TWO ORACLES THAT FORCE A BIG CRUNCH

HARRY BUHRMAN, STEPHEN FENNER, LANCE FORTNOW,
AND LEEN TORENVLIET

Abstract. We construct an oracle $A$ such that $\text{NEXP}^A = \text{P}^{\text{NP}^A}$. The construction of this oracle answers a long standing open question first posed by Heller, and unsuccessfully attacked many times since. For the first construction of the oracle, we present a new type of injury argument that we call “resource bounded injury.” In the special case of the construction of this oracle, a tree method can be used to transform unbounded search into exponentially bounded, hence recursive, search. This transformation of the construction can be interleaved with another construction so that relative to the new combined oracle also $\text{P} = \text{UP} = \text{NP} \cap \text{coNP}$. This leads to the curious situation where $\text{LOW}(\text{NP}) = \text{P}$, but $\text{LOW}(\text{NP}^A) = \text{NEXP}$, and the complete $\leq_p^m$-degree for $\text{P}^{\text{NP}}$ collapses to a single $p$-isomorphism type.

Keywords. Relativization, Complexity Classes, Polynomial Hierarchy, Exponential Hierarchy, Priority Arguments, Diagonalization

Subject classification. 68Q05, 68Q10, 68Q15

1. Introduction

In 1978, Seiferas, Fischer and Meyer (SFM78) showed a very strong separation theorem for nondeterministic time: For time constructible $t_1(n)$ and $t_2(n)$, if $t_1(n+1) \in o(t_2(n))$ then $\text{NTIME}(t_1(n))$ does not contain $\text{NTIME}(t_2(n))$. Thus we have a huge gap between nondeterministic polynomial time (NP) and nondeterministic exponential time NEXP. We would also expect then a separation between $\text{P}^{\text{NP}}$ and $\text{P}^{\text{NEXP}}$. Indeed, we have some evidence for that direction: Micas (Moc96), improving upon work of Fu, Li and Zhong (FLZ94) showed that for any fixed $c$, NEXP is not contained in $\text{P}^{\text{NP}}$ if the polynomial machine can only ask $n^c$ queries to the NP oracle. Her technique can however not be extended to prove that NEXP is not contained in $\text{P}^{\text{NP}}$ unconditionally. Indeed the present paper shows that no relativizing technique can show this unconditionally.
Mocas' result however does put a restriction on the simplicity of our oracle construction. It is well-known that $P^{\text{NP}[\log]} = P^\text{NP}$ (Hem89). Since $n^c$ majorizes log for any $c$, Mocas' result also shows that $\text{NEXP} \not\subseteq P^{\text{NP}}$. Our oracle construction proceeds by encoding the standard complete set $K$ into the oracle $A$ such that a fixed $P^{\text{NP}}$ oracle machine can decide membership in $K^A$ by asking queries of $A$. Mocas' results shows that this oracle machine can neither be non-adaptive nor linearly bounded. In many oracle papers, the construction is such that the test language only requires a bounded number of queries (often even one).

There are also some other obstacles that our construction must avoid, and that might explain why this problem has remained open for such a long time. Coding $\text{NEXP}^A$ into $P^{\text{NP}^A}$ implies that $\text{EXP}^A = \text{EXP}^\text{NP}$. It is not known whether the Exponential Time Hierarchy (EXPH), defined below, has the downward separation property—a separation of two adjacent levels implying separations at all lower levels—as does the Polynomial Time Hierarchy (PH). For PH, this property relativizes. If EXPH should have this property, then it follows from a relativization of the deterministic time hierarchy (HS85) that $P^{\text{NP}} \neq \text{NEXP}$, since $P^{\text{NP}} \neq \text{EXP}^{\text{NP}}$. Hartmanis, Immerman and Sewelson (HIS85) observed that if EXPH should have this property, then this property does not relativize. They constructed an oracle $A$ such that $\text{EXP} = \text{NEXP}$, yet the second level of EXPH is proper. A much weaker property for EXPH, known as Sewelson's Conjecture (Sew83), is the following statement: $\text{EXP} = \text{NEXP} \implies \text{NEXP} = \text{EXP}^{\text{NP}}$. In a world where Sewelson's conjecture holds, $P^{\text{NP}} \neq \text{NEXP}$, since $P^{\text{NP}} \neq \text{EXP}^{\text{NP}}$. Our construction has to avoid this problem, i.e., in the world we construct Sewelson's conjecture must not hold. In Section 3 we will introduce a language for encoding the complete set $K^A$ and the resource bounded injury method, which overcome these difficulties. In that section, we will also give the first construction of the oracle.

In Section 4 we will transform the construction to a recursive construction using a tree-based method (see (Soa87), Chapter XIV) where, if we can get an injury from one leaf in a tree we can then throw away that leaf and start using the next one. We arrange the tree so that we have more leaves than injuries. Also, this new recursive method can be combined with techniques developed by Rackoff (Rac82), Hartmanis and Hemachandra (HH91) and Blum and Impagliazzo (BI87) to get $P = U P = NP \cap \text{coNP}$, while still having $P^{\text{NP}} = \text{NEXP}$ relative to a recursive oracle. We eliminate machines that do not categorically accept on at most one path by forcing the acceptance on one of the leaves of our tree and then using a future leaf for encoding.

The relativized world we create has two interesting properties. The first is
Two Oracles that Force a Big Crunch

about low sets. The low sets of a class \( \mathcal{C} \) consists of those oracles \( A \) such that \( \mathcal{C}^A = \mathcal{C} \). Essentially, the low sets of a class \( \mathcal{C} \) give no help to the class \( \mathcal{C} \).

Relative to our oracle, we get a surprising relationship between the low sets for \( \mathsf{NP} \) and the low sets for \( \mathsf{P}^{\mathsf{NP}} \). The first consist of exactly the languages in \( \mathsf{P} \), whereas the second consists of the languages in \( \mathsf{NEXP} \).

The second property of our relativized world is that the complete \( \leq^p_m \)-degree for \( \mathsf{P}^{\mathsf{NP}} \) collapses to a single \( p \)-isomorphism type. That is, all \( \mathsf{P}^{\mathsf{NP}} \)-complete sets under polynomial-time many-one reductions are polynomial-time isomorphic. This advances a program for getting complete degrees to collapse: if \( \mathcal{C} \subseteq \mathsf{NEXP} \) is a class with complete sets, then one can make all \( \mathcal{C} \)-complete sets \( p \)-isomorphic in a relativized world by setting \( \mathsf{P} = \mathsf{UP} \) and \( \mathcal{C} = \mathsf{EXP} \). Homer and Selman (HS92) were able to do this for \( \mathcal{C} = \Sigma^p_2 \), and our present oracle works for \( \mathcal{C} = \mathsf{P}^{\mathsf{NP}} \). An oracle for \( \mathcal{C} = \mathsf{NP} \) would confirm the isomorphism conjecture (BH77). Even though oracles making the conjecture true are known (FFK92; BBF98), it is still an interesting open question whether there is an oracle relative to which \( \mathsf{P} = \mathsf{UP} \) and \( \mathsf{NP} = \mathsf{EXP} \).

2. Preliminaries

We assume the reader familiar with standard notions in structural complexity theory, as are defined, e.g., in (BDG88). Nonetheless, we will in this section recall some notions that we feel are not common knowledge, and fix on some notation.

Sets are denoted by capital letters and are subsets of \( \Gamma^* \), where \( \Gamma = \{0, 1\} \). The cardinality of a set \( A \) is denoted as \( |A| \). Strings are denoted as small letters \( x, y, u, v, \ldots \). We sometimes identify a set with its characteristic function, that is, \( A(x) = 1 \) if \( x \in A \), and \( A(x) = 0 \) otherwise.

The length of a string \( x \) is denoted by \( |x| \). For a set \( A \) and \( n \in \omega \), we let the notation \( A^{\leq n} \) stand for the set consisting of all the strings in \( A \) of length \( n \). We write \( \Gamma^n \) for \( (\Gamma^*)^n \). Strings are ordered first by length and then lexicographically. By definition it holds for any string \( z \) that \( z > \text{max}\emptyset \) in this ordering. This ordering also lets us identify strings with natural numbers \( 0, 1, 2, \ldots \).

