Monotonic oracle machines and binary search reductions*

Martin Mundhenk†

May 7, 1996

Abstract

Polynomial-time oracle machines being restricted to perform a certain search technique in the oracle are considered. These search techniques (e.g. binary search, prefix search) are expressed as monotonicity properties of the queries computed by the oracle machine. The power of different kinds of restricted machines is systematically investigated. It turns out that restrictions are comparable in different ways if the class of oracles is restricted to NP resp. to sparse sets, or if restricted machines being additionally positive (as defined by [Sel82]) are considered.

1 Introduction

The notion of a polynomial-time oracle machine is a basic tool in complexity theory. Different types of machines having restricted access to the oracle are well-studied (see e.g. [LLS75, Wag90]). For example, non-adaptive oracle machines generate their queries independent from the oracle; bounded oracle machines may ask not more than a certain number of queries; conjunctive oracle machines accept exactly when all queries were answered positively. These restrictions are independent from the oracle used by the machine, so that they can be seen as "syntactical" restrictions. We will investigate now more "semantical" types of restrictions. They are defined by certain search methods used by the oracle machine. How can we say that an oracle machine performs a binary search in an oracle? A binary search tree enables us to sort the labels of the nodes in a way, that if a label of a node fulfills a certain property the labels of all "smaller" nodes also fulfill this property. For any input, all the possible computations of an oracle machine also induce a binary *query-tree*, on which we will impose a property as above. The query-tree $M(x)$ induced by an oracle machine M on some input x is defined as follows. Its root is the first query of M on input x. Every left branch is labeled "no" and every right branch is labeled "yes", and every internal node is labeled with the query asked by M, when the preceding queries are answered corresponding to the branches of the path leading from the root to that node. Imagine this binary tree, and let all the labels (=queries) drop from their nodes in the tree down to the floor. Then these queries arrive from left ro right ordered w.r.t. inorder of their nodes. This sequence of queries is called monotonic w.r.t. a set B, if it can be cut into two parts such that the left part consists of elements in B, and the right part consists of elements not in B. If for every x, the sequence of queries obtained as above from the query-tree $M(x)$ is monotonic w.r.t. B, then M is said to be *tree-inorder monotonic w.r.t. B*. This forms the basic idea for our definition of monotonic oracle machines and the binary search reducibility defined by these machines. We can weaken this definition by considering in each query-tree $M(x)$

*A preliminary version of this work was presented at LATIN'95.

†mundhenk@ti.uni-trier.de, mundhenk@cs.engr.uky.edu, http://www.cs.engr.uky.edu/~mundhenk/

only the path $M^B(x)$, which is the path where a "yes" branch from a node is taken iff the label of the node is in B, and we also consider preorder and postorder instead of inorder.

In Section 3 we will give the formal definitions of monotonic oracle machines, the monotonic reducibilities defined by them, and we will prove their basic properties. We compare the absolut power of all the monotonic reducibilities, where we also consider oracle machines fulfilling more than one monotonicity property. It turns out, that monotonic reducibilities can be used to characterize several reducibilities being already considered under specific aspects e.g. in [Wag87, Wag90, AKM96a]. This work gives a unified approach and systematic study of monotonic reducibilities.

In Section 4 we consider monotonic machines which are additionally positive. In [Sel82], an oracle machine M is said to be positive, if $A \subseteq B$ implies that the set accepted by M using oracle A is contained in the set accepted by M with oracle B. For a positive M, if we list the accept/reject decisions of any paths $M^{A_1}(x), M^{A_2}(x), \ldots, M^{A_k}(x)$ where $A_i \subseteq A_{i+1}$, we get a sequence being monotonic w.r.t. the set {reject}. An oracle machine being both positive as monotonic, then fulfills two different monotonicity conditions. Interestingly, we show that conjunctive, disjunctive, and many-one reductions machines can be completely characterized by those machines. We argue that positiveness and monotonicity are fundamental properties of oracle machines, since the different combinations of these properties yield well-known and natural notions of reducibilities.

In Section 5 the closures of NP under monotonic reducibilities are shown to coincide with well-studied classes from the Polynomial-Time Hierarchy. The order of the reducibility makes the difference, namely Δ_2^p is the closure of NP under any inorder monotonic reducibility, and Θ_2^p is the closure under any preorder or postorder monotonic reducibility. Reductions to sparse sets – i.e. sets of small density and thus of only little use for polynomial-time oracle machines – and consequences of the existence of sparse hard sets for several complexity classes are an intensively studied topic in complexity theory (cf. [HOW92]). For reductions to sparse sets we show, that all the tree monotonic reducibilities coincide and are weaker than the path monotonic reducibilities.

In Section 6 we consider the consequences of the existence of sparse hard sets for NP or ETIME under monotonic reducibilities. As a consequence of results from Section 5 we get that the Polynomial-Time Hierarchy collapses to Δ_2^p if a sparse NP hard set exists for monotonic reducibilities. Finally we extend results from [Wat87] showing that ETIME cannot have sparse hard sets under monotonic reducibilities.

