Sparse Selfreducible Sets and Nonuniform Lower Bounds

Harry Buhrman[1,2], Leen Torenvliet[2], Falk Unger[1], Nikolay Vereshchagin[3]

[1] CWI Amsterdam, [2] Universiteit van Amsterdam,

[3] Moscow State University, National Research University Higher School of Ecomomics

April 25, 2017

Abstract

It is well-known that the class of sets that can be computed by polynomial size circuits is equal to the class of sets that are polynomial time reducible to a sparse set. It is widely believed, but unfortunately up to now unproven, that there are sets in $\mathrm{EXP}^{\mathrm{NP}}$, or even in EXP that are not computable by polynomial size circuits and hence are not reducible to a sparse set. In this paper we study this question in a more restricted setting: what is the computational complexity of sparse sets that are *selfreducible*? It follows from earlier work of Lozano and Toran [LT91] that $\mathrm{EXP}^{\mathrm{NP}}$ does not have sparse selfreducible hard sets. We define a natural version of selfreduction, tree-selfreducibility, and show that NEXP does not have sparse tree-selfreducible hard sets. We also construct an oracle relative to which all of EXP is reducible to a sparse tree-selfreducible set. These lower bounds are corollaries of more general results about the computational complexity of sparse sets that are selfreducible, and can be interpreted as super-polynomial circuit lower bounds for NEXP.

Keywords Computational Complexity, Sparseness, Selfreducibility

1 Introduction

Finding techniques to separate complexity classes is one of the, if not *the*, main open problem in complexity theory. Our understanding towards solving problems like the P versus NP problem is very limited. Not only do we not know how to separate P from NP, we don't even know how to separate $\mathrm{EXP}^{\mathrm{NP}}$ from the class of sets that have polynomial size circuits. [1] Work on derandomization assumes much stronger separations than this, like for example that EXP requires exponential size circuits.

It is long known that the class of sets that have polynomial size circuits equals the class of sets that are polynomial time Turing reducible to a sparse set [Mey77]. In this paper we address the question of whether $\mathrm{EXP}^{\mathrm{NP}}$ and smaller classes are Turing reducible to a sparse set by restricting the sparse set to be selfreducible and even *tree selfreducible* (see Section 4). A set S is selfreducible if there exists a polynomial time machine that can decide membership of x in S by making queries to S that are smaller than x in some well-defined way (see Definition 2.2). A selfreduction is a tree selfreduction if the pattern of queries generated by expanding all selfreductions is a tree. For example, the well-known selfreduction of satisfiability (explained again in more detail below) where queries are obtained by replacing variables by constants until all variables have been replaced and thus a leaf in the reduction is reached. Is a 2-disjunct tree selfreduction.

We do not know of any examples of selfreductions that are not essentially tree-selfreductions.

It follows from work of Lozano and Toran [LT91] that there are no sparse selfreducible sets that are hard for $\mathrm{EXP}^{\mathrm{NP}}$. We extend this result by showing that NEXP does not have sparse hard sets that are tree-selfreducible. This

[1]Formally, the reader might notice, these questions are independent. They are related however as follows. If EXP^{NP} or even NEXP has polynomial size circuits then $\mathrm{P} \neq \mathrm{NP}$ follows. Therefore, it seems that it should be easier to settle the former question, in the negative, than it does to settle the latter

result is optimal [2] with respect to relativizing proof techniques, since we also obtain a relativized world where EXP has a sparse tree-selfreducible hard set. These results can be interpreted as super-polynomial lower bounds for NEXP with respect to a restricted class of circuits.

These lower bounds are consequences of more general results on the complexity of sparse selfreducible sets. Lozano and Toran showed that sparse selfreducible sets are in P^{NP}, we give a different proof of this result that allows us to generalize it to sets of smaller density. We also show a relativized world in which this result is optimal, i.e., we construct an oracle relative to which there exists a sparse selfreducible set that cannot be recognized by a P oracle machine that has only a linear number of queries to put to its NP oracle (but has unlimited direct access to the oracle constructed). We further show that tree-selfreducible sparse sets are in $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$, the class of languages that can be decided with logarithmically many queries to an NP oracle. It follows from this result, that NEXP does not have sparse tree-selfreducible hard sets. Connecting this with recent results of Fortnow et al. [FK05, SU04] it follows that if EXP has a sparse tree-selfreducible hard set, then it is in NP/log. We next exhibit a relativized world where there exists a sparse 2-parity selfreducible set in $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ that is not in any lower complexity class. This solves an open question from [LT91].

A 2-parity selfreduction is a selfreduction where membership can of a string in the set can be computed by computing the parity of the answers to the two queries generated by the reduction, i.e., the answer is yes if and only if exactly one of the answers to the two queries is yes. Not in any lower complexity class means here that the total number of queries to compute the set by a P^{NP} machine is $\Omega(\log n)$.

We also show a relativized world where there is a sparse Turing selfreducible set that is not truth-table selfreducible, and present some absolute results about the complexity of selfreducible sets that have sub-polynomial densities. Finally, we discuss log-sparse selfreducible sets and show sharp upper bounds on their computational complexity. Bounded truth-table, i.e., the number of queries is bounded by some constant, log-sparse selfreducible sets are even in P. Summarizing our results:

[2] In several places in this paper we use "optimal" where this is not an exact statement. If we prove a problem to be in NEXP and show an oracle relative to which it is not in EXP then it could still be in many intermediate classes, and even a non-relativizing proof might still show it to be in EXP. Though we always make the exact meaning of optimal precise in theorems following the statement, the reader should be cautioned.

- Every sparse set that is tree-selfreducible can be computed in $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ (Theorem 3.3). This allows us to prove that NEXP does not have sparse tree-selfreducible hard sets (Theorem 4.1). On the other hand we show a relativized world where EXP *does* have tree-selfreducible sparse hard sets (Theorem 4.2).
- We construct a relativized world where there exists a sparse (tree) selfreducible set in $\mathrm{P}^{\mathrm{NP}[\log n]}$, that can not be computed with fewer queries to NP (Theorem 3.7). This partially answers an open question from [LT91].
- We construct a relativized world where there is a sparse selfreducible set that cannot be recognized by a P machine that has only a linear number of queries for its NP oracle (Corollary 3.4).
- Every log-sparse selfreducible set is in $\mathrm{P}^{\mathrm{NP}[\mathrm{O}(\log \mathrm{n})^2]}$ (Theorem 5.1), and every log-sparse btt-selfreducible set is in P (Theorem 5.2).

2 Definitions and Notation

We assume the reader to be familiar with standard complexity theory notation, as for example in [Pap94, BDG88]. Let $\Sigma = \{0,1\}$. We write λ for the empty word. For a set $A \subseteq \Sigma^*$, let $A^{=n}$ be the set of strings from A of length n and $A^{\leq n} = \bigcup_{i=0}^{n} A^{=i}$. Note that $\Sigma^n = (\Sigma^*)^{=n}$ by this notation. Pairing functions will be denoted by $\langle .,. \rangle$ and concatenation of strings x and y by xy. Implicitly using a standard mapping between numbers and strings in binary, we will use numbers as arguments to functions where strings are required and vice versa.

Definition 2.1 *A partial order $\prec$ on Σ^* is called* polynomially related *if and only if there exists a k such that for all $x, y \in \Sigma^*$*

1. *$y \prec x \rightarrow |y| \leq |x|^k$*
2. *$x \prec y$ is decidable in time $(|x| + |y|)^k$*
3. *Every descending chain starting with x has length at most $|x|^k$.*

Let $\prec$ be polynomially related. The Directed Acyclic Graph that represents the weak initial segment dominated by x, i.e., the graph with nodes $\{y \mid y \in \Sigma^* \wedge y \prec x\} \cup \{x\}$ and edges between y and y' if $y \prec y'$, is denoted by S_x. In this paper we will happily make use of type-conflicting notations, like $Y \subseteq S_x$ where Y is a set of strings and S_x is the graph just defined. Here we mean that the strings from Y are nodes in S_x. First we define selfreducibility for some ordering $\prec$.

Definition 2.2 *Let r be some reduction type, e.g., btt, tt, T etc. A set $S \subseteq \Sigma^*$ is called* $\leq_r^P$-selfreducible *with respect to the polynomially related ordering $\prec$ on Σ^* if and only if*

1. *$S \leq_r^P S$ and*

2. *For any input $x \in \Sigma^*$ the reduction queries only elements y with $y \prec x$*

An example of a selfreducible set is SAT, the set of satisfiable boolean formulas. There exists a well-known two-query disjunctive selfreduction for SAT, which is even length decreasing, where queries are formed by assigning values to variables.

