Introduction to Complexity Theory

Bernhard Häupler

May 2, 2006

Abstract

This paper is a short repetition of the basic topics in complexity theory. It is not intended to be a complete step by step introduction for beginners but addresses to readers who want to refresh their knowledge efficiently. We start with the definition of the standard (non)deterministic time and space bounded complexity classes. Next the important concept of reduction and completeness is discussed intensively. After a short excursion on Boolean circuits several completeness results in P, NP and $PSPACE$ strengthen the routine of these methods and give a broad base for further hardness results. Besides that we have a look at optimization problems in P^{NP} and classify these problems within the polynomial hierarchy. The polynomial hierarchy is then characterized through the notion of certificates, which make it more comfortable and intuitive to handle. With this characterization we close with some facts about PH collapses.

Contents

1 Complexity Classes

1.1 Space and time bounds

A complexity class[Pa94] is defined by four parameters, the model and mode of computation, a bounded resource and an asymptotic worst case bound:

$TIME(f(n)) :=$ Languages decidable in time $O(f(n))$ by a DTM

$NTIME(f(n)) :=$ Languages decidable in time $O(f(n))$ by a NTM

$SPACE(f(n)) :=$ Languages decidable in space $O(f(n))$ by a DTM
(besides the (read only) input and the (write only) output)

$NSPACE(f(n)) :=$ Languages decidable in space $O(f(n))$ by a NTM

1.2 Important complexity classes

Having these notations we can easily define the important classical complexity classes:

$L := SPACE(\log n)$ $\quad NL := NSPACE(\log n)$

$P := \bigcup_k TIME(n^k)$ $\quad NP := \bigcup_k NTIME(n^k)$
$coNP := \mathrm{P}(\Sigma^*) \setminus NP$

$PSPACE := \bigcup_k SPACE(n^k)$

$EXP := \bigcup_k TIME(2^{n^k})$ $\quad NEXP := \bigcup_k NTIME(2^{n^k})$

$2\text{-}EXP := \bigcup_k TIME(2^{2^{n^k}})$

$ELEMENTARY := \bigcup_k k\text{-}EXP$

1.3 Relationships between complexity classes

Important relationships:

- Hierarchy in $PSPACE$
  - $L \subseteq P$
  - $NL \subseteq P$
  - $\Rightarrow L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$
  - $L \subset PSPACE$
- Linear Speedup
  - $TIME(f(n)) = TIME(\epsilon f(n) + n + 2)$
  - $SPACE(f(n)) = SPACE(\epsilon f(n) + 2)$
  - same for nondeterministic classes
- Nondeterministic Space
  - $coNSPACE = NSPACE$
  - $NSPACE(f(n)) \subseteq SPACE(f^2(n))$
  - $\Rightarrow PSPACE = NPSPACE = coNPSPACE$

1.4 Function problems

Definition 1.1 *(Function problems)*
A function problem is abstracted by a binary relation $R \subseteq \Sigma^* \times \Sigma^*$.
The task is: Given an input x, find an output y with $(x, y) \in R$.
FC denotes the class of all function problems computable by a TM in C

Definition 1.2 *(Decision problems)*
A decision problem is abstracted by a language $L \subseteq \Sigma^*$.
The task is: Given an input x, decide whether $x \in L$. The decision problem related to the function problem R is
$L(R) := \{x \mid \exists y : (x, y) \in R\}$

1.5 Oracles

Definition 1.3 *(Oracle TM)*
An oracle TM $M^?$ has 3 additional states (q_{query}, q_{yes} and q_{no}) and one additional query-string qs.
After being in state q_{query} $M^?$ continues in state q_{yes} / q_{no} depending on the answer of the oracle on input qs.

Definition 1.4 *(Oracle Complexity Class)*
C^O = Languages decidable by an oracle TM $M^? \in C$ with oracle language O
$C^{C'}$ = Languages decidable by an oracle TM $M^? \in C$ with oracle language $O \in C'$

2 Reductions

2.1 Idea

The idea behind Reductions is to relate the complexity of languages by trying to transform instances of the domain A to a domain B. If we can do such a transformation we can solve A with the help of B. Therefore it seems reasonable to say B is at least as hard as A and write $A \leq B$.

