Oracle and BPP

In this recitation, we covered the definitions of oracle Turing machines and probabilistic Turing machines, as well as the complexity classes \( P^A \), \( NP^A \), \( PSPACE^A \), and \( BPP \). We discussed the relationships between these classes and other complexity classes, and explained how to design algorithms for oracle Turing machines.

Oracle Turing Machine

Oracle Turing machine is a new computation model that extends the standard Turing machine model. To introduce this model, we will discuss its definition and the intuition behind its computational strength. We will also introduce reasonable complexity classes under this model and their relationships with existing complexity classes, and discuss methods for designing algorithms in the oracle setting.

Definitions and intuitions of Oracle Turing Machines

**Definition 1.** A deterministic Turing Machine with \( A \) oracle \( M^A \) is a deterministic Turing machine, equipped with an additional oracle tape. This tape allows the machine to read and write just as the work tape. It also has an extra transition: let the content of oracle type be \( x \), then it can "ask" the oracle whether \( x \in A \), replacing the content of oracle tape with the answer.

Similarly, a nondeterministic Turing Machine with \( A \) oracle \( N^A \) is a nondeterministic Turing machine, with the same oracle tape and extra transition as the deterministic version.

**Intuition** Intuitively, a (nondeterministic) Turing machine with \( A \) oracle is a (nondeterministic) Turing machine, that has the additional ability to determine whether a given string is in the language \( A \) by invoking the oracle.
Several Complexity Classes: \(P^A, NP^A, PSPACE^A\)

With the definition of oracle Turing Machines, we can define classes \(P^A, NP^A, PSPACE^A\).

**Definition 2.** \(P^A\) consists of all the languages decided by some deterministic Turing machine with an \(A\) oracle in polynomial time. \(NP^A\) consists of all the languages decided by some nondeterministic Turing machine with an \(A\) oracle in polynomial time. \(PSPACE^A\) consists of all the languages decided by some deterministic Turing machine with an \(A\) oracle using polynomial space.

**Examples.** \(PSAT\) consists of all the languages that can be decided by some deterministic poly-time Turing Machine with a \(SAT\) oracle. \(NP_{SAT}\) consists of all the languages that can be decided by some poly-time nondeterministic Turing Machine with a \(SAT\) oracle.

**Some relationships between oracle complexity classes and existing classes**

The following theorem illustrates an interesting relationship between the complexity classes \(P^{TQBF}, NP^{TQBF}, PSPACE^{TQBF}\) and \(PSPACE\).

**Theorem 3.** \(P^{TQBF} = NP^{TQBF} = PSPACE^{TQBF} = PSPACE\)

**Proof.** First, note that \(P^{TQBF} \subseteq NP^{TQBF} \subseteq PSPACE^{TQBF}\), since we can simulate the poly-time TM using a poly-time NTM with the same oracle. Similarly, we can enumerate all the branches of a poly-time NTM using a poly-space TM with the same oracle.

Second, we want to show that \(PSPACE \subseteq P^{TQBF}\). For any \(L \in PSPACE\), \(L \leq_p TQBF\) since TQBF is PSPACE complete. We can build poly-time deterministic Turing machine \(M^{TQBF}\) deciding \(L\) as follows:

- On instance \(x\) of \(L\), compute \(f(x)\) where \(f\) is the reduction function from \(L\) to \(TQBF\).
- Use the \(TQBF\) oracle to check if \(f(x) \in TQBF\), accepts if true.

The correctness follows from the fact that \(x \in L\) if and only if \(f(x) \in TQBF\). Therefore, \(L \in P^{TQBF}\), and we have that \(PSPACE \subseteq P^{TQBF}\).

Third, it’s easy to see \(PSPACE^{TQBF} \subseteq PSPACE\), since we can simulate the poly-space TM with TQBF oracle using poly-space by computing the actual answers of oracle queries.

In conclusion, \(PSPACE \subseteq P^{TQBF} \subseteq NP^{TQBF} \subseteq PSPACE^{TQBF} \subseteq PSPACE\), so all the classes are equal.

\(\square\)

**Remark.** Essentially, the part of the proof that shows \(P^{TQBF} \subseteq NP^{TQBF} \subseteq PSPACE^{TQBF}\) relies on the fact that the inclusion relationships \(P \subseteq NP \subseteq PSPACE\) is preserved when the same oracle is added.
to the classes. This is known as the property of relativization. The diagonalization proof of time hierarchy theorem from lectures also has this property of relativization. However, Theorem 3 implies that the relationship \( P \neq NP \) (if true) does not have this property, and therefore cannot be proved using a diagonalization argument that can relativize.

**Design algorithms for oracle TM**

Designing algorithms for \( P^A \) and \( PSPACE^A \) is relatively straightforward. For \( NP^A \), the old paradigm of designing algorithms still applies: we can come up with a certificate that can be quickly verified by a checker. The only difference is that the checker now has the ability to query the \( A \) oracle. To use this approach, we can simply guess a certificate and then use the checker to verify its correctness using the \( A \) oracle. Here is an example of this process in action.

**Proposition 4.** \( \text{MinFormula} = \{ B | B \text{ is a boolean formula, such that there exists a shorter boolean formula } B_1 \text{ equivalent to } B \} \).

