BPP

BPP is the class of languages which have probabilistic polynomial time Turing machines that decide them with a bounded error of at most $\frac{1}{3}$ in both the Yes and No instances of the language. This means that a BPP machine for a language when input a Yes instance accepts with probability at least $\frac{2}{3}$ and when input a No instance rejects with probability at least $\frac{2}{3}$. The probability is over the coins tossed by the BPP machine. Since it runs in polynomial time, it can only toss polynomially many coins. Thus, the probability is over a uniformly random string of polynomial length. One way to think about this is to build a computation tree for the BPP machine. Every time the machine tosses a coin, we branch in the tree, going left if the coin toss resulted in a Heads (or 0) and going right if the coin toss resulted in a Tails (or 1). This is a tree of height at most polynomial in the length of the input. Each path from the root of the tree to a leaf represent a possible computation path that was taken by the BPP machine. In this setting, we can think of the error constraints on the machine as follows: In the cases of both the Yes and No instances, at least $\frac{2}{3}$ of the paths from the root of the tree to a leaf must result in the right answer.

This gives us a way to think about non-deterministic machines and compare it with probabilistic machines as well. Every time the non-deterministic machine makes a non-deterministic guess, we branch in the tree, going left if the guess resulted in a 0 and going right if the guess resulted in a 1. This is a tree of height at most the running time of the machine. If particular, if we consider an NP machine, this is a tree of height at the most polynomial in the length of the input. Each path from the root of the tree to a leaf represent a possible computation path that was taken by the NP machine. In this setting, we can think of the constraints on the NP machine as follows: In the case of the Yes instances, at least 1 of the paths from the root of the tree to a leaf must result in the right answer, namely, accept. In the case of the No instances, all of the paths from the root of the tree to a leaf must result in the right answer, namely, reject.

From this exposition, we make the following observations:

- $\mathbf{P} \subseteq \mathbf{BPP}$. This is true is since $\mathbf{P}$, the error probability is 0. There are no coin tosses and thus there is only one leaf, namely the root itself, which always outputs the correct answer. Whether $\mathbf{P} = \mathbf{BPP}$ is an open question. This is the well-known problem of derandomization. Here, we are looking at correctness as a resource. Does allowing a bit of error suddenly allow us to solve more problems? The amplification lemma (repeat and take majority) lets us get the error down to $\frac{1}{2^{\text{poly}(n)}}$ while running in polynomial time, but not any further. The question as to whether the error can be got down to 0 remains open.

- $\mathbf{BPP}$ and $\mathbf{NP}$ (or $\mathbf{coNP}$) are incomparable. $\mathbf{BPP}$ makes tiny error on both sides, while $\mathbf{NP}$ (or $\mathbf{coNP}$) make (potentially) a lot of error on one side (rendering amplification useless) and no error on the other.

- $\mathbf{BPP} \subseteq \mathbf{PSPACE}$. One can go over the entire computation tree using polynomial space.
IP represents the class of languages which have interactive proof systems. The proof systems consist of a probabilistic polynomial time verifier, \( V \), a computationally unbounded prover, \( P \) and a polynomially long protocol \( \Pi \). The protocol involves rounds of communication between the prover and the verifier. The requirements from the protocol are as follows. Consider a language \( L \) and an instance \( x \). The verifier wishes to know whether \( x \in L \) or not. Informally, we would like that if \( x \in L \), then a prover should be able to convince the verifier of this fact, while if \( x \not\in L \), the prover should not be able to convince the verifier that \( x \in L \). Formally,

- If \( x \in L \), there exists a (honest) prover \( P \) such that \( P \) makes \( V \) accept \( x \) with probability at least \( \frac{2}{3} \).
- If \( x \not\in L \), there does not exist a (malicious) prover \( \tilde{P} \) such that \( \tilde{P} \) makes \( V \) accept \( x \) with probability more than \( \frac{1}{3} \).

From this exposition, we make the following observations:

- **P \subseteq IP.** In this case, the verifier does not need the prover. It can simply run the P algorithm. Notice here that there is no use of interaction or randomness.