2.2 Reductions

Definition 2.1 *(Reductions)*
Let f, g, h be functions, then $A \leq B \; :\Longleftrightarrow$

- ***Cook:*** $A \in P^B$
- ***Karp:*** $\exists f \in FP : \; x \in A \iff f(x) \in B$
- ***Logspace:*** $\exists f \in FL : \; x \in A \iff f(x) \in B$
- ***Levin:*** $\exists f, g, h \in FP :$
  $x \in L(R_1) \iff f(x) \in L(R_2)$
  $\forall x, z : \; (f(x), z) \in R_2 \Longrightarrow (x, g(x, z)) \in L(R_1)$
  $\forall (x, y) \in R_1 : \; (f(x), h(x, y)) \in L(R_2)$
- ***L-Reduction:*** *like Karp but preserves approximability*

2.3 Hierarchy and closure

Lemma 2.2 $A \leq_{\log} B \implies A \leq_K B \implies A \leq_C B$

Proof:

1. $L \subseteq P$
2. compute $f(x)$ and ask oracle

$\square$

Definition 2.3 *(Closure under reduction)*
C is closed under reduction $:\iff \quad A \leq B \wedge B \in C \Longrightarrow A \in C$

Proposition 2.4
L, NL, P, NP, coNP, PSPACE, EXP are closed under $\leq_{\log}$

2.4 Transitivity

Lemma 2.5 *(Transitivity)*
$\leq_C$, $\leq_K$, $\leq_{\log}$, *and* $\leq_{Levin}$ *are transitive.*

Proof: $(A \leq B \wedge B \leq C \Longrightarrow A \leq C)$

1. **Cook:** $A \in P^B \wedge B \in P^C \Longrightarrow A \in P^C$
   - run the P^B TM
   - instead of asking the oracle compute answer with P^C TM
   - polynomial queries which take polynomial time can be computed in P
2. **Karp:** $f_{AC} = f_{BC} \circ f_{AB}$
3. **Logspace:**
   - like Karp
   - but $f_{AB}(x)$ could be polynomial long
   - $\Rightarrow$ each time f_{BC} needs input compute only this char with f_{AB}

$\square$

3 Boolean circuits

For a short introduction to boolean Circuits look at [Bl05] or [Pa94].

3.1 Expressive power

Shortly spoken boolean circuits are a potentially more economical way of representing boolean functions than tables or Boolean expressions. While tables represent a function without compression, boolean expressions can describe many natural dependencies with short formulas. A Boolean circuit extends the expressive power of Boolean expressions by compressing shared subexpressions in additional edges. It is an amazing fact that on the one hand this representation is so efficient that nobody has been able to come up with a natural family of Boolean functions that require more than a linear number of gates to compute but on the other hand the next lemma shows that there must exist many exponentially difficult functions.

Lemma 3.1
For $n > 2$ there is a n-ary boolean function which needs more than $m = \frac{2^n}{2n}$ gates.

Proof: (number of circuits < number of boolean functions)

- sorts of gates: $(n+5)$
- number of gates: $\leq m$
- possible inputs: $\leq m^2$

$\Longrightarrow \quad \left((n+5)m^2\right)^m = \left((n+5)\frac{2^{2n}}{4n^2}\right)^{\frac{2^n}{2n}} < \left(2^{2n}\right)^{\frac{2^n}{2n}} = 2^{2^n}$ □

3.2 Reduction to Boolean expressions

It is clear, that for every Boolean expression Φ we can construct a circuit C with the same functionality by following the inductive definition of Φ. The other direction is not so simple. In fact we can have an exponential blow-up if we do not allow introducing new variables. Without this restriction the linear sized transformation can be done this way:

Lemma 3.2
Every Boolean circuit C is equivalent to a boolean expression with size $O(|C|)$.

Proof:
Give each gate a variable and "‘translate"’

- **variable gate:** $g \iff x$
- **True gate:** g
- **False gate:** $\neg g$
- **not gate:** $g \iff \neg h$
- **and gate:** $g \iff a \wedge b$
- **or gate:** $g \iff a \vee b$
- **output gate:** g

The conjunction of these clauses is equivalent to the circuit. □

4 Completeness

4.1 Definition

Definition 4.1 *(Completeness)*
A is complete for C : $\iff \quad A \in C \;\wedge\; \forall L \in C : \; L \leq A$
(maximal elements of the preorder given by $\leq$)

4.2 P completeness

Problem 4.2 *CIRCUIT_VALUE*
Given a Boolean circuit C without variable gates, does C compute to True?

