15. Alternation vs. Oracles: A Characterization of the Polynomial Hierarchy

Goal:
- Introduce oracle Turing machines and relativized complexity classes.
- Provide a characterization of the polynomial hierarchy.

15.1 Oracle Machines and Relativized Complexity Classes

Idea:
- Oracle Turing machines are defined like Turing machines but they are allowed to invoke other problems as an oracle.
- The oracle is a means of answering membership questions about some set B, say SAT.
- This invocation to B is counted as a single step, which means we study computation *relative* to free knowledge about B.

So we get a complexity like NP^{SAT}
- NP decider with oracle access to SAT
- Problems in NP, *relative* to free knowledge about SAT.

Definition:
- An oracle Turing machine with oracle set B is a Turing machine with
  - an additional oracle query tape,
  - a special oracle query state, and
  - a yes-state and a no-state.
- When the machine enters the oracle query state with string y written on the oracle query tape, the machine magically moves
  - to the yes-state if $y \in B$,
  - to the no-state if $y \notin B$,

  and clears the oracle query tape.

Remark:

- There is no explicit representation of B.
- The oracle must give the same answer when queried with y again in the future.
- The same oracle machine may be used with different oracles B. The result of the queries will usually be different

Alternative Definition of Oracle Machines:

- An oracle TM can also be defined as a TM with a right-infinite, read-only input tape on which an infinite string is written.
- This type is called the oracle type, the string is the oracle.
- The machine can make decisions based on the symbols written on the oracle tape.

Understanding the alternative definition:

- We think of the oracle as a specification of a set $B \subseteq \mathbb{N}$.
- If the oracle is an infinite string over $\{0,1\}$, we can see it as the characteristic function of B:

$$n \in B \quad \text{iff} \quad n\text{th bit of the oracle string is } 1.$$

Definition (Relativized Complexity Classes):

Let B be a set and let $\mathcal{C}$ be a complexity class.

Define

- $P^B := \{ L(M) \mid M \text{ is a deterministic, polynomial-time-bounded oracle Turing machine with oracle } B \}$

- $NP^B := \{ L(M) \mid M$ is a non-deterministic, polynomial-time-bounded oracle Turing machine with oracle $B \}$

- $P^{\mathcal{C}} := \bigcup_{B \in \mathcal{C}} P^B$ - $NP^{\mathcal{C}} := \bigcup_{B \in \mathcal{C}} NP^B$.

In a similar fashion, one can define other relativized complexity classes, like $PSPACE^{\mathcal{C}}$.

Lemma:
If B is $\leq_m^p$-complete for $\mathcal{C}$,
then $P^{\mathcal{C}} = P^B$ and $NP^{\mathcal{C}} = NP^B$.

Proof:
- The polynomial-time reduction from some $A \in \mathcal{C}$ to B can be computed before the oracle query is made.
- Indeed, performing a polynomial-time computation (the reduction) a polynomial number of times (the computation time of A (with oracle A)) yields another polynomial.

Examples:
- $P^{NP} = P^{SAT}$, $NP^{NP} = NP^{SAT}$, by the lemma.
- $P^P = P$, by the argument in the proof of the lemma.
- $NP \subseteq P^{SAT}$ and co-$NP \subseteq P^{SAT}$, because P^{SAT} is closed under complement (it is a deterministic complexity class).
- If $P^{SAT} = NP$, then $NP =$ co-NP, by the previous statement.

Oracle machines are a proper formalization of the concept of Turing reducibility (mentioned earlier, problem you reduce to can be invoked multiple times).

Definition:
A problem A is polynomial-time Turing reducible to B, $A \leq_T^P B$, if $A \in P^B$.

Lemma:
- $A \leq_m^P B$ implies $A \leq_T^P B$.
- If the reverse would hold, then $NP = co\text{-}NP$.
- $SAT \leq_T^P \overline{SAT}$.

15.2 A Characterization of the Polynomial Hierarchy

Goal: Characterize the levels of the polynomial hierarchy in terms of relativized complexity classes:

$$NP \subseteq NP^{NP} \subseteq NP^{NP^{NP}} \subseteq \dots$$

Definition:
Let $NP_1 := NP$ and $NP_{k+1} := NP^{NP_k}$ for all $k \geq 1$.