At this point it may help the intuition to consider the different interpretations of selfreduction. First there is the definition where the selfreduction is performed by an oracle Turing machine that, on the basis of the answers of the oracle to a polynomial number of queries that are smaller in the polynomially related order, decides whether or not the string is in the set. Here we consider only one step of the reduction. In the case of SAT this translates to: A formula $\phi(x_1, \ldots, x_n)$ is satisfiable if and only if $\phi(0, x_2, \ldots, x_n)$ is satisfiable or $\phi(1, x_2, \ldots, x_n)$ is satisfiable. Both $\phi(0, x_2, \ldots, x_n)$ and $\phi(1, x_2, \ldots, x_n)$ are smaller than $\phi(x_1, \ldots, x_n)$ in the order that simply counts the number of free variables. Then, there is the picture where the queries themselves are reinserted to the oracle machine and new queries are produced, until queries can be answered without the intervention of the oracle. This pictures a graph with polynomial length paths. In the case of SAT, this graph is obtained by putting edges between the reduced formulae. All paths end in two nodes, 0 and 1. Any selfreduction can be represented by such a graph, with at most exponentially many nodes, in which paths are at most of polynomial length. Such a graph can be explored in PSPACE and in fact this outlines the proof that any selfreducible set is in PSPACE as can be found in many textbooks. The graph just mentioned can often be transformed into a tree. In fact, we know of no natural selfreducible problem where this cannot be done. In the case of SAT, this can be achieved by *not* reducing intermediate queries. The leaves of the tree then consist of exponentially many formulae without free variables (that can be easily valued true or false). Finally, it is worth noting that the polynomially related ordering of Definition 2.1 is often just the length of the queries, though it is unlikely that this is always the case (see [FO05]).

We will say that a reduction M that witnesses the selfreducibility of S *obeys* $\prec$, if queries by M respect the ordering $\prec$, i.e., y is queried on input x only if $y \prec x$. We will denote the set of strings that is queried by oracle machine M on input x—the *query set* of M on input x—by $Q_M(x)$. If M is a non-adaptive machine then this query set is independent of the oracle. If M is an adaptive machine then this notation is sometimes enriched with the oracle, e.g., $Q_M^A(x)$, or, if the oracle is left out, the set of all *potential* queries is meant by this notation (of exponential size for polynomial time bounded oracle machines, but sometimes even bigger). This notation can also be used to denote an even bigger set. If V is a set of strings, then $Q_M(V) = \bigcup_{v \in V} Q_M(v)$.

We now define a very natural variant of self-reductions, see further below for some comments. Given a string x and a selfreducible set S, we can define a graph on S "around x", consisting of all strings y that are smaller than x in the ordering. We put an edge between two strings y and z if z may be queried by the selfreduction on input y. Note that this may depend on a particular oracle. In the case of SAT this structure is rather simple and straightforward, since the question whether y can be queried on input z is not dependent on the outcome of other queries, but this can be more complicated. Note also that this definition deals with the "one step" interpretation of the selfreduction.

Definition 2.3 *Let S be a self-reducible set, witnessed by the deterministic polynomial time oracle machine M, which obeys the ordering $\prec$. For a string x define the graph G as follows:*

1. *The nodes of G are all strings y with $y \prec x$.*
2. *for $y, z \in G$, there is a edge from y to z if and only if $z \in Q_M^O(y)$ for some oracle O.*

Let S_x^M be the (connected) component of G that contains x. We say that M is a tree-selfreduction *if for all x, S_x^M is always a tree.*

If $L(M^S) = S$ for some tree-selfreduction M, then this S is called tree-selfreducible. *Note $S_x^M \subseteq S_x$ for all x and M.*

Note that S_x^M contains all strings y which could be possibly queried in the self-reduction of x, no matter which (possibly wrong) answers M gets.

Definition 2.4 *Let M be a selfreduction obeying $\prec$ and T some set. Consider a labeling $l : T \mapsto \{0,1\}$ of T. We call l* consistent *with M, or M-consistent, if and only if $(\forall y \in T)[l(y) = 1 \Leftrightarrow M(y)$ accepts when queries of $M(y)$ in T are answered according to l and queries outside T are answered NO$]$.*

Definition 2.5 *Let $\prec$ be a polynomially related order and let $T_x \subseteq S_x$ be a tree that has root x. For $y \in T_x$ we define the* depth *of y as $d_x(y) = \#$nodes on the path from x to y in T_x. Consider a labeling $l : T_x \mapsto \{0,1\}$. Let T_D be the set of nodes from $l^{-1}(1)$ that are minimal w.r.t. $\prec$, i.e., $T_D = \{y \in l^{-1}(1) |$ there is no $z \in l^{-1}(1) - \{y\}$ such that y is on the path from z to the root $x\}$. We define the* weight *of T_x as $weight(T_x) = \sum_{y \in T_D} d_x(y)$.*

Note that for each set T there is a unique M-consistent labeling for T, which can be easily found in a bottom-up fashion, i.e., starting from the leaves and working towards the root.

We want to mention that all selfreducible sets we know of can be made (or even are) tree-selfreducible. Take SAT as a simple example. Consider the standard reduction which on input $\phi(x_1, \ldots, x_n)$ with $n > 0$ queries $\phi(0, x_2, \ldots, x_n)$ and $\phi(1, x_2, \ldots, x_n)$, but does not simplify the terms, and accepts iff one of the queries is true. For $n = 0$ it outputs the truth value of ϕ. Then this reduction is obviously a tree-selfreduction.

We call a set $S \subseteq \Sigma^*$ *sparse* if and only if $\|S^{\leq n}\| \in O(Pol(n))$. S is called *log-sparse* if and only if $\|S^{\leq n}\| \in O(\log(n))$.

It is well-known that the class $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ (P-machines that can make $O(\log n)$ adaptive queries to an NP-oracle) is equivalent to the class $\mathrm{P}^{\mathrm{NP}_\|}$

(P-machines that can only make non-adaptive queries to an NP-oracle), see [Wag88]. This class is commonly referred to as Θ_2^{P}. The class P^{NP} is commonly referred to as Δ_2^{P}.

3 Sparse selfreducible sets

3.1 Upper Bounds

We start by citing a result from [LT91].

Theorem 3.1 *If $S \subseteq \Sigma^*$ is sparse and selfreducible then S is in P^{NP}.*

We will later state a theorem (Theorem 5.1), whose proof can be easily adapted to yield the same result, thus giving an alternative proof for Theorem 3.1.

An open question from [LT91] is whether P^{NP} in Theorem 3.1 is optimal. We will show in Corollary 3.4 that this is true at least for relativizing techniques, by constructing a relativized world where there is a sparse selfreducible set that is not in $\mathrm{P}^{\mathrm{NP}[n]}$. Concerning tree selfreducibility, we will show in Theorem 3.3 that for the natural case of sparse, tree-selfreducible sets we can find a better bound than P^{NP}, namely $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$. Optimality of this result is supported by Corollary 3.9 that shows a relativized world in which a sparse, tree-selfreducible set S exists such that $S \in \mathrm{P}^{\mathrm{NP}[O(\log n)]}$ but S is not in any lower class. This will follow from Theorem 3.7 and Lemma 3.8. We will first isolate and prove a crucial lemma.

Lemma 3.2 *Let M be a tree-selfreduction obeying $\prec$ and witnessing the selfreducibility of some set S and let x be some input. Let $T \subseteq S_x^M$ be a tree with root x that has maximal weight among all trees $T \subseteq S_x^M$, which can be labeled M-consistently. Then it holds that $S \cap S_x^M \subseteq T$.* [3]

Proof Let l be the M-consistent labeling of T. Assume that $(S \cap S_x^M) - T$ is nonempty. Let y be the deepest node in $(S \cap S_x^M) - T$. Note, that this implies that y has no children in S. Let p be the (unique) path from y to the root x in S_x^M. Let $p = p_{out}p_{in}$ such that p_{out} is the part of p outside of T and p_{in} is the part of P inside T. Note that p_{out} contains at least y. Let

[3]Here S_x^M resp. T denote the nodes of the graphs $S_x^{S,M}$, T.