Lemma 4.3
$CIRCUIT\_VALUE$ is P complete

Proof:

1. $CIRCUIT\_VALUE \in P$:
   The circuit can be easily evaluated in polynomial time.
2. $\forall L \in P: \; L \leq_{\log} CIRCUIT\_VALUE$:
   Having an arbitrary language $L \in P$ decided by a TM M in time n^k and an input x we want to build a boolean circuit that is satisfiable $\iff x \in L \iff M$ accepts x.
   - W.L.O.G. M has only one string
   - interpret the computation on x as a $|x|^{k+1} \times |x|^{k+1}$ computation table with alphabet $\Sigma \cup \Sigma \times K$

$\sqcup$	$\triangleright$	O_{q_0}	T	t	O	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T_O	t	O	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T	t_O	O	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T	t	O_O	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T	t	O	$\sqcup_O$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T	t	$O_{O'}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T	t_{q_r}	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T_{q_r}	t	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	$@_{q_r}$	T	t	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T_{q_0}	t	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	@	t_T	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	@	t	$@_T$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	@	$t_{T'}$	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	@	no	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	@	no	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$

   - a char depends only on the 3 chars above it
   - translate it into binary and write a circuit C which computes one char
   - with polynomial copies of C build a circuit G which computes the table
   - add a circuit which tests for accepting states
   - the leftest and rightest columns are (set to) $\sqcup$ and the input x is known
   - $\Rightarrow$ no free variables occur
   - the value of G is True $\iff M$ accepts x

$\square$

4.3 NP completeness

Problem 4.4 *CIRCUIT_SAT*
Given a Boolean circuit C. Is there a truth assignment to the variable gates of C such that C computes to true?

Lemma 4.5
$CIRCUIT\_SAT$ is NP complete

Proof:
Similar to proof of lemma 4.3

- W.L.O.G. NTM M has single string and 2 nondeterministic choices (0,1) at each step
- a char depends only on the 3 chars above it and the choice
- for each choice build an computation table and translate it into binary
- build a circuit which computes one char using an free variable gate for the choice
- build a circuit for the whole computation table
- this circuit is satisfiable $\iff$ an truth assignment to the choice gates exists that leads to an accepting state $\iff$ M accepts x

$\square$

Problem 4.6 *SAT*
Given a Boolean expression in CNF, is it satisfiable?

Corollary 4.7
SAT is NP complete

Proof:

- can be solved in NP by guessing and verifying
- for $CIRCUIT_SAT \leq_{\log} SAT$ see lemma 3.2 and replace gate formulas by their CNF for example: $(g \iff x) = ((\neg g \vee x) \wedge (g \vee \neg x))$

$\square$

Problem 4.8 *HAMILTON_PATH*
Given a directed graph, is there a path that visits each node exactly once?

Lemma 4.9
$HAMILTON\_PATH$ is NP complete

Proof: $HAMILTON\_PATH \in NP \;\wedge\; SAT \leq_{\log} 3SAT \leq_{\log} HAMILTON\_PATH$

1. guess and verify
2. without proof (simple logic)
3. short sketch: (for full proof see [Pa94])
   given a boolean expression Φ, construct a graph G:
   G has a Hamilton path $\iff$ Φ is satisfiable:
   - each variable $\mapsto$ choice gadget
     (allowing the true or false path to traverse)
   - each clause $\mapsto$ constraint gadget
     (forming a circle iff all variables are false)
   - consistency guaranteed through xor-gadgets
     (substitutes two edges so that only one can be traversed)

$\square$

Problem 4.10
TSP : *Given a undirected complete weighted graph, find the shortest tour (circle visiting each node once)*

Like for every other optimization problem we can define a related decision problem:

Problem 4.11
$TSP(D)$: Given a undirected complete weighted graph and an integer budged B, is there a tour of length at most B?