Theorem:
$NP_k = \Sigma_k^P$ for all $k \geq 1$.

Note that
- the left-hand side of the equation refers to oracle machines
whereas
- the right-hand side talks about alternation.

Proof: We proceed by induction on k.

Base case $k=1$: $NP_1 = NP = \Sigma_1^P$.

Induction step $k \rightsquigarrow k+1$: We assume that $NP_k = \Sigma_k^P$ and show that

$$NP_{k+1} = NP^{NP_k} = \Sigma_{k+1}^P.$$

With an application of the hypothesis, this means we have to show

$$NP^{\Sigma_k^P} = \Sigma_{k+1}^P.$$

The two inclusions are done separately.

$\supseteq$: Assume we have a $\Sigma_{k+1}^P - NTM$ M running in time n^c.
We have to show $L(M) \in NP^{\Sigma_k^P}$.

- Let the configurations of M be encoded as strings over Δ.
The configurations reachable from input x of length n can be represented as strings in Δ^{n^c}.
Indeed, we use $\binom{a}{p}$ if the head is at letter a and the machine is in state p.

- Consider the set

$$D := \{ \alpha \mid \alpha \text{ is an } \wedge\text{-configuration of } M,\ |\alpha| = n^c,\ \alpha \text{ leads to acceptance via a } \Pi_k^P\text{-computation in at most } n^c \text{ steps} \}$$

Then $D \in \Pi_k^P$ and so $\overline{D} \in \Sigma_k^P$.
The polynomial-time-bounded $\Pi_k^P - NTM$ that decides D simulates M starting from α.

- With the definition of D, we have
M accepts x iff $\exists$ computation from the initial configuration through $\vee$-configurations to some $\alpha \in D$.

This means $L(M)$ can be accepted
by a non-deterministic polynomial-time-bounded oracle machine M'
with oracle $\bar{D}$.
On input x, M' guesses the computation from the initial configuration
through only ν-configurations to some λ-configuration α.
Then M' consults the oracle to check $\alpha \in \bar{D}$.
The oracle gives a yes-no-answer that M' inverts.

$\subseteq$: For this inclusion, we consider an n^c-time-bounded
non-deterministic oracle machine with oracle $B \in \Sigma_k^P$.
The task is to show that $L(M) \in \Sigma_{k+1}^P$.

- We build a Σ_{k+1}^P-NTM M' as follows.
On input x, M' simulates M except for the oracle queries.
Whenever M queries the oracle with string y,

M' guesses the answer that the oracle would return
(yes if $y \in B$, no if $y \notin B$)

and stores the pair (y, answer_y).
M' continues the simulation until it arrives
at an accept or at a reject state.
This must happen after n^c steps (otherwise we force termination
with a reject).
If M wants to reject, M' rejects.
If M wants to accept, M' has to
verify the guessed oracle answers.

- The computation tree of M' so far coincides with the computation tree of M — except for the non-deterministic branching at oracle queries.
All branches so far are existential.
At each leaf, there is a list of oracle queries and responses that have to be verified.
The list is different for each path.
Why? Later queries may depend on responses to earlier queries.
All lists are at most n^c long.

- A list has the shape
$y_1 \dots y_m$ for positively evaluated queries
and $z_1 \dots z_\ell$ for negatively evaluated queries.
The combined length of y_1 to z_ℓ is $\leq n^c$,
because the machine had to write them onto the oracle tape.
The task is now
to verify with a Σ^p_{k+1}-computation in polynomial time
that $y_i \in B$ for $1 \leq i \leq m$
and $z_j \notin B$ for $1 \leq j \leq \ell$.
This Σ^p_{k+1}-computation combined with the previous Σ^p_1-computation
still yields a Σ^p_{k+1}-computation.
So we have reduced the problem to showing that
for $B \in \Sigma^p_k$, the set
$$\{ y_1 \# \dots \# y_m \# \# z_1 \# \dots \# z_\ell \mid y_i \in B,\ 1 \leq i \leq m,\ z_j \notin B,\ 1 \leq j \leq \ell \}$$
is in Σ^p_{k+1}.

