ON THE COMPLEXITY STRUCTURE OF \( NP \cap \text{CO}NP \)

by

Shimon Even* and Yacov Yacobi**

Technical Report #201

March 1981

* Computer Science Dept., Technion, Haifa, Israel

** Electrical Engineering Dept., Technion, Haifa, Israel

This research was supported by the Fund for the Promotion of Research at the Technion.
ON THE COMPLEXITY STRUCTURE OF $NP \cap CONP$

by

Shimon Even* and Yacov Yacobi**

March, 1981

ABSTRACT

Assuming $NP \neq CONP$ we show that there are problems in $NP$ which neither belong to $CONP$ nor to $NPC_{COOK}$.

We further show that for each such problem there exists a problem which is essentially not harder, and does not belong to $NP \cup CONP$.

That is, although under the NDTM model these problems and their complements are harder than NPC problems, this is not the case when the DTM model is considered.

* Computer Science Dept., Technion-Israel Institute of Technology, Haifa, Israel.

** Electrical Engineering Dept., Technion-Israel Institute of Technology, Haifa, Israel.

This research was supported by the Fund for the Promotion of Research at the Technion.
1. INTRODUCTION

Ladner [1] proved, assuming P ≠ NP, that NP - NPC\textsubscript{COOK} ≠ P - ∅, where NPC\textsubscript{COOK} denotes the class of NPC problems defined by using Cook's reduction [2,3]. Using the same proof technique we show that NP ≠ CONP implies A \triangleleft NP - NPC\textsubscript{COOK} ≠ CONP ≠ ∅.

We further show that for every problem \( \alpha \in I(\sim\text{NP} \cap \text{CONP}) \) there exists a problem \( \beta \in U(\sim\text{NP} \cup \text{CONP}) \), such that \( \beta \)'s time-complexity is roughly that of \( \alpha \).

These two results imply that if NP ≠ CONP then there are problems external to \( U \) which are not harder than NPC problems.

2. \( (\text{NP} ≠ \text{CONP}) \Rightarrow (\text{NP} - I - \text{NPC}_{\text{COOK}} ≠ ∅) \)

Let \( \{\psi_1\} \) be an effective enumeration of the pairs of Non-Deterministic-Turing-Machines (NDTM) which run in polynomial time over input domain \( \{0,1\}^* \). For every \( i \), \( \psi_i = (\phi_{i1}, \phi_{i2}) \), where \( \phi_{i1} \)'s running time may differ from that of \( \phi_{i2} \). (Every pair of two different machines appears twice in the enumeration, in the two possible orders.)

**Definition 2.1:** \( \psi_i \) is a matched pair if \( \phi_{i2} \) accepts the complement of the language accepted by \( \phi_{i1} \).

**Definition 2.2:** A word \( z \in \{0,1\}^* \) witnesses \( \psi_i \)'s mismatch if \( z \) is accepted by both \( \phi_{i1} \) and \( \phi_{i2} \) or by none of them, or in abbreviated form:

\[
(z \in \phi_{i1} \cap \phi_{i2}) \lor (z \in \phi_{i1} \cup \phi_{i2})
\]

here \( \phi_{i1} \) and \( \phi_{i2} \) stand for the languages accepted by the machines \( \phi_{i1} \) and \( \phi_{i2} \).
Definition 2.3: Let $\psi_i$ be a matched pair. $\psi_i$ agrees with language $A$ if $A = \phi_i$.

From Definition 2.3 it follows that $A \notin I$ iff there exists a matched pair $\psi_i$ such that $A = \phi_i$, i.e. $\psi_i$ agrees with $A$. This follows from the fact that every pair appears twice in the two possible orders.

Let us denote by $A \not\leq B$ and $A \not\subseteq B$ the existence of Karp and Cook reductions from $A$ to $B$, respectively. (See, for example, [2,3] for discussion on reductions.)