Lemma 4.12
$TSP(D)$ is NP complete

Proof: $TSP(D) \in NP \ \wedge \ HAMILTON\_PATH \leq_{\log} TSP(D)$

1. guess and verify
2. given graph G with n nodes, construct a complete weighted graph G' with n nodes and a budget B:
   - edges in G' have weight 1 if they exist in G else 2
   - Budget $B = n + 1$
   - G' has a TSP-Tour with budged $B \iff G$ has a Hamilton path

□

4.4 PSPACE completeness

Problem 4.13 *$IN\_PLACE\_ACCEPTANCE$*
Given a DTM M and an input x, does M accept x without ever leaving the $|x|+1$ first symbols of its string?

Lemma 4.14
$IN\_PLACE\_ACCEPTANCE$ is $PSPACE$ complete

Proof: $IN\_PLACE\_ACCEPTANCE \in PSPACE \ \wedge$
$L \in PSPACE \implies L \ \leq_{\log} \ IN\_PLACE\_ACCEPTANCE$

1. simulate M on x and count steps
   reject $\iff M$ rejects, leaves the place, or operates more than $|K||x||\Sigma|^{|x|}$ steps
2. DTM M decides L in n^k space:
   $x \in L \iff M$ accepts x in $|x|^k$ space $\iff M$ accepts $x\sqcup^{|x|^k}$ in place
   $\iff (M, x\sqcup^{|x|^k}) \in IN\_PLACE\_ACCEPTANCE$

□

5 Polynomial Hierarchy

5.1 Optimization problems in FP^{NP}

Lemma 5.1
TSP is FP^{NP} complete

Proof: TSP is FP^{NP} hard $\wedge$ $TSP \in FP^{NP}$

1. without proof
2. Construct TM $M^? \in FP$ which decides TSP with $TSP(D)$ oracle
   - optimum cost C is an integer between 0 and $2^{|x|}$
   - $\Rightarrow$ exact cost C can be computed by binary search asking $|x|$ queries

- test every edge:
  - set its cost to $C+1$
  - ask $TSP(D)$ oracle whether now an tour with budged C exists
  - reset the cost only if the answer is "'no"'
- all edges with cost $< C+1$ form an optimal tour

□

Corollary 5.2
$MAXIMUM\_WEIGHTED\_SAT \in FP^{NP}$

Proof:
Construct TM $M^? \in FP$

- compute the largest possible weight of satisfied clauses by binary search
- test each variable one-by-one

□

Corollary 5.3
$WEIGHTED\_MAX\_CUT \in FP^{NP}$
$KNAPSACK \in FP^{NP}$
$WEIGHTED\_BISECTION\_WIDTH \in FP^{NP}$
$\ldots$

5.2 Polynomial Hierarchy

After we have seen that P^{NP} captures many important problems it seems reasonable to consider the corresponding nondeterministic class NP^{NP}. As a nondeterministic class it will naturally not be closed under complement. We also can have a look at classes using NP^{NP} and so on. This leads us directly to the definition of the polynomial hierarchy:

Definition 5.4 *(Polynomial hierarchy)*
$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$
and for all $i \geq 0$*:*

- $\Delta_{i+1}^P = P^{\Sigma_i^P}$
- $\Sigma_{i+1}^P = NP^{\Sigma_i^P}$
- $\Pi_{i+1}^P = coNP^{\Sigma_i^P}$

$PH = \bigcup_i \ \Sigma_i^P$ *is the cumulative polynomial hierarchy*

Looking at the first level of the polynomial hierarchy with $\Delta_1^P = P^P = P$, $\Sigma_1^P = NP^P = NP$ and $\Pi_1^P = coNP^P = coNP$ we find our familiar important complexity classes as a special case within the PH. The second level contains $\Delta_2^P = P^{NP}$ studied in the previous subsection $\Sigma_2^P = NP^{NP}$ and its complement $\Pi_2^P = coNP^{NP}$.

