Quantum Algorithms
Lecture 7
The hierarchy of complexity classes

Zhejiang University

Complements of languages

Definition

Recall that we identify languages (sets of strings) and predicates (and $x \in L$ means $L(x) = 1$).

Let A be some class of languages. The dual class $co - A$ consists of the complements of all languages in A. Formally,

$$L \in co - A \iff (B^* \backslash \mathrm{L}) \in A.$$

Equivalence of complements

It follows immediately from the definitions that

P = co-P,

BPP = co-BPP,

PSPACE = co-PSPACE.

Games machines play

Two-player game

Consider a game with two players called White (W) and Black (B). A string x is shown to both players. After that, players alternately choose binary strings: W starts with some string $w1$, B replies with $b1$, then W says $w2$, etc. Each string has length polynomial in $|x|$. Each player is allowed to see the strings already chosen by his opponent.

Description of the game

There is a predicate $W(x, w1, b1, w2, b2, ...)$ that is true when W is the winner, and we assume that this predicate belongs to P. If this predicate is false, B is the winner (there are no ties). This predicate (together with polynomial bounds for the length of strings and the number of steps) determines the game.

The game is completed after some prescribed number of steps.

The referee, who knows x and all the strings and who acts according to a polynomial-time algorithm, declares the winner.

Number of moves

The termination rule can be more complicated, but we always assume that the number of moves is bounded by a polynomial.

Therefore, we can "pad" the game with dummy moves that are ignored by the referee and consider only games where the number of moves is known in advance and is a polynomial in the input length.

Defining classes

Since this game is finite and has no ties, for each x either B or W has a winning strategy.

Therefore, any game determines two complementary sets,

$Lw = \{x\colon W\ has\ a\ winning\ strategy\}$,

$Lb = \{x\colon B\ has\ a\ winning\ strategy\}$.

Many complexity classes can be defined as classes formed by the sets Lw (or Lb) for some classes of games.

Examples of classes

P: the sets Lw (or Lb) for games of zero length (the referee declares the winner after he sees the input)
NP: the sets Lw for games that are finished after W's first move. In other words, NP-sets are sets of the form
$\{x\colon \exists w1\ W(x, w1)\}$.
co-NP: the sets Lb for games that are finished after W's first move. In other words, co-NP-sets are sets of the form
$\{x\colon \forall w1\ B(x, w1)\}$.
(here B = ¬W means that B wins the game.)

Examples of classes

Σ_2: the sets Lw for games where W and B make one move each and then the referee declares the winner. In other words, Σ_2-sets are sets of the form
$\{x\colon \exists w1\, \forall b1\, W(x, w1, b1)\}$.
(W can make a winning move $w1$ after which any move $b1$ of B makes B lose).
Π_2: the sets Lb for the same class of games, i.e., the sets of the form
$\{x\colon \forall w1\, \exists b1\, B(x, w1, b1)\}$.

Examples of classes

Σ_k: the sets Lw for games of length k (the last move is made by W if k is odd or by B is k is even), i.e., the sets

$\{x\text{: } \exists w1\, \forall b1 \ \ldots\ Qk\ yk\ W(x, w1, b1, \ldots)\}$

(if k is even, then $Qk = \forall, yk \ = \ b_{k/2}$; if k is odd, then $Qk = \exists$, $yk = w_{(k+1)/2}$).

Π_k: the sets Lb for the same class of games, i.e., the sets

$\{x\text{: } \forall w1\, \exists b1 \ \ldots\ Qk\ yk\ B(x, w1, b1, \ldots)\}$

(if k is even, then $Qk = \exists$, $yk = b_{k/2}$; if k is odd, then $Qk = \forall$, $yk = w_{(k+1)/2}$).

Relation between Σ and Π sets

Complements of Σ_k-sets are Π_k-sets and vice versa: $\Sigma_k = co - \Pi_k$, $\Pi_k = co - \Sigma_k$.