T' be the same as T, but with path p_{out} added and let l' be the (unique) M-consistent labeling of T'. Note that $l'(y) = 1$, because y has no children in S. For nodes in T, labelings l and l' differ at most on the path p_{in}. The total weight that p_{in} contributes to the weight of T can be at most $|p_{in}|$. Path p_{out} contributes $|p| > |p_{in}|$ to the weight of T', so the weight of T' is larger than that of T. □

Theorem 3.3 *If S is sparse and $\leq_T^P$-tree-selfreducible then $S \in \mathrm{P}^{\mathrm{NP}[O(\log n)]}$.*

Proof Fix x and a selfreduction machine M. We will give a $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$-algorithm to compute $x \in S$. First find the maximum weight $w_{max}(x)$ of any M-consistently labeled T. It is clear that $w_{max}(x) \in O(Pol(|x|))$. Further, there is a $k > 0$ such that for any x there is a maximally weighted tree $T \subseteq S_x^M$ with $\|T\| \leq |x|^k$. We can find $w_{max}(x)$ with logarithmically many queries to an NP oracle of the following type: "Given a weight w, guess a tree T of size at most $|x|^k$, a labeling l such that $weight(T) \geq w$. Accept if the labeling of l is M-consistent."

Lemma 3.2 guarantees that any tree T of maximum weight will contain all nodes in $S_x^M \cap S$. Recall that there is only one M-consistent labeling for any T. The true labeling of such maximally weighted T, i.e. $l(y) = 1 \leftrightarrow y \in S$, is of course M-consistent, so the (unique) M-consistent labeling of T labels all nodes correctly.

The final query will then be "Guess a tree T of size at most $|x|^k$ and an M-consistent labeling l such that $weight(T) = w_{max}(x)$. Accept iff $l(x) = 1$." Our $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$-algorithm then just outputs the result of this query. □

3.2 Polynomial Lower Bounds

In this section we will show that there can be no relativizing proof that shows sparse selfreducible sets to be in $\mathrm{P}^{\mathrm{NP}[n]}$.

Theorem 3.4 *There exists an oracle A and a sparse selfreducible relative to A set S such that S is not in $\mathrm{P}^{A,\mathrm{NP}^A[n]}$.*

In the rest of this section we prove this theorem. We will first show how to fool one pair (M, N), where M is a deterministic poly time oracle machine and N is a non-deterministic poly time oracle machine.

We start with explaining how the set S and the oracle A are related. We will define a P-set $X \subset \{0\}^*$ and for each $0^n \in X$ we will define a circuit C_n without inputs. The oracle A will encode all the circuits C_n, $n \in X$, and S will encode the values of all the gates in C_n, $n \in X$. Moreover we let $0^n \in S$ iff the value of the output gate of C_n is 1.

We will use the following definition of a circuit without outputs. A circuit C is identified by (1) a number k of *gates*, (2) a mapping that assigns to each gate g_i of C a Boolean function f_i computed in that gate and called the *type of* g_i and (3) a mapping that assigns to each gate g_i a tuple of preceding gates $g_{j_1}, \dots, g_{j_l}$, called *inputs to* g_i, where l the arity of f_i. We say that gate g_i *precedes* gate g_k if $i < k$. As functions f_i we will use constants 0, 1, and-of-nots of fan-in at most $n+2$, $\neg x_1 \wedge \dots \wedge \neg x_i$, and ORs of fan-in two, $x_1 \vee x_2$. The *value* of a gate g_i is defined by induction on i. We will call a gate g a *1-gate*, if its value is 1 and *0-gate* otherwise.

The circuit C_n will have $2^{p(n)}$ gates for some polynomial p. We will thus identify gates of C_n with binary strings of length $p(n)$. The type of gate g_i will be computable in polynomial time given n and g_i. An the oracle $A(C_n)$ will provide the information about the inputs to x: when queried a pair $\langle n, g\rangle$ the oracle provides the sequence of inputs to g.

Let $S(C_n)$ stand for the set of all $\langle n, g\rangle$ such that g is a 1-gate in C_n. There will be a fixed polynomially related order $\prec$ such that $\langle n, u\rangle \prec \langle n, v\rangle$ for all gates u and w in C_n such that u is an input to v. In particular, the depth of C_n will by bounded by a polynomial of n. Clearly this implies that the set $S = \bigcup_{n\in X} S(C_n)$ is selfreducible relative to oracle $A = \bigcup_{n\in X} A(C_n)$. Indeed, given a string $x = \langle n, g\rangle$ we can find in polynomial time the type of g. Querying the oracle A, we find inputs $g_1, \dots, g_l$ to g. Then by querying S for $\langle n, g_1\rangle, \dots, \langle n, g_l\rangle$ we can find values of $g_1, \dots, g_l$ and compute the output of g.

The density of S is bounded by the number of 1-gates in the family of circuits C_n. Our construction below will ensure that this number is polynomially bounded. This partially explains the choice of OR and and-of-nots functions, as the only non-constant functions allowed in C_n. Indeed, the functions $x_1 \vee \dots \vee x_k$ and $\neg x_1 \wedge \dots \wedge \neg x_k$ have the following feature: both values 0,1 can be forced by assigning only one 1 to $x_1, \dots, x_k$. One can notice that the second function is the negation of the first one. However, we cannot replace the and-of-not gate $\neg x_1 \wedge \dots \wedge \neg x_k$ by a circuit with the NOT gate on the top applied to a tree of $k-1$ ORs. Indeed, in the case when $x_1 \vee \dots \vee x_k = 1$ some gates in this circuit will evaluate to 1, whereas the gate $\neg x_1 \wedge \dots \wedge \neg x_k$ evaluates to 0. As C_n has exponentially many and-of-not gates, such a replacement would add exponentially many strings in S.

The oracle model presented here is for convenience of the proof only. It can be replaced by the standard model by storing the inputs in the form of sequences $\langle n, g, L\rangle$ where L is a list of inputs to g. If we also store $\langle n, g, K\rangle$ for all K that are prefixes of L, then a standard YES, NO polynomial time bounded oracle machine can recover L using a number of queries linear in $|L|$. Clearly the number of queries in the standard model is lower bounded by the number of queries in our preferred model.

Now we are able to present the first theorem stating that using such technique we can fool one pair (M, N) of a deterministic poly time oracle machine M and a non-deterministic poly time oracle machine N.

Theorem 3.5 *For every deterministic polynomial time bounded Turing machine M and every nondeterministic polynomial time bounded Turing machine N for all sufficiently large n there exists a circuit $C = C_n$ as described above such that the value of output gate of C differs from $M^{A(C),N^{A(C)}[n]}(0^n)$. The number of 1-nodes in C_n is at most $O(n^3)$.*

Proof We will start with defining a class of circuits, called *normal* circuits, which will contain all circuits C_n for $n \in X$. Describing the design of a normal circuit we will provide intuitive reasons for it.

Let us first try the following. Let $C = C_n$ be a binary tree of depth n consisting of OR gates where inputs to leaves are constants 0 and 1. Such a design suffices to fool the pair (M, N) provided M does not query N. Indeed, the only information provided by $A(C)$ is the information about inputs to leaves of C. Run $M^{A(C)}(0^n)$ and answer all the queries by saying that the queried input is connected to constant 0. When M has halted and returned a result, fool M as follows. If the result is 1 then connect all yet unconnected leaves to constant 0. Otherwise pick a non-queried leaf (if n is large enough there is such a leaf) and connect it to constant 1, and connect all other yet unconnected leaves to constant 0. Notice that the number of 1-nodes in the constructed circuit C_n is at most $n + 1$.

This design does not work, if M can query N. Indeed, one NP query is enough to find the output of C ("is there a leaf in C connected to 1?"). Nevertheless OR trees of depth n are helpful and we use them as sub-circuits of C_n. Intuitively, evaluating a root of an OR tree costs one NP query even if the enemy can evaluate in polynomial time each its leaf. Another useful property of OR trees of depth n is the following: we can force its root evaluate to one by setting only $n + 1$ its gates to 1.

Let us outline the coarse structure of C_n. It consists of an OR tree T of depth n on the top, n copies of a special circuit S_n (to be defined later) and two constants 0,1. Constants 0,1 sit on the bottom level 0 of C_n. The next level 1 contains a copy of S_n, called S_n^1, inputs to which are constants 0,1 from level 0. The next $n - 1$ levels contain the remaining $n - 1$ copies of S_n so that the inputs to S_n^{i+1} are certain outputs of S_n^i or the constant 0. The OR tree is on the top level $n + 1$ and its inputs are certain outputs of S_n^n or the constant 0. The circuit S_n will depend on m only. Its property is stated in the following Lemma 3.6. Thus the only freedom in designing C_n consists in choosing a mapping from inputs of the OR tree T and inputs of all copies of S_n to outputs of copies of S_n and constants 0,1. If we decide that input g is mapped to output g' we will say that *g' and g are connected.* If we decide that input g is mapped to constant 0 or 1 we say that it is *connected to 0 or 1.*

To state the desired properties of S_n let us introduce the following notation. For a set P of inputs of S_n let 1_P stand for the assignment of 0/1 to inputs of S_n where we set all inputs in P to 1 and all remaining inputs to 0. In the next lemma we call a set *small* if it has less than 2^n elements and *tiny* if it has less than 2^{n-1} elements.