2 Preliminaries

Let $\Sigma = \{0, 1\}$ be the standard alphabet, and let $A \subseteq \Sigma^*$ be a set. $\sqsubseteq$ denotes the prefix relation between strings, i.e. $u \sqsubseteq v$ iff $v = uv'$ for some $v' \in \Sigma^*$. $u_{[n]}$ denotes the length n prefix of u. The length of a string x is denoted by $|x|$. ε denotes the empty string, i.e. $|\varepsilon| = 0$. $A^{=n}$ ($A^{\leq n}$) denotes the set of all strings in A of length n (up to length n, respectively). A set S is called sparse if its census is bounded above by a polynomial. A set T is called a tally set if $T \subseteq 0^*$. We use TALLY and SPARSE to denote the classes of tally and sparse sets, respectively. ETIME denotes the complexity class DTIME$(2^{O(n)})$. FP denotes the class of polynomial-time computable functions. We will use the following polynomial-time reducibilities. $\leq_T^p$ denotes the Turing reducibility, $\leq_{k\text{-}tt}^p$ the bounded Turing reducibility with at most k queries to the oracle, and $\leq_m^p$ the many-one reducibility.

Definition 2.1 *1. [Wag87] A is Hausdorff reducible to B (denoted $A \leq_{hd}^p B$), if there exists a polynomial-time computable function f mapping every string x to a sequence of an even number of queries, such that for all $x \in \Sigma^*$, for $f(x) = \langle y_1, \ldots, y_{2m} \rangle$ it holds that*

- *$y_{k+1} \in B \Rightarrow y_k \in B$, for all $k \in \{1, 2, \ldots, 2m-1\}$, and*

- $x \in A \iff \max(\{j \mid 1 \le j \le 2m : y_j \in B\} \cup \{0\})$ *is odd.*

2. *[LLS75] A set A is conjunctively reducible to a set B (denoted $A \le_c^p B$) if there exists an* FP *function f such that for every $x \in \Sigma^*$, $x \in A$ if and only if $f(x) \subseteq B$. A disjunctively reduces to B ($A \le_d^p B$) iff $\overline{A} \le_c^p \overline{B}$.*

$R_\alpha^p(B)$ denotes the class of sets $\le_\alpha^p$ reducible to B, $R_\alpha^p(\mathcal{C}) = \bigcup_{B \in \mathcal{C}} R_\alpha^p(B)$.
For further definitions confer [BDG90].

3 Monotonic machines

An oracle (Turing) machine M is a polynomial-time oracle machine, if for a polynomial q, every oracle B, and every input x, the computation of M on input x using oracle B is $q(|x|)$ time bounded. $L(M, B)$ denotes the set accepted by oracle machine M if the queries are answered by oracle set B. If we consider a computation of M on input x under all possible oracles, we obtain several computations which build the finite binary query-tree $M(x)$ of depth at most $q(|x|)$: its root is labeled with the first query of M on input x. Every left branch is labeled 0 (corresponding answer "no" from the oracle on the query) and every right branch is labeled 1 (corr. answer "yes"), and every internal node is labeled with the query asked by M when the preceding queries are answered corresponding to the branches of the path leading from the root to the node. Finally, every leaf is labeled with M's decision on that path, i.e. either "accept" or "reject". $M^B(x)$ denotes the path of $M(x)$ which corresponds to the answers from oracle B. We now formally define the labels of the nodes in the query-tree as a function of the node's location in the query-tree.

Definition 3.1 *Let M be a polynomial-time oracle machine. Then M defines function f_M and predicate acc_M on pairs of strings as follows.*

$$f_M(x,w) = \begin{cases} \text{the } |w|+1\text{st query asked by } M \text{ on input } x \text{ using oracle} \\ \{f_M(x,v) \mid v1 \sqsubseteq w\} \text{ (i.e. } M \text{ answers the oracle queries} \\ \text{according to the bitstring } w\text{),} \\ \hfill \text{if } M \text{ asks that many queries} \\ \text{undefined,} \hfill \text{otherwise} \end{cases}$$

$$acc_M(x,w) \Leftrightarrow \begin{cases} \text{the label of the node reached on path } w \text{ in } M(x)\text{,} \\ \hfill \text{if it is a leaf} \\ \text{undefined,} \hfill \text{otherwise} \end{cases}$$

For simplicity, we extend acc_M by saying that $acc_M(x,w) \Leftrightarrow acc_M(x,v)$, if v is a prefix of w and $acc_M(x,v)$ is defined. Clearly, f_M and acc_M are polynomial-time computable, if M is a polynomial-time oracle machine. We will call f_M the *query-function associated to M*. We use the following total orders of Σ^*: the inorder $\prec_{in}$ (defined by $u0v \prec_{in} u \prec_{in} u1w$), the preorder $\prec_{pre}$ ($u \prec_{pre} u0v \prec_{pre} u1w$), and the postorder $\prec_{post}$ ($u0v \prec_{post} u1w \prec_{post} u$). We now define the notion of paths in query-trees being monotonic w.r.t. some set. For the following definitions, let M be a polynomial-time oracle machine, and let f_M be the query-function associated to M.