We assume a 1-1, onto, easy-to-compute pairing function (the function \( \langle x, y \rangle = y + (x + y)(x + y + 1)/2 \) will do). Sets can also appear as arguments, e.g., \( \langle x, B \rangle = \{\langle x, y \rangle : y \in B\} \). We assume standard enumerations of recursively presentable classes by (oracle) Turing machines. An oracle machine is a multi-tape Turing machine with an input tape, an output tape, work tapes, and a query tape. Oracle machines have three distinguished states \texttt{QUERY},
YES and NO, which are explained as follows: at some stage(s) in the computation the machine may enter the state QUERY, and then goes to the state YES, or goes to the state NO, depending on the membership of the string currently written on the query tape in a fixed oracle set. Computation paths of a nondeterministic machine can be ordered by ordering the state set of that machine. An accepting computation that is minimal in this sense can thus be defined. Such a computation is called the leftmost computation of $M$ on input $x$. If no accepting computation exists then the leftmost computation is undefined. If the machine is deterministic then the leftmost computation is the unique computation if the machine accepts. We let $M(x)$ stand for the leftmost computation of machine $M$ on input $x$. We use the same notation for oracle machines $(M^A(x))$. We let $Q(M^A(x))$ be the set of queries that is asked along $M^A(x)$ if $M^A$ accepts $x$, and $\emptyset$ otherwise. The length of a computation, i.e., the number of steps, is denoted by $|M^A(x)|$. For a nondeterministic machine this stands for the length of its leftmost accepting computation if it exists, and 0 otherwise. By focusing on a particular computation path we may seem to get a nonstandard interpretation of time bounds here. However, since we implicitly use clocked machines, this is no objection. The set of strings accepted by a Turing (oracle) machine $M$ (with oracle $A$), is called the language of $M$ (relative to $A$) and is denoted by $L(M)$ ($L(M^A)$). A relativized complexity class is denoted by writing the oracle as a superscript, e.g., $P^A$. A complexity class may also appear as a subscript to another complexity class, denoting the class of languages emerging from the operation of equipping a machine from the class with an oracle from the superscript class, e.g., $P^{NP} = \{L(M^X) : M$ is a deterministic polynomial time oracle machine and $X \in NP\}$.

The main complexity classes in this paper are $P$, $NP$, $EXP$, and $NEXP$, where exponential time is taken to be two to the power polynomial $(2^{n^{i+1}}$, for $i \in \omega)$. These classes are members of the so-called Polynomial Time Hierarchy and the Exponential Time Hierarchy respectively, which we define below.

The Polynomial Time Hierarchy was introduced by Stockmeyer (Sto76) and consists of the infinite collection of classes $\Sigma_i^p$, $\Delta_i^p$ and $\Pi_i^p$, which are defined inductively as follows:

$$
\begin{align*}
\Sigma_0^p &= \Pi_0^p = \Delta_0^p = \Delta_1^p = P \\
\Sigma_{i+1}^p &= \text{NP}^{\Sigma_i^p} \\
\Pi_{i+1}^p &= \text{Co}\Sigma_{i+1}^p \\
\Delta_{i+1}^p &= \text{P}^{\Sigma_i^p} 
\end{align*}
$$

The Exponential Time Hierarchy as an analog of the Polynomial Time Hierarchy is defined as follows:

$$
\begin{align*}
\Sigma_0^{exp} &= \Pi_0^{exp} = \Delta_0^{exp} = \Delta_1^{exp} = EXP 
\end{align*}
$$
\[
\Sigma_{i+1}^{\text{exp}} = \text{NEXP}_{\Sigma_i^{\text{exp}}}
\]
\[
\Pi_{i+1}^{\text{exp}} = \text{CoNEXP}_{\Sigma_{i+1}^{\text{exp}}}
\]
\[
\Delta_{i+1}^{\text{exp}} = \text{EXP}_{\Sigma_i^{\text{exp}}}
\]

Relativizations of these hierarchies will be denoted by, e.g., \(\Sigma_i^{\text{exp},A}\), where the interpretation is that the highest level oracle is changed from \(\text{NP}\) to \(\text{NP}^A\).

3. The Resource Bounded Injury Method

3.1. Resource Bounded Injury. A first solution to Post’s problem was obtained by Friedberg (Fri57) and independently Muchnik (Muc56). They presented a means of construction that later became known as the “Finite Injury Priority Method.” Usually, an r.e. set that has some property with respect to all recursive oracle machines is constructed by stages. The set of all oracle machines forms an infinite set of requirements that have to be satisfied by the construction. At each stage one of these requirements is satisfied, and so the process guarantees that in the limit all the requirements are satisfied. The difference between earlier constructions and the finite injury priority method is that requirements that are satisfied at some stage, may become unsatisfied again at some later stage. This is called injuring a satisfied requirement. Here indexing the requirements becomes of major importance. A requirement may be injured at some stage only if this action is taken to satisfy a requirement of higher priority (i.e., with smaller index). For a given requirement there are only finitely many requirements of higher priority (indexing starts with 0). If there are infinitely many stages at which a given requirement may be satisfied, it follows that every requirement eventually will be satisfied permanently (i.e., will not be injured after being satisfied). Soare’s book (Soa87) contains an excellent exposé of different methods of constructing r.e. sets. In complexity theory, especially in oracle construction, it is seldom necessary to resort to more complicated methods of construction than standard, slow, so-called wait-and-see arguments. The method, at stage \(s\), takes into consideration all requirements with index less than \(s\), and satisfies from the subset of these requirements that can be satisfied the one with the smallest index.

Satisfying a requirement usually means adding a set of strings to the oracle under construction. The requirements in our coding construction are indexed with the length of strings. Requirement \(R_{[x]}\) asks that strings \(x\) of length \(|x|\) be correctly encoded in the oracle. That is if \(x \in K^A\) then there has to be some code that says that it is and if \(x \notin K^A\) than there has to be some code that says it isn’t. In recursion theoretical constructions this code can, because recursive machines have unbounded time, be spread out through the entire oracle. In
our case however, we need to be able to retrieve the encoding by a resource-
bounded machine. In particular, a polynomial time oracle machine. Therefore, 
all of the encoding pertinent to \( x \) has to be done “close to” \( x \). That is, the 
length of the encoding strings may not be greater than a fixed polynomial in 
the length of \( x \). On a binary alphabet there are only \( 2^{p(|x|)} \) strings available of 
length \( p(|x|) \). Therefore a method that permits injuries, that is re-encoding of 
\( x \), should also have an implicit bound on the number of times that an injury 
is permitted. Otherwise, the space of encoding would simply fill up. Because 
our method has such a built in security, we call the method “resource bounded 
injury.”

3.2. Encoding and Retrieval. Let \( M_1, M_2, \ldots \) be some standard enumeration 
of nondeterministic oracle machines, and let \( K^A = \{ \langle i, x, n \rangle : x \in 
L(M_i^A) \land |M_i^A(x)| \leq n \} \), be the standard complete set for \( \text{NEXP}^A \). We plan to 
encode \( K^A \) into \( A \) such that it can be retrieved by a polynomial time bounded 
machine with the help of an \( \text{NP}^A \) oracle. The encoding of a string \( x \) will consist 
of strings \( \langle x, z \rangle \), where \( |z| = |x^2 + 2z| + 5 \). On input \( x \), the polynomial time 
oracle machine will retrieve the maximum (in the lexicographical order) string 
\( z \) such that \( \langle x, z \rangle \in A \), and will accept iff this \( z \) is odd.

Since for given \( x \) and \( z \) the query \( (\exists z' > z)|z'| = |z| \land \langle x, z \rangle \in A \) is an \( \text{NP}^A \) 
query, a \( P \) machine with an \( \text{NP}^A \) oracle can find \( \langle x, z \rangle \in A \) with maximum \( z \) 
by binary search and then decide whether this \( z \) is odd.