Let $\{M_i\}$ denote an effective enumeration of polynomial-time Oracle Deterministic Turing Machines (ODTM) acceptors. $M_i(A)$ will denote the language defined by $M_i$ if it uses $A$ as its oracle.

Finally, $N$ denotes the set of non-negative integers.

Theorem 2.1: $(\forall B \notin I)(\exists A \in I)[A \not\leq B \land B \not\subseteq A]$.

Proof: Given $B$ we define simultaneously a polynomial time Deterministic Turing Machine (DTM) $T$, with domain $\{0,1\}^*$ and $\text{Range} \subseteq \{1\}^*$, and a language $A \triangleq \{x \in B \mid |T(x)| \text{ is even}\}$.

Since $B \notin I$ there exists a word $\omega \in B$. Assuming the existence of $T$, it is easy to show that $A \not\leq B$, as follows: Given an input word $x$, first compute $T(x)$. If $|T(x)|$ is even feed $x$ to the oracle $B$; otherwise feed $\omega$ to $B$.

The definition of $T$ will imply that the following infinite list of conditions holds.

0) Either $\psi_o$ is a mismatch or it disagrees with $A$, 

1) $B \not\subseteq M_o(A)$, 

2) Either $\psi_1$ is a mismatch or it disagrees with $A$, 

3) $B \not\subseteq M_1(A)$, 

etc.
Clearly, the even numbered conditions guarantee that $A \notin I$, and the odd numbered conditions guarantee that $B \notin A$. 

$T$ is defined as follows: $T(\lambda) = \lambda$, and $T$'s output depends on the length of its input only; thus a definition of $T(0^n)$ ($0^n$ is the word consisting of $n$ zeroes) is all we need. The definition is recursive:

begin
(1) Within $n$ steps try to compute $T(\lambda), T(0), T(0^2), \ldots$, let $T(0^m)$ be the last output word computed in this sequence.
(2) Allot $n$ steps for the following task. $T(0^n) := T(0^m)$ if the task does not terminate within the allotted time.

Case (i): $|T(0^m)|$ is even. $i := |T(0^m)|/2$. For $z = \lambda, 0, 1, 00, 01, \ldots$
if $z$ is a witness for $\psi_i$'s mismatch or for $A \neq \phi_i$, then $T(0^n) := T(0^m)1$, end.

Case (ii): $|T(0^m)|$ is odd. $i := (|T(0^m)| - 1)/2$. For $z = \lambda, 0, 1, 00, 01, \ldots$
if $z$ is a witness for $B \neq M_i(A)$ then $T(0^n) := T(0^m)1$, end.

Remarks:
(1) In order to decide if $z \in A$, $T$ simulates $B$ and $T$ recursively ($|z| < n$, since there are $2^n$ words shorter than $n$, but only $n$ moves allotted).
(2) In case (ii) the running time of oracle $A$ is also counted within the $n$ steps allotted.
(3) Notice that in case (i) $T$ "tries to verify" that one of the even numbered conditions in our infinite list, holds, and in case (ii) - that one of the odd numbered conditions holds.

Our first purpose is to show that for every $n$

$$|T(0^n)| \leq |T(0^{n+1})| \leq |T(0^n)| + 1.$$

To do this, that Thue-Provensal condition to be checked - Second
we show that whenever \( T \) starts a verification of some condition for 
\( 0^n \), it continues to check the same condition for larger and larger \( n \)'s, 
until eventually the time allotted suffices and the desired witness 
is found.

We prove that \( |T(0^n)| \leq |T(0^{n+1})| \leq |T(0^n)| + 1 \) by induction on \( n \). 
Surely \( |T(\lambda)| \leq |T(0)| \leq |T(\lambda)| + 1 \). Assume inductively that for all \( \lambda < n \) 
\( |T(\lambda)| \leq |T(\lambda\lambda)| \leq |T(\lambda\lambda)| + 1 \).

Let \( m \) and \( m' \) be the lengths of the largest input for which \( T \) 
completes the calculation in step (1), for inputs \( 0^n \) and \( 0^{n+1} \), respectively.