- **BPP \subseteq IP.** In this case, the verifier does not need the prover. It can simply run the BPP algorithm. Notice here that there is no use of interaction.

- **NP \subseteq IP.** In this case, the prover can simply send over the NP certificate or witness to the verifier and the verifier can run the NP verification algorithm. In the case of the Yes instances, a certificate is always guaranteed to exist and thus an unbounded prover can find it and force the verifier to accept with probability 1. In the case of the No instances, there is no certificate that the prover can come up with that would make the verifier accept, just by the definition of NP. Thus, in the case of No instances, the verifier always rejects. Notice here that there is no use of randomness.

- **IP \subseteq PSPACE.** One can go over all possible transcripts of the protocol using polynomial space.

We will in fact learn that IP = PSPACE. The first non-trivial observation in this regard was showing that the problem of graph non-isomorphism, \( \text{NON-ISO} = \{ G, H : G, H \text{ are not isomorphic} \} \) is in IP. The protocol for NON-ISO is rather simple. The verifier picks one of \( G, H \) at random, scrambles (randomly permutes) it and sends it along with the prover, expecting the prover to answer with which of \( G, H \) the verifier chose to scramble and accepting if the prover succeeds (and equivalently, rejecting when the prover fails). It is easy to see that if \( G, H \) are indeed non-isomorphic, that is, constitute a Yes instance of NON-ISO, then the prover can determine, with probability 1, which of \( G, H \) the verifier chose to scramble, thus making the verifier accept with probability 1. If \( G, H \) are isomorphic, that is, constitute a No instance of NON-ISO, then, no matter what strategy the prover employs, the prover makes the right choice with probability exactly \( \frac{1}{2} \). This is because, if \( G, H \) are isomorphic, the scrambled graph the verifier sends over could have from \( G \) or \( H \) unconditionally with equal chance. Thus, in the case of the No instances, the prover can make the verifier accept with probability at most \( \frac{1}{2} \). Note that NON-ISO \( \in \text{coNP} \) and is not yet known.
to be in $P$. Also note that this protocol used both interaction and randomness. This gives us the sense that $IP$ could be a pretty expressive class.

The proof of $IP = PSPACE$ makes use of the technique of arithmetization which was used to show that $EQ_{ROBP} \in BPP$. We first show that $\#SAT = \{\phi, k : k = \#\phi\} \in IP$. The protocol involves the prover iteratively convincing the verifier of the number of satisfying assignments to $\phi$ with partial assignments and using the technique of arithmetization to compress the interaction. The proof does indeed make extensive use of randomness and interaction, thus exploiting the complete power of the class $IP$. This can be further generalized to show that $TQBF \in IP$, thus completing the proof of $IP = PSPACE$.

The State of Affairs

Refer back to the notes of Recitation 6 for topics covered prior to the midterm. Since then, we have been looking into the decidable world and studying time and space complexity. We have

$$
\begin{align*}
\text{REG} & \subset \text{CFL} \subset P \subset \text{NP} \subset \text{PSPACE} = \text{NPSPACE} \subset \text{EXP} \subset \text{EXPSPACE} \subset \text{Decid} \\
\text{REG} & \subset L \subset NL = \text{coNL} \subset L^2 \subset \text{PSPACE} \\
\text{NL} & \subset P \subset \text{NP} \cap \text{coNP} \\
\text{coNP} & \subset \text{PSPACE} \\
P & \subset \text{EXP} \\
\text{PSPACE} & = \text{NPSPACE} \subset \text{EXPSPACE}
\end{align*}
$$

We’ve also discussed randomness in computation. Here

$$
P \subseteq BPP \subseteq \text{PSPACE}
$$

The relationship between $BPP$ and $NP$ is unknown. A good way to think about this is the following: $NP$ algorithms produce no false positives but may produce a lot of false negatives (when the answer to the computation is yes, only one of the branches needs to accept). A $BPP$ algorithm on the other produced false negatives with a probability of only $\frac{1}{3}$ but it also produces false positives with the same probability.

Finally, we looked at interactive protocols and observed that

$$
IP = PSPACE
$$

Highlights

- **Cook-Levin Theorem**
  $\text{SAT}$ is $NP$-complete.