3.3. Construction. We first give an informal description of the construction. The construction of the oracle proceeds by stages. Let \( A_s \) be the oracle 
after stage \( s \). Let \( M_s^X \) be an oracle machine computing \( K_s^X \), and let \( M_s^X \) and 
\( N^X \) be polynomial time oracle machines (deterministic and nondeterministic, 
respectively) such that \( M_s^{L(N^X)} \) computes the \( \text{NP}^A \) binary search algorithm 
described above. Let \( C_s^X = L(M_s^{L(N^X)} \rangle \). The goal of stage \( s + 1 \) is to guarantee 
that at the end of this stage \( C_s^X \cap \Gamma^n = K^A \cap \Gamma^n \), for some \( n \) depending 
on the stage. To achieve this goal the construction builds a set \( B \) at stage \( s + 1 \) 
which is initially empty. Then it repeatedly selects a string \( x \) of length \( n \) and 
a minimal \( z \) of length \( n^2 + 2n + 4 \) such that

\( x \in K^A \cup B \Delta C^A \cup B, \) and

\( \{ \langle x, z0 \rangle, \langle x, z1 \rangle \} \cap Q(M^A (u)) = \emptyset \) for any \( u \) with \( |u| \leq |x| \).

The construction proceeds by letting \( b = K^A \cup B (x) \), then setting \( B \) to \( B \cup 
\{ \langle x, z0 \rangle \} \). Stage \( s + 1 \) ends when no more \( x \) in \( (K^A \cup B \Delta C^A \cup B) \cap \Gamma^n \) can be
found, i.e., when all strings of length \( n \) are correctly encoded in the oracle. We will prove that indeed each stage ends after a limited number of steps.

We call the need for the construction to correctly encode all strings of length \( n \) a requirement \( R_n \). We then say that requirement \( R_n \) is satisfied at stage \( s \) if \( C^A_s \cap \Gamma^n = K^A_s \cap \Gamma^n \). A requirement requires attention at stage \( s \) if it is not satisfied. A requirement that was satisfied at stage \( s \) may require attention at some stage \( t > s \), due to changes in the oracle. We say that this requirement is injured.

After stage \( s \) the construction acts upon the highest priority requirement that requires attention by setting \( n \) to the minimal \( m \) such that requirement \( R_m \) is not satisfied. This can be recognized by the fact that \( \Gamma^n \cap (K^{A_{s+1}} \Delta C^{A_{s+1}}) \neq \emptyset \). Note that since \( ||A_{s+1}|| < \infty \) such an \( m \) must exist for any given \( s \), for otherwise \( P^{NP} = \text{NEXP} \) with the empty oracle and we are done. We will prove that each requirement \( R_n \) will be satisfied and injured only a limited number of times (i.e., the encoding fits in the encoding space). As \( R_n \) is satisfied only if all strings of length \( n \) are correctly encoded, this proves the correctness of the construction. First we will present the construction slightly more formally.

**Construction**

**Stage \(-1\):** \( A_0 = \emptyset \).

**Stage \( s \geq 0 \):**

\( n_s = \min \{ p > 0 : (C^A_s \Delta K^A_s) \cap \Gamma^p \neq \emptyset \} \);

\( m_s = n_s^2 + 2n_s + 5 \);

\( t = 0 \); \( B_{s,0} = \emptyset \);

**While** \( \exists x \in (K^{A_s \cup B_s} \Delta C^{A_s \cup B_s}) \cap \Gamma^n \)

**Do**

- Pick the least such \( x \);

- Let \( z \) be minimum such that

  - \( |z| = m_s \),

  - \( \langle x, z \rangle \notin \bigcup_{|u| \leq n_s} Q(M_{K}^{A_s \cup B_s}(u)) \),

  - \( z \geq \max \{ u : |u| = m_s \wedge \langle x, u \rangle \in A_s \} \), and

  - \( z = z' \rightarrow x \in K^{A_s \cup B_s} \).

If no such \( z \) exists, then the construction ends;

Otherwise, let \( B_{s,t+1} = B_{s,t} \cup \{ \langle x, z \rangle \} \); /* \( x \) is encoded with \( z \) */

Set \( t = t + 1 \);

**Endwhile**

\( A_{s+1} = A_s \cup B_{s,t} \);

**Endstage \( s \)**

End of Construction
3.4. Correctness. The correctness proof consists of a series of lemmas. Most importantly, we must show that the construction goes on forever, that is, a \( z \) is found each time in the loop. The following lemma will show that the set of \( M_K \) queries is easy to avoid.

**Lemma 1.** \((\forall X \subseteq \Gamma^\star)(\forall n \in \omega)[\|\bigcup_{|u| \leq n} Q(M_K^X(u))\| \leq 2^{2n+1}].\)

**Proof.** On input \( u \) of length \( \leq n \) the number of steps of \( M_K^X \) is bounded by \( 2^n \). It follows that the number of queries in the leftmost computation on this input is bounded by \( 2^n \). As there are at most \( 2^{n+1} \) strings \( v \) of length \( \leq n \), the lemma is proved. \( \square \)

Next we show that the encoding done at a single level does not take too many encoding strings. For each stage \( s \) and each string \( x \) of length \( n_s \), we may find first that \( x \) is not in \( K^{A_s \cup B_s, s} \) at some point, and then have to encode this fact in \( B_{s,t+1} \). Next we may find for some \( t' > t \) that \( x \in K^{A_s \cup B_s, t'} \) and have to encode this fact in \( B_{s,t'+1} \). Since in subsequent iterations \( t'' \geq t' \) of the While loop the \( z \)'s are picked such that the new strings in \( B_{s,t''+1} \) are not queried in the leftmost computation of \( M_K^{A_s \cup B_s, t''}(x) \), this is the last time that we need to encode \( x \). We therefore have the following lemma:

**Lemma 2.** For every \( s \in \omega \) and \( x \in \Gamma^{n_s} \), \( x \) is encoded at most twice in stage \( s \).

The following lemma is central to the correctness of the construction. It states that the number of times that the strings of a given length are encoded in the oracle is limited. And therefore that the room to code is present.

**Lemma 3.** \((\forall m)[\|\{s : n_s \leq m\}\| < 2^{m^2+1}].\)

**Proof.** We prove this by induction on \( m \). Let \( S_m = \{s_0 < s_1 < \ldots\} \) be the maximal set such that \( n_{s_i} = m \) for \( s_i \in S_m \). Thus \( \|\{s : n_s \leq m\}\| = \|S_m\| + \|\{s : n_s < m\}\| \). For \( m = 1 \), since there are no stages \( s \) such that \( n_s < 1 \), we have \( \|\{s : n_s \leq 1\}\| = \|S_1\| \). Let \( i \) be arbitrary such that \( s_i \) and \( s_{i+1} \) both exist in \( S_1 \). From the construction it follows that \( K^{A_i} \cap \Gamma^1 \subseteq K^{A_{i+1}} \cap \Gamma^1 \). Since accepting paths of \( M_K \) are preserved, \( K^{A_i} \cap \Gamma^1 \) can never lose elements at any stage, and the final proper containment follows from the fact that we are visiting length \( 1 \) at stage \( s_{i+1} \). Since there are only two strings of length \( 1 \), \( \|S_i\| \leq 3 \).