Lemma 3.6 *There is an explicit circuit S_n that has the following property. For every tiny set O of outputs of S_n there are outputs $v, w \notin O$ such that for every small set I of inputs to S_n there is a set P disjoint with I such that for every small set $J \supset$ of inputs to S_n there is a set Q disjoint with J with the following properties. (1) For assignment 1_P outputs v, w evaluate to 1 all other outputs evaluate to 0. (2) For assignment $1_{P\cup Q}$ the output v evaluates to 1 and all other outputs evaluate to 0. (3) For both assignments 1_P, $1_{P\cup Q}$ the total number of 1-gates in S_n is $O(n^2)$.*

We will assume the lemma and finish the proof. A circuit that has the design as explained above is called *normal* if for every $i = 1, \ldots, n$ an output s_i of S_n^i is singled out (called the *distinguished* 1-output) so that s_i evaluates to 1 and each other output of S_n^i evaluates to 0 or has zero fan-out (that is, s_i is the only 1-node of C_n connected to inputs of S_n^{i+1}). A normal circuit is shown in Fig. 1.

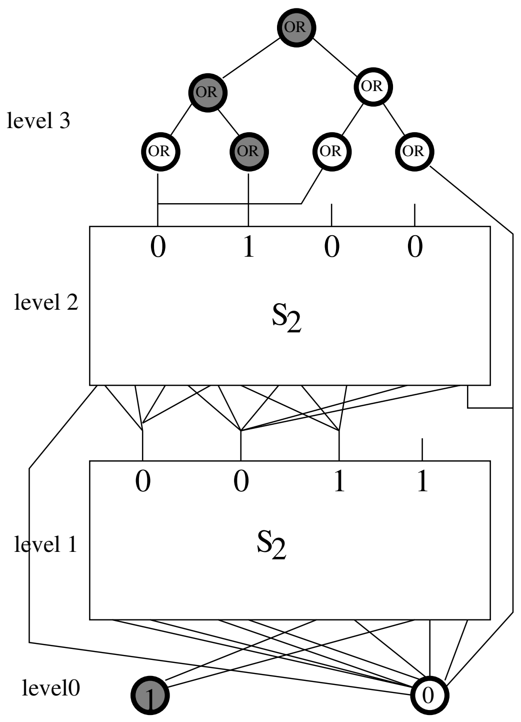


Figure 1: A normal circuit for $n = 2$. 1-gates are marked grey. 1s and 0s indicate values of outputs of S_n.

A normal circuit C_n satisfying the theorem, is built in steps. Fix a large n and drop the subscript n. Roughly speaking, after step $i-1$ we will have a "partial" circuit C^{i-1}, a circuit where some inputs have not been connected. On step i we will construct a partial circuit C^i extending C^{i-1} so that to "fix" the answer of $M(0^n)$ to ith query to oracle N^A. That means that for some $a_i \in \{0,1\}$ for every normal circuit C extending C^i the answer of $N^{A(C)}$ to ith query is a_i. We will pay for that by connecting all inputs of S^i and connecting at most polynomially many inputs of $S^{i+1}, \dots, S^n$ and T. To construct C^i we will apply Lemma 3.6 to the set O of all already connected outputs of S^i and the set I of all already connected inputs of S^i. Thus both v, w will have zero fan-out and we will be able to apply Lemma 3.6 for S^{i+1}.

Now we proceed to the formal proof. W.l.o.g. we may assume that the only allowed queries to oracle $A(C)$ have the form "which output is assigned to x?", where x is an input of S^i or of the OR tree T.

We will define by induction a sequence $C^0, \ldots, C^n$ of "partial circuits", the sequence of distinguished outputs $s_0, \ldots, s_n$ and a sequence $a_1, \ldots, a_n$ of answers to queries to N^A that have the following properties.

(1) In the partial circuit C^i all inputs of $S^1, \ldots, S^i$ are connected to some outputs and also at most $ip(n)$ remaining inputs are connected to some outputs (with p a fixed polynomial).

(2) s_i is an output of S^i that evaluates to 1 in C^i and all outputs of S^i evaluating to 1 (including s_i) have zero fan-out in C^i.

(3) All outputs of S^{i-1} evaluating to 1, except s_{i-1}, have zero fan-out in S^i.

(4) The answers $a_1, \ldots, a_i$ are the actual answers to first i queries of $M^{A(C)}(0^n)$ to $N^{A(C)}$ for every normal circuit C extending C^i.

See Fig. 2 for possible connections in C^1.

Initially C^0 is the partial circuit with no connections yet specified, the sequence $a_1, \ldots, a_i$ is empty and s_0 is the constant 1 in level 0. Thus all conditions (1)–(4) are satisfied for $i = 0$.

Step i. Assume that $C^{i-1}, a_1, \ldots, a_{i-1}, s_1, \ldots, s_{i-1}$ satisfy conditions (1)–(4). Let $C^i = C^{i-1}$ and let O be the set of all outputs of S^i that have positive fan-out in C^i. Let I be the set of all inputs of S^i that are connected in C^i. By assumption (1) both $|I|, |O|$ are at most $(i-1)p(n)$. Thus we can apply Lemma 3.6 to O, I provided n is large enough. Let P, Q, v, w be set of outputs and inputs existing by Lemma 3.6. Connect each $j \in P$ to s_{i-1} and connect each input $j \notin P \cup I$ to 0. Enter in C^i all connections made. Note that all inputs from I are connected to 0-gates (assumption (3)). Thus both v, w evaluate to 1 in C^i and all other outputs of S^i evaluate to 0. And by construction v, w have zero fan-out in C^i.

Run $M^{A(C^i)}(0^n)$ and answer all queries to A about not yet connected inputs by saying that they are connected to 0. Update C^i according to these answers. During this run answer $a_1, \ldots, a_{i-1}$ to queries to N^A. Proceed in this way until $M^{A(C^i)}(0^n)$ makes the ith query q_i to N^A. Then consider two cases.

case 1: For every normal circuit C extending C^i such that v has zero fan-out in C we have $q_i \notin N^{A(C)}$. We then let $s_i = w$. As v evaluates to 1 in C^i, it has zero fan-out in every normal circuit extending C^i. Thus the condition (4) is satisfied with $a_i = 0$.

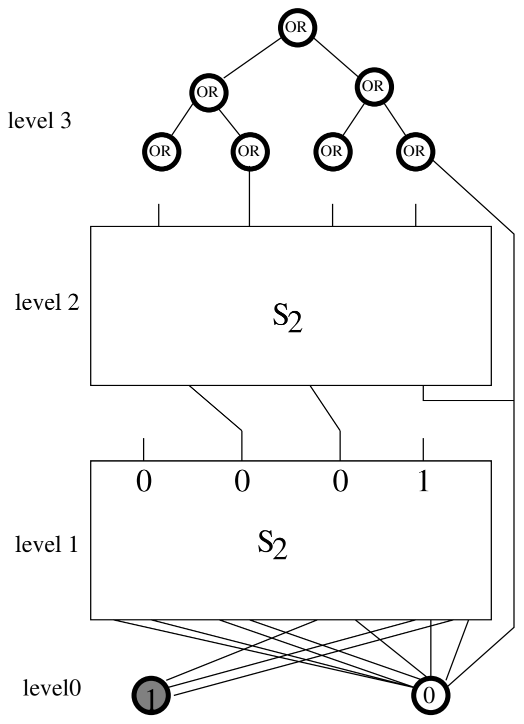


Figure 2: Possible connections inside C^1 for $n = 2$.

case 2: There is a normal circuit C extending C^i such that v has zero fan-out in C and $q_i \in N^{A(C)}$. Then pick such a C and an accepting computation of $N^{A(C)}(q_i)$ and include in C^i all connections of inputs in C queried along this computation; call W the set of all of those inputs that belong to S^i. Notice that we have added in C^i at most $p(n)$ connections, where p is a polynomial. Thus property (1) holds. For every normal circuit C extending C^i we will certainly have $q_i \in N^{A(C)}$, thus property (4) holds for $a_i = 1$. Let $s_i = v$. We can do it, as v has zero fan-out in C^i. Now we have a problem: the output w may have non-zero fan-out and it evaluates to 1 in C_i. Thus property (3) may be violated by the output w. To resolve the problem we use the set Q from the lemma. Apply the lemma for $J = I \cup W$ and obtain Q. Re-connect each input $j \in Q$ to s_{i-1}. As Q is disjoint with W, we still will have $q_i \in N^{A(C)}$ for every normal circuit C extending C^i. Moreover, as Q is disjoint with I, C^i extends C^{i-1}. Therefore every normal circuit extending C^i extends C^{i-1} as well and thus property (4) holds. Now v is the only output of S^i that evaluates to 1 and it has zero fan-out in C^i.