- **Savitch’s Theorem**
  For any function $f = \Omega(\log n)$,

$$
\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))
$$
• **Immerman-Szelepcsényi Theorem**
  For any function $f = \Omega(\log n)$,
  \[
  \text{NSPACE}(f(n)) = \text{coNSPACE}(f(n))
  \]

• **Time Hierarchy Theorem**
  For any time-constructible\(^1\) function $f$,
  \[
  \text{TIME}\left(o\left(\frac{f(n)}{\log f(n)}\right)\right) \subset \text{TIME}(f(n))
  \]

• **Space Hierarchy Theorem**
  For any space-constructible function\(^2\) $f$,
  \[
  \text{SPACE}(o(f(n))) \subset \text{SPACE}(f(n))
  \]

**Some complete problems**

- **NL, coNL**: PATH, PATH
- **NP**: SAT, 3SAT, HAMPATH, CLIQUE, INDSET, VERTEXCOVER, 3COLOR, SUBSUM
- **PSPACE, NPSPACE**: TQBF, GENEO
- **EXPSPACE**: EQ\(_{\text{REX}}\)\(\dagger\)

**Methodologies**

- **L**
  - To show membership: Logspace DTM; you should be able to solve the problem with a fixed number of pointers into the input, counters, simple arithmetic and (possibly extensive) reuse of space; closure properties
  - To show non-membership: Reduction from a problem complete for a class known to be larger than L, closure properties
  - To postulate non-membership: Reduction from a problem complete for a class postulated to be different than L, closure properties

- **NL, coNL**
  - To show membership: Logspace DTM; you should be able to solve the problem with a fixed number of pointers into the input, counters, simple arithmetic, non-determinism and (possibly extensive) reuse of space; recall that NL = coNL so you may also attempt to solve the complement problem; closure properties

---

\(^1\)A function $f$ is called time-constructible if there exists a Turing machine which, given a string $1^n$, outputs the binary representation of $f(n)$ in $O(f(n))$ time.

\(^2\)A function $f$ is called space-constructible if there exists a Turing machine which, given a string $1^n$, outputs the binary representation of $f(n)$ while using $O(f(n))$ space.
To show non-membership: Reduction from a problem complete for a class known to be larger than NL, coNL; closure properties

To postulate non-membership: Reduction from a problem complete for a class postulated to be different than NL, coNL; closure properties

• P

To show membership: Polynomial time DTM, closure properties

To show non-membership: Reduction from a problem complete for a class known to be larger than P, closure properties

To postulate non-membership: Reduction from a problem complete for a class postulated to be different than P, closure properties

• NP

To show membership: Polynomial time NTM; the problem must be of the form ∃x : π(x) where π is a predicate that can be verified deterministically in polynomial time, then x is a witness for membership that can be guessed and then verified. Another way to view it is that the NTM you construct must accept on at least one branch if the input is in the language and reject on all branches if the input is not in the language; closure properties

To show non-membership: Reduction from a problem complete for a class known to be larger than NP, closure properties

To postulate non-membership: Reduction from a problem complete for a class postulated to be different than NP, closure properties

• coNP

To show membership: Polynomial time NTM for the complement; the problem must be of the form ∀x : π(x) where π is a predicate that can be verified deterministically in polynomial time, then there is no witness x for the predicate ¬π. Another way to view it is that the NTM you construct must accept on all branches if the input is in the language and reject on at least one branch if the input is not in the language; closure properties

To show non-membership: Reduction from a problem complete for a class known to be larger than coNP, closure properties

To postulate non-membership: Reduction from a problem complete for a class postulated to be different than coNP, closure properties

• PSPACE, NPSPACE

To show membership: Polynomial space (N)TM; several computations in (N)P that reuse space; closure properties

To show non-membership: Reduction from a problem complete for a class known to be larger than PSPACE, NPSPACE; closure properties
To postulate non-membership: Reduction from a problem complete for a class postulated to be different than $\text{PSPACE}$, $\text{NPSPACE}$; closure properties

