Today

- \( \#P \subseteq \text{IP} \).
- Polynomial straightline programs and interactive proofs.
- Straightline programs for \( \text{PSPACE} \).

Recall \( \#P, \text{IP} \)

- \( \#P \) is the class of counting functions. Prototypical example: \( \#\text{SAT} \) - \( \# \) of satisfying assignments of a 3CNF formula.
- \( \text{IP} \) is the class of languages with interactive proofs. So far know that \( \text{IP} \) contains \( \text{NP} \) and \( \text{GNI} \) (graph nonisomorphism).
- Anything else? Today will show \( \#P \) has interactive proofs. Also try showing \( \text{PSPACE} \) has \( \text{IP} \).
- Remarks:
  - Need to use multiple rounds of interaction (so not an “AM” proof system.).

- \( \text{PSPACE} \) is closed under complement. Any reason to believe \( \text{IP} \) is?

Basic Idea

- Suppose Prover wishes to prove \( \phi \) has \( A \) satisfying assignments.
- Can use self-reducibility:
  - Can prove \( \phi_{x_1=0} \) has \( A_0 \) assignments and \( \phi_{x_1=1} \) has \( A_1 \) assignments, and that \( A_0 + A_1 = A \).
  - Unfortunately \( \# \) statements to be proved is growing exponentially.
  - Any way to commit to \( \#\phi_{x_1=0} \) and \( \#\phi_{x_1=1} \) jointly and then prove only one claim?
  - How does \( \#\phi_{x_1=\alpha} \) behave as a function of \( \alpha \) - naturally?
**Arithmetizing SAT**

Literal polynomials: $x \mapsto x$, $\overline{x} \mapsto (1 - x)$.

Clause polynomial: $C(x, y, z)$ converted to $P(x, y, z); x \lor y \lor z \mapsto 1 - (1 - x)(1 - y)(1 - z)$.

SAT polynomial: $\phi(x_1, \ldots, x_n) \mapsto Q(x_1, \ldots, x_n)$ where $Q(x) = \prod_{i=1}^{m} P_i(x)$ if $\phi = \land_{i=1}^{m} C_i$.

Property $Q(x_1, \ldots, x_n)$: for $a \in \{0, 1\}^n$, $Q(a) = 1$ if $a$ satisfies $\phi$ and 0 otherwise.

$Q$ is a polynomial of degree $m$ in each variable.

$\#\phi = \sum_{a \in \{0, 1\}^n} Q(a)$.

**#SAT tree & Q-tree**

Draw tree of $Q$-values:

Root = value of $\sum_{a \in \{0, 1\}^n} Q(a)$.

Node = value of sum on suffix, with prefix set to some fixed value.

$Q_b = \sum_{c \in \{0, 1\}^n} Q(b, c)$.

Verifier verifies $Q_b = Q_{b0} + Q_{b1}$.

Now need to to verify $Q_{b0}$ and $Q_{b1}$.

Can’t afford to do this!

---

**#SAT in IP**

Will arbitrarily consider $Q_b$ for every $b \in \mathbb{Z}_p^2$ for some prime $p$.

What meaning does it have? None seemingly, but $Q_b$ is well defined!

Suppose prover claims $Q_\lambda = \#\phi = N$. Will ask prover to prove $Q_\lambda = N (\text{ mod } p)$.

**IP protocol for #SAT**

Recursively Arthur is verifying: $Q_b = K (\text{ mod } p)$.

Consider the function $p_b(x) = \sum_{c \in \{0, 1\}^n} Q(b, x, c)$ $p_b$ is a univariate polynomial of degree $m$.

Arthur asks Merlin for $p_b(x)$.

Merlin responds with $h(x)$.

Arthur verifies $h(0) + h(1) = K$.

Arthur picks random $\alpha \in \mathbb{Z}_p$ and sends to Merlin,

Now recursively verify $Q_{b\alpha} = h(\alpha)$.

At end Arthur can compute verify $Q_b = K$, since $Q_b = Q(b)$.
Soundness

Completeness obvious.

For soundness, will claim:

Claim: If $Q_b \neq K$, then $\Pr_{\alpha}[Q_{b\alpha} = h(0) + h(1) = K] \leq m/p$.

Proof: CRT to get initialization right over $p$. Schwartz Lemma for inductive step.

Theorem follows (modulo details).

Abstracting the proof

- Proof uses very little specific to $\#P$.
- More about “downward self-reducibility and polynomials”.
- Specifically, downward self-reducibility leads to the tree.
- Algebra compresses questions down to one question.
- In fact, don’t need any structure on the questions!

Extending compression: Low-degree curves

Suppose computing $Q_b(x)$ involves computing $Q_c(y)$ and $Q_c(z)$, where $y$ and $z$ are not related. Can we extend our idea to this case?

