1 IP

The topic is covered in Chapter 8 of the text, but we will see only a small part: roughly Section 8.1, and the beginning of Section 8.3 (8.3.1 and 8.3.2).

**Definition 1.** An interactive proof system is a protocol that enables two actors, a prover $P$ (Merlin in the previous lecture) and a verifier $V$ (Arthur) to decide (probabilistically) whether a string is in a given language $L$ (the protocol depends on $L$). It proceeds in phases. In each phase, the verifier and the prover exchange messages. The messages of $V$ are computed by a fixed probabilistic, polynomial time bounded Turing machine $V$. The messages of $P$ can be arbitrary functions.

Initially $V$ sends $P$ a message $a_1 = f(x)$ (where $f$ is computed by the probabilistic Turing machine.) $P$ responds with a message $a_2 = g(x, a_1)$ (where $g$ is determined by the protocol, but has no restrictions on the complexity of computing it.)

Now $V$ computes $a_3 = f(x, a_1, a_2)$ and sends it to $P$ who responds with a string $a_4$, and so on. At some point the protocol produces an output $out(V, P) \in \{0, 1\}$. The output must satisfy Completeness and Soundness conditions:

(Completeness) if $x \in L$ $\exists$ prover $P$ : $Pr[ out(V, P) = 1 ] \geq 2/3$

(Soundness) if $x \notin L$ $\forall$ prover $P$ : $Pr[ out(V, P) = 0 ] \geq 2/3$

The second condition is equivalent to

$\forall$ prover $P$ : $Pr[ out(V, P) = 1 ] \leq 1/3$
In this definition the coin tosses of \( V \) are private – \( P \) does not see them. This restriction can be removed (but we will not see how to prove this in this course.) Also, the probability of Completeness can be boosted to 1.

**Fact 2.** It is easy to see that \( IP \subseteq PSPACE \)

The proof follows from direct simulation.

The important results on this topic are

- The model where coins are public (\( P \) sees the results of \( V \)'s coin tosses) is equivalent to the one we defined

- \( IP = PSPACE \) The proof is a clever protocol for TQBF.

These results have important consequences. Unfortunately, we will not have time to pursue any of this.

We will study a proof that is the ‘baby version’ of the result that \( TQBF \in IP \). It uses a new (to us) technical device: **algebraization**

**Definition 3.** \( \#SAT_D = \{(\phi,k) : \text{Boolean formula } \phi \text{ has exactly } k \text{ satisfying assignments}\} \)

It is easy to see that this language is \( NP \)-hard, since \((\phi,0) \in \#SAT_D \) iff \( \phi \) is unsatisfiable.

It is the decision version of the class \( \#P \), defined by Valiant, the counts the number of solutions to to NP-complete problems. A slight variant, \( \{(\phi,k) : \text{formula } \phi \text{ has at least } k \text{ satisfying assignments}\} \)

is the complete language for the class \( PP \) (probabilistic polynomial) class of languages where the probability of acceptance is only required to be strictly greater than 1/2. This is a nice class, but the advantage of random choice (probability = 1/2) is so small that it cannot be exploited to give reliable answers through polynomially many repetitions and taking majorities, as in the case of \( BPP \).

The new important technique we will use in our proof is **arithmetization**. We will extend Boolean formulas to algebraic ones.

If \( x_i \) are Boolean variables, we will ‘translate’ them to variables that range over a field (we will make Boolean formulas into polynomials) as follows:

- \( x_i \rightarrow X_i \)
- \( \bar{x}_i \rightarrow (1 - X_i) \)
- \( u \land v \rightarrow UV \) (multiplication)
\[ u \lor v \rightarrow 1 - (1 - U)(1 - V) \] (actually follows from the previous ones by DeMorgan Laws)

So, for example, the Boolean formula \( x_i \land \bar{x}_j \land x_k \) becomes \( X_i(1 - X_j)X_k \), and an \( n \)-variable 3-CNF formula with \( k \) clauses becomes a polynomial of degree \( 3m \) in \( n \) variables.

Note that if we substitute the values 0 or 1 (values from the field) into the polynomial, we get 0 or 1 exactly when substituting the same values (viewed as Boolean) into the Boolean formula results in a 0 (or 1), respectively. In other words, the values we compute with the polynomial are exactly the same as the values we compute with the Boolean expression, if we only evaluate the polynomial at 0-1 valued inputs.

The power of the technique comes from the fact that we can substitute other values, and this, as we’ll see, help us catch Merlin’s lies if he is dishonest.

We will now prove

**Theorem 4.** \( \#SAT_D \in IP \)

The proof consists of an \( IP \) protocol (and its proof of correctness).