Note that in both cases C^i has acquired at most $O(n^2)$ 1-gates in level i on step i. After n steps we have processed all n queries to N^A and have defined C^n and s_n so that C^n has $O(n^3)$ 1-gates.

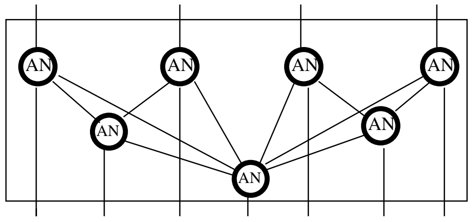
Besides, the output s_n of S^n evaluates to 1 and has zero fan-out. Continue running $M^A(0^n)$ saying that all queried leaves of the OR tree are connected to constant 0. When M has halted we obtain a partial circuit C^{n+1} such that for every normal circuit C extending C^{n+1} all the queries to both its oracles are fixed. However we still can force the circuit C output 0 (connect all yet non-connected leaves of T to constant 0) or 1 (connect one yet non-connected leaf of T to s_n and others to constant 0). We thus can fool M, N by choosing C to output the negation of M's result. $\square$

Proof of Lemma 3.6 We first construct a circuit G_n that satisfies the property of Lemma 3.6 for "small" meaning empty (and unchanged meaning of "tiny"). This basically means that there we are free in putting any inputs to P, Q. The design of G_n is shown in Fig. 3 (a). It has $2^{n+1}-1$ inputs and 2^n outputs and consists of not-of-ands gates only. They are arranged into a binary tree of depth n. Every gate of the tree is and-of-nots of all its ancestors and of an input to G_n.

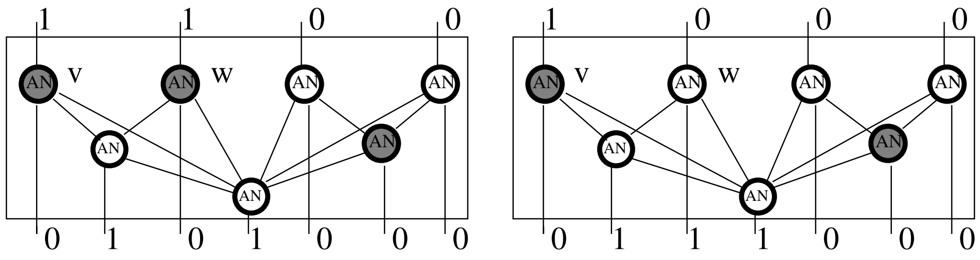
Let us show that G_n satisfies Lemma 3.6 (with "small" meaning empty). Indeed, fix a tiny set O of outputs of G_n. Partition all outputs of G_n into 2^{n-1} pairs of outputs; each pair (v, w) consists of outputs of two leaves of the tree that have common father. Obviously, there is a pair (v, w) that is disjoint with O. Pick such a pair. Let the set P consist of all input wires to all the gates g that are inputs to v or w. Let the set Q consist of only one wire — the input wire to w. (See Fig. 3 (b) and (c).)

Now we construct S_n. Its design is shown in Figure 4. Circuit S_n is obtained from G_n by connecting an OR tree of depth n to each input of G_n. The resulting circuit has $(2^{n+1}-1)2^n$ inputs. For all small sets I and $J \supset I$ of inputs to S_n each OR tree T_i has at least one input outside I and one input outside J. Setting each of these two inputs to 1 we will set to 1 the corresponding input i to G_n. Since G_n satisfies Lemma 3.6 (for "small" meaning empty), S_n satisfies Lemma 3.6 as written. $\square$

Proof of Theorem 3.4 We construct normal circuits for numbers n sufficiently far apart, such that the ith pair (M_i, N_i) of deterministic and non-deterministic polynomial time machines fails on input $0^{n(i)}$ as is standard



(a)



(b)

(c)

Figure 3: (a) Circuit G_n for $n = 2$. (b) Assignment 1_P. (c) Assignment $1_{P \cup Q}$. All 1-gates are marked grey.

in diagonalizations. The sequence $n(0), n(1), \dots$ is chosen so that $M_i(0^{n(i)})$ cannot query strings of length $n(i+1)$ and $N_i(q)$ cannot query strings of length $n(i+1)$ for any query q of $M_i(0^{n(i)})$. It is not hard to choose such a sequence in such a way that the set $X = \{n(i) \mid i = 0, 1, \dots\}$ is in P. Assume that the pairing function is chosen so that $|\langle n, x \rangle| > n$. Then all strings in the oracle $A(C_n)$ have length at least n, and thus $M_i(0^{n(i)})$, $N_i(q)$ cannot make any queries in A encoding $C_{n(i+1)}$. So we can apply Theorem 3.5 for all pairs independently and then let $A = \bigcup_i A(C_{n(i)})$ and $S = \bigcup_i S(C_{n(i)})$. $\square$

Remark We can easily improve the lower bound n for the number of queries in Theorem 3.4 to n^k for every constant k. Indeed, instead of the circuit C_n in the construction of A and S we can use C_{n^k}.

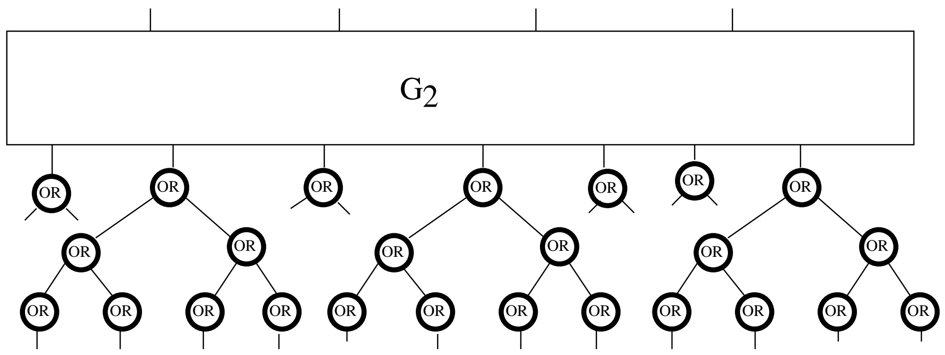


Figure 4: Circuit S_n for $n = 2$.

3.3 Exponential Lower Bounds

Let us now prove a relativized lower bound on sparse, tree-selfreducible sets, see Corollary 3.9. The proof follows easily from Theorem 3.7 and Lemma 3.8. In Theorem 3.7 we show that if there are NE-machines with a certain structural property, then one can easily derive an S as desired. In Lemma 3.8 we will then show that there is a relativized world in which NE-machines have this property.

Theorem 3.7 *Assume there is an* NE*-machine M and a set B such that*

1. *M has at most 2^n accepting paths for all inputs of length n*
2. *$x \in B$ if and only if the number of accepting paths of $M(x)$ is odd*
3. *$B \notin \mathrm{EXP}^{\mathrm{NP}[n]}$.*

Then there is a sparse, tree-selfreducible set with $S \in \mathrm{P}^{\mathrm{NP}[\log n+1]} - \mathrm{P}^{\mathrm{NP}[\log n]}$.

Proof Define

$$S' := \{\langle x, Pad'(x)\rangle \mid x \in B\},$$

where $Pad'(x)$ and the pairing function are chosen such that $|\langle x, Pad'(x)\rangle| = 2^{|x|}$. First note that S' is sparse.