Now fix \( m > 1 \) and suppose as induction hypothesis \((\forall r < m)[\|\{s : n_s \leq r\}\| \leq 2^{r^2+1} - 1] \). Observe that \((\forall r)(\exists s)[s_i < s < s_{i+2m+1} \land n_s < m] \). This is
due to the fact that, for any \( i \) such that \( s_i \) and \( s_{i+1} \) exist in \( S_m \), if \( (\forall s_i < s < s_{i+1} \implies n_s \geq m) \) then \( K_{A_s} \cap \Gamma^m \subseteq K_{A_{i+1}} \cap \Gamma^m \) as in the \( m = 1 \) case. As long as \( n_s \geq m \), all accepting computations of \( M_k \) at length \( m \) are preserved by the construction. Hence the construction can only fall back to length \( m \) to encode a new accepting computation. Since there are only \( 2^m \) strings of length \( m \), \( K_{A_{m+1}} \cap \Gamma^m \) cannot be properly larger than \( K_{A_s} \cap \Gamma^m \) for more than \( 2^m \) consecutive \( i \). The induction hypothesis gives us an upper bound on how often \( n_s < m \). We find that \( \|S_m\| \leq \|\{s : n_s < m\}\| \times (2^m + 1) + 1 \). Since \( \|\{s : n_s < m\}\| \leq 2^{(m-1)^2 + 1} - 1 \), it follows that \( \|\{s : n_s < m\}\| \leq 2^{(m-1)^2 + 1} - 1 + (2^{(m-1)^2 + 1} - 1) \times (2^m + 1) + (2^m + 1) = (2^{(m-1)^2 + 1} - 1) \times (2^m + 2) + (2^m + 1) < 2^{(m-1)^2 + 1} \times (2^m + 2) \leq 2^{(m-1)^2 + 1} + (m + 1) = 2^{m^2 - m + 3} \). Since \( m \geq 2 \), this is less than or equal to \( 2^{m^2 + 1} \).

We are now ready to prove

**Lemma 4.** No stage ends for lack of coding strings.

**Proof.** Fix an \( n \) \( \geq 1 \), and let \( s_1 < s_2 < \cdots < s_k \) be all the stages \( s \) with \( n_s = n \). By Lemma 3, we have \( k \leq 2^{n^2 + 1} \). We show that if the construction reaches the start of a stage \( s_i \), then it continues beyond \( s_i \).

Fix any \( x \) of length \( n \), and let \( m = n^2 + 2n + 5 \). At the start of stage \( s_1 \), we have \( \langle x, \Gamma^m \rangle = \emptyset \). Suppose \( x \) first needs to be encoded at some iteration \( t_1 \) of the loop. There are \( 2^{m-1} \) pairs \( \{z'0, z'1\} \subseteq \Gamma^m \), and by Lemma 1, we have at least \( 2^{m-1} - 2^{2n+1} \) of these pairs available for encoding \( x \) while avoiding \( \bigcup_{|u| \leq n} Q(M_K^{A_{t_1}}(u)) \). Thus \( x \) can be encoded with a minimal string \( z_1 \in \Gamma^m \) leaving at least \( 2^{m-1} - 2^{2n+1} - 1 \) pairs of strings \( w > z_1 \) available for encoding \( x \) in the future. In the worst case, \( x \) needs to be encoded once more in stage \( s_1 \), say at some iteration \( t_2 > t_1 \) with a minimal string \( z_2 \). Among the \( 2^{m-1} - 2^{2n+1} - 1 \) available pairs of coding strings, we now have at least \( 2^{m-1} - 2 \times 2^{n+1} - 1 \) that we can use while avoiding \( \bigcup_{|u| \leq n} Q(M_K^{A_{t_2}}(u)) \) (again using Lemma 1). After encoding \( x \) the second time, we then have at least \( 2^{m-1} - 2^{2n+2} - 2 \) pairs of coding strings \( w > z_2 \) for encoding \( x \) in future stages.

Continuing the same argument through stages \( s_2, s_3, \ldots \), we see that after stage \( s_i \) we have at least \( 2^{m-1} - i(2^{n+i+1} + 1) \) pairs of strings available for encoding \( x \) at future stages. Since \( i \leq k \leq 2^{n+1} \), we have at least \( 2^{m-1} - 2^{n+i+1} = 0 \) pairs of coding strings left for \( x \) after every stage \( s \) with \( n_s = n \). Thus we always have enough coding strings to complete every stage.

\( \square \)

From this Lemma it follows immediately that indeed each stage ends with the level correctly encoded in the extension of the oracle. That is,
Corollary 5. \((\forall s)(C^{A_{s+1}} \Delta K^{A_{s+1}}) \cap \Gamma_{n_s} = \emptyset)\).

Corollary 6. \(\lim_{n \to \infty} n_s = \infty\).

Proof. This follows directly from Lemma 3. \(\square\)

From these lemmas the correctness of the construction follows. The oracle is constructed through a recursive procedure, hence it is recursively enumerable. As the construction can "fall back" to a length \(n\) at any point in the construction (through a chain of changes in the oracle) we cannot conclude recursiveness.

Theorem 7. There exists a recursively enumerable oracle \(A\) such that \(P^{NPA} = NEXPA\).

3.5. Consequences. The following corollary was the initial motivation for the construction of the oracle.

Corollary 8. There exists an oracle \(A\) such that \(P^{NPA} = P^{NEXPA}\).

Proof. The oracle constructed in Theorem 7 makes \(P^{NPA} = NEXPA\). It now follows that \(P^{NPA} = P^{NEXPA}\). \(\square\)

Corollary 8 suggests that there is no tight hierarchy theorem (at least not one that relativizes) for \(P^{\text{NTIME}(f(n))}\) time classes, for suitable functions \(f\). Whereas such a theorem is true and relativizes for \(\text{NTIME}(f(n))\) (SFM78; Zák83). An even further collapse follows directly from the observation that the techniques in (Hem89) relativize.

Corollary 9. There exists an oracle \(A\) such that \(P^{NPA} = \text{NP}^{NEXPA}\).

An equivalent statement is the following.

Corollary 10. There exists an oracle \(A\) such that \(\text{EXP}^{NPA}[\text{poly}] = P^{NPA}\).

Proof. Again use the techniques in (Hem89) together with oracle \(A\) in Theorem 7. \(\square\)

Again this is a strange but not contradictory consequence. We know from the deterministic hierarchy theorem that \(\text{EXP} \nsubseteq \text{P}\) and that this theorem relativizes. The previous corollary shows that if the exponential time oracle machines are restricted to querying only polynomially many queries (as are the polynomial time oracle machines), then the two classes coincide relative to \(A\). The following corollaries are also mentioned as open problems in (Hel84).
Corollary 11. There is an oracle $A$ such that $\text{EXP}^A = \text{NEXP}^A \subseteq \text{EXP}^{\text{NP}^A} = \Sigma_2^{\text{exp}^A}$.

Proof. Take again oracle $A$ in the proof of Theorem 7. It follows from the fact that the Polynomial Time Hierarchy collapses to $\text{P}^{\text{NP}^A}$, by padding, that the Exponential Time Hierarchy collapses to $\text{EXP}^{\text{NP}^A}$. Furthermore, the relativized time hierarchy theorem for deterministic oracle machines implies $\text{P}^{\text{NP}^A} \neq \text{EXP}^{\text{NP}^A}$. $\Box$

This corollary implies the following property.

Corollary 12. (IT89).

There exists an oracle $A$ such that Savitch's conjecture fails relative to $A$.

Another consequence is the following.

Corollary 13. (BH91).

There exists an oracle set $A$ such that $\text{P}^{\text{NP}^A} \not\subseteq \text{P}_{\text{it}}^{\text{NP}^A}$.

4. The Tree Method

We now proceed by explaining the tree method that will help us obtain an injury-free version of the construction in Section 3. The main reason we needed injuries as we argued in Section 3 was the following.

On an individual path on input $x$, a nondeterministic exponential time machine may query only exponentially many strings. However on the combination of all possible paths, even a linear exponentially bounded machine may have $2^{|x|}$ distinct queries. As the coding region for $x$ (i.e., the region of strings in which some string in the oracle codes whether $x$ is in the test language or not) is limited to strings of length $n^k$ for some fixed $k$, indeed all $2^{n^k}$ strings in this region may appear on some path. Therefore, whatever coding strategy we decide on, the nondeterministic machine may turn out to have an accepting path if and only if the encoding says that $x$ is not in the test language.

The workaround here, borrowed from recursion theory, is again that:

1. there are not too many strings of length $|x|$.
2. an accepting computation on input $x$ can be “frozen” using not too many strings.
3. if bounded search shows that none of our current candidate oracles can force an accepting computation, then we can safely code in acceptance/rejection of $x$. 