As we expect it for a hierarchy the following containment relationship holds:

Corollary 5.5
$\forall i \geq 0 : \quad \Delta_i^P \ \subseteq \ \Sigma_i^P \cap \Pi_i^P \ \subseteq \ \Delta_{i+1}^P$

5.3 Characterization

5.3.1 Certificates and verification

Finding problems for the polynomial hierarchy by using its definition is hard because an arbitrary long scope of oracles must be taken into consideration. Therefore we are more likely to argue in terms of witnesses or certificates than in terms of nondeterministic TM. These certificates encode the accepting paths of NTM and are so proofs for the containment in the accepted language. Our first step into this direction leads us an alternative characterization of NP which makes this class so much more intuitive that it is for example used in the informal description of the $P = NP$ millennium problem.[Clay]

Definition 5.6 *(polynomial bounded relation)*
A polynomial bounded relation is a relation $R \subseteq (\Sigma^*)^{l+1}$ *with* $\exists k \in \mathbf{N} : \ \forall (x, y_1, y_2, ..., y_l) \in R : \ |y_i| \leq |x|^k$.

Definition 5.7 *(C-verifiable relation)*
A C-verifiable relation R is a polynomial bounded relation, which is decidable in C: $\{x; y_1; y_2; ...; y_l \mid (x, y_1, y_2,$
$R\} \in C$

5.3.2 Characterization of NP

Lemma 5.8 *(Characterization of NP)*
$NP = \{ \ \{x \mid \exists y : (x, y) \in R\} \ \mid \ R$ *is P-verifiable*$\}$

Proof:
”‘$\Leftarrow$”’: R is P-verifiable $\Longrightarrow \{x \mid \exists y : (x, y) \in R\} \in NP$

- construct NTM M' which on input x
  - guesses polynomial bounded y
  - verify whether $(x, y) \in R$
  - accept $x \iff (x, y) \in R$
- $M' \in NP$
- M' accepts $x \iff \exists y : (x, y) \in R$

”‘$\Rightarrow$”’: $L \in NP \Longrightarrow \exists R$ P-verifiable : $L = \{x \mid \exists y : (x, y) \in R\}$

- have TM $M \in NP$ deciding L
- for input $x \in L$ encode the choices of an accepting path of M into a witness y
- $R = \{(x, y) \mid y$ is witness for x$\}$ is the searched relation
  - polynomial bounded y (because of the polynomial running time of M)
  - polynomial decidable (by DTM M’ using y to determine the computation path of M)
  - $\exists y : (x, y) \in R \iff M$ accepts $x \iff x \in L$

$\square$

5.3.3 Characterization of Σ_i^P and Π_i^P

Lemma 5.9 *(Characterization of Σ_i^P)*
$\Sigma_i^P = \{\ \{x \mid \exists y : (x,y) \in R\}\ \mid\ R \textit{ is } \Pi_{i-1}^P\textit{-verifiable}\}$

Proof: (by induction on i)
$i = 1$: exactly the characterization of NP

$(i-1) \rightarrow i$:
"'$\Leftarrow$"': R is Π_{i-1}^P-verifiable $\Longrightarrow \{x \mid \exists y : (x,y) \in R\} \in \Sigma_i^P$

- construct NTM $M^? \in NP$ which on input x
  - guesses polynomial bounded y
  - aks an oracle $K \in \Sigma_{i-1}^P$ whether $(x,y) \in R$
  - accepts $x \iff (x,y) \in R$
- $M^K \in \Sigma_i^P$ (since $\Sigma_{i-1}^P \subseteq \Sigma_i^P$)
- M^K accepts $x \iff \exists y : (x,y) \in R$

"'$\Rightarrow$"': $L \in \Sigma_i^P \Longrightarrow \exists R\ \Pi_{i-1}^P$-verifiable : $L = \{x \mid \exists y : (x,y) \in R\}$

- have NTM $M^? \in NP^?$ deciding L with oracle $K \in \Sigma_{i-1}^P$
- for input $x \in L$ encode all choices and queries of $M^?$ into a certificate y of x (example: $y = (c_0,\, c_4,\, qs_1 \notin K,\, c_1,\, qs_2 \in K$ + cert, ...))
- define $R = \{(x,y) |\ y$ is certificate for $x\}$
  - R is polynomial bounded
  - $x \notin K$ is Π_{i-1}^P-decidable
  - $x \in K$ is Π_{i-2}^P-verifiable (by induction)
  - $\Rightarrow R$ is Π_{i-1}^P-verifiable
- $\exists y : (x,y) \in R \iff M^K$ accept $x \iff x \in L$