Σ_k: $\{x\text{: } \exists w1\, \forall b1\ \ldots\ Qk\ yk\ W(x, w1, b1, \ldots)\}$

Π_k: $\{x\text{: } \forall w1\, \exists b1\ \ldots\ Qk\ yk\ B(x, w1, b1, \ldots)\}$

BPP ⊆ Σ2 ∩ Π2

Since BPP=co-BPP, it suffices to show that $BPP \subseteq \Sigma_2$.

Σ_2: $\{x\colon \exists w1\ \forall b1\ W(x, w1, b1)\}$.

BPP: the fraction of strings r of length $p(|x|)$ giving correct answer for $R(x, r)$ is greater than $1 - \varepsilon$.

BPP ⊆ Σ2 ∩ Π2

Let us assume that $L \in BPP$. Then there exist a predicate R (computable in polynomial time) and a polynomial p such that the fraction $|Sx|/2^N$ is either large (greater than $1-\varepsilon$ for $x \in L$) or small (less than ε for $x \notin L$).

$Sx = \{y \in B^N\text{: } R(x, y)\}$, $N = p(|x|)$.

BPP ⊆ Σ2 ∩ Π2

To show that $L \in \Sigma_2$, we need to reformulate the property "X is a large subset of G" (where G is the set of all strings y of length N) using existential and universal quantifiers.

This could be done if we impose a group structure on G. Any group structure will work if the group operations are polynomial-time computable. For example, we can consider an additive group formed by bit strings of a given length with bit-wise addition modulo 2.

BPP ⊆ Σ2 ∩ Π2

The property that distinguishes large sets from small ones is the following: “several copies of X shifted by some elements cover G”, i.e.,

$$\exists g1, \ldots, gm \bigcup_i (gi + X) = G,$$

where "+" denotes the group operation. To choose an appropriate value for m, let us see when this formula is guaranteed to be true (or false).

BPP ⊆ Σ2 ∩ Π2

It is obvious that condition is false if $m|X| < |G|$.

On the other hand, condition is true if for independent random $g1, \ldots, gm \in G$ the probability of the event $\cup_i (gi + X) = G$ is positive; in other words, if $\Pr[\cup_i (gi + X) \neq G] < 1$.

Let us estimate this probability.

BPP ⊆ Σ2 ∩ Π2

The probability that a random shift $g + X$ does not contain a fixed element $u \in G$ (for a given X and random g) is $1 - |X|/|G|$. When $g1, \ldots, gm$ are chosen independently, the corresponding sets $g1 + X, \ldots, gm + X$ do not cover u with probability $(1 - |X|/|G|)^m$. Summing these probabilities over all $u \in G$, we see that the probability of the event $\cup_i(gi + X) \neq G$ does not exceed $|G|(1 - |X|/|G|)^m$.

BPP ⊆ Σ2 ∩ Π2

Thus condition $\exists g1, \ldots, gm \bigcup_i (gi + X) = G$ is true if $|G|(1 - |X|/|G|)^m < 1$.

BPP ⊆ Σ2 ∩ Π2

Let us now apply these results to the set $X = Sx = \{y \in B^N \colon R(x, y)\}$. We want to satisfy $m|X| < |G|$ and $|G|(1 - |X|/|G|)^m < 1$ when $|Sx|/2^N < \varepsilon$ (i.e., $x \notin L$) and when $|Sx|/2^N > 1 - \varepsilon$ (i.e., $x \in L$), respectively. Thus we get the inequalities $\varepsilon m < 1$ and $2^N \varepsilon^m < 1$, which should be satisfied simultaneously by a suitable choice of m.

BPP ⊆ Σ2 ∩ Π2

This is not always possible if N and ε are fixed. Fortunately, we have some flexibility in the choice of these parameters. Using "amplification of probability" by repeating the computation k times, we increase N by factor of k, while decreasing ε exponentially. Let the initial value of ε be a constant, and λ given error probability.