Now

\[
m \leq m' \leq n. \tag{1}
\]

The left inequality follows from the definition of step (1) of \( T \), and 
the right inequality follows from the fact that only \( n+1 \) steps are 
allotted for step (1) of \( T \) for input \( 0^{n+1} \), and it takes at least \( n+1 \) time 
units to express \( T(\lambda), T(0\lambda), T(0\lambda\lambda), \ldots, T(0^n) \).

From (1) and the l.h.s. of the inductive hypothesis it follows that

\[
|T(0^m)| \leq |T(0^m)| \leq |T(0^n)|. \tag{2}
\]

Case A: \( |T(0^m)| = |T(0^m')| \).

Case A1: \( |T(0^n)| = |T(0^m')| \):

\[
|T(0^n)| = |T(0^m')| = |T(0^m')| \leq |T(0^{n+1})| \leq |T(0^m')| + 1 =

= |T(0^m)| + 1 = |T(0^n)| + 1.
\]

Case A2: \( |T(0^n)| = |T(0^m')| + 1: \)

Since the condition to be checked is the same for \( m \) and \( m' \), and 
since \( n \) time units were sufficient for the completion of the task, it 
follows that \( n+1 \) time units suffice and \( |T(0^{n+1})| = |T(0^m')| + 1 =

\[
|T(0^m)| + 1 = |T(0^n)| + 1.
\]

Case B: \( |T(0^m')| = |T(0^m)| + 1: \)

by (2) and the inductive hypothesis \( |T(0^m)| = |T(0^m)| + 1 \), hence

\[
|T(0^n)| = |T(0^m')| \leq |T(0^{n+1})| \leq |T(0^m')| + 1 \leq |T(0^n)| + 1.
\]
In order to complete the proof we have to show that \( T(\cdot) \) is unbounded, i.e.
\[
\neg \left( \exists j, n \right) \left( \forall n \geq n \right) \left[ T(n) = 1 \right].
\] (3)
Assume the contrary.

Case (i): \( j \) is even. Let \( i = \lfloor j/2 \rfloor \). The negation of (3) implies that \( \psi_1 \) is matched, and \( \psi_1 \) agrees with \( A \), i.e. \( A \in I \). Also, since \( j \) is even \( A = B \) except for finitely many words, and \( B \in I \), contradicting our assumption on \( B \).

Case (ii): \( j \) is odd, let \( i = \lfloor j/2 \rfloor \). Now, (3) implies that \( B = M_i(A) \), and since \( j \) is odd \( A \) is finite.

These two facts imply \( B \notin P \), a contradiction.

Q.E.D.

**Definition 2.4:** Let \( A \xrightarrow{C} B \) denote \( A \xrightarrow{C} B, A \xrightarrow{C} A \).

Since \( \leq \) is reflexive and transitive relation, and since \( \leq_{C} \) is symmetric \( \leq_{C} \) is an equivalence relation.

This relation imposes a partition of the languages into equivalence classes. We call these classes 'Cook equivalence classes'. (In [1] they are called 'polynomial \( T \) degrees'.)

**Corollary 2.1:** Assume \( NP \neq \text{CONP} \). It is well known that members of \( NPC_{\text{COOK}} \) cannot belong to \( I_\text{p} \). Hence from Theorem 2.1 we conclude that \( NP-I_{\text{p}}-NPC_{\text{COOK}} \neq \emptyset \).

In fact we conclude that \( NP-I_{\text{p}}-NPC_{\text{COOK}} \) contains infinitely many Cook-equivalence-classes. This follows from Theorem 2.1 applied first to \( B \in NPC_{\text{COOK}} \), the resulting \( A \) is in \( NP-I_{\text{p}}-NPC_{\text{COOK}} \). Next, apply Theorem 2.1 to \( A \), etc...