The key of the workaround is item 3. Each time in the construction we ask whether strings can be added to the oracle—*in a certain region*—to force a new accepting computation. If so, we force and fix the accepting computation and move on to the next region. If not, then all future strings added to the oracle will be this region. Hence rejection is also preserved. We make sure that there are more regions than there are computations to be preserved at a given stage, so that we always have room enough to encode.

The regions are defined as follows. Given a string $w$ denote by $w \Gamma^*$ the set of all strings in $\Gamma^*$ that extend $w$. In some papers sets like this are also called *cylinders*. Now a sequence of cylinders can be given as $w_1 \Gamma^* \supset w_2 \Gamma^* \supset \ldots$, where $w_i$ has $w_{i-1}$ as a prefix. The $w_i$ can also be thought of as nodes on a finitely branching tree. A region will be a set of the form $\langle \Gamma^*, w \Gamma^*, \Gamma^*, i \rangle$ for some $w \in \Gamma^*$. The entire oracle will be constrained to be a subset of $\langle \Gamma^*, \Gamma^*, \Gamma^*, i \rangle$.

For given $x$ of length $n > 0$ there will be strings in the oracle of the form $\langle x, w_{2n}, y, b \rangle$ where

$$w_{2n} = m_1 m_2 \cdots m_{2n}$$

is such that $|m_2i| = |m_{2i+1}| = i + 1$ for all $1 \leq i \leq n$, $|y| = 2n + 2$, and $b \in \Gamma$. Let us denote the set of all such strings by $C_x$. The $b$ components of these strings encode whether $x \in K^A$. Each $m_j$ will be identified with a number in the range $0$ to $2^{3j/2 + 1} - 1$ in the usual way.

The test language $L^B$ for oracle $B$ is defined as follows: $x \in L^B$ if and only if the lexicographically maximum element in $C_x \cap B$ (if it exists) is of the form $\langle x, w, y, 1 \rangle$. Here, our lexicographical ordering on the elements $\langle x, w, y, b \rangle$ of $C_x$ is: first lexicographically order by $w$, then by $y$, then by $b$. Evidently, $L^B \in \text{P}^\text{NP}_B$ for all oracles $B$. We will construct a recursive oracle $A$ such that $L^A = K^A$.

Our construction will build the oracle by adding strings that belong to regions as defined above. Contrary to the construction in Section 3, the construction here will preserve as well as just avoid accepting computations. Therefore not only will strings be put *into* the oracle during the construction, but strings will be reserved for the complement of the oracle as well. For this purpose we maintain a special set, the restraint set. For each $s \geq 1$, we will have $A_s$ be the set of strings put into the oracle before stage $s$. We will also have a restraint set $R_s$ which contains all the strings that we have *not* committed to the complement of the oracle before stage $s$. We will always have $A_s \subseteq R_s$, as well as $A_s \subseteq A_{s+1}$ and $R_s \subseteq R_{s+1}$. Thus the oracle will be sandwiched between $A_s$ and $R_s$. Within each stage $s > 1$, we will also use local variables $\tilde{A}$ and $\tilde{R}$ to interpolate between $A_s$ and $A_{s+1}$ and between $R_s$ and $R_{s+1}$ respectively. At
any point in the construction, we will say that an oracle $B$ is a candidate iff $\widetilde{A} \subseteq B \subseteq \widetilde{R}$.

One final note before we start: we will sometimes say, “Let $B$ be least such that . . . .” or “look for the minimum $B$ such that . . . .” or some such, where $B \subseteq \Gamma^*$. In such cases, we assume the usual lexicographical ordering on subsets of $\Gamma^*$ by their infinite characteristic sequences. Each condition on $B$ will depend on only a finite set of bits of $B$’s characteristic sequence, so a minimum $B$ will always exist if the condition can be satisfied at all.

Now we are ready to present the construction. Stages are numbered using $n$ for $n = 1, 2, \ldots$

Construction

**Stage 1**: $m_1 = 0 \in \Gamma^2$; $w_1 = m_1$; $A_1 = \emptyset$; $R_1 = (\Gamma^*, w_1 \Gamma^*, \Gamma^*, \Gamma)$;

**Stage $2n$ ($n \geq 1$)**:

**Phase 1** Forcing accepting computations

$m = 0 \in \Gamma^{n+1}$; $S = \Gamma^n$; $\tilde{A} = A_{2n-1}$; $\tilde{R} = R_{2n-1}$;

**Repeat**

$w = w_{2n-1}$;

$\tilde{R}_w = \tilde{A} \cup (\tilde{R} \cap (\Gamma^*, w \Gamma^*, \Gamma^*, \Gamma))$;

$C = \{ x \in S : (\exists B)[\tilde{A} \subseteq B \subseteq \tilde{R}_w \land x \in K^B] \}$

**If** $C \neq \emptyset$ **Then**

$x = \min C$;

$B = \min \{ B' : \tilde{A} \subseteq B' \subseteq \tilde{R}_w \land x \in K^{B'} \}$;

$\tilde{A} = \tilde{A} \cup (Q(M^B_K(x)) \cap B)$;

$\tilde{R} = \tilde{R} - (Q(M^B_K(x)) - B)$;

$S = S - \{ x \}$;

$m = m + 1$;

**Endif**

**Until** $C = \emptyset$

$m_{2n} = m$; $w_{2n} = w$; $\tilde{R} = \tilde{R}_w$

**Phase 2** Coding

**For each** $x \in \Gamma^n$ **Do**

/* Invariant: $\tilde{A} \cap \langle x, w_{2n}, \Gamma^* \rangle = \emptyset$ */

$y = \min \{ y' \in \Gamma^{2n+2} : \langle x, w_{2n}, y', \Gamma \rangle \subseteq \tilde{R} \}$;

**If** $x \in K^A$ **Then** $b = 1$ **Else** $b = 0$;

$\tilde{A} = \tilde{A} \cup \{ \langle x, w_{2n}, y, b \rangle \}$ /* $x$ is encoded by $\langle x, w_{2n}, y, b \rangle$ */

**Endfor**
Phase 3 Cleanup

\[ A_{2n} = \tilde{A}; \quad R_{2n} = (\tilde{R} - \langle \Gamma^n, \Gamma^s, \Gamma \rangle) \cup \tilde{A} \]

End Stage 2n

Stage 2n + 1:

\[ m_{2n+1} = 0 \in \Gamma^{n+2}; \quad w_{2n+1} = w_{2n}m_{2n+1}; \]

\[ A_{2n+1} = A_{2n}; \quad R_{2n+1} = R_{2n} \cap (\langle \Gamma^s, w_{2n+1} \Gamma^s, \Gamma \rangle \cup A_{2n}) \]

End Stage 2n + 1

End of Construction

Lemma 14. For all s > 0, we have \( A_s \subseteq A_{s+1} \), \( R_s \supseteq R_{s+1} \), and \( A_s \subseteq R_s \).

Proof. Straightforward induction on s. Note that just before \( \tilde{R} \) is changed at the end of the Forcing phase of stage \( s = 2n \), because \( \tilde{A} \subseteq \tilde{R} \), we have \( \tilde{R}_w = \tilde{A} \cup (\tilde{R} \cap \langle \Gamma^s, w_{2n} \Gamma^s, \Gamma \rangle) = \tilde{R} \cap (\tilde{A} \cup \langle \Gamma^s, w_{2n} \Gamma^s, \Gamma \rangle) \subseteq \tilde{R} \).

Lemma 15. For each \( n \geq 1 \) and \( x \in \Gamma^n \), a value for \( y \) is found in the Coding phase of Stage 2n.

