It is not known how BPP and NP are related to each other. They both belong to PSPACE, however. (For BPP, we just enumerate all random strings of poly-size, counting how many of them are accepted by our randomized polytime algorithm. This can be done in polyspace.) Actually, even more is known about BPP.

**Theorem 1** (Sipser). $\text{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$

**Proof.** Since $\text{BPP} = \text{coBPP}$ (check this!), it suffices to prove that $\text{BPP} \subseteq \Sigma_2^p$.

We’ll follow the proof due to Lautemann. The idea is simple. Assume that a given $L \in \text{BPP}$ is decided by a probabilistic TM $M$ with error probability less than $2^{-n}$, where $n$ is the input size. As we saw in the last lecture, we can always reduce the error probability to be that low. Fix an input $x$ of size $n$. Consider the set of $A$ of random strings $r$ on which our TM $M$ accepts $x$, i.e., $A = \{r \mid M(x,r) \text{ accepts} \}$. Let $R$ be the set of all random strings $r$, for an input of size $n$.

There are two cases. Case I: $x \in L$. Then, by our assumption, $\frac{|A|}{|R|} > 1 - 2^{-n}$. We will consider translations of the set $A$: given a binary string $s$, where $|s| = |r|$, we define the set $A \oplus s = \{a \oplus s \mid a \in A\}$, where $a \oplus s$ means the bitwise XOR of the strings $a$ and $s$. We will argue that, since $A$ is big, there will be a small number of strings $s_1, \ldots, s_k$, for $k = |r|$, such that

$$R = \bigcup_{i=1}^{k} (A \oplus s_i),$$

i.e., translating the set $A$ for $k$ times will cover the entire set $R$.

**Claim 2.** If $|A|/|R| > 1 - 2^{-n}$, then there exist strings $s_1, \ldots, s_k$, for $k = |r|$, such that $R = \bigcup_{i=1}^{k} (A \oplus s_i)$.

**Proof of Claim.** The proof is an easy application of the **Probabilistic Method**. We’ll show that a random collection of $k$ strings will have the required property with non-zero probability, and so a desired collection of $s_i$’s certainly exists.
So, let’s pick a random sequence $s_1, \ldots, s_k$ uniformly and independently. Let $S = \cup_{i=1}^{k} (A \oplus s_i)$. For a fixed string $r \in R$, we have

$$\Pr[r \notin S] = \prod_{i=1}^{k} \Pr[r \oplus s_i \notin A] < (2^{-n})^k = 2^{-nk}.$$ 

Hence, applying the “union bound”,

$$\Pr[\exists r \in R \text{ such that } r \notin S] \leq |R|2^{-nk} = 2^k2^{-nk} \ll 1.$$ 

It follows that a randomly chosen sequence $s_1, \ldots, s_k$ is good with probability $1 - 2^{-nk+k} > 0$, and hence a good sequence exists.

Now, in case II: $x \notin L$. We have $|A|/|R| < 2^{-n}$. We claim that there is no sequence $s_1, \ldots, s_k$ such that $R = \cup_{i=1}^{k} A \oplus s_i$ in this case. The proof is very simple: $|\cup_{i=1}^{k} A \oplus s_i| \leq k|A| < \frac{k}{2^n}|R| \ll |R|$. So, we’ll never be able to cover the set $R$ by few “translated” copies of the small set $A$.

To summarize, what we proved above is the following: $x \in L$ iff there exist $s_1, \ldots, s_k$ such that for every $r \in R$ at least one of $M(x, r \oplus s_i)$ accepts. But this is exactly a $\Sigma_2^p$ formula. Hence, $L \in \Sigma_2^p$. \hfill \Box

### 2 Nondeterminism + randomization

#### 2.1 Class MA

Randomness is a computational resource that can be combined with other resources, e.g., nondeterminism. We will consider such a combination next.

Recall that a language $L \in \text{NP}$ if there is a polytime verifier such that, for every $x \in L$, there is a proof (or witness) of small size that makes the verifier accept. (The verifier is the polytime relation $R(x, y)$, where $x$ is the input, and $y$ is supposed to be a “proof” that $x \in L$.) In this definition, the verifier is a deterministic polytime algorithm. By letting the verifier be randomized polytime algorithm, we obtain an extension of $\text{NP}$, called $\text{MA}$ (which stands for Merlin-Arthur). Here, Merlin is an all-powerful wizard that tries to convince a probabilistic polytime verifier Arthur that an input string is in the language. If a string is indeed in the language, Merlin can make Arthur accept most of the time. If, on the other hand, the string is not in the language, then no matter what kind of proof Merlin shows to Arthur, Arthur will reject most of the time.

