier evaluates at random elements $r \in \mathbb{F}$ and we are only dealing with polynomials of degree at most m. It is quite unlikely that the fake polynomials will match the real one on random inputs.

More Precisely . . .

The likelihood that two such polynomials agree on r is at most

$$\Pr[P(r) = \widetilde{\mathcal{P}}(r)] < m^{-2}$$

for $m \geq 10$.

The reason is that we operate over a field, so any degree d, univariate polynomial has at most d roots, or is already identically zero. We're good since $m/2^m \leq m^{-2}$ for $m \geq 10$.

So to cheat in the first round, the malicious prover must probably cheat in all other rounds: one can show that the prover will get lucky only with probability at most nm/q, an exponentially small number by our choice of q.

1 Interactive Proofs

2 IP $\subseteq$ **PSPACE**

3 Warmup Exercise

4 ∗ **PSPACE** $\subseteq$ IP

Mental Health Warning

☠ ☠ ☠ ☠ ☠ ☠ ☠

Don’t worry about all the details in the following argument, just try to get an idea of what the main strategy is.

☠ ☠ ☠ ☠ ☠ ☠ ☠

Are We Done?

Sadly, no: we have only handled satisfiability so far. To grab all of PSPACE we need more.

It is tempting to extend the finite field trick from $\#_D$SAT to QBF, the problem of testing validity of quantified Boolean formulae (which is enough to capture all of PSPACE).

Alas, the construction is quite a bit more challenging.

Towards PSPACE $\subseteq$ IP

After all, $\forall$ is just a glorified $\wedge$, and similarly for $\exists$ and $\vee$.

More precisely, consider a QBF in prenex normal form, say

$$\varphi = \exists\, x_1\ \forall\, x_2\ \ldots \exists\, x_{m-1}\, \forall\, x_m\, \psi(\boldsymbol{x})$$

Let $S(\boldsymbol{x}) = \mathsf{B}_\psi(\boldsymbol{x})$ be the arithmetization of the matrix (aka the quantifier-free part) ψ of the formula, so for $\boldsymbol{a} \in \mathbf{2}^m$ we have

$$S(\boldsymbol{a}) = \begin{cases} 1 & \text{if } \psi(\boldsymbol{a}) \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

Recursion

Write Q_i for a quantifier $\exists$ or $\forall$.

Let $\boldsymbol{a} = a_1, \ldots, a_i$, $i \leq m$. We want a map $S : \mathbb{F}^{\leq m} \to \mathbb{F}$ such that

$$S(\boldsymbol{a}) = \begin{cases} 1 & \text{if } Q_{i+1}x_{i+1} \ldots Q_m x_m \, \psi(\boldsymbol{a}, x_{i+1}, \ldots, x_m) \text{ is valid,} \\ 0 & \text{otherwise.} \end{cases}$$

so that $S(\text{nil})$ determines validity of φ.

The prover needs to convince the verifier that $S(\text{nil}) = 1$, where all the arithmetic takes place over $\mathbb{F}$.

First Try

To get from $S(\boldsymbol{a}, b)$ to $S(\boldsymbol{a})$ let Q be the corresponding quantifier and do this:

$$\begin{array}{ll} Q = \forall & S(\boldsymbol{a}) = S(\boldsymbol{a}, 0) \cdot S(\boldsymbol{a}, 1) \\ Q = \exists & S(\boldsymbol{a}) = S(\boldsymbol{a}, 0) \amalg S(\boldsymbol{a}, 1) \end{array}$$

So this is just the usual translation into conjunctions and disjunctions.

Clearly, this backward recursion produces the right function S.

Trouble: Alas, when we are writing the right-hand-sides as polynomials, the degrees increase (recall that last time we only had to deal with addition). In fact, they might double at each step, leading to exponential degrees.

A Trick

We will get the degree of the polynomials under control by applying the linearization reduction $x^k \rightsquigarrow x$ mentioned above. First, we insert copious degree reduction/linearization operations everywhere.

Here R_i means: clean up all the variables $x_1, \ldots, x_i$ (by sequentially applying linearization operators R_z).

$$\coprod_{x_1} R_1 \prod_{x_2} R_2 \coprod_{x_3} R_3 \ldots \prod_{x_m} R_m \, S(\boldsymbol{x})$$

Slight Rewrite

Note that the semantics for R_z is really a no-op for Boolean polynomials, unlike the semantics of a quantifier, but that's OK.

So now we can write the whole Boolean polynomial as

$$\mathcal{Q}_1 y_1 \, \mathcal{Q}_2 y_2 \, \ldots \mathcal{Q}_\ell y_\ell \, S(\boldsymbol{x})$$

where $\ell = (m^2 + 3m)/2$ and $\mathcal{Q}_i \in \{\coprod, \prod, R\}$ is one of our operators.

Protocol Outline

The protocol is based on the prover convincing the verifier that

$$v_j = \mathcal{Q}_{j+1} y_{j+1} \, \ldots \mathcal{Q}_\ell y_\ell \, S_j(\boldsymbol{x})$$

where the arithmetic is over $\mathbb{F}$ and S_j is some polynomial ($S_m = S$).

To this end, the verifier produces v_{j+1} and the prover has to establish

$$v_{j+1} = \mathcal{Q}_{j+2} y_{j+2} \, \ldots \mathcal{Q}_\ell y_\ell \, S_{j+1}(\boldsymbol{x})$$

and so on and so forth.

In the end, the verifier should be convinced that $v_0 = 1$.

The Rounds

There are three cases, depending on whether $Q = \coprod, \prod, R$. The first two are entirely similar, we will only discuss $Q = \coprod$.

The prover tries to convince the verifier that

$$v_j = \coprod_{x_i} R_i \prod_{x_{i+1}} \ldots \prod_{x_m} R_m \, S(r_1, \ldots, r_{i-1}, x_i, \ldots, x_m)$$

- To this end, the prover sends a univariate polynomial $p(z)$.
- The verifier checks $v_j = \coprod_z p(z)$.
- Then the verifier chooses r_i at random in $\mathbb{F}$ and sets $v_{j+1} = p(r_i)$.

Next, the prover has to convince the verifier that

$$v_{j+1} = R_i \prod_{x_{i+1}} \ldots \prod_{x_m} R_m \, S(r_1, \ldots, r_i, x_{i+1}, \ldots, x_m)$$

Reduction Rounds

This is the place where the prover has to convince the verifier that

$$v_j = R_{x_i} \mathcal{Q}_y \dots \prod\nolimits_{x_m} R_{x_1} \dots R_{x_m} S(r_1, \dots, r_i, x_{i+1}, \dots, x_m)$$

- To this end, the prover again sends a univariate polynomial $p(z)$.
- The verifier checks $v_j = R_{x_i} p(x_i)[r_i]$.
- Then the verifier chooses a new r_i at random in $\mathbb{F}$, sets $v_{j+1} = p(r_i)$ and challenges the prover to establish

$$v_{j+1} = \mathcal{Q}_y \dots \prod\nolimits_{x_m} R_{x_1} \dots R_{x_m} S(r_1, \dots, r_i, x_{i+1}, \dots, x_m)$$

Aspirin, Anyone?

One can show that this protocol satisfies both the completeness and soundness conditions.

All the operations are duly polynomial time.