Conditions 1 and 2 suggest the following $\mathrm{P}^{\mathrm{NP}[\log n+1]}$-algorithm A for S' on input $\langle x, y\rangle$: If $y \neq Pad'(x)$ reject. Set $n = |x|$ and $m = 2^n =$

$|\langle x, Pad'(x)\rangle|$. Then $M(x)$ runs in time m and has at most m accepting paths. Therefore the number of accepting paths of $M(x)$ can be computed with $\log m + 1$ queries to a suitable NP-oracle. Accept if this number is odd, otherwise reject.

Let us now prove $S' \notin \mathrm{P}^{\mathrm{NP}[\log n]}$. Assume there was a $\mathrm{P}^{\mathrm{NP}[\log n]}$-algorithm A' which decides S'. Then the following $\mathrm{EXP}^{\mathrm{NP}[n]}$-algorithm for B shows a contradiction to condition 3. On input x compute $\langle x, Pad'(x)\rangle$, which has length $m = 2^n$. Start A' on $\langle x, Pad'(x)\rangle$ which can by assumption decide $x \in B$ with $\log m = n$ queries to an NP-oracle, using binary search.

Now we define the selfreducible set S.

$$S := \{\langle x, Pad(x), v\rangle \mid \oplus\|\{w \mid vw \text{ is an accepting path of } M(x)\}\| = 1\},$$

where this time Pad and $\langle\cdot,\cdot,\cdot\rangle$ are chosen such that $\langle x, Pad(x), \lambda\rangle = 2^{|x|}$. We have

$$\chi_S(\langle x, Pad(x), v\rangle) = \chi_S(\langle x, Pad(x), v0\rangle) \oplus \chi_S(\langle x, Pad(x), v1\rangle),$$

which means that S is 2-parity-selfreducible (χ_S is the characteristic function of S). The fact that $S \in \mathrm{P}^{\mathrm{NP}[\log n+1]} - \mathrm{P}^{\mathrm{NP}[\log n]}$ follows immediately from the proof for S'. $\square$

Lemma 3.8 *There is an oracle O, an* NE*-machine M and a set B such that*

1. *M^O has at most 2^n accepting paths for all inputs of length n*
2. *$x \in B$ if and only if the number of accepting paths of $M^O(x)$ is odd*
3. *$B \notin \mathrm{EXP}^{O,\mathrm{NP}^O[n]}$*

Proof First we define the NE-machine M. On input 0^n it non-deterministically guesses all paths y with $|y| = 2^n$ and accepts on path y iff $y \in O$. On inputs other than 0^n it always rejects. Each oracle O defines B uniquely by condition 2.

Now we use a diagonalization argument to construct O such that conditions 1 and 3 hold. For any oracle O let K_O be the standard linear time NP^O-complete set

$$\langle x, i, 1^t\rangle \in K_O \leftrightarrow \text{the } i\text{-th } \mathrm{NP}^O\text{-machine accepts } x \text{ after} \leq t \text{ steps.}$$

Let N^O be an NP^O-machine accepting K_O which runs in time $O(n)$ on inputs of length n. We will prove that $B \notin \mathrm{EXP}^{O,\mathrm{N}^O[n]}$ and by our choice of N this is equivalent to condition 3.

Let $\{M_i\}_i$ be an enumeration of all exponential time bounded oracle machines such that M_k on inputs of length n runs in time 2^{n^k} and for any oracle O makes at most 2^{n^k} queries to O and at most n queries to N^O.

We now describe the k-th stage of the diagonalization. Set the function $m(k)$ to

$$m(k) = 2^{2^{m(k-1)}} + m(k-1) \tag{1}$$

where $m(0) = 2$. For ease of notation we also write m for $m(k)$. We will ensure that

$$M_k^{O,N^O}(0^m) \text{ accepts } \leftrightarrow 0^m \notin B. \tag{2}$$

As will be clear from the construction none of the later stages will change this property. This implies condition 3.

Initially set $F := \emptyset$. In each stage of the diagonalization we will add elements to F. These elements are "frozen" and are not allowed to be put into O later.

Let u be an m-bit string. Since M_k asks at most m queries to N^O, this string induces a computation of $M_k^{O,N^O}(0^m)$, if we define that the answer of the i-th query to N^O is given by the i-th bit of u. Let Q_m be the set of all possible queries of $M_k^{O,N^O}(0^m)$ to N^O, for any such u. Note that $|Q_m| \leq 1 + 2 + 4 + \ldots 2^{m-1} = 2^m - 1$. For any such u, freeze all direct queries from M_k to O in $M_k^{O,N^O}(0^m)$ and put them into F.

We now put some elements of length 2^m into O such that we get (2).

```
1. $Q := Q_m$
2. WHILE there is $w \in (\Sigma^{2^m} - F) \cup \{\lambda\}$ and $q \in Q$ such that $N^{O\cup\{w\}}(q) = 1$ DO
    (a) $Q := Q - \{q\}; O := O \cup \{w\}$
    (b) Add all queries on the left-most accepting path of $N^{O\cup\{w\}}(q)$ to $F$
3. IF $\left(M_k^{O,N^O}(0^m) \text{ accepts and } |O^{=2^m}| = \text{odd}\right)$ or $\left(M_k^{O,N^O}(0^m) \text{ rejects and } |O^{=2^m}| = \text{even}\right)$
   THEN take any $w \in \Sigma^{2^m} - F$ and set $O := O \cup \{w\}$
```

The idea behind this algorithm is very simple: In the WHILE-loop we try to find as many potential queries $q \in Q$ to N^O, for which $N^O(q)$ already accepts ($w = \lambda$) or $N^O(q)$ becomes accepting if we add one element w of length 2^m to O. We do not want to undo the acceptance of $N^O(q)$ in later iterations, so we "freeze" all queries on the left-most accepting path and put them into F.

Note that the WHILE-loop terminates after at most $|Q_m| \leq 2^m - 1$ iterations. Observe that after the completion of the WHILE-loop adding one of these unfrozen elements of length 2^m to O also cannot change the acceptance of $N^O(q)$ for any of the remaining $q \in Q$. Since all direct queries from M_k to O in $M_k^{O,N^O}(0^m)$ were also frozen initially, the acceptance of $M_k^{O,N^O}(0^m)$ cannot change in step 3. On the other hand, adding a 2^m-long element to O adds an accepting path to $M^O(0^m)$, which changes the predicate $0^m \in B$. Thus, line 3 ensures (2) if we show that

CLAIM 1 *There is at least one unfrozen string of length 2^m at the beginning of line 3.*

Proof We first observe that by (1) none of the 2^{2^m} strings of length 2^m were frozen in one of the previous stages. We then freeze at most $2^m \times 2^{m^k}$ direct queries of M_k to O and in each of the at most $2^m - 1$ iterations of the WHILE-loop at most 2^{m^k} strings, altogether less than 2^{m^k+m+1} strings. This is smaller than 2^{2^m} by (1). □

Now, condition 3 follows from (2), since in the k-th stage we add only elements to O, which by (1) are too long to be queried anywhere in any $M_l^{O,N^O}(0^{m_l})$ for $l < k$. Furthermore, our procedure adds at most 2^m elements of size 2^m to O and thus by construction of M condition 1 also holds. Condition 2 holds by definition. □

From Theorem 3.7 and Lemma 3.8 we get

Corollary 3.9 *There is an oracle O and a sparse set S, which is 2-parity-tree-selfreducible in the relativized world O, but $S \notin \mathrm{P}^{O,\mathrm{NP}^O[\log n]}$.*

4 Applications: lower bounds for NEXP

In this section we apply the results about sparse tree selfreducible sets to obtain lower bounds for NEXP. It is well-known that $\mathrm{P}^{\mathrm{SPARSE}} = \mathrm{P}/poly$.

However the question whether $\mathrm{EXP}^{\mathrm{NP}}$ is contained in P/*poly* is still open. The best known lower bound along these lines shows that $\mathrm{MA}_{\mathrm{exp}}$, the class of languages that allow for exponentially long Arthur-Merlin games, is not in P/*poly* [BFT98].

We will show that Theorem 3.3 can be used directly to show that NEXP does not reduce to a sparse set that is tree selfreducible. The class of sets that reduce to sparse tree-selfreducible sets can be interpreted as sets that are computed by some restricted form of polynomial size circuits, and hence this result yields some lower bound for NEXP with respect to this class of polynomial size circuits. We will show moreover that there exists a relativized world where EXP has a sparse tree-selfreducible hard set.

Theorem 4.1 *Let K be a Turing complete set for* NEXP*. There is no sparse tree-selfreducible set S such that $K \leq_T^p S$.*

Proof This follows directly from [Moc93], where it is shown that NEXP is not contained in $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ and Theorem 3.3, which relativizes. $\square$

We next show that with relativizing techniques, considering only standard complexity classes, this is optimal.