BPP ⊆ Σ2 ∩ Π2

The amplification changes N and ε to $N' = kN$ and $\varepsilon' = \lambda^k$. Thus we need to solve the system

$$\lambda km < 1,\ 2^{kN}\lambda^{km} < 1$$

by adjusting m and k. It is obvious that there is a solution with $m = O(N)$ and $k = O(logN)$.

BPP ⊆ Σ2 ∩ Π2

We have proved that $x \in L$ is equivalent to the following Σ_2-condition:
$\exists g1, \ldots, gm\ \forall y((|y| = p'(|x|)) \Rightarrow ((y \in g1 + S'x) \vee \ldots \vee (y \in gm + S'x)))$
Here $p'(n) = kp(|x|)$ (k and m also depend on $|x|$), whereas $S'x$ is the “amplified” version of Sx.

BPP ⊆ Σ2 ∩ Π2

In other words, we have constructed a game where W names m strings (group elements) $g1, \ldots, gm$, and B chooses some string y. If y is covered by some $gi + S'x$ (which is easy to check: it means that $y - gi$ belongs to $S'x$), then W wins; otherwise B wins. In this game W has a winning strategy if and only if $S'x$ is big, i.e., if $x \in L$.

The class PSPACE

PSPACE - introduction

This class contains predicates that can be computed by a TM running in polynomial (in the input length) space. The class PSPACE also has a game-theoretic description.

PSPACE – alternative definition

L ∈ PSPACE if and only if there exists a polynomial game such that
$L = \{x\colon W\ has\ a\ winning\ strategy\ for\ input\ x\}$.

By a polynomial game we mean a game where the number of moves is bounded by a polynomial (in the length of the input), players' moves are strings of polynomial length, and the referee's algorithm runs in polynomial time.

Game => PSPACE

We show that a language determined by a game belongs to PSPACE. Let the number of turns be $p(|x|)$. We construct a sequence of machines $M_1, \ldots, M_{p(|x|)}$. Each M_k gets a prefix $x, w1, b1, \ldots$ of the play that includes k moves and determines who has the winning strategy in the remaining game.

Game => PSPACE

The machine $M_{p(|x|)}$ just computes the predicate $W(x, w1, \ldots)$ using referee's algorithm. The machine M_k tries all possibilities for the next move and consults M_{k+1} to determine the final result of the game for each of them.

Game => PSPACE

Then M_k gives an answer according to the following rule, which says whether W wins. If it is W’s turn, then it suffices to find a single move for which M_{k+1} declares W to be the winner. If it is B’s turn, then W needs to win after all possible moves of B.

Game => PSPACE

The machine M_0 says who is the winner before the game starts and therefore decides $L(x)$. Each machine in the sequence $M_1, \ldots, M_{p(|x|)}$ uses only a small (polynomially bounded) amount of memory, so that the composite machine runs in polynomial space. (Note that the computation time is exponential since each of the M_k calls M_{k+1} many times.)

PSPACE => Game

Let M be a machine that decides the predicate L and runs in polynomial space s. We may assume that computation time is bounded by $2^{O(s)}$. Indeed, there are $2^{O(s)}$ different configurations, and after visiting the same configuration twice the computation repeats itself, i.e., the computation becomes cyclic.

Number of configurations

To see why there are at most $2^{O(s)}$ configurations note that configuration is determined by head position (in $\{0,1,\ldots,s\}$), internal state (there are $|Q|$ of them) and the contents of the s cells of the tape ($|A|^s$ possibilities where A is the alphabet of TM); therefore the total number of configurations is $|A|^s \cdot |Q| \cdot s = 2^{O(s)}$.

PSPACE => Game

Therefore, we may assume without loss of generality that the running time of M on input x is bounded by 2^q, where $q = poly(|x|)$.

In the description of the game given below we assume that TM keeps its configuration unchanged after the computation terminates.