Definition 3.2 *A path w in $M(x)$ is called* inorder monotonic (resp. preorder monotonic, resp. postorder monotonic) w.r.t. the set B, *if for all $u, v \sqsubseteq w$ it holds that $u \prec_{in} v$ (resp. $u \prec_{pre} v$, resp. $u \prec_{post} v$) implies that $f_M(x,v) \in B \Rightarrow f_M(x,u) \in B$.*

Since the considered orders of strings are transitive, if w in $M(x)$ is α monotonic w.r.t. B, then for $w_{[i_0]} \prec_\alpha w_{[i_1]} \prec_\alpha \cdots \prec_\alpha w_{[i_{|w|-1}]}$ – i.e. the list of prefixes of w ordered w.r.t. $\prec_\alpha$ – the sequence $f_M(x, w_{[i_0]}),\ f_M(x, w_{[i_1]}),\ \ldots,\ f_M(x, w_{[i_{|w|-1}]})$ is monotonic w.r.t. B. The following observation is easy to verify.

Proposition 3.3 *1. For all x and B, the path $M^B(x)$ is inorder monotonic w.r.t. B.*

2. If w is preorder monotonic w.r.t. B, then $w \in 1^*0^*$, and if w is postorder monotonic w.r.t. B, then $w \in 0^*1^*$.

If the paths in all query-trees for an oracle machine M fulfill certain monotonicity conditions, then M is said to be monotonic.

Definition 3.4 *1. M is called* path-inorder monotonic (resp. path-preorder monotonic, resp. path-postorder monotonic) w.r.t. *B, if for every x, the path $M^B(x)$ is inorder monotonic (resp. preorder monotonic, resp. postorder monotonic) w.r.t. B.*

2. M is called tree-inorder monotonic (resp. tree-preorder monotonic, resp. tree-postorder monotonic) w.r.t. *B, if for every x, every path w in the tree $M(x)$ is inorder monotonic (resp. preorder monotonic, resp. postorder monotonic) w.r.t. B.*

It follows from Proposition 3.3 that path-inorder montonicity is not a restriction on an oracle machine.

Proposition 3.5 *$A = L(M, B)$ for a path-inorder monotonic polynomial-time oracle machine M if and only if $A \leq_T^p B$ via M.*

For short, we use *t-in* for tree-inorder monotonic, *p-in* for path-inorder monotonic, and so on for the other types of monotonicities.

Proposition 3.6 *Let M be tree-inorder monotonic w.r.t. B. Then for all x and $u, v \in M(x)$ it holds that $u \prec_{in} v$ implies $f(x, v) \in B \Rightarrow f(x, u) \in B$.*

Proof Let $w \sqsubseteq M^B(x)$ be the longest common prefix of u, v, and $M^B(x)$ for $u \prec_{in} v$. If $f_M(x, w) \in B$, then $u \prec_{in} w$ implies $f_M(x, u) \in B$ following the definition of an inorder monotonic path. If $f_M(x, w) \notin B$, then $w \prec_{in} v$ implies $f_M(x, v) \notin B$. ∎

As a consequence, we observe that if M is t-in w.r.t. B, then for every x, the path $M^B(x)$ "cuts" the query tree $M(x)$ into two parts: the part to the left of $M^B(x)$ consists only of nodes labeled with elements of B, and all labels of nodes in the part to the right of $M^B(x)$ are not in B. Since this is a property of a binary search tree, we say that a tree-inorder monotonic machine performs a *binary search* reducibility.

If M is path-preorder monotonic w.r.t. B, then for every x, $M^B(x) \in 1^*0^*$. Therefore, we can construct a machine M_0 which on input x simulates M on input x until it gets a negative answer from the oracle, and then does not ask any further queries to the oracle but makes the same decision as in $M(x)$ on the path 1^k0^l, where k is the number of positive answers obtained during the simulation, and l is chosen so that 1^k0^l is a leaf in $M(x)$. It is straightforward to see that $L(M, B) = L(M_0, B)$. In a similar way, a path-postorder monotonic machine M can be simulated by a machine M_1 which simulates M until it gets the first positive answer from the oracle. We will call machines like M_0 resp. M_1 to be in *0-normal-form* resp. *1-normal-form*. The reducibilities performed by these machines will be called *positive search* resp. *negative search* reducibility.