Proof. It suffices to show that at the start of the Coding phase of Stage 2n, \( \| \langle x, w_{2n}, \Gamma^{2n+2}, \Gamma \rangle - \tilde{R} \| < 2^{n+2} \). It is clear by the construction that for any \( z \in \langle x, w_{2n}, \Gamma^{2n+2}, \Gamma \rangle \), if \( z \) was ever removed from \( \tilde{R} \) before the start of the Coding phase of Stage 2n, then this must have happened in the Repeat loop of the Forcing phase of some stage \( s \leq 2n \). This happens for at most \( 2^n \) many \( z \) (queries answered “no” along a single path of \( M_R(x') \)) for each \( x' \) with \( |x'| \leq n \), and thus \( \| \langle x, w_{2n}, \Gamma^{2n+2}, \Gamma \rangle - \tilde{R} \| \leq 2^n (2^{n+1} - 1) < 2^{n+1} \).

We define \( A = \bigcup_{s>0} A_s \).

Lemma 16. For each \( n \) and for all \( x \) of length \( n \), \( A \) contains exactly one string of the form \( \langle x, w_{2n}, y, b \rangle \) for \( y \in \Gamma^{2n+2} \) and \( b \in \Gamma \), which is the lexicographically maximum element of \( C_x \cap A \).

Proof. By Lemmas 14 and 15, \( A \) contains at least one such string.

Let \( \tilde{A}, m_0, \ldots, m_{2n}, y \) and \( b \) be as in the Coding phase of stage \( 2n \) just before \( x \) is encoded. Consider any string \( z \in C_x \) where \( z \neq \langle x, w_{2n}, y, b \rangle \). The string \( z \) can only enter \( A \) either (i) during the Coding phase of some stage other than \( 2n \), (ii) during the Forcing phase of some stage \( 2n' > 2n \), or (iii) during the Forcing phase of some stage \( 2n' \leq 2n \). Case (i) can never happen, because any strings entering then would be of the form \( \langle x', u', y', b' \rangle \) with \( x' \neq x \). Case (ii) cannot happen by Lemma 14 and the fact that \( z \) is removed from \( R_{2n} \) during the Cleanup phase of Stage 2n (if it wasn’t already removed earlier).
In case (iii), \( z \) must be a query of the form \( \langle x, m_1 \cdots m_{2n-1}mu, y', b' \rangle \) (for some \( m, u, y' \) and \( b' \)) made on some accepting path of \( M_K(x') \) for some \( x' \). This is because when \( z \) enters \( A \), \( z \in Q(M^B_K(x')) \cap B \subseteq \tilde{R}_w \), where \( w = m_1 \cdots m_{2n-1}m \). Since \( m \) is subsequently incremented, we have \( m < m_{2n} \), and so \( m_1 \cdots m_{2n-1}mu \) is lexicographically strictly less than \( m_1 \cdots m_{2n} = w_{2n} \). Thus, \( z \) is also lexicographically strictly less than \( \langle x, w_{2n}, y, b \rangle \). This proves the lemma. \( \square \)

**Lemma 17.** \( L^A = K^A \).

**Proof.** Fix an \( n > 0 \) and an \( x \in \Gamma^n \). Let \( \tilde{A}, \tilde{R}, C \), and \( S \) be as in the Coding phase of Stage \( 2n \) just before \( x \) is encoded, and let \( \tilde{A}_0 \) be the value of \( \tilde{A} \) at the start of the same Coding phase. By Lemma 16, it suffices to show that \( x \in K^A \iff x \in K^A \), that is, \( x \) stays correctly encoded in the oracle. Since \( C = \emptyset \), we have

\[
x \in S \implies (\forall B)[\tilde{A}_0 \subseteq B \subseteq \tilde{R} \implies x \not\in K^B] .
\]

Suppose \( x \in S \). Then, because \( \tilde{A}_0 \subseteq \tilde{A} \subseteq A \subseteq \tilde{R} \) by Lemma 14, we have \( x \not\in K^A \) and \( x \not\in K^A \).

Now suppose \( x \not\in S \). Then \( x \) was removed from \( S \) in some iteration of the Repeat loop, at which time an accepting path of \( M_K(x) \) was explicitly preserved, ensuring that \( (\forall B)[\tilde{A}_0 \subseteq B \subseteq \tilde{R} \implies x \in K^B] \).

So in this case we have \( x \in K^A \) and \( x \in K^A \). \( \square \)

**Lemma 18.** \( A \) is recursive.

**Proof.** All searches done in the construction are recursively bounded, including the calculation of \( B \) in the Repeat loop. Further, every string \( z = \langle x, w, y, b \rangle \) with \(|x| = n\) is committed by the end of Stage \( 2n \) by being out of \( R_{2n} - A_{2n} \). \( \square \)

### 4.1. Getting More Mileage

The astute reader may have noticed that in the construction in Section 4 we really used only the even stages to construct the oracle. Now the time has come to use the odd stages in that construction to get an extra interesting property of the oracle. We will use the odd stages of the construction to force that

\[
P^A = UP^A = NP^A \cap \text{coNP}^A
\]

in addition to \( P^{NP^A} = \text{NEXP}^A \).
Actually, this is not quite correct. We can prove that the oracle $A$ that
we construct has the property ((4.1)) above only under the assumption that
$P = \text{PSPACE}$ (unrelativized). That is, we will instead show the following:

\[(4.2) \quad P = \text{PSPACE} \implies (P^A = \text{UP}^A = \text{NP}^A \cap \text{coNP}^A \land P^{\text{NP}^A} = \text{NEXP}^A),\]

where $A$ is the oracle we are constructing. We can then re-relativize to obtain
an oracle $A'$ such that $P^{A'} = \text{UP}^{A'} = \text{NP}^{A'} \cap \text{coNP}^{A'} \land P^{\text{NP}^{A'}} = \text{NEXP}^{A'}$ with
no assumptions.

Re-relativization is a technique that has been used before in a number of
places (see (FFKL93), for example), but it deserves a brief explanation here. We
can cast our entire construction relative to any oracle $X$ by letting all the
machines involved have access to oracle $X$. By doing so, we build an oracle $A_X$
such that

\[(4.3) \quad P^X = \text{PSPACE}^X \implies (P^X)^{A_X} = \text{UP}^X)^{A_X} = (\text{NP}^X)^{A_X} \cap (\text{coNP}^X)^{A_X}
\land (P^{\text{NP}^X})^{A_X} = (\text{NEXP}^X)^{A_X}\]

because our proof of ((4.2)) below can clearly be relativized to $X$. In ((4.3))
above, each class of the form $(C^X)^{A_X}$ is just equal to $C^{X \oplus A_X}$ by combining
the two oracles into one. Now let $H$ be an oracle such that $P^H = \text{PSPACE}^H$
(a PSPACE-competitive set, for instance). Setting $X = H$ in ((4.3)) gives

\[P^{A'} = \text{UP}^{A'} = \text{NP}^{A'} \cap \text{coNP}^{A'} \land P^{\text{NP}^{A'}} = \text{NEXP}^{A'},\]

where $A' = H \oplus A_H$. The $P = \text{PSPACE}$ assumption has been effectively
discharged by first relativizing to $H$, then to $A_H$. It therefore suffices for our
construction to ensure ((4.2)) above, via a proof that relativizes. So from now
on, we will freely assume that $P = \text{PSPACE}$ unrelativized.

For simplicity of presentation, the following construction only ensures that
$P^{\text{NP}^A} = \text{NEXP}^A$ and $P^A = \text{UP}^A$. Getting $P^A = \text{NP}^A \cap \text{coNP}^A$ uses similar
techniques and can easily be interleaved into the construction. As the even
stages are identical to the construction in Section 4, we only present the odd
stages here.