PSPACE => Game

During the game, W claims that M’s result for an input string x is “yes”, and B wants to disprove this. The rules of the game allow W to win if $M(x)$ is indeed “yes” and allow B to win if $M(x)$ is not “yes”.

PSPACE => Game

In his first move, W declares the configuration of M after 2^{q-1} steps dividing the computation into two parts. B can choose any of the parts: either the time interval $[0, 2^{q-1}]$ or the interval $[2^{q-1}, 2^{q}]$. (B tries to catch W by choosing the interval where W is cheating.) Then W declares the configuration of M at the middle of the interval chosen by B and divides this interval into two halves, B selects one of the halves, W declares the configuration of M at the middle, etc.

PSPACE => Game

The game ends when the length of the interval becomes equal to 1. Then the referee checks whether the configurations corresponding to the ends of this interval match (the second is obtained from the first according to M’s rules). If they match, then W wins; otherwise B wins.

PSPACE => Game

If M’s output on x is really “yes”, then W wins if he is honest and declares the actual configuration of M. If M’s output is “no”, then W is forced to cheat: his claim is incorrect for (at least) one of the halves. If B selects this half at each move, than B can finally catch W “on the spot” and win.

Nondeterministic PSPACE

Any predicate $L(x)$ that is recognized by a nondeterministic machine in space $s = poly(|x|)$ belongs to PSPACE. (A predicate L is recognized by an NTM M in space $s(|x|)$ if for any $x \in L$ there exists a computational path of M that gives the answer “yes” using at most $s(|x|)$ cells and, for each $x \notin L$, no computational path of M ends with “yes”.)

Polynomial hierarchy

L ∈ PSPACE if and only if there exists a polynomial game such that
$L = \{x\colon W\ has\ a\ winning\ strategy\ for\ input\ x\}$.
This shows that all the classes Σ_k, Π_k are subsets of PSPACE.

The class PSPACE allows the number of moves in a game to be polynomial in $|x|$.

Polynomial hierarchy

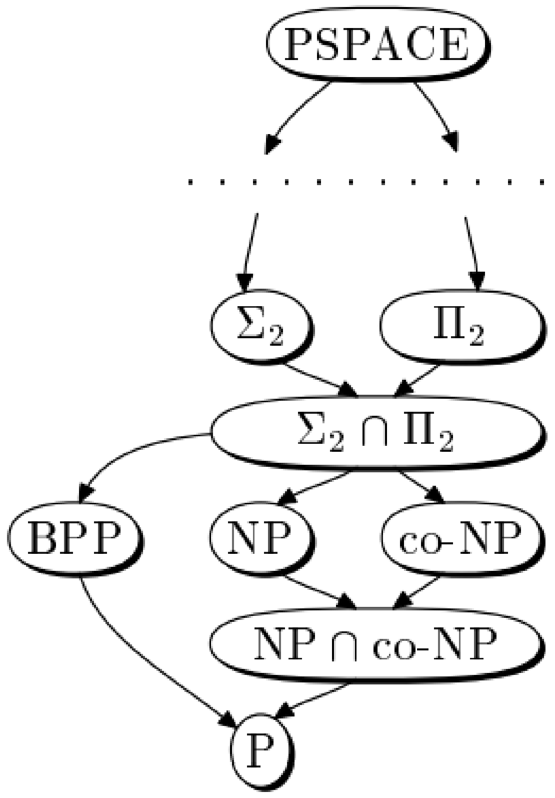


Fig. 5.1. Inclusion diagram for computational classes. An arrow from A to B means that B is a subset of A.

Polynomial inclusions

We do not know whether the inclusions in the diagram are strict. Computer scientists have been working hard for several decades trying to prove at least something about these classes, but the problem remains open. It is possible that P = PSPACE (though this seems very unlikely).