By the above observation about p-pre machines, one can see that a tree-preorder monotonic or a tree-postorder monotonic machine performs a search which can be simulated by a binary search of order of logarithmic depth. Since both these types of monotonic oracle machines are similar powerful, this reducibility will be called *fast binary search*.

Proposition 3.7 *$A = L(M, B)$ for a tree-preorder monotonic polynomial-time oracle machine if and only if $A = L(M, B)$ for a tree-postorder monotonic polynomial-time oracle machine.*

Now we are ready to define the monotonic reducibilities.

Definition 3.8

1. *$A \leq^p_{posS} B$ (resp. $A \leq^p_{negS}$), if $A = L(M, B)$ for a polynomial-time oracle machine M being p-pre (resp. p-post) w.r.t. B.*
2. *$A \leq^p_{bS} B$ (resp. $A \leq^p_{fbS} B$), if $A = L(M, B)$ for a polynomial-time oracle machine M being t-in (resp. t-pre or t-post) w.r.t. B.*

We are now going to investigate the strength of the different types of monotonic reducibilities. Clearly, the tree monotonic reducibilities are more restricted that the path monotonic ones. They are also "robust" versus complementing the oracle set.

Proposition 3.9

1. *If $A \leq^p_{fbS} B$, then $A \leq^p_{\alpha} B$, for $\alpha \in \{posS, negS\}$.*
2. *$A \leq^p_{\alpha} B \Leftrightarrow A \leq^p_{\alpha} \overline{B}$, for $\alpha \in \{bS, fbS\}$.*
3. *$A \leq^p_{posS} B \Leftrightarrow A \leq^p_{negS} \overline{B}$.*
4. *If $A \leq^p_{\alpha} B$, then $\overline{A} \leq^p_{\alpha} B$, for any monotonic reducibility α*

From the normal-form for tree-preorder monotonic oracle machines we get

Proposition 3.10 *If $A \leq^p_{fbS} B$, then $A \leq^p_{bS} B$.*

Machines in 0-normal-form where already considered in [AKM96a]. Simple modifications of their proofs yield the following results.

Proposition 3.11

1. *$A \leq^p_{fbS} B$ if and only if $A \leq^p_{hd} B$.*
2. *$A \leq^p_{posS} B$ if and only if there exists set C such that $A \leq^p_{hd} C$ and $C \leq^p_{c} B$.*
3. *$A \leq^p_{negS} B$ if and only if there exists set C such that $A \leq^p_{hd} C$ and $C \leq^p_{d} B$.*

What happens, if an oracle machine has two different types of monotonicity? Consider a computation of a machine being both path-preorder as path-postorder monotonic. Then the answers on all queries have to be equal. Thus, it suffices to ask one of the queries.

Proposition 3.12 *$A \leq^p_{1\text{-}tt} B$ if and only if for some M: $A \leq^p_{posS} B$ via M and $A \leq^p_{negS} B$ via (the same) M.*

Proofsketch Let $A \leq^p_{posS} B$ via M and $A \leq^p_{negS} B$ via the same M, and consider the first query $q = f_M(x, \varepsilon)$ of $M(x)$. If $q \in B$, then $M^B(x) \in 1^*$ (Prop. 3.3), since M is p-post; similarly, otherwise $M^B(x) \in 0^*$, since M is p-pre. Thus the answer on the first query to the oracle suffices to compute $M^B(x)$. The other proof direction is straightforward, since every oracle machine asking at most one query is monotonic. ■

Corollary 3.13 *$A \leq^p_{1\text{-}tt} B$ if and only if for some M: $A \leq^p_{posS} B$ via M, $A \leq^p_{negS} B$ via M, and $A \leq^p_{bS} B$ via M.*

A machine being both path-preorder as tree-inorder monotonic is as powerful as a machine being tree-preorder monotonic; i.e. the more restrictive properties (path vs. tree, preorder vs. inorder) "beat" the less restrictive ones in that combination. Observe that preorder can become exchanged with postorder here.

Proposition 3.14 *$A \leq^p_{fbS} B$ if and only if for some M: $A \leq^p_{posS} B$ via M and $A \leq^p_{bS} B$ via (the same) M.*

Proofsketch The "forward" direction follows straightforward from the normal-form remark. Now, assume $A \leq^p_{posS} B$ via M and $A \leq^p_{bS} B$ via M. For any x, $M^B(x) = 1^k 0^l$, since M is p-pre. M's t-in monotonicity implies that $M^B(x)$ cuts the query-tree $M(x)$ into two parts (cf. Proposition 3.6) – to the left of $M^B(x)$ are only queries *in* B, and to the right are only queries *not* in B. Therefore it follows that for every path in $M(x)$ the sequence of queries of the preorder sorted nodes is monotonic w.r.t. B. Thus M is t-pre monotonic w.r.t. B. ■

Figure 1 summarizes the results of this section. It shows how the strength of the different monotonic reducibilities are related. The closer to the top of the figure a reducibility is located, the less restricted it is. The lines indicate that the "lower" reducibility is properly more restricted than the "higher" one. The rest of this section is devoted to the proofs of these inclusions.