**Proof strategy.** One possible certificate for the language \( \text{MinFormula} \) is the shorter boolean formula, \( B_1 \). To check whether \( B \) and \( B_1 \) are equivalent, we can use the SAT oracle to determine whether the formula \((B \land \neg B_1) \lor (\neg B \land B_1)\) is satisfiable. If this formula is unsatisfiable, then \( B \) and \( B_1 \) are equivalent, and the certificate is valid.

**Proof.** We can build a nondeterministic Turing machine with a SAT oracle, denoted \( \text{NSAT} \), to solve the language \( \text{MinFormula} \) as follows:

- Given a boolean formula \( B \) with variables \( x_1, \ldots, x_n \), the machine nondeterministically guesses a boolean formula \( B_1 \) on the same variables such that \(|B_1| < |B|\).

- The machine uses the SAT oracle to check whether \( B_1 \) is equivalent to \( B \). This can be done by determining whether the formula \((B \land \neg B_1) \lor (\neg B \land B_1)\) is satisfiable. If this formula is unsatisfiable, then \( B_1 \) and \( B \) are equivalent, and the machine accepts.

\( \square \)

**Probabilistic Turing Machine and BPP**

**Probabilistic Turing Machine**

A probabilistic Turing machine is a variation of the standard Turing machine model that allows for probabilistic choices during the computation. Here is a formal definition of this computational model.
Definitions and intuitions of Oracle Turing Machines

Definition 5. A probabilistic Turing machine (PTM) $P$ is a nondeterministic Turing machine in which each computation step has one or two possible choices. The probability $\Pr[\text{branch } B]$ of a given branch $B$ is defined as $2^{-k}$, where $k$ is the number of coin flips made during the computation of that branch. The overall probability of $P$ to accept a given string $x$ is defined as the sum of the probabilities of all branches that accept $x$.

We say that a $P$ decides a language $L$ with error $\epsilon$ if, for any string $w$, the probability that $P$ produces the incorrect answer on $w$ is no greater than $\epsilon$.

Intuition Intuitively, a PTM is a Turing Machine with the access to random coin flips.

Complexity Class: BPP

With the definition of PTM, we can define the complexity class $BPP$.

Definition 6. The complexity class $BPP$ is defined as the set of all languages that can be decided by a PTM in polynomial time with error probability at most $1/3$.

The lemma below allows us to amplify $1/3$ to $1/2^{-p(n)}$ for any polynomial $p$.

Lemma 7. (Amplification lemma) If $M_1$ is a poly-time PTM with error $\epsilon_1 < 1/2$, $p$ is a polynomial function, then there is an equivalent poly-time PTM $M_2$ with error $< 2^{-p(n)}$.

Proof idea. We can run $M_1$ for polynomial many times, then pick the majority answer as the output. The details require some bounds on probabilities (e.g. Chernoff bound), hence is omitted here.

The following corollary of amplification lemma is often useful for bounding error probabilities.

Corollary 8. For any language $L \in BPP$, for any polynomial $p$, there exists poly-time PTM $P$ decides $L$ with error $\leq 2^{-p(n)}$.

Proof. By definition, there exists poly-time PTM $M$ decides $L$ with error $\leq 1/3$. By lemma 7, we can pick equivalent poly-time PTM $P$ deciding $L$ with error $\leq 2^{-p(n)}$.

Some relationships between BPP and existing classes

The following theorem illustrates the relationship between $BPP$, $P$ and $PSPACE$. 

Here, $n$ stands for the input length.
Theorem 9. $P \subseteq BPP \subseteq PSPACE$

Proof. $P \subseteq BPP$ is easy, since the deterministic poly-time TM is a PTM with 0 error.

For a poly-time PTM $M$ deciding language $L$ with error $\leq 1/3$, we can construct a deterministic Turing machine $N$ that also decides $L$ and runs in polynomial space. This can be done by enumerating all possible branches of $M$ and computing the accept probability. Since $N$ runs in polynomial space, we have $BPP \subseteq PSPACE$.

**Remark** The Sipser–Lautemann theorem states that $BPP \subseteq NP^{SAT} \cap CoNP^{SAT}$, which implies that $BPP$ is contained within the polynomial hierarchy. There is evidence to suggest that $BPP$ and $P$ might be equal, however, even the relationship between the complexity class $BPP$ and the classes $NP$ and $CoNP$ stays open.

**Designing algorithms in BPP**

Many powerful algorithms belong to the complexity class $BPP$. One example is the arithmetization method, which we will discuss in more detail in the next recitation.

**Summary**

In this recitation, we introduced two new computation models: oracle Turing machines and probabilistic Turing machines (PTMs). Oracle Turing machines are able to query a specific language, while PTMs have access to coin flips.

We discussed how the oracle setting can be used to define the classes $P^A$, $NP^A$, and $PSPACE^A$, and showed that when $A = TQBF$, all three classes are equal to $PSPACE$. We also defined the class $BPP$ in the PTM setting, which lies between $P$ and $PSPACE$.

Designing algorithms for $NP^A$ follows the standard certificate-checking paradigm, while designing algorithms for $BPP$ often involves clever uses of randomness, which we will discuss in more detail in future lectures.