Note that $y_i \in B$ can be verified with a Σ_k^P-computation and $z_j \notin B$ can be verified with a Π_k^P-computation.

First idea:

- Do an $\wedge$-branch of degree $n+l$.
  Each process verifies (for some i or j) that
  - ↳ $y_i \in B$ with a Σ_k^P-computation and
  - ↳ $z_j \notin B$ with a Π_k^P-computation.
- This does not work:
  - ↳ The $\wedge$-branching + Σ_k^P results in a Π_{k+1}^P-computation.
  - ↳ Π_{k+1}^P cannot in general be simulated by Σ_{k+1}^P.

Better idea:

- Let n^d be the time bound on a Σ_k^P-machine for B.
  The key idea is to
  not only guess the answer to y_i
  but also a <u>certificate</u> for the first round of existential guessing
  in the Σ_k^P-computation for $y \in B$.
  Formally, the certificate is a binary string (assume binary branching)
  v_i of length n^d.
  Guessing the strings $w_1 \dots w_m$ takes time
  $m \cdot n^d \leq n^c \cdot n^d = n^{c+d}$.
- With this, we are prepared to do the simulation.
  We do an $(n+l)$-branch,
  each process getting
  - (y_i, w_i) to check for $y_i \in B$ or
  - z_j to check for $z_j \notin B$.

So far, the computation is Σ_2^P.

↳ For z_j, we check $z_j \notin B$ by a Π_k^P-computation for $\overline{B}$.
Combined with the Σ_2^P-computation, this yields Σ_{k+1}^P.

↳ For (y_i, w_i), we simulate the Σ_k^P-computation
checking $y_i \in B$.
The guessed w_i guides the first level of existential branching,
thus making this level deterministic.
The remaining computation is $\overline{\Pi}_{k-1}^P$.

The total computation is Σ_{k+1}^P. □

Skolemization:

- The key trick in the above proof is the following identity:

$$\bigwedge_{i \in A} \bigvee_{w \in B} C(i,w) \iff \bigvee_{f: A \to B} \bigwedge_{i \in A} C(i, f(i)),$$

which can be understood as a distributive law of Boolean algebra.

- In our setting:

$A = \underbrace{\{1, \dots, m\}}_{\text{the } y_i}$ $\qquad B = \underbrace{\{0,1\}^{n^d}}_{\text{the certificates}}$

$C(i,w)$ states that
y_i is accepted by a Π_{k-1}^P machine
that simulates the Σ_k^P-computation of B
but uses w to direct
the first level of non-deterministic choices.

- Usually, Skolemization yields an exponential blow-up in the size of the formula:

$|A| = n$ and $|B| = n$ gives $|A \to B| = n^n$.

Here, it works as

$|A| = n \leq n^c$, $|B| = 2^{n^d}$,

and so $|A \to B| = (2^{n^d})^{n^c} \leq (2^{n^d})^{n^c}$

$= 2^{n^d \cdot n^c} = 2^{n^{d+c}}$.

15.3 A Logical Characterization of the Polynomial Hierarchy

Goal: Provide a third characterization of the polynomial hierarchy.

Idea: Use certificates (like for NP).

Note: The size of the certificates has to be bounded by a polynomial.

Definition: Let $t \in \mathbb{N}$.

$\exists^t y : \mathcal{C}(y) \quad :\Leftrightarrow \quad \exists y : |y| \leq t \wedge \mathcal{C}(y)$

$\forall^t y : \mathcal{C}(y) \quad :\Leftrightarrow \quad \forall y : |y| \leq t \to \mathcal{C}(y)$.

Theorem:
The set A is in Σ_k^P if and only if there is a deterministic, polynomial-time computable $(k+1)$-ary predicate R and a constant c:

$$A = \{ x \mid \exists^{|x|^c} y_1 \, \forall^{|x|^c} y_2 \, \exists^{|x|^c} y_3 \ldots Q^{|x|^c} y_k : R(x, y_1, \ldots, y_k) \}.$$

Here, $Q = \exists$ if k is odd, $Q = \forall$ if k is even.