Proposition 3.15

1. *There exist sets A,B such that $A \leq^p_T B$, $A \not\leq^p_{bS} B$, $A \not\leq^p_{posS} B$, $A \not\leq^p_{negS} B$.*
2. *$\leq^p_{posS}$ and $\leq^p_{negS}$ are incomparable. I.e. (1) there exist sets A, B such that $A \not\leq^p_{posS} B$ and $A \leq^p_{negS} B$, and (2) there exist sets A, B such that $A \leq^p_{posS} B$ and $A \not\leq^p_{negS} B$.*
3. *$\leq^p_{posS}$ and $\leq^p_{bS}$ are incomparable. In fact, there exist sets A, B such that (a) $A \leq^p_{posS} B$ and $A \not\leq^p_{bS} B$, and (b) $B \not\leq^p_{posS} A$ and $B \leq^p_{bS} A$.*
4. *$\leq^p_{negS}$ and $\leq^p_{bS}$ are incomparable.*
5. *There exist sets A, B such that $A \not\leq^p_{fbS} B$ and $A \leq^p_{bS} B$.*
6. *There exist sets A, B such that $A \not\leq^p_{fbS} B$ and $A \leq^p_{posS} B$.*
7. *There exist sets A, B such that $A \not\leq^p_{1\text{-}tt} B$ and $A \leq^p_{posS} B$.*
8. *There exist sets A, B such that $A \leq^p_{bS} B$, and $A \not\leq^p_{tt} B$, and $A \notin \mathrm{P}^B[O(\log n)]$.*

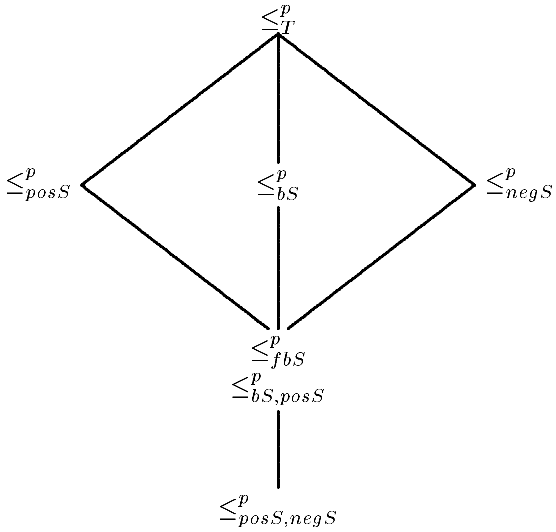

Figure 1: The strength of monotonic reducibilities

In the proof of the Proposition we will make use of the information encoded into strings, which is measured by means of Kolmogorov complexity. Time bounded generalized Kolmogorov complexity were introduced by Hartmanis [Har83]. We give a slightly modified definition for notational convenience. For a Turing machine M and integers l and s let

$$\mathrm{KT}_M[l,t] = \{x \mid \exists y : |y| \leq l \text{ and } M(y) \text{ outputs } x \text{ using at most } t \text{ steps } \}$$

be the time bounded Kolmogorov complexity (relative to M). For fixed integers this is a finite set, but we can still let l and t be unbounded functions. The following fact can be easily verified [Har83]:

There exists a Universal Turing machine U such that, for every Turing machine M, there is a constant c such that, for every l and t,

$$\mathrm{KT}_M[l,t] \quad \subseteq \quad \mathrm{KT}_U[l+c, c \cdot t \cdot \log t + c].$$

In the following we fix such a Universal machine U and omit the subscript U when no confusion arises.

Proof To prove part 1, we proceed by diagonalization. Let $M^0_1, M^0_2, \ldots$ be an enumeration of all oracle machines in 0-normal-form, and $M^1_1, M^1_2, \ldots$ be an enumeration of all oracle machines in 1-normal-form, such that M^b_i is time bounded by the polynomial $n^i + i$. We construct sets B_i in stages.

stage 0**:** let $B_0 = \emptyset$, $n_0 = 0$.

stage $i > 0$**:** Choose $n_i > n^i_{i-1} + i$. Thus no machine against we diagonalized in a former step asks the oracle for a string of length $\geq n_i$. If i is odd, then this step diagonalizes against the path-preorder monotonic machine M^1_j for $j = \frac{i+1}{2}$. Consider the computation of M^1_j on input 0^{n_i} using oracle B_{i-1}. If it is accepting, then $B_i := B_{i-1}$. If it is rejecting, then chose $x \in \{0^{n_i}, 1^{n_i}\}$ which is not queried during the computation and let $B_i := B_{i-1} \cup \{x\}$. Since M^1_j is path-preorder monotonic, it asks at most one query out of B, and thus such an x exists.