Theorem 4.2 *There exists an oracle A and a sparse tree-selfreducible set S such that for every set $B \in \mathrm{EXP}^{\mathrm{A}}$, $B \leq_T^{p^A} S$.*

Proof It is sufficient to show that K^A, the standard 2^n-time complete set for $\mathrm{EXP}^{\mathrm{A}}$, reduces to S. The proof goes along the same lines as [Wil85], where an oracle is constructed relative to which EXP is in P/poly.

For length n we will code for all the 2^n strings x_i of length n, $K^A(x_i)$ into A. Assume that we correctly coded all strings of length $\leq n-1$ into A. Let M be such that $L(M^X) = K^X$ for all X. Since M^A runs in time 2^n it can query at most 2^n strings to A on any input x_i of length $\leq n$. Let $Q = Q_M^A((\Sigma^*)^{\leq n})$.

Then $\|Q\| \leq 2^{3n}$ and so $(\exists z_n \in \Sigma^{4n})(\forall v)[\langle z_n, v\rangle \notin Q]$. Now we are able to code for every string x_i of length n, $K^A(x_i)$ into A as follows.

$$\langle z_n, x_i\rangle \in A \leftrightarrow K^A(x_i) = 1$$

It is clear that the above construction will yield a z_n for every length n. We now will code z_n into a sparse tree selfreducible set S as follows:

$$S = \{\langle 0^n, v\rangle \mid \exists w : vw = z_n\}$$

It is easy to see that given access to S one can recover z_n and then decide for every string x of length n whether it is in K^A by querying $\langle z_n, x\rangle$. In order to make the set S tree-selfreducible we put $\langle z_n, \lambda\rangle$ in A as well. The selfreduction for S is now as follows: on input $\langle 0^n, v\rangle$ query whether $\langle 0^n, v0\rangle$ or $\langle 0^n, v1\rangle$ is in S, if $|v| < 4n$, otherwise decide $\langle 0^n, v\rangle$ for $|v| = 4n$ by querying whether $\langle z_n, \lambda\rangle \in A$. □

We don't know how to prove that EXP does have a sparse tree-selfreducible hard set, but we can connect this question to a recent line of research by Fortnow, Klivans, Shaltiel, and Umans [FK05, SU04]. The argument runs as follows. If EXP has a sparse tree-selfreducible hard set, then from Theorem 3.3 we would get that EXP $\subseteq \mathrm{P}^{\mathrm{NP}[O(\log n)]}$. It then follows from [FK05] that EXP $\subseteq$ NP/log.

5 Log-sparse selfreducible sets

We will next prove that log-sparse selfreducible sets are in $L \in \mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$. The proof of the theorem can easily be adapted to yield Theorem 3.1, which was first proven in [LT91] with a different proof. The proof idea is the following. Given a log-sparse selfreducible set S and a string x, then $S_x \cap S$ has at most $O(\log|x|)$ elements. We will show a $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$ algorithm that recovers these elements in a "depth first" fashion. The structure S_x is now no longer a tree, since different paths can lead to the same element, but with the help of an NP oracle, the longest path to such a string can be recovered. The length of such a longest path is the *depth* of this string. Note that there can be different strings with the same depth in S_x, but if there are, then their longest paths from x split. Having recovered all elements in $S_x \cap L$ in this way, there can be only one string in this set of depth 0, namely x.

Theorem 5.1 *Let $L \subseteq \Sigma^*$ be log-sparse and $\leq_T^P$-selfreducible. It follows that $L \in \mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$.*

Proof Choose a constant c' such that $\|L \cap (\Sigma^*)^{\leq n}\| \leq c' \log n$. Let $\prec$ be the underlying polynomially related ordering and let M be a polynomial-time oracle machine witnessing the selfreduction. Fix some input x. Choose s such that $S_x \subseteq (\Sigma^*)^{\leq |x|^s}$. Now $\|S_x \cap L\| \leq c's \log n$. Let $c's = c$.

For $y \prec x$ let $d_x(y) = \max\{d \mid x \succ a_1 \succ \cdots \succ a_{d-1} \succ y\}$, where $d_x(x) = 0$. We call $d_x(y)$ the depth of y.

We define oracle O as $\langle x, d_1, \ldots, d_l \rangle \in O$ if and only if there are distinct strings $a_1, \ldots, a_l$ such that $(\forall 1 \leq i < j \leq l)[(d_i \geq d_j) \wedge (a_i \prec x) \wedge (d_x(a_i) \geq d_i) \wedge (M^{\{a_1,\ldots,a_{i-1}\}}(a_i) = 1)]$.

Clearly, $O \in \mathrm{NP}$. We now give a $\mathrm{P}^{\mathrm{NP}[\log^2 n]}$-algorithm that decides whether $x \in L$.

```
1. $i := 0$
2. WHILE $\langle x, d_1, \ldots, d_i, 0 \rangle \in O$ DO
    (a) $i := i + 1$
    (b) Use binary search to find the maximum value $d_i$ such that $\langle x, d_1, \ldots, d_{i-1}, d_i \rangle \in O$
3. ACCEPT if $i > 0$ and $d_i = 0$; otherwise REJECT
```

The following claim is immediate.

CLAIM 2 *After the i-th iteration of line 2b, the algorithm has recovered $d_1, \ldots, d_i$ such that $(\forall y \in L \cap S_x)[(\exists j \leq i)[d_j = d(y)]$ or $(d(y) < d_i])$.*

CLAIM 3 *The algorithm stops after at most $c \log|x|$ iterations.*

Proof After $c \log|x|$ iterations, it has built a string of $c \log|x|$ values d_i, The query in step 2b requires $L \cap S_x$ to have $c \log|x|$ distinct strings that are accepted by M using this set of strings as an oracle. By Claim 2 these are the deepest strings in $L \cap S_x$ since the second part of the disjunct can no longer be true. Hence acceptance of M means that these strings are indeed in $L \cap S_x$. So after $\|S_x\| - 1 < c \log|x|$ iterations, the next query requires recovering *all* strings in $L \cap S_x$. Furthermore, there is at most one string of depth 0. □

The proof of the theorem is now completed by observing that the depth of any string in S_x is at most polynomial in $|x|$. Hence binary search can be performed in $O(\log|x|)$ steps. □

If in Theorem 5.1 we assume $\leq^P_{btt}$-selfreducibility then we get a stronger conclusion.

Theorem 5.2 *If L is log-sparse and $\leq^P_{btt}$-selfreducible then $L \in \mathrm{P}$.*

Proof Let c be a constant such that $\|L^{\leq n}\| \leq c \log n$ and let s be a constant such that $(\forall x)[S_x \subseteq (\Sigma^*)^{\leq |x|^s}]$. This implies that $\|S_x \cap L\| \leq cs \log |x|$. Let M be an oracle machine that witnesses the $\leq_{btt}^P$ reduction and assume that M asks no more than b queries on any input. Because of the fact that queries are asked non-adaptively, we can limit the structure S_x to queries "of interest," i.e., we can assume that the set of nodes that are direct descendants of x is $Q_M(x)$, the set of nodes that are direct descendants of these nodes is $Q_M(Q_M(x))$ etc. This is what will make the algorithm below polynomial time bounded.

Of course, a string may be queried on different paths and therefore S_x is still a DAG. For a node y in S_x this time define the depth $d_x(y)$ as the *minimal* length of a path from x to y in S_x, where $d_x(x) = 0$. Let S_x^k be the part of S_x which contains all nodes up to depth k (inclusive). Set $level_x(k) = \{y \in S_x : d_x(y) = k\}$. Note that for $i \neq j$ it holds $level_x(i) \cap level_x(j) = \emptyset$. Nodes in level k of S_x^k can only have nodes in level k or $k-1$ as ancestors. Therefore nodes in level k are the sinks in the DAG S_x^k.

CLAIM 4 *Consider a partial labeling $l : S_x^k \mapsto \{0,1\}$. Call l correct if $l(x) = 1 \Leftrightarrow x \in L$. If all nodes in level k are labeled correctly, then S_x^k can be labeled correctly using M.*

Proof With induction on the number of unlabeled nodes remaining. It is clear that if this number is 1, i.e., only x remains unlabeled, then we know the answer to the queries $Q_M(x)$, so we can label x correctly. If this number is m, then starting from x we can walk down a path to end up in a node y that has only (correctly labeled) sinks as descendants, i.e., we know the answers to the queries $Q_M(y)$ and therefore can label y correctly. The DAG $S_x^{k\prime}$ that is S_x^k with y additionally labeled has one less unlabeled node. $\square$

Surprisingly, the fact that makes the proof complete is that, for large enough k, x can also be labeled correctly if some or all nodes in level k of S_x^k are labeled *incorrectly*. Therefore, the following algorithm decides whether $x \in L$.