EXPTIME

It is also possible that PSPACE = EXPTIME, where EXPTIME is the class of languages decidable in time $2^{poly(n)}$. Note, however, that P ≠ EXPTIME — one can prove this by a rather simple "diagonalization argument".

P vs EXPTIME

Deterministic time hierarchy theorem. If $f(n)$ is a time-constructible function, then there exists a decision problem which cannot be solved in worst-case deterministic time $f(n)$ but can be solved in worst-case deterministic time $f(n)^2$. In other words,

$$DTIME(f(n)) \subsetneq DTIME(f(n)^2)$$

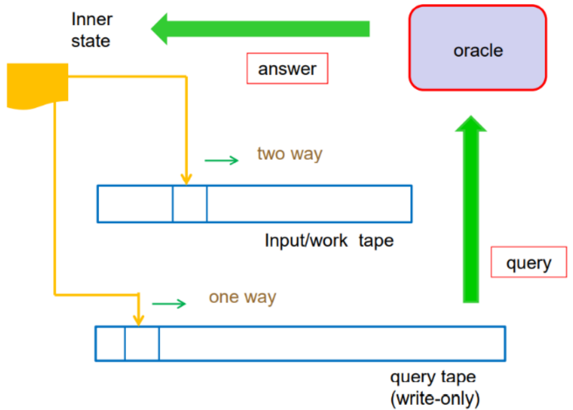
Therefore, we have:

$$P \subseteq DTIME(2^n) \subsetneq DTIME(2^{2n}) \subseteq EXPTIME$$

Oracle TM

A Turing machine with oracle for language L uses a decision procedure for L as an external subroutine. The machine has a supplementary oracle tape, where it can write strings and then ask the "oracle" whether the string written on the oracle tape belongs to L.

Any language that is decidable in polynomial time by a TM with oracle for some $L \in \Sigma_k$ (or $L \in \Pi_k$) belongs to $\Sigma_{k+1} \cap \Pi_{k+1}$.



TQBF(x)

x is a True Quantified Boolean Formula, i.e., a true statement of type $Q1y1$ ... $Qn\, yn\, F(y1, ..., yn)$, where variables yi range over $B = \{0,1\}$, F is some propositional formula (involving $y1, ..., yn$, ¬, ∧, ∨), and Qi is either ∀ or ∃.

By definition, $\forall y\, A(y)$ means (A(0) ∧ A(1)) and $\exists y\, A(y)$ means (A(0) ∨ A(1)).

TQBF is PSPACE-complete

We reduce an arbitrary language L ∈ PSPACE to TQBF. We construct a game that corresponds to L. Then we convert a TM that computes the result of the game (a predicate W) into a circuit. Moves of the players are encoded by Boolean variables.

TQBF is PSPACE-complete

Then the existence of the winning strategy for W can be represented by a quantified Boolean formula

$$\exists w_1^1 \exists w_1^2 \ldots \exists w_1^{p(|x|)} \forall b_1^1 \ldots \forall b_1^{p(|x|)} \exists w_2^1 \ldots \exists w_2^{p(|x|)} \ldots S(x, w_1^1, w_1^2, \ldots),$$

where $S(\cdot)$ denotes the Boolean function computed by the circuit. (Boolean variables $w_1^1 \ldots w_1^{p(|x|)}$ encode the first move of W, variables $b_1^1 \ldots b_1^{p(|x|)}$ encode B’s answer, $w_2^1 \ldots w_2^{p(|x|)}$ encode the second move of W, etc.)

TQBF is PSPACE-complete

In order to convert S into a Boolean formula, recall that a circuit is a sequence of assignments $yi = Ri$ that determine the values of auxiliary Boolean variables yi. Then we can replace $S(\cdot)$ by a formula

$\exists y1, \ldots, \exists ys \; (y1 \;\Leftrightarrow\; R1) \;\wedge \cdots \wedge\; (ys \;\Leftrightarrow\; Rs) \;\wedge\; ys$,

where s is the size of the circuit.