Lines in $\mathbb{F}^n$: $\ell : \mathbb{F} \to \mathbb{F}^n$.

Geometrically - a line is a line.

Algebraically: it is a collection of $n$ functions, each of which is a degree 1 polynomial.

For any two points $y$ and $z$, there is a line $\ell$ s.t. $\ell(0) = y$ and $\ell(1) = z$. Specifically $\ell(t) = (1 - t)y + tz$.

Why are lines nice?

\[ Q \circ \ell : \mathbb{F} \to \mathbb{F} \text{ is a polynomial of (at most) same degree as } Q. \]
Extending the protocol’s capabilities

- At $i$th level, to verify $Q(x) = a$, the verifier generates $y$ and $z$ and $\ell$ containing $y$ and $z$. Asks prover for $Q \circ \ell$.

- Prover responds with degree $d$ univariate polynomial $h$.

- Verifier verifies consistency assuming $h$ is right, and then verifies $h(\alpha)$ is correct for random $\alpha$.

Straightline program of polynomials

Defn: $p_0, \ldots, p_L$ is an $(n, d, L, w)$-straightline program of polynomials if

- Every $p_i$ is on at most $n$ variables.

- Every $p_i$ is of degree at most $d$.

- $p_i$ is constructed from $p_{i-1}$ in a simple form. (Formally, there is a polynomial time algorithm $A$ that, given $i$, $x$ and an oracle for $p_{i-1}$ can compute $p_i(x)$ making at most $w$ non-adaptive queries to $p_i$.)

- $p_0$ is computable in polynomial time.

Polynomial program satisfiability

Defn: Polynomial straightline program polynomial satisfaction is the language whose instances are $(x, a, \langle p_0, \ldots, p_L \rangle)$ such that $p_L(x) = a$, where $x \in \mathbb{Z}^n$, $a \in \mathbb{Z}$ and $p_0, \ldots, p_L$ is an $(n, d, L, w)$-straightline program of polynomials.

Polynomial program is in IP for $w = 2$

Verifier runs in time $\text{poly}(n, d, L, \log ||x||)$.

- Verifier picks random prime $p \approx \text{poly}(n, d, L, \log ||x||)$ and sends to prover. Sets $a_L \leftarrow a$, and $x_L \leftarrow x$.

- For $i = L - 1$ downto 0 do:
  - Let $y_i$ and $z_i$ be queries of $A$ on input $i + 1$, $x_{i+1}$. Let $\ell_i$ be line thru $y_i$ and $z_i$. Verifier asks prover for $p_i \circ \ell_i$. Prover responds with $h_i$.
  - Verifier verifies that $A$’s answer on oracle values $h(0)$ and $h(1)$ is $a_{i+1}$.
  - Verifier picks random $\alpha \in \mathbb{Z}_p$ and sets $x_i \leftarrow \ell_i(\alpha)$ and $a_i \leftarrow h_i(\alpha)$.
- At end verifier verifies \( h_0(\alpha) = p_0(\ell_0(\alpha)) \).

Completeness = 1.

Soundness \( \leq \ell d/p + \text{CRT} \).

---

**PSPACE-completeness**

Define longer sequence:

- \( g_i = g_{i,s} = f_i \).
- \( g_i(a, b, c) = g_{i-1,s}(a, c) \cdot g_{i-1,s}(c, b) \).
- \( g_{ij}(a, b, c) = g_{i,j-1}(a, b, c0) + g_{i,j-1}(a, b, c1) \), where \( c \in \mathbb{Z}_p^{s-j} \).

- \( g \) has degree at most \( C' \) in the variables of \( a, b \), and at most \( 2C' \) in the variables of \( c \).

- \( g_0, g_{10}, g_{11}, \ldots, g_{1s}, g_{20}, \ldots, g_{ss} \) is a sequence of width \( w = 2 \).

- PSPACE completeness follows.

---

**Poly program sat. is PSPACE complete**

- Basic idea: \( f_i(a, b) \) has configurations \( a \) and \( b \) as inputs (if from \( \{0, 1\}^s \)), and \( f_i(a, b) = 1 \) if get from \( a \) to \( b \) in exactly \( 2^i \) steps.

- \( f_0 \) is a constant-degree polynomial, of degree \( C' \) in each variable.

- \( f_{i+1}(a, b) = \sum_{c \in \{0, 1\}^s} f_i(a, c) f_i(c, b) \) is also a polynomial of degree \( C' \) in each variable.

- Unfortunately \( w \neq 2 \).

- Can fix easily: Will do summation slowly.

---

**Conclusion**

- PSPACE complete problem (Poly. program sat.) has an IP.

- PSPACE \( \subseteq \text{IP} \).

- Can generalize lines argument even “wider”, for \( w > 2 \).

- Exercise: Do this, and thus give direct proof that the permanent has an interactive proof, where the prover only needs to be able to compute permanent.