For $i = \langle j, k \rangle = 1, 2, 3, \ldots$, let $N_i$ be the machine $M_j$ equipped with a clock
so that it runs in time $n^i$. 
Stage \( 2n + 1 \) \((n \leq 1)\):

\[ m = 0 \in \Gamma^{n+2}; \ S = \{1, \ldots, n\}; \ \hat{A} = A_{2n}; \ \hat{R} = R_{2n}; \]

Repeat

\[ w = w_{2n}m; \]
\[ \hat{R}_w = \hat{A} \cup (\hat{R} \cap (\Gamma^*, w\Gamma^*, \Gamma^*, \Gamma)); \]
\[ C = \{ i \in S : (\exists B)[\hat{A} \subseteq B \subseteq \hat{R}_w \land (\exists y)[|y| \leq 2^{n/i} \land N^B_i has at least two accepting paths on input y]] \}

If \( C \neq \emptyset \) Then

\[ i = \min C; \]
\[ B = \min \{ B' : \hat{A} \subseteq B' \subseteq \hat{R}_w \land (\exists y)[|y| \leq 2^{n/i} \land N^B_i has at least two accepting paths on input y] \}

Pick the least such \( y \) and the least two distinct accepting paths \( p \) and \( q \) of \( N^B_i \) on input \( y \);

Let \( Q \) be the set of all queries made along paths \( p \) and \( q \);
\[ \hat{A} = \hat{A} \cup (Q \cap B); \]
\[ \hat{R} = \hat{R} - (Q - B); \]

/* \( \hat{N}_i \) is no longer a \( \text{UP}^B \) machine for any candidate \( B' */

\[ S = S - \{ i \}; \]
\[ m = m + 1; \]

Endif

Until \( C = \emptyset; \)
\[ m_{2n+1} = m; \ w_{2n+1} = w; \ A_{2n+1} = \hat{A}; \ R_{2n+1} = \hat{R}_w \]

End of Stage \( 2n + 1 \)

The oracle \( A \) is defined as before, and it is clearly recursive. The proof that \( \text{P}^{N_A} = \text{NEXP}^A \) requires little change from the previous proof: Lemma 14 is still clearly true. Lemma 17 is true by exactly the same proof. Lemma 15 is still true, but we must revise our upper bound on \( ||(x, w_{2n}, \Gamma^{n+2}, \Gamma) - \hat{R}|| \) to account for strings being removed from \( \hat{R} \) during odd stages. Each

odd stage \( 2n' + 1 < 2n \) removes from \( \hat{R} \)—at most—queries made along two paths of \( N_i(y_i) \) for each \( i \in \{1, \ldots, n'\} \),

where \( y_i \) is a string with length \( \leq 2^{n'/i} \). The number of queries on each path is at most \( (2^{n'/i})^i \leq 2^n \). So the

total number of strings removed from \( \hat{R} \) in all the odd stages \( 2 < s < 2n \) is at most \( n^2 2^n \leq 2^{2n+1} \). Adding to this the number of strings counted in the even stages gives a total strictly less than \( 2^{2n+1} + 2^{2n+1} = 2^{2n+2} \) as desired. Lemma 16 is true when the proof is modified to account for strings \( z \) entering \( A \) at odd stages. This case is entirely similar to case (iii) in that proof, since the odd stages closely resemble the Forcing phases of the even stages.
Note that the $P = \text{PSPACE}$ assumption was not needed in any of the considerations above. So in fact, $P^{NP^A} = \text{NEXP}^A$, and this also clearly holds when we relativize the construction to any prior oracle $X$.

It remains to show that $P^A = \text{UP}^A$. Fix an $i$ such that $N_i$ is a $\text{UP}^A$ machine. We will describe a $P^A$ algorithm to compute $L(N_i^A)$. Fix an input $y$ and let $n = i|\log|y||$. We first run the construction through stage $2n + 1$. After a little thought, it can be seen that $w_{2n+1}$ and $A_{2n+1}$ (as a list of strings) can both be computed in space linearly exponential in $n$ and thus polynomial in $|y| \leq 2^{n/i}$. In particular, the searched paths of the machines $M_K$ and $N_i$ are each of length $\leq 2^n$ and only query strings of length $\leq 2^n$. In simulating the construction, besides maintaining a list of the strings in $\hat{A}$, we must also maintain a $2^{O(n)}$-size list $\ell_{\widehat{R}}$ of the strings removed from $\widehat{R}$ during odd stages and in the Repeat loops of even stages. Given these lists and the $w_s$, membership of any string in $\widehat{R}$ can easily be computed at any time “on the fly” as needed. Define

$$c(y) = \langle w_{2n+1}, A_{2n+1}, \ell_{\widehat{R}_{2n+1}} \rangle$$

as above. The function $c$ can be computed (without access to an oracle) in polynomial space (in $|y|$), and its output is polynomial size. It follows from the $P = \text{PSPACE}$ assumption that $c$ can be computed in polynomial time.

By the construction and the fact that $N_i^A$ is a $\text{UP}^A$ machine, $N_i^B(y)$ has at most one accepting path on input $y$ for every $B$ such that $A_{2n+1} \subseteq B \subseteq R_{2n+1}$. We can now recover the accepting path of $N_i^A(y)$, if there is one, through a deterministic polynomial time bounded procedure using oracle $A$, as we show next. We adapt a standard technique developed and used by Rackoff (Rac82), Tardos (G.T89), Hartmanis and Hemachandra (HH91) and Blum and Impagliazzo (BI87) to compute $N_i^A(y)$. Our Algorithm uses a function $d$ as a subroutine, where

$$d(y, w_{2n+1}, \hat{A}, \ell_{\hat{R}}) = \begin{cases} Q(N_i^B(y)) & \text{if } (\exists \text{ a least candidate } B)[N_i^B(y) = 1], \\ "\text{no}" & \text{otherwise}. \end{cases}$$

where $\hat{A}$ and $\ell_{\hat{R}}$ are lists of strings, and $\hat{R}$ is defined from $\ell_{\hat{R}}$ and $w_{2n+1}$ by the construction through stage $2n + 1$ as described above. Clearly, $d$ can be computed in polynomial space without access to an oracle, and its output is polynomial size. Since $P = \text{PSPACE}$, $d$ is computable in polynomial time.
ALGORITHM FOR $N_i^A(y)$
Compute $\tilde{A} = A_{2n+1}, \ell_{\tilde{R}} = \ell_{R_{2n+1}}$ (which defines $\tilde{R}$),
and $w = u_{2n+1}$ as above via $c(y)$;

$Q = \emptyset$;
Repeat $|y|^i$ times
\[ \text{If } d(y, w, \tilde{A}, \ell_{\tilde{R}}) \neq \text{"no" then} \]
\[ Q = Q \cup d(y, w, \tilde{A}, \ell_{\tilde{R}}); \]
\[ \tilde{A} = \tilde{A} \cup (Q \cap A); \]
\[ \tilde{R} = \tilde{R} - (Q - A); \]
Endif
End Repeat
Accept iff $N_i^A(y) = 1$ and $Q(N_i^A(y)) \subseteq Q$;
END

The Algorithm can be run in time polynomial in $|y|$ relative to oracle $A$: $w, \tilde{A}, \ell_{\tilde{R}},$ and $Q$ are all of size polynomial in $|y|$, and we only query $A$ on strings in $Q$, which are all of size at most $|y|^i$. (We update $\ell_{\tilde{R}}$ to account for the strings removed from $\tilde{R}$ as before.) The final evaluation of $N_i^A(y)$ and $Q(N_i^A(y))$ can be done in polynomial space without oracle access, and so can be done in polynomial time.

We define any oracle to be a candidate just as before. Note that $A$ is a candidate throughout the Algorithm, and $A$ never loses elements nor does $\tilde{R}$ gain any. If the Algorithm accepts, then after the For loop there is an accepting path of $N_i^A(y)$ which only makes queries in $Q$. But this implies $N_i^A(y) = 1$ also, because $A, \tilde{R},$ and hence $\tilde{A}$ all agree within $Q$.

Conversely, suppose $N_i^A(y) = 1$, and let $Q_0 = Q(N_i^A(y))$. Clearly, $d$ never returns “no.” We claim that $Q_0 \subseteq Q$ after the loop finishes. It follows immediately that $\tilde{A} \cap Q_0 = \tilde{R} \cap Q_0 = A \cap Q_0$, and so $N_i^A(y) = 1$ and the Algorithm accepts.