**EXP**
- To show membership: Exponential time DTM; exhaustive search; closure properties
- To show non-membership: Reduction from a problem complete for a class known to be larger than EXP, closure properties
- To postulate non-membership: Reduction from a problem complete for a class postulated to be different than EXP, closure properties

**EXPSPACE**
- To show membership: Exponential space DTM; several exhaustive searches that reuse space; closure properties
- To show non-membership: Reduction from a problem complete for a class known to be larger than EXPSPACE, closure properties
- To postulate non-membership: Reduction from a problem complete for a class postulated to be different than EXPSPACE, closure properties

- To show the existence of languages outside certain classes, use the hierarchy theorems.

**Techniques**

- **Gadgets:** In coming up with reductions for NP-completeness, etc., it is useful (especially in graph problems) to develop gadgets for each part of the problem. For instance, if you are working with a formula, you could come up with gadgets for variables and clauses. If you were working with a graph, you could come up with gadgets for nodes and edges.

- **Configuration graphs:** We may model the computation of a Turing machine $M$ on an input $w$ by a configuration graph $G_{M,w}$ whose nodes are all the configurations of $M$ and there is a directed edge from one node to another if the latter configuration may be reached from the former by a single step of $M$. We may also think of the initial configuration as the start node and final configurations as target nodes. Furthermore, this graph can be computed on the fly in the sense that given any node, one can determine its neighbors in logspace.

- **Amplification Lemma:** We used this to show that the constant $\frac{1}{3}$ in the definition of $\text{BPP}$ is arbitrary, one can choose any constant less than $\frac{1}{2}$ and in fact something as small as $\frac{1}{2\text{poly}(n)}$.

- **Arithmetization:** We used this technique to show that $\text{EQ} \text{ROBP} \in \text{BPP}$ and $\text{IP} = \text{PSPACE}$. 

Closure Properties

<table>
<thead>
<tr>
<th>Class of Languages</th>
<th>$\rightarrow$</th>
<th>$*$</th>
<th>$\cap$</th>
<th>$\cup$</th>
<th>$\cap$</th>
<th>$\circ$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>Y</td>
<td>?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NL = coNL</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>$P$</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NP</td>
<td>?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>?</td>
</tr>
<tr>
<td>coNP</td>
<td>?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>?</td>
</tr>
<tr>
<td>BPP</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>IP = PSPACE = NPSPACE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>EXP</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>EXPSPACE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Remarks:

- In the table above, $Y = yes$, $N = no$
- General set-theory: $A \setminus B = A \cap \overline{B}$; $\overline{A \cup B} = \overline{A} \cap \overline{B}$; $\overline{A \cap B} = \overline{A} \cup \overline{B}$
- (Problem Set 6, Problem 7) $L$ is closed under $*$ if and only if $L = NL$.
- Recall the use of dynamic programming as a technique to show closure under $*$.

Reminders

- You may cite results that have been proved in the textbook/recitation and do not need to reprove them. For instance, for Problem 4 in Problem Set 6 where we are given a SAT oracle and asked to solve CLIQUE, we know that CLIQUE can be reduced to SAT (by Cook-Levin) since CLIQUE is in NP and SAT is NP-complete. One need not write down the explicit reduction in this case.
- When showing a problem is complete for a class, remember that you still have to prove membership of the problem in the class aside from showing that all problems in the class are reducible to the problem.

An Educational Game

ROUND I : RAPID FIRE

You are given two grammars $G_1$ and $G_2$. Which of these problems are decidable and which undecidable?

1. Does there exist a string $w$ generated by neither of them?
   Undecidable. Hint. This is equivalent to testing that $L(G_1) \cup L(G_2) = \Sigma^*$. 

2. Does there exist a string $w$ generated by either of them?
   Decidable, in $P$. Hint. This is equivalent to testing that $L(G_1) \cup L(G_2) = \emptyset$. 

3. Does there exist a string \( w \) generated by exactly one of them?
   \textbf{Undecidable.} \textit{Hint. This is equivalent to testing that} \( L(G_1) = L(G_2) \).