TQBF example

$$\forall x_1 \exists x_2 \forall x_3 \big((x_1 \vee NOT x_2)\ \wedge (NOT x_1 \vee x_3)\big)$$

This formula is a false QBF. To see this, note that, if x_1 is true, then x_3 must be true in order to satisfy the clause $(NOT x_1 \vee x_3)$, but x_3 is a universally quantified variable.

Remarks

There are similarities between proofs of polynomial space computation and poly-logarithmic space computation.

Also note that a polynomial-size quantified Boolean formula may be regarded as a polynomial depth circuit (though of very special structure): the $\forall$ and $\exists$ quantifiers are similar to the $\wedge$ and $\vee$ gates.

Remarks

The reduction from NTM to TQBF is similar to the parallel simulation of a finite-state automaton.

In the case of TQBF we could afford reasoning at a more abstract level: with greater amount of computational resources we were sure that all bookkeeping constructions could be implemented. This is one of the reasons why “big” computational classes (like PSPACE) are popular among computer scientists, in spite of being apparently impractical. In fact, it is sometimes easier to prove something about big classes, and then scale down the problem parameters while fixing some details.

PH definition

Level i - Σ_i:

$x \in L \;\Rightarrow\; \exists y_1 \forall y_2 \exists y_3 \ldots Q_i y_i$ such that M accepts $(x, y_1, \ldots, y_i)$,

$x \notin L \;\Rightarrow\; \forall y_1 \exists y_2 \forall y_3 \ldots \overline{Q_i} y_i$ such that M rejects $(x, y_1, \ldots, y_i)$.

PH is union over all Σ_i for all $i \in \mathbb{N}$.

PH alternative definition

For the oracle definition of the polynomial hierarchy, define
$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$,
where P is the set of decision problems solvable in polynomial time. Then for $i \geq 0$ define
$\Delta_{i+1}^P = P^{\Sigma_i^P}$
$\Sigma_{i+1}^P = NP^{\Sigma_i^P}$
$\Pi_{i+1}^P = co-NP^{\Sigma_i^P}$
where P^A is the set of decision problems solvable in polynomial time by a Turing machine augmented by an oracle for some complete problem in class A; the classes NP^A and $co-NP^A$ are defined analogously. For example, $\Sigma_1^P = NP$, $\Pi_1^P = co-NP$, and $\Delta_2^P = P^{NP}$ is the class of problems solvable in polynomial time with an oracle for some NP-complete problem

PH

$$\begin{tikzcd}
 & \vdots & \\
\Sigma_3^{\mathrm{P}} & & \Pi_3^{\mathrm{P}} \\
 & \Delta_3^{\mathrm{P}} \arrow[ul] \arrow[ur] & \\
\Sigma_2^{\mathrm{P}} \arrow[uu] \arrow[ur] & & \Pi_2^{\mathrm{P}} \arrow[uu] \arrow[ul] \\
 & \mathrm{P}^{\mathrm{NP}} = \Delta_2^{\mathrm{P}} \arrow[ul] \arrow[ur] & \\
\mathrm{NP} = \Sigma_1^{\mathrm{P}} \arrow[uu] \arrow[ur] & & \Pi_1^{\mathrm{P}} = \mathrm{coNP} \arrow[uu] \arrow[ul] \\
 & \Delta_0^{\mathrm{P}} = \Sigma_0^{\mathrm{P}} = \mathrm{P} = \Pi_0^{\mathrm{P}} = \Delta_1^{\mathrm{P}} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

PH related problems

P = NP if and only if P = PH.
If NP = co-NP then NP = PH. (co-NP is Σ_1^P.)
Whether PH = PSPACE is open question.

PSPACE-complete problems

Actually, a lot of them:
https://en.wikipedia.org/wiki/List_of_PSPACE-complete_problems

- Generalized versions of games and puzzles
- Logic
- Automata and language theory
- Graph theory

Thank you for your attention!