We prove the claim: Let $m = |y|^i$. For $j = 1, 2, \ldots, m$, let $B_j$ be the set found by $d$ and let $Q_j$ be the query set returned by $d$ on the $j$th iteration of the loop. Suppose first that there is a $j$ such that $B_j$ and $A$ agree within $Q_0 \cap Q_j$, that is, $B_j \cap Q_0 \cap Q_j = A \cap Q_0 \cap Q_j$. Then we must have $Q_0 = Q_j$: If not, then let

$C = \tilde{A} \cup (B_j \cap Q_j) \cup (A \cap Q_0)$.

Clearly, $C$ is a candidate, but $N_i^C(y)$ has two distinct accepting paths, one making queries in $Q_j$ and the other making queries in $Q_0$. This violates the condition enforced by the end of Stage $2n + 1$, and so the claim holds.

Now suppose that $B_j \cap Q_0 \cap Q_j \neq A \cap Q_0 \cap Q_j$ for all $j$, and let $z_j$ be the
minimum string in \((B_j \triangle A) \cap Q_0 \cap Q_j\). Then since \(A(z_j) \neq B_j(z_j)\) we know that \(z \notin \bigcup_{k=1}^{j-1} Q_k\), because \(B_j\) and \(A\) agree within \(\bigcup_{k=1}^{j-1} Q_k\). This means that

\[
\left( Q_0 - \bigcup_{k=1}^{j} Q_k \right) \supseteq \left( Q_0 - \bigcup_{k=1}^{j-1} Q_k \right).
\]

That is, \(Q_0 - Q\) loses at least one element \((z_j)\) on each iteration of the loop. Before the loop starts, we have \(||Q_0 - Q|| = ||Q_0|| \leq |y|^i\), and so after the loop finishes, we have \(Q_0 - Q = \emptyset\). This proves the claim and the correctness of the Algorithm.

To get \(P^A = \text{NP}^A \cap \text{coNP}^A\), we can easily interleave \(P^A = \text{NP}^A \cap \text{coNP}^A\) stages (of the form \(4n + 1\)) with the \(P^A = \text{UP}^A\) stages (now of the form \(4n - 1\)). For the former stages, which are handled similarly to the latter ones, we consider each \(N_i^X\) as a proper \(\text{NP}^X \cap \text{coNP}^X\) machine if, for each input, it has at least one accepting path, and either all accepting paths start with a left branch or they all start with a right branch. \(N_i^X\) accepts an input if and only if it has an accepting path starting with a left branch. In the \(P^A = \text{NP}^A \cap \text{coNP}^A\) stages we look for a \(B\) with \(A \subseteq B \subseteq R\) that forces both types of accepting path for the same input \(y\). The corresponding \(P^A\) Algorithm is essentially unchanged, except that \(d\) now searches for two candidates \(B\) and \(B'\) such that \(N_i^B(y)\) and \(N_i^{B'}(y)\) each have an accepting path and the two paths start on opposite branches. If successful, \(d\) returns the union of the respective query sets. The proof of efficiency and correctness proceeds largely as before. If \(d\) ever returns “no,” then we search for any single accepting path of \(N_i^{B}(y)\) (for any candidate \(B\)). The first branch of the path we find must agree with that of any accepting path of \(N_i^A(y)\). The construction promises that, for all \(C\) with \(A_{4n+1} \subseteq C \subseteq R_{4n+1}\), any two accepting paths of \(N_i^C(y)\) start with the same branch. So if \(d\) never returns “no,” then at least one of the queries in the set returned by \(d\) must also be in \(Q_0\) and have been answered differently from \(A\). So \(Q_0 - Q\) will shrink as before. We thus have the following theorem:

**Theorem 19.** There is a recursive oracle \(A\) such that \(P^{\text{NP}^A} = \text{NEXP}^A\) and \(P^A = \text{UP}^A = \text{NP}^A \cap \text{coNP}^A\).

### 4.2. Lowness and Collapsing Degrees.

For any relativizable class \(C\) we let \(\text{LOW}(C)\) denote the class of all \(C\)-low sets, i.e., the class of all \(B\) such that \(C^B = C\). This concept itself relativizes, i.e., we say that a set \(B\) is low for \(C\) relative to an oracle \(A\) if \(C^A \triangleq B = C^A\).

Since the classes \(\text{NP}\) and \(P^{\text{NP}}\) are “close” to each other, one might expect that \(\text{LOW}(\text{NP})\) should likewise be close to \(\text{LOW}(P^{\text{NP}})\), but instead we get the
following curious lowness property relative to oracle $A$ of Theorem 19.

**Corollary 20.** Relative to the set $A$ of Theorem 19, $\text{LOW}(\text{NP}) = P$ and $\text{LOW}(\text{P}^\text{NP}) = \text{NEXP}$.

*Proof.* It is well-known (see (Sel79)) that $\text{LOW}(\text{NP}) = \text{NP} \cap \text{coNP}$, and the proof relativizes; thus $\text{LOW}(\text{NP}) = P$ relative to $A$. Moreover, relative to $A$ we have

$$(\text{P}^\text{NP})^\text{NEXP} = (\text{P}^\text{NP})^{\text{NP}} \subseteq \text{PH} \subseteq \text{NEXP} \subseteq \text{P}^\text{NP}$$

and thus $\text{NEXP} \subseteq \text{PH}$. $\text{LOW}(\text{P}^\text{NP}) \subseteq \text{P}^\text{NP} = \text{NEXP}$.

Homer and Selman (HS92) constructed an oracle relative to which $P = \text{UP}$ and $\Sigma_2^P = \text{EXP}$, and thus the complete $\leq^m_P$-degree for $\Sigma_2^P = \text{EXP}$ collapses. The oracle $A$ of Theorem 19 brings the collapse further down in the polynomial hierarchy.

**Corollary 21.** Relative to the set $A$ of Theorem 19, all $\leq^m_P$-complete sets for $\text{P}^\text{NP}$ are polynomial time isomorphic.

*Proof.* Berman (Ber77) showed via a relativizable proof that the complete $\leq^m_P$-degree for EXP collapses to a 1-li-degree (one-to-one length-increasing $m$-reductions). By results in (GS88) and (KLD87), all 1-li-degrees collapse if and only if $P = \text{UP}$, again by a relativizable proof. Relative to $A$ we have $P = \text{UP}$ and $\text{P}^\text{NP} = \text{EXP}$.

5. Conclusions

Having come to this point, the reader may wonder why in this paper we presented two constructions of essentially the same oracle. The reasons for this are historical, aesthetic and technical. Historically, the injury construction was the first to be presented at a conference (BT94). As often before, the construction was soon made recursive by force of the tree method we presented in Section 4 (FF95). Both methods have their own intrinsic beauty and therefore both deserve their place in the literature. In other constructions both methods may prove their separate use. Technically, it seemed that there were very good arguments here why indeed an injury method is necessary for this construction as we explained before. However, again it turned out that injury arguments could be replaced by bounded search. There is an old thesis that any oracle construction in complexity theory can be made recursive simply because of the fact that the construction deals with resource bounded machines. The present
constructions failed to provide a counter example—or, equivalently, provided more support for that thesis.

Of course, we would like to see the $P^{NP}$ versus $NEXP$ question answered in the unrelativized world. We conjecture that the classes in fact are different, but such a proof will require vastly new techniques.

In the relativized setting, we would like to see an oracle relative to which $P^{NP} = NEXP$ but $P = BPP$. We think a proof similar to the proof of Theorem 19 might work, but trying to force all BPP machines to be categorical may put too many strings in the oracle.

References


Manuscript received ???

HARRY BUHRMAN
CWI
Kruislaan 413
1098 SJ Amsterdam
the Netherlands
buhrman@cwi.nl

STEPHEN FENNER
University of South Carolina
Department of Computer Science
Columbia, SC 29208
fenner@cse.sc.edu

LANCE FORTNOW
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
fortnow@research.nj.nec.com

LEEN TORENVLIET
ILLC
University of Amsterdam
24 Plantage Muidergracht
1018 TV Amsterdam
the Netherlands
leen@science.uva.nl