**Proof.** A very high view of our protocol is the follows:

Given an \( n \)-variable 3-CNF Boolean formula \( \phi \) with \( m \) clauses, and integer \( k \), we perform the arithmetization described above, and obtain a polynomial \( P_\phi(X_1, X_2, \cdots X_m) \) of (total) degree \( 3m \). This polynomial evaluates to 1 whenever the 0-1 values assigned to the corresponding variables of the Boolean formula \( \phi \) make it equal to 1. So the number of satisfying assignments of \( \phi \) is given by the formula

\[
\sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_k \in \{0,1\}} \phi(b_1, b_2, \cdots b_k)
\]

The prover acts as follows:

First it generates a prime \( p \in (2^n, 2^{2n}) \) and sends it to the verifier \( V \).

The verifier verifies that \( p \) is indeed a prime. We will now work in the finite field \( \mathbb{Z}_p \) of the integers modulo \( p \).

\( P \) and \( V \) now run the SUMCHECK PROTOCOL.

This protocol has as input a polynomial \( P \), and an integer \( k \) and certifies true statements of the form of the sum above.

Note that \( p \) is big enough so that the results computed in \( \mathbb{Z}_p \) are the same as the results computed over the integers. Also, actually evaluating the sum is out of the question: if we have \( n \) nested sums, we have to make \( 2^n \) polynomial evaluations...
1.1 The Sumcheck Protocol

Input: integer $k$, polynomial $g(X_1, X_2, \cdots, X_n)$ of degree $d$ (polynomial in $n$).
We assume that given $n$ values $b_1, b_2, \cdots, b_n$, with $b_i \in \mathbb{Z}_p$, we can evaluate $g(b_1, b_2, \cdots, b_n)$ in polynomial time.

The protocol will guarantee that, with high probability,
either
\[
k = \sum_{b_1 \in \{0, 1\}} \sum_{b_2 \in \{0, 1\}} \cdots \sum_{b_n \in \{0, 1\}} g(b_1, b_2, \cdots, b_n)
\]
or, if the equality does not hold, we will discover this fact.

1.2 SUMCHECK PROTOCOL

checks the equality above.

for $n = 1$ check that $k = g(0) + g(1)$ [if not, reject]
for $n > 1$
ask the prover to ‘send over’ the polynomial $h(X_1) = \sum_{b_2 \in \{0, 1\}} \cdots \sum_{b_n \in \{0, 1\}} g(X_1, b_2, \cdots, b_n)$

Prover sends a polynomial $s(X_1)$ [comment: this is linear polynomial, so we can evaluate it. What we cannot do efficiently is to compute it when given as a sum of exponentially many monomials. If the prover follows the protocol, $s(x)$ is identical to $h(x)$, but $P$ can cheat!–and we have to be able to discover this.]

if $s(0) + s(1) \neq k$ reject. Otherwise, select $a \in \mathbb{Z}_p$ uniformly at random.
Use SUMCHECK PROTOCOL to check that
\[
s(a) = \sum_{b_2 \in \{0, 1\}} \cdots \sum_{b_n \in \{0, 1\}} g(a, b_2, \cdots, b_n)
\]

Lemma 5. Assume the verifier tries to prove
\[
s(a) = \sum_{b_1 \in \{0, 1\}} \sum_{b_2 \in \{0, 1\}} \cdots \sum_{b_n \in \{0, 1\}} g(a, b_2, \cdots, b_n)
\]

yet the equality does not hold. Then she will discover this with probability at least $(1 - d/p)^n$

Proof. We do induction on $n$, the number of nested sums.
If $n = 1$
V computes $g(0) + g(1)$ and verifies this is not $k$. The probability of error is 0.

Assume that for all $n - 1$-variable polynomials, a false claim is discovered with probability at least $(1 - d/p)^{n-1}$.

Consider what happens after $V$ received the polynomial $s()$. If the prover is cheating (in the overall protocol) if the polynomial $s$ that was returned is $h$, then the cheat will be revealed, as $h(0) + h(1) \neq k$. So $P$ must return a polynomial $s$ that is not $h$. Since the degree of the polynomials is at most $d$, $s(X_1) - h(X_1)$ has degree $d$ and has at most $d$ roots. If the randomly chosen $a$ is not a root, the prover will in the subsequent phases of the protocol have to prove the false statement

$$s(a) = \sum_{b_2 \in \{0, 1\}} \cdots \sum_{b_n \in \{0, 1\}} g(a, b_2, \cdots, b_n)$$

and by the induction hypothesis $V$ discovers the cheat with probability at least $(1 - d/p)^{n-1}$.

The probability that a random $a$ is not a root is $1 - d/p$, so the probability that the cheat is not discovered in the whole protocols is at least $(1 - d/p)(1 - d/p)^{n-1} = (1 - d/p)^n$.

Since $p > 2^n$ the expression $(1 - d/p)^n$ is well approximated by the first two terms of its Taylor expansion, namely $1 - dn/p$ which is exponentially close to 1.