UCT

Interactive Proofs

Klaus Sutner

Carnegie Mellon University

Spring 2022

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, underneath we are really dealing with projections and nondeterministic Turing machines, but 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.

Proofs

Witnesses may carry a lot of information, just think about Pratt's result that primality is in $\mathbb{NP}$.

Examples like Pratt's theorem suggest that it might be useful to think of w as a **proof** of the assertion $x \in L$. The proof has to be sufficiently simple that it can be verified in polynomial time, and it has to have polynomial length itself.

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

Lonely Verifier

In the usual math scenario, someone spends a lot effort to build a proof, publishes it, 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 real world proofs are not formal (though a few have indeed been formalized). Instead, the reader has to work a bit harder, find typos, resolve ambiguities, fill in gaps, and so on.

For this process it is immeasurably useful for the reader to be able to ask questions, to the author or some other authority.

So?

6

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 (but later we will have to deal with malicious provers).

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

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

Prover and Verifier

7

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 several 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 can fool the verifier into acceptance.

More Formally . . .

8

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 $a_i \in \mathbf{2}^\star$:

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

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

Output and Step 1

9

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

The prover has unlimited power, so it can easily compute $a_1 = f(x)$.

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

10

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 $\mathrm{res}(\mathcal{V}, \mathcal{P})(x) = 1$

Soundness $x \notin L$ implies $\mathrm{res}(\mathcal{V}, \widetilde{\mathcal{P}})(x) = 0$ 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.

Proof Systems

11

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.

The proof for a true statement cannot be too complicated or long here, 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, even using something like 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.

In addition, in the end it turns out that only provers in PSPACE are relevant.

Oracle Prover

Suppose the prover has access to $\emptyset'$, the Halting Set, and we would like to exploit this to define an interaction to check membership in $\emptyset'$.

On input some index e we set

$\mathcal{V}$: $a_1 = f(e) =$ "hello"
$\mathcal{P}$: $a_2 = g(e, a_1) = 1$ if $e \in \emptyset'$, 0 otherwise.
$\mathcal{V}$: Accept if $a_2 = 1$, reject otherwise.

Completeness is fine, but this protocol fails the soundness condition, miserably: the prover could cheat, and the verifier has no way of checking the correctness of a_2.

Exercise

Convince yourself that there is no easy fix to this problem.
Then show that indeed Halting is 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.

$\mathcal{V}$: $a_1 = f(x) =$ "hello"
$\mathcal{P}$: $a_2 = g(x, a_1)$ 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 $(a_1, \ldots, a_k)$ 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 $\mathcal{V}(x, a_1, \ldots, a_{2i}) = a_{2i+1}$.

Case 2: $x \notin L$.

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

Probabilistic Verifiers

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

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

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

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

Set

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

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

So What?

We know $\mathbb{NP} \subseteq$ 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 $\mathbb{P} =$ 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 question of whether two undirected graphs are isomorphic, here in its negated form.

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 (possibly an intermediate problem). By contrast, GNI is only 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 that $\mathbb{Z}_m^\star$ 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, see homework.

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.

Connections

It is clear that IP contains $\mathbb{NP}$ and BPP (but remember: it might be that $\mathbb{P}$ = BPP).

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

So the question 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 = 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.

Workaround

Instead of attempting a direct simulation, we study more carefully the structure of an IP protocol, and in particular a transcript $(a_1, a_2, \ldots, a_k) \in (\mathbf{2}^\star)^k$.

To this end consider a partial transcript $t = (a_1, a_2, \ldots, a_j)$. We are interested in extensions to a full transcript such that

$$\begin{aligned} a_{i+1} &= f(x, r, a_1, \ldots, a_i) && 0 \le i < k, \text{ even} \\ a_{i+1} &= g(x, a_1, \ldots, a_i) && 0 \le i < k, \text{ odd} \\ a_k &= \text{Yes} \end{aligned}$$

Extensions

Let's say that a partial transcript t is extensible if there exists an extension $t' = t \oplus (a_{j+1}, \ldots, a_k)$ with these properties. Here $\oplus$ here means join or append.