1. FOR $k = 2cs \log|x|$ to $3cs \log|x|$ DO
   (a) Compute the DAG S_x^k.
   (b) Label all nodes in S_x^k as follows. Label all nodes in S_x^k of depth k with 0. Compute from that the labels of all nodes in S_x^k with lower depth using the selfreduction.

(c) IF the root, i.e., x is labeled 1 and S_x^k does not contain more than $cs \log |x|$ 1-nodes THEN accept and HALT.

2. Reject.

Note that each S_x^k contains at most $\frac{b^{3cs \log |x|+1}-1}{b-1} \in O(Pol(|x|))$ nodes. Thus, this algorithm works in polynomial time. We now show that it is also correct.

CLAIM 5 *If $x \in L$ then A accepts*

Proof It is clear that (the correct) selfreduction-DAG S_x always contains a level k with $2cs \log |x| \leq k \leq 3cs \log |x|$ such that $level_x(k) \cap L = \emptyset$, because there can be at most $cs \log |x|$ elements from L in S_x. For such k A labels the nodes in $level_x(k)$ correctly. The claim now follows from Claim 4. □

CLAIM 6 *If A accepts then $x \in L$.*

Proof Assume contrarily that A accepts an $x \notin L$, during iteration k. By Claim 4 this can only happen if $level_x(k)$ is not correctly labeled, which means:

$$level_x(k) \cap L \neq \emptyset.$$

Let S_x^k be labeled as given by step 1b. Since S_x^k contains $k \geq 2cs \log |x| + 1$ levels and A accepts x, the condition in 1c implies that S_x^k contains at least $cs \log |x| + 1$ levels whose nodes are all labeled with 0. Assume we now change the labels of the nodes in $level_x(k) \cap L$ (correctly) from 0 to 1 and compute from that the labels of all other levels in S_x^k. By Claim 4 S_k^x is then correctly labeled. We now want to prove that this cannot have an effect on the label of x.

Suppose it changes the label of the root x. Changing the label of a node y only has an effect if this changes the label of at least one of the parents of y and nodes in level i can only have parents in levels $\leq i - 1$. So for each $i = k - 1, \ldots, 0$ there must be at least one node in $level_x(i)$ which is changed. Thus, the changed S_x^k has at least one 1-node in each of the $\geq cs \log |x| + 1$ levels which before were completely labeled with 0. But this contradicts that S_x contains at most $cs \log|x|$ 1-nodes. □

This completes the proof of Theorem 5.2. □

The same proof idea also establishes that log-sparse sets L, which are $\leq_{tt}^P$-selfreducible can be decided in time $O(n^{\log n})$.

6 Conclusions

Selfreducible sets are all in PSPACE. In fact many natural PSPACE complete sets, like Quantified Boolean Formulae and infinite versions of two-player games are selfreducible. There is no upper bound on the computational complexity of sparse sets. The intersection of these classes turns out to be of considerably less computational complexity. Since selfreducibility, and in particular tree-selfreducibility, is a property that many problems share and it is a crucial property that allows for recursive programs and divide and conquer strategies, it is interesting to investigate properties of selfreducible sets in different complexity classes and of different densities. Many open questions remain here, especially with respect to different forms of selfreducibility and the corresponding upper bounds on the computational complexity of problems. This paper is just a starting point that shows some interesting and sometimes unexpected cases.

Our results are also somewhat surprising with respect to structural properties of complexity classes. Sparse sets show, concerning their structural and computational properties, great resemblance to P-selective sets [HT02]. Sparse sets and P-selective sets [4]. are equivalent with respect to polynomial-time Turing reductions [Sel79, Sel82]. Both P-selective sets [Ko83] and Sparse sets have polynomial size circuits [Mey77] and their difference in lowness (if any) is limited (see [KS85]). Both P-selective sets [AA96, BKS95, Ogi95] and Sparse sets [OW91] have the property that if NP btt reduces to such a set, then P = NP. The situation becomes drastically different when we limit these classes to their selfreducible subclasses. Where the class of selfreducible P-selective sets is just another name for P [BT96], the computational complexity of the sparse selfreducible sets is quite a different matter as we have shown in this paper.

Some specific open problems are the following:

1. Does there exist a sparse selfreducible set that is not in $\mathrm{P}^{\mathrm{NP}[\log n]}$? This would yield a sparse selfreducible set that is not tree-selfreducible.

2. Does there exist a relativized world where NEXP has a sparse selfreducible hard set?

3. Prove that EXP does not have a sparse tree selfreducible hard set. This proof needs to be non-relativizing, but it may be within reach

[4]A set S is called P-selective if there exists a polynomial time function f such that $f(x,y) \in \{x,y\}$ and $[x \in S \vee y \in S] \Rightarrow f(x,y) \in S$

using non-relativizing techniques from for example the MIP = NEXP proof.

4. Can the super polynomial lower bounds for NEXP be used to prove some kind of derandomization result? Is the selfreducibility restriction on the sparse set a real restriction or could one show that if NEXP has a sparse hard set then there also is a sparse hard set that is (tree) selfreducible?

References

[AA96] M. Agrawal and V. Arvind. Quasi-linear truth-table reductions to p-selective sets. *Theoretical Computer Science*, 158:361–370, 1996.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer-Verlag, 1988.

[BFT98] H. Buhrman, L. Fortnow, and T. Thierauf. Nonrelativizing separations. In *IEEE Conference on Computational Complexity*, pages 8–12. IEE Computer Society Press, 1998.

[BH77] L. Berman and H. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM J. Comput.*, 6:305–322, 1977.

[BKS95] R. Beigel, M. Kummer, and F. Stephan. Approximable sets. *Information and Computation*, 120(2):304–314, 1995.

[BT96] H. Buhrman and L. Torenvliet. P-selective self-reducible sets: A new characterization of P. *J. Computer and System Sciences*, 53(2):210–217, 1996.

[FK05] L. Fortnow and A. Klivans. NP with small advice. In *Proceedings of the 20th IEEE Conference on Computationa Complexity.* IEEE Computer Society Press, 2005. to appear.

[FO05] Piotr Faliszewski and Mitsunori Ogihara. Separating the notions of self- and autoreducibility. In *MFCS*, pages 308–315, 2005.

[HT02] L.A. Hemaspaandra and L. Torenvliet. *Theory of Semi-Feasible Algorithms.* Monographs in Theoretical Computer Science. Springer-Verlag, Heidelberg, 2002.

[Ko83] K.-I. Ko. On self-reducibility and weak P-selectivity. *J. Comput. System Sci.*, 26:209–211, 1983.

[KS85] K. Ko and U. Schöning. On circuit-size and the low hierarchy in NP. *SIAM J. Comput.*, 14(1):41–51, 1985.

[LT91] A. Lozano and J. Torán. Self-reducible sets of small density. *Mathematical Systems Theory*, 1991.

[Mey77] A. Meyer. oral communication. cited in [BH77], 1977.

[Moc93] S. Mocas. *Separating Exponential Time Classes from Polynomial Time Classes*. PhD thesis, Northeastern University, 1993.

[Ogi95] M. Ogihara. Polynomial-time membership comparable sets. *SIAM Journal on Computing*, 24(5):1168–1181, 1995.

[OW91] M. Ogiwara and O. Watanabe. On polynomial time bounded truth-table reducibility of NP sets to sparse sets. *SIAM J. Comput.*, 20:471–483, 1991.

[Pap94] C.H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[Sel79] A. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Math. Systems Theory*, 13:55–65, 1979.

[Sel82] A. Selman. Analogues of semicursive sets and effective reducibilities to the study of NP complexity. *Information and Control*, 52(1):36–51, January 1982.

[SU04] R. Shaltiel and C. Umans. Pseudorandomness for approximate counting and sampling. Technical Report TR04-086, ECCC, 2004.

[Wag88] K. Wagner. Bounded query computations. In *Proc. 3rd Structure in Complexity in Conference*, pages 260–278. IEEE Computer Society Press, 1988.

[Wil85] C.B. Wilson. Relativized circuit complexity. *J. Comput. System Sci.*, 31:169–181, 1985.