If i even, then step i diagonalizes against machine M_j^0, $j = \frac{i}{2}$. Consider the computation of M_j^0 on input 0^{n_i} using oracle $B_{i-1} \cup \{0^{n_i}, 1^{n_i}\}$. If M_j^0 accepts, then $B_i = B_{i-1} \cup \{0^{n_i}, 1^{n_i}\}$. Otherwise, during the computation at most one element of $\{0^{n_i}, 1^{n_i}\}$ is asked. Let x be the other one element of this set, and let $B_i = B_{i-1} \cup \{x\}$.

Let $B = \bigcup_{i \geq 0} B_i$. Define $A = \{0^m \mid |B \cap \Sigma^m| = 1\}$. Assume $A = L(M, B)$ for M being p-pre or p-post w.r.t. B. Then for some i and $b \in \{0, 1\}$, $L(M, B) = L(M_i^b, B)$. Let $m = 2i - b$ and consider the computation of M_i^b on input 0^m. By the definition of B, M_i^b does not ask any query of length m which is contained in B. Since this computation accepts if and only if B contains an even number of elements of length m, it follows that $A \neq L(M_i^b, B)$, contradicting that M_i^b reduces A to B.

Since t-in monotonic machines are not recursively presentable, we cannot prove $A \not\leq_{bS}^p B$ by diagonalization. Since B is sparse, we can apply Theorem 5.2, saying that $A \leq_{bS}^p B \Leftrightarrow A \leq_{fbS}^p B$. Since $A \not\leq_{posS}^p B$ implies $A \not\leq_{fbS}^p B$ (Prop. 3.9), it follows that $A \not\leq_{bS}^p B$.

The proof of part 2 case (1) uses a part of the above construction, namely the odd stages. Let B_{odd} be the set obtained, if the above construction is executed only for odd stages, and let A_{odd} be defined as A for B_{odd} instead of B. Then $A_{odd} = \{0^m \mid B_{odd} \cap \{0^m, 1^m\} \neq \emptyset\}$, and thus $A_1 \leq_{negS}^p B_{odd}$. On the other hand, $A_{odd} \not\leq_{posS}^p B_{odd}$ by the above argumentation. Case (2) then follows from Case (1) Proposition 3.9.

Observe that the answers on the queries $0^n, 1^n$ from the oracle B suffice to decide A. Thus, $A \leq_{2\text{-}tt}^p B$ follows.

We proceed with part 3. For every m, let w_m be the lexicographically first string of length m such that $w_m \notin \mathrm{KT}[\log^2 m, 2^m]$. These w_m exist by a counting argument. Define a fast growing function t by $t(0) = 0$ and $t(n+1) = 2^{t(n)}$. Let A be a set containing for lengths $t(n)$ all strings of that length up to $w_{t(n)}$, i.e. $A = \{w \mid \exists n : |w| = t(n) \wedge w \leq w_{t(n)}\}$. The set B is a bitwise encoding of strings $w_{t(n)}$, e.g. $B = \{0^{\langle t(n), i, b\rangle} \mid$ the i-th bit of $w_{t(n)}$ equals $b\}$, where $\langle \cdot, \cdot, \cdot \rangle$ is a polynomial-time computable and invertible tupling function. Observe, if $A = L(M, B)$ then M on input $0^{\langle t(m), i, b\rangle}$ does not need to ask queries $0^{\langle n, i', b'\rangle}$ for $n < t(m)$, since for $n > t(m-1)$ all these instances can be rejected, and for $n \leq t(m-1) = \log t(m)$ these instances can be decided in time polynomial in $t(m)$.

To show that $A \leq_{posS}^p B$ we use that a p-pre monotonic machine can perform a prefix search. The following machine M using oracle B, computes the longest common prefix u of input y and $w_{|y|}$ and accepts iff $u = w_{|y|}$ or the $|u| + 1$st bit in y equals 0. Since in the latter case the $|u| + 1$st bit of w_n equals 1, one can conclude from both cases together that y is lexicographically smaller than $w_{|y|}$.

> **input** y (* let $y_1 \cdots y_n$ be the bitwise representation of y *)
> **if** $n \neq t(m)$ for any m **then** *reject* **end**
> **for** $i := 1$ **to** n **do**
>   **if** $0^{\langle n, i, y_i\rangle} \notin B$ (* the common prefix of y and w_n has length $i - 1$ *)
>   **then if** $y_i = 0$ **then** *accept* **else** *reject* **end**
>   **end**
> **end**
> *accept*

If M gets a negative answer from the oracle, then M decides its input. Therefore, M is in 0-normal-form. M runs in polynomial time, and by the above discussion, M accepts an input y with oracle B iff $|y| = t(m)$ for some m and $y \leq w_{|y|}$. Thus, *Ap-preB* via M.