4. Does there exist a string \( w \) generated by both of them?
   \textbf{Undecidable.} \textit{Hint. This is equivalent to testing that} \( L(G_1) \cap L(G_2) = \emptyset \). \textit{Valid computation histories can be expressed as an intersection of context-free languages.}

\textbf{ROUND II : PICTONARY}

You will be given a language. Your goal is to get your team to guess the language. You must express a \textbf{minimal} decider for the language. For instance, if the language is regular, you may right down a regular expression or draw a DFA or NFA. If it is context-free but not regular, you may write a grammar or a a description of a PDA. If it is decidable but not context-free, you may write down the description of a decider.

1. \{0\(^i\)1\(^j\) : \(i + j\) is prime\}
   \textit{Hint. This language is not context-free, but primality can be tested in} \( P \).

2. \{0\(^i\)1\(^j\) : \(i + j\) is even\}
   \textit{Hint. This language is regular.}

3. \{0\(^i\)1\(^j\) : \(i - j\) is even and positive\}
   \textit{Hint. This language is not regular, but context free.}

4. \{0\(^i\)1\(^j\) : \(|i - j|\) is even\}
   \textit{Hint. Same as 2.}

\textbf{ROUND III : THROWING DARTS}

Place the following problems in the most appropriate class among the ones you have seen.

1. Given a finite \( S \subseteq \mathbb{Z} \) and a \( T \in \mathbb{Z} \), does there exist a sextuplet in \( S \) that sums to \( T \)?
   \( P \). \textit{Hint. Try all sextuplets.}

2. Given a finite \( S \subseteq \mathbb{Z} \) do all subsets of \( S \) have non-zero sum?
   \( \text{coNP} \). \textit{Hint. Testing for a subset of zero sum is NP-complete.}

3. Given a finite \( S \subseteq \mathbb{Z} \), do there exist two sextuplets in \( S \) with the same sum?
   \( P \). \textit{Hint. Try all sextuplets.}

4. Given a finite \( S \subseteq \mathbb{Z} \), do there exist two disjoint subsets in \( S \) whose sums have the same magnitude but opposite signs?
   \( \text{NP} \). \textit{Hint. This is equivalent to testing for a subset of zero sum.}

\textbf{ROUND IV : ARRANGING PLATES}

Arrange the following classes in order of containment and specify when it is known that the containment is strict.

1. \( \text{REG, NP, EXP} \)
   \( \text{REG} \subseteq \text{NP} \subseteq \text{EXP} \)
2. coNP, PSPACE, Decid
   
   \[ \text{coNP} \subseteq \text{PSPACE} \subseteq \text{Decid} \]

3. NL, NONCFL, EXPSPACE, where NONCFL is the set of all languages that are not CFLs.
   
   \[ \text{NL} \subseteq \text{EXPSPACE}. \quad \text{Hint. There is a language in L, and hence NL, that is a member of NONCFL, however there are also members of NL that are not in NONCFL. Similarly, there are members of NONCFL that are not in NL. Furthermore, there is a language in NONCFL that is a member of EXPSPACE, however there are also members of NONCFL that are not in EXPSPACE. Similarly, there are members of EXPSPACE that are not in NONCFL.} \]

4. BPP, NEXPSPACE, EEXP, where

   \[ \text{EEXP} = \bigcup_{k \in \mathbb{N}} \text{TIME}(2^{2^{nk}}) \]

   \[ \text{BPP} \subseteq \text{NEXPSPACE} \subseteq \text{EEXP}. \quad \text{Hint. NEXPSPACE = EXPSPACE by Savitch’s Theorem. EXPSPACE \subseteq EEXP using a simple simulation.} \]

**ROUND V : BACK TO THE CHALKBOARD**

1. Show that NP \neq \text{SPACE}(n).
   
   \[ \text{Hint. Padding!} \]

2. How hard is the following problem: Given the description of a Turing machine \( M \) and a string \( w \), how much space does \( M \) use while running on \( w \)?

   \[ \text{Uncomputable. Hint. If we knew how much space} \ M \ \text{used while running on} \ w, \ \text{we would know how long} \ M \ \text{took to run on} \ w. \]