3. ON THE COMPLEXITY OF PROBLEMS NOT IN \( \text{U} \).

**Notation:**

For every one-tape \( \text{DTM-}M, \gamma_{M} \) represents the set of control states, \( \gamma_{M} \) represents the set of tape symbols, \( \delta_{M} \) is the next state function,
\( q_a, q_r \in Q_M \) are the final accepting and rejecting states, respectively. \( S_M \subseteq Q_M \times T_M \) is defined as follows: if \((q, t) \in S_M\) then \( \delta_M(q, t) \in \{q_a, q_r\}\); also we assume that \( q_M, q_a, q_r \in S_M \). \( \text{Des}(D) \) denotes the description of a DTM-D.

**Definition 3.1:** A function \( h : \{0,1\}^* \to \mathbb{N} \) is a running time function if there exists a DTM-N such that for every input \( x \), M runs exactly \( h(x) \) steps.

Clearly, a running time function, as defined above, is a total function. (This definition is different from that of 'time constructible function' [2], in that the latter has \(|x|\), the length of \( x \), as an operand. Clearly, if \( f(\cdot) \) is time constructible, then \( g(x) \), defined by \( g(x) = f(|x|) \), is a running time function.

For every total running time function \( h \) we define problem \( \alpha_h \) as follows:

**Input:** \( \text{Des}(N), x \), where \( N \) is a DTM and \( x \) a binary string.

**Property:** \( N \) accepts \( x \) in time \( \leq h(x) \).

Clearly, \( \alpha_h \) denotes the following problem:

**Input:** \( \text{Des}(M), y \).

**Property:** \( M \) does not accept \( y \) in time \( \leq h(y) \).

We show a Karp reduction from \( \alpha_h \) to \( \alpha_h \). The main idea in the reduction is that while we are not allowed to negate the oracle's answer, we can impose the desired negation into the construction of \( N \).

**Lemma 3.1:** \( \alpha_h \leq_k \alpha_h \).

**Proof:** Given \((\text{Des}(N), y)\), input instance of \( \alpha_h \), we construct \((\text{Des}(N), x)\), input instance of \( \alpha_h \), as follows: First \( x := y \). The DTM-N is basically a parallel operation of \( M \) and a DTM-D whose running time is exactly \( h(x) \). More precisely, let \( Q_N, N \) set of states, be \( Q_N = Q_M \times Q_D \cup \{q_r, q_a\} \).
We define \( N \)'s next state function as follows:

\[
\delta_N((q_M, q_D), (t_M, t_D)) = \begin{cases} 
(q_a_N, t_M', t_D) & \text{if } (q_M, t_D) \in S_D \\
(q_r_N, t_M', t_D) & \text{if } (q_M, t_D') \in S_D \land (q_M, t_M') \in S_M \land \delta_M(q_M, t_M') = q_r_M,
\end{cases}
\]

(Remark: according to the definitions in the beginning of this chapter
\[
\delta_N(q_r_N, (t_M, t_D)) = q_r_N, \quad \text{and} \quad \delta_N(q_a_N, (t_M, t_D)) = q_a_N.
\]

The definition of \( \delta_N \) implies that for every input \( x \), \( N \) runs exactly \( h(x) \) steps. In addition, whenever \( M \) halts within \( h(y) \) steps \( N \)'s answer is opposite to \( M \)'s answer, and if \( M \) does not halt within \( h(y) \) steps \( N \) accepts. Therefore, the oracle's answer for the instance \(( \text{Des}(N), x)\) of \( \alpha_h \) is the right answer for the instance \(( \text{Des}(M), y)\) of \( \tilde{\alpha}_h \). The mapping defined is clearly polynomial since \( |\text{Des}(D)| \), the length of \( \text{Des}(D) \), is fixed.

Q.E.D.

We proceed with a variation on a well known lemma due to A. Meyer.

Lemma 3.2: If \( \tilde{A} \in P \) then \( A \in U \) implies \( A \in I \).