Define a rational number $N(t)$, the acceptance value of t, for any partial transcript t as follows.

$|t| = k.$ Then $N(t) = 1$ if t is consistent with some random string r and accepting; 0 otherwise.

$|t| = j < k.$ Then

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

The weighted average in the second case is supposed to be

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

WTF?

Though the definition is slightly complicated, we can verify that the whole computation of the $N(t)$ is in polynomial space (though not polynomial time): the strings r and a cannot be too long, we can try them all and count.

Why would anyone care about $N(t)$? Because

Proposition

$N(\text{nil}) = \Pr[\mathcal{V} \text{ accepts } x] := \max\big(\Pr[(\mathcal{V},\mathcal{P}) \text{ accepts } x] \mid \mathcal{P}\big)$.

Proof.

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

The case $|t| = k$ is by definition.

Step

Suppose $j = |t| < k$ is even.

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

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

Step

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

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

This proves the proposition. □

It follows that IP $\subseteq$ PSPACE.

The Opposite Direction

We still need to show that 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 PSPACE $\subseteq$ IP directly, here is a little warmup exercise.

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

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

Obviously φ fails to be satisfiable iff $\varphi\#0 \in \#_D$SAT, so this 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 polynomial p 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.

Upcoming Attractions: Later we will evaluate these polynomials also on other, a priory non-nonsensical arguments.

Conversion

We can easily translate a Boolean formula φ into corresponding polynomial P_φ by induction as follows:

$$\begin{aligned} P_x &= x \\ P_{\neg\varphi} &= 1 - P_\varphi \\ P_{\varphi \wedge \psi} &= P_\varphi \cdot P_\psi \\ P_{\varphi \vee \psi} &= 1 - (1 - P_\varphi) \cdot (1 - P_\psi) = P_\varphi + P_\psi - P_\varphi \cdot P_\psi \end{aligned}$$

We write $x \amalg y$ for the last “multiplication” operation.

Note that Boolean polynomials for a given formula φ are not uniquely determined; a fact that we will casually ignore.

Example: Counting

The formula “exactly one out of” for three variables produces

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

For four variables we get

$$u+x+y+z-2ux-2uy-2xy-2uz-2xz-2yz+3uxy+3uxz+3uyz+3xyz-4uxyz$$

Exercise

Figure out a fast test for Boolean polynomials.

Example: CNF

Suppose we have a formula in 3-CNF:

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

For each conjunct we have

$$P_{x \vee y \vee z} = x + y + z - xy - xz - yz + xyz$$

and for the whole formula the arithmetization is

$$P_\varphi = \prod (x_{i1} + x_{i2} + x_{i3} - x_{i1}x_{i2} - x_{i1}x_{i3} - x_{i2}x_{i3} + x_{i1}x_{i2}x_{i3})$$

Note that the degree is $3m$, and generating the explicit coefficient vector by multiplying everything out requires exponential work.

Degrees and Arguments

Clearly, if $n = |\varphi|$, then the degree of P_φ cannot be larger than n.

In fact, note that in a Boolean polynomial we can replace x^k by x without affecting the proper representation of a Boolean function. We get for example

$$\begin{aligned} P_{x \wedge \neg x} &= x(1-x) = x - x^2 \rightsquigarrow 0 \\ P_{x \vee \neg x} &= 1 - x(1-x) = 1 - x + x^2 \rightsquigarrow 1 \end{aligned}$$

In general, we can linearize every monomial to the form

$$x_1^{e_1} x_2^{e_2} \ldots x_m^{e_m}$$

where $e_i \in \mathbf{2}$.

By performing this simplification, we can safely assume that the degree of P_φ is no higher than m, the number of variables in φ.

Linearization Operator

If you don't like rewrite systems, 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.