$\square$

Corollary 5.10 *(Characterization of Π_i^P)*
$\Pi_i^P = \{\ \{x \mid \forall y :\ |y| < |x|^k \Rightarrow (x,y) \in R\ \}\ \mid\ R \textit{ is } \Sigma_{i-1}^P\textit{-verifiable}\}$

Corollary 5.11
$L \in \Sigma_i^P \iff \exists R :\ R \textit{ is } P\textit{-verifiable} \wedge L = \{x \mid \exists y_1 \forall y_2 \exists y_3\ \dots : (x, y_1, y_2, ..., y_i) \in R\}$
$L \in \Pi_i^P \iff \exists R :\ R \textit{ is } P\textit{-verifiable} \wedge L = \{x \mid \forall y_1 \exists y_2 \forall y_3\ \dots : (x, y_1, y_2, ..., y_i) \in R\}$

5.4 Problems in PH

Problem 5.12 *MINIMUM_CIRCUIT*
Given a Boolean circuit C, is it true that there is no circuit with fewer gates computing the same Boolean function?

Lemma 5.13
$MINIMUM\_CIRCUIT \in \Pi_2^P$

Proof:

- C is accepted $\iff \forall C' : \; |C'| < |C| : \; \exists$ input $x : \; C'(x) \neq C(x)$
- and $C'(x) \neq C(x)$ can be checked in polynomial time

$\square$

Problem 5.14 $QSAT_i$
Decide whether a quantified boolean expression with i alternations of quantifiers (beginning with an existential quantifier) is satisfiable

Lemma 5.15
$QSAT_i$ is Σ_i^P complete

Proof:
Sketch: Combine the above characterization of Σ_i^P with the equivalence of accepting computation and satisfiability of Boolean circuits/expressions shown at page 7 $\square$

5.5 PH collapses

The recursive reuse of each level as an oracle to define the next level leads naturally to an extremely fragile structure. Therefore any jitter, at any level of this hierarchy yields to disastrous consequences further up.[Pa94]

Definition 5.16 *(Collapse of PH)*
PH collapses to the ith level means: $\forall j > i : \; \Sigma_j^P = \Pi_j^P = \Delta_j^P = \Sigma_i^P$

Lemma 5.17 *(Collapse of PH)*
If for some $i \leq 1$ $\Sigma_i^P = \Pi_i^P$ then PH collapses to the ith level.

Proof: $\Sigma_i^P = \Pi_i^P \Longrightarrow \Sigma_{i+1}^P = \Sigma_i^P$

$$\begin{aligned}
L \in \Sigma_{i+1}^P \quad &\iff L = \{x \mid \exists y : \; (x,y) \in R\} \text{ with } R \text{ is } \Pi_i^P\text{-verifiable} \\
&\iff L = \{x \mid \exists y : \; (x,y) \in R\} \text{ with } R \text{ is } \Sigma_i^P\text{-verifiable} \\
&\iff L = \{x \mid \exists y : \; (x,y) \in R\} \text{ with} \\
&\qquad \big[(x,y) \in R \iff \exists z : (x,y,z) \in S \text{ with } S \text{ is } \Pi_{i-1}^P\text{-verifiable}\big] \\
&\iff L = \{x \mid \exists y, z : \; (x,y,z) \in S\} \text{ with } S \text{ is } \Pi_{i-1}^P\text{-verifiable} \\
&\iff L \in \Sigma_i^P
\end{aligned}$$

$\square$

Corollary 5.18 *(PH complete Problems)*
If PH has complete problems, then it collapses to some finite level.

Corollary 5.19 *(PH and PSPACE)*
$PH \subseteq PSPACE$ and $PH = PSPACE \Longrightarrow PH$ collapses

Proof:

1. trivial
2. $PSPACE$ has complete problems

$\square$

References

[Go99] O. Goldreich: Introduction to Complexity Theory
Lecture Notes, 1999.

[Pa94] Christos H. Papadimitriou: Computational Complexity
Addison Wesley, 1994.

[Bl05] Markus Bläser: Complexity Theory
Lecture Notes, 2005.

[Clay] Clay Mathematics Institute, Cambridge, Massachusetts: P vs NP Problem
http://www.claymath.org/millennium/P_vs_NP/.