Proof: Let \( M \) be a DTM which performs the mapping from \( \tilde{A} \) to \( A \) in polynomial time. Suppose \( A \in \text{NP} \) and let \( N \) be NDTM which accepts \( A \) in polynomial time. The NDTM constructed by first applying \( M \) to the input and then applying \( N \) to \( M \)'s output accepts \( \tilde{A} \) in polynomial time, i.e. \( \tilde{A} \in \text{NP} \). Hence \( \tilde{A} \in I \).

On the other hand, if \( A \in \text{CONP} \) then \( \tilde{A} \in \text{NP} \) and it is sufficient to show that \( A \in K \tilde{\alpha} \), which is clearly the case, since \( \tilde{\beta} \in K \gamma \) implies \( \tilde{\beta} \in K \gamma \).

Q.E.D.

Corollary 3.1: From Lemmas 3.1 and 3.2 it follows that for every total running time function \( h \), \( \alpha_h \in U \) implies \( \alpha_h \in I \).
The following theorem is meaningful under the assumption \( NP \neq \text{CONP} \).

**Theorem 3.1:** If \( \beta \in \mathbb{U}^1 \), and DTM-\( D \) solves \( \beta \) in running time \( h(x) \), then:

(a) \( \beta \overset{K}{\rightarrow} \alpha_h \)

(b) \( \alpha_h \in \mathbb{U} \)

(c) Let \( N \), the DTM whose description is included in the input of \( \alpha_h \) and \( D \) be one-tape DTM's. There exists a four-tape DTM-\( M \), which solves \( \alpha_h \) in \( O(|\text{Des}(D)| + |\text{Des}(N)|).h(x) \) time.

**Proof:**

(a) There exists a polynomial DTM-\( T \), which given \( x \) (the input instance of \( \beta \)) produces \( (\text{Des}(D),x) \). This follows from the fact that \( D \) is independent of \( x \), and a fixed \( D \) can be used. [This proof of \( T \)'s existence, is clearly non-constructive.] \( (\text{Des}(D),x) \) can now be used as the input instance for \( \alpha_h \). Clearly, \( (\text{Des}(D),x) \in \alpha_h \) iff \( x \in \beta \).

(b) If \( \alpha_h \in \mathbb{I} \) then, by part (a), \( \beta \in \mathbb{I} \). Thus, \( \alpha_h \notin \mathbb{I} \). By Corollary 3.1, \( \alpha_h \notin \mathbb{U} \).

(c) Use a four-tape universal DTM-\( M \), to simulate alternately, \( \tilde{N} \) and \( \tilde{D} \), each with the input \( x \). If, while simulating a step of \( \tilde{N} \), \( q_{\tilde{N}} \) or \( q_{a_{\tilde{N}}} \) is reached, then \( M \) halts with a corresponding answer. If while simulating a step of \( \tilde{D} \), \( q_{\tilde{D}} \) or \( q_{a_{\tilde{D}}} \) is reached then \( M \) halts with a negative answer.

The simulation of each step of \( \tilde{N} \) \( (\tilde{D}) \), takes \( O(|\text{Des}(N)|) \) \( O(|\text{Des}(D)|) \) time, and the number of simulation steps is at most \( h(x) \).

Q.E.D.
CONCLUSION

By Theorem 2.1, we know that if $\text{NP} \neq \text{CONP}$ then there exist problems $\beta \in \text{NP}-\text{I-NPC}_{\text{COOK}}$. For each such $\beta$ there exists a corresponding problem $\alpha_h$ which is not in $U$. Its complexity is roughly that of $\beta$; by applying Theorem 3.1 to an efficient $\text{DTM-D}$ which solves $\beta$, part (c) implies that the time complexity of $\alpha_h$ is similar to that of $\beta$.

This suggests the interesting possibility that while one problem can be "harder" than another in the non-deterministic model ($\alpha_h$ "harder" than $\text{NPC}_{\text{COOK}}$ problems) it may be "easier" if one uses the deterministic model ($\alpha_h$ has, approximately, the time complexity of $\beta$, which is "easier" than the $\text{NPC}_{\text{COOK}}$ problems).

REFERENCES