More formally, we say that a language $L \in \text{MA}$ if there is a constant $c$, and a polytime relation $R(x, y, z)$ such that, for every $x$ of length $n$, we have

$$x \in L \Rightarrow \exists y \Pr_z[R(x, y, z) = 1] \geq 3/4$$
$$x \notin L \Rightarrow \forall y \Pr_z[R(x, y, z) = 1] \leq 1/4,$$

where $|y| = |z| \leq nc$. In other words, if $x \in L$, then there is a short proof $y$ that will convince Arthur with probability at least 3/4, and if $x \notin L$, then every $y$ will be rejected by Arthur with probability at least 3/4.
2.2 Class AM

Suppose we change the order in which Merlin and Arthur communicate by letting Arthur go first, and Merlin go second. More formally, we say that a language \( L \in \text{AM} \) if there is a constant \( c \) and a polytime relation \( R(x, y, z) \) such that, for every \( x \) of length \( n \),

\[
\begin{align*}
    x \in L \Rightarrow \Pr_{z}[\exists y : R(x, y, z) = 1] & \geq \frac{3}{4}, \\
    x \not\in L \Rightarrow \Pr_{z}[\exists y : R(x, y, z) = 1] & \leq \frac{1}{4},
\end{align*}
\]

where \( |y| = |z| \leq n^c \).

In other words, if \( x \in L \), then Merlin can successfully answer with \( y \) almost every random challenge \( z \) from Arthur. If \( x \not\in L \), then, for most random challenges from Arthur, Merlin does not have a good answer.

While the class \( \text{MA} \) is a natural generalization of \( \text{NP} \) (where we allow randomized verifiers), the definition of \( \text{AM} \) does not appear very natural: why should we change the order of moves of the prover and verifier? How does it help?

We’ll see later that there is an \( \text{AM} \) protocol for deciding the problem Graph Non-Isomorphism: Given graphs \((G_1, G_2)\), accept iff \( G_1 \) and \( G_2 \) are not isomorphic.

Before seeing the \( \text{AM} \) protocol for this problem, we consider a different setup for interaction between the prover and the verifier where the verifier uses private randomness, i.e., the random coinflips that the verifier is not obliged to reveal to the prover.

2.3 Interactive Protocols

The interactive protocols (IP) are also protocols between a probabilistic polytime Verifier and an all-powerful Prover, where after a certain number of rounds of communication, the Verifier accepts or rejects. The difference from Arthur-Merlin protocols is that the Verifier in an Interactive Protocol does not reveal its randomness to the Prover. Sometimes, these protocols are called private-coin protocols, as opposed to public-coin Arthur-Merlin protocols.

In interactive protocols, the Verifier moves first. An IP protocol with \( k \) rounds of communication is the protocol where at most \( k \) messages are exchanged in a conversation between Verifier and Prover.

We say that a language \( L \in \text{IP}[k] \) if there is a probabilistic polytime verifier such that, for every \( x \in L \), there is a Prover that convinces the Verifier to accept with probability \( \geq 3/4 \) after at most \( k \)-rounds; and for every \( x \not\in L \), each Prover can convince Arthur to accept with probability at most \( 1/4 \) after \( k \) rounds.

Here’s an example of an IP protocol for the Graph Nonisomorphism Problem (NISO). Define \( NISO = \{(G_1, G_2) | G_1 \text{ and } G_2 \text{ are not isomorphic}\} \).

**Theorem 3.** \( NISO \in \text{IP}[2] \)

**Proof.** Here’s a protocol. Given an input \((G_1, G_2)\) of two graphs on \( n \) vertices, the Verifier will randomly pick \( i \in \{1, 2\} \), and a random permutation \( \pi \) of the set \( \{1, 2, \ldots, n\} \). The verifier will send \( \pi(G_i) \) to the Prover (i.e., the Verifier sends a randomly permuted copy of a randomly chosen graph in \((G_1, G_2)\)). The Prover sends back \( j \in \{1, 2\} \). The Verifier accepts iff \( j = i \).
Analysis:

• If $G_1$ and $G_2$ are non-isomorphic, the computationally unbounded Prover can always find a correct $j = i$ by checking which of $G_1$ and $G_2$ is isomorphic to the graph received from the Verifier. So, in this case, the Verifier can be made to accept with probability 1.

• If $G_1$ and $G_2$ are isomorphic, then a graph sent to the Prover by the Verifier in case $i = 1$ is from the same distribution as the graph sent in the case $i = 2$. Hence, the Prover has no way of determining $i$, and his $j$ will be equal to $i$ with probability $1/2$. (This can be shown formally after some simple probability calculations; it’s left as an exercise.)

So, if the graphs are non-isomorphic, the Verier accepts with probability 1. If the graph are isomorphic, the Verifier accepts with probability at most $1/2$. (It is possible to reduce the error probability to $1/4$.)