We now show that $A \not\leq^p_{bS} B$. Assume for contradiction that $A \leq^p_{bS} B$ via M. Since B is tally, it follows from Theorem 5.2 and Proposition 3.11 that $A \leq^p_{hd} B$ via a function f in FP. Consider $f(w_m) = \langle y_1^m, \ldots, y_{2l}^m \rangle$. Since $w_m \in A$, for some k $(1 \leq k \leq l)$ it holds that $y_{2k-1}^m \in B$ and $y_{2k}^m \notin B$. Since any $z \in \Sigma^m$ being lexocigraphically greater than w_m is not in A, $f(z)$ cannot contain the subsequence y_{2k-1}^m, y_{2k}^m starting at any odd position in the list $f(w_m)$. Thus we can describe every w_m by "the lexicographically greatest string w of length m such that for $f(w) = \langle y_1, \ldots, y_{2r} \rangle$ it holds that $y_{2i-1} = y_{2k-1}^m$ and $y_{2i} = y_{2k}^m$ for an i, $1 \leq i \leq r$". Since w_m is one of the lexicographically first $m^{\log m}$ strings of length m, it takes time $O(m^{a+\log m})$ for a constant a to compute w_m. Since y_{2k-1}^m, y_{2k}^m have length polynomial in m, it follows that there exists a constant b such that for every m, $w_m \in \mathrm{KT}[m^b, bm^{b+\log m}]$, contradicting the complexity of w_m in its definition almost everywhere. Thus, $A \not\leq^p_{bS} B$.

We continue showing $B \leq^p_{bS} A$. Let M be an oracle machine which on input $0^{\langle t(m), i, b \rangle}$ performs a binary search for $w_{t(m)}$ and decides appropriately. I.e. the query- and decision-functions for M are defined by $f_M(0^{\langle t(m),i,b \rangle}, u) = u10^{t(m)-|u|-1}$ for u with $|u| < t(m)$, and $acc_M(0^{\langle t(m),i,b \rangle}, u) \Leftrightarrow$ the i-th bit of u equals b, for $|u| = t(m)$. It is straightforward to see that $B = L(M, A)$, and that M is a polynomial-time oracle machine being tree-inorder monotonic w.r.t. A.

To prove $B \not\leq^p_{posS} A$, we need a technical claim about the number of queries asked by an oracle machine deciding B with oracle A.

Claim 1 *Let M be a polynomial-time oracle machine. If $B = L(M, A)$, then for almost every n, for $1 \leq i \leq t(n)$, $b \in \{0, 1\}$, the set of labels of nodes in the query-tree $M(0^{\langle t(n),i,b \rangle})$ cannot be computed in time polynomial in m.*

Proof Assume the contrary. Let $B = L(M, A)$, w.l.o.g. M does not ask "short" queries (as mentioned above), and let t_M be the function which on input $0^{\langle m,i,b \rangle}$ computes all the queries in the query-tree $M(0^{\langle m,i,b \rangle})$ in time bounded by the polynomial q for infinitely many $m = t(n)$. Then for infinitely many n the following holds. For $m = t(n)$, let Q_m be the set $\bigcup_{1 \leq i \leq m} t_M(0^{\langle m,i,1 \rangle})$ of all queries in the query-trees $M(x)$ for $x \in \{0^{\langle m,i,1 \rangle} \mid 1 \leq i \leq m\}$. Since the size of Q_m is bounded by $mq(m)$ and since Q_m can be generated from simple inputs, it follows that $Q_m \subseteq \mathrm{KT}[c \log m, m^c]$ for some constant c. Let v_m be the lexicographically greatest string in $Q_m \cap A$, and let M_{v_m} be the machine which simulates M but answers all queries $y \in \Sigma^m$ during this simulation with "yes" iff $y \leq v_m$. Then clearly $0^{\langle m,i,1 \rangle} \in L(M_v) \Leftrightarrow 0^{\langle m,i,1 \rangle} \in L(M, A)$, because all answers are simulated correctly. Then $w_m = \chi_{L(M_{v_m})}(0^{\langle m,i,1 \rangle}) \cdots \chi_{L(M_{v_m})}(0^{\langle m,m,1 \rangle})$, where $\chi_{L(M_{v_m})}$ is the characteristic function of the set $L(M_{v_m})$. Since every M_{v_m} is polynomially time bounded, we conclude $w_m \in \mathrm{KT}[c \log m, m^c]$ for a constant c, contradicting the choice of w_m. □

Since every $\leq^p_{posS}$ reduction can be performed by a machine in 0-normal-form, and these machines have polynomially bounded query-trees, the proof part 3 is completed.

Part 8 also follows from the above Claim.

Part 4 follows from Proposition 3.9 and part 3. Part 5 follows from part 3. Part 7 follows from part 2 and Proposition 3.12. Part 6 follows from Proposition 3.9, part 5, and part 3. ■

4 Monotonic and positive oracle machines

Positive oracle machines were defined by Selman [Sel82].