Proof.

Suppose we have a Boolean formula $\varphi = \varphi(x_1, \ldots, x_m)$.

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

$$S(\boldsymbol{a}) = \#\big(\boldsymbol{b} \mid \varphi(\boldsymbol{a}, \boldsymbol{b}) = 1\big)$$

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

Computing S

Clearly

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

So $S(a_1, \ldots, a_m)$ is easy to compute.

Then we can inductively work our way backwards towards $S(\text{nil})$ which would allow us to check whether $\varphi\#k$ is a yes-instance.

We need to express this 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.

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

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

Needless to say, the prover has no problem computing the various $S(w)$.

Alas, this won't work ...

Correctness

First the good news: our sumcheck protocol is correct.

If $\varphi\#k$ is a yes-instance, then the "canonical" 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 (potentially 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 $m+1$.

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.

Needless to say, an exponential verifier is not very exciting.

More importantly, note that the verifier is entirely deterministic, so if it were polynomial this would put us into $\mathbb{NP}$, 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 form.

Ultimately we want to argue about the roots of polynomials, which might suggest to work in a field rather than just the ring $\mathbb{Z}$. But, 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, see below.

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 suitable $\mathbb{F}$.

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.

Counting, Again

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

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

where $|\boldsymbol{z}| = m - |\boldsymbol{a}|$ 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 $S(a_1, \ldots, a_m) = P_\varphi(a_1, \ldots, a_m)$ 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

$$S(x, y, z) = x \cdot (y \amalg \overline{z}) = xyz - xz + x$$

We get

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

Dire Warning: Our polynomial verifier cannot actually compute the coefficient representation of P_φ, only the compact representation using $\cdot$ and $\amalg$.

Sumcheck Protocol

The correct version of the sumcheck protocol uses the extended map S over $\mathbb{F}$, $S : \mathbb{F}^\star \to \mathbb{F}$. The verifier $\mathcal{V}$ picks elements in $\mathbb{F}$ at random.

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:

$\mathcal{P}$ sends the coefficient vector of the polynomial

$$P_\ell(z) = \sum_{\boldsymbol{b}} S(\boldsymbol{r}, z, \boldsymbol{b})$$

where the sum is over all $\boldsymbol{b} \in \mathbf{2}^{m-\ell} \subseteq \mathbb{F}^{m-\ell}$.

$\mathcal{V}$ checks $v_{\ell-1} = P_\ell(\boldsymbol{r}, 0) + P_\ell(\boldsymbol{r}, 1)$.

More

$\mathcal{V}$ also checks that the degrees are bounded by n; rejects if anything fails.

Lastly, $\mathcal{V}$ does the following:

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

In the last round, $\mathcal{V}$ compares $v_m = S(r_1, \ldots, r_m)$ to $P_\varphi(r_1, \ldots, r_m)$; accepts/rejects accordingly.

Note that $\mathcal{P}$ now sends polynomials, rather than values.

Correctness

As usual, the honest prover uses the actual S function, albeit it over $\mathbb{F}$. Everything works just fine on yes-instances.

But how about a malicious prover when $\varphi\#k$ is a no-instance? Then $\widetilde{\mathcal{P}}$ must send the wrong coefficient vector, representing a different polynomial, to fool $\mathcal{V}$ into acceptance. But then in the first round

$$\Pr[\widetilde{S}(r_1) = S(r_1)] < n^{-2}$$

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

The same holds for longer sequences of arguments to $\widetilde{S}$ and S.

So to cheat at round 0, $\mathcal{P}$ must probably cheat at all other levels: it will get lucky only with probability at most $m \cdot n^{-2} \leq 1/n$.

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).

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

Towards PSPACE ⊆ 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 $S(\boldsymbol{x}) = P_\psi(\boldsymbol{x})$ 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

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

- 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

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

Aspirin, Anyone?

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

All the operations are duly polynomial time.