Definition 4.1 *1. An oracle machine M is called* positive*, if for all C, C': $C \subseteq C' \Rightarrow L(M, C) \subseteq L(M, C')$.*

2. *A polynomial-time positively reduces to B ($A \leq^p_{pos} B$), if $A = L(M, B)$ for a polynomial-time positive oracle machine M.*

This restriction implies monotonic acceptance behaviour of paths in the query-tree ordered by a partial order as follows. Informally, for any x and oracle machine M we say that $u \subseteq_{M(x)} v$, if there exists sets $C \subseteq D$ such that $u = M^C(x)$ and $v = M^D(x)$. If we have a sequence of paths ordered w.r.t. this order, then we yield a monotonic sequence of decisions made by the considered positive oracle machine at the end of each path. We will make this formal now.

Definition 4.2 *Let M be a polynomial-time oracle machine, f_M its query-function. For $x \in \Sigma^*$, we write $u \subseteq_{M(x)} v$, if no positively answered queries on u is answered negatively on v, i.e.*

$$\{f_M(x, u_{[i]}) \mid 0 \leq i < |u|, u_{[i+1]} = u_{[i]}1\} \cap \{f_M(x, v_{[i]}) \mid 0 \leq i < |v|, v_{[i+1]} = v_{[i]}0\}$$

is empty.

The following is easy to verify.

Proposition 4.3 *M is positive if and only if for all x, u, v, if $acc_M(x, u)$ and $acc_M(x, v)$ are defined, then $u \subseteq_{M(x)} v \Rightarrow [acc_M(x, u) \Rightarrow acc_M(x, v)]$.*

One can show that positive and monotonic reductions are independent.

We investigate the properties of machines being both monotonic and positive. As main result we prove that binary search reducibility coincides with many-one reducibility, if it is are positive.

Theorem 4.4 *If $A \leq^p_{bS} B$ via a positive M', then $A \leq^p_{fbS} B$ via a positive M'.*

Proof Let y be the first query during the computation of M on input x. We will argue, that either answer "no" or answer "yes" (or both) allows to decide x without any additional oracle query. Thus we only need to store the query y and continue the search in the subtree of $M(x)$ having that successor of the considered node as root, which does not allow to decide x. Repeating this process until a leaf is reached yields a list of polynomially many queries. The answers on these queries suffice to simulate the correct computation of M.

Let $A = L(M, B)$ for a positive and t-in (w.r.t. B) oracle machine M time bounded by the polynomial q, let f be the query-function associated to M. Fix some x, and let $l = q(|x|)$ be the depth of $M(x)$. (W.l.o.g. we assume that all paths in $M(x)$ have equal length.) Then every path v of length $|v| = l$ is either accepting or rejecting. At first we will classify the nodes w of the query-tree $M(x)$ into three different types, depending on the acceptance behaviour of the right-most path of the subtree with the left successor of w as root and of the left-most path of the subtree with the right successor of w as root.

- If $w01^{l-|w0|}$ is accepting $\not\Leftrightarrow$ $w10^{l-|w1|}$ is accepting, then w is of type 1.
- If $w01^{l-|w0|}$ is accepting and $w10^{l-|w1|}$ is accepting, then w is of type 2.
- If $w01^{l-|w0|}$ is rejecting and $w10^{l-|w1|}$ is rejecting, then w is of type 3.

For each type of node we can show which of both the successors of the node in the query-tree yields a decision of the node.

Claim 2 *Let $w \sqsubseteq M^B(x)$.*

```
input x
w := ε
for i := 1 to q(|x|) do
  if f(x, w) is undefined then exit the loop end
  if w is of type 1
    then if f(x, w) ∈ B then accept else reject
  else if w is of type 2
          then if f(x, w) ∈ B then accept
                  else w := w0
          end
  else if w is of type 3
          then if f(x, w) ∉ B then reject
                  else w := w1
          end
  end
end (* of the for loop *)
accept ⟺ acc(x, w)
```

Figure 2: Oracle machine $\hat{M}$

---

1. *If w is of type 1, then $x \in A \Leftrightarrow f(x,w) \in B$.*
2. *If w is of type 2, then $f(x,w) \in B \Rightarrow x \in A$.*
3. *If w is of type 3, then $f(x,w) \notin B \Rightarrow x \notin A$.*

Proof Let $w \sqsubseteq M^B(x)$. Note that $w \subseteq_{M(x)} M^B(x)$ and $M^B(x) \subseteq_{M(x)} w$.

1. Assume w is of type 1. Then $acc(x, w01^{l-|w|-1}) \not\Leftrightarrow acc(x, w10^{l-|w|-1})$. First consider the case that $f(x,w) \in B$. Then $w10^{l-|w|-1} \subseteq_{M(x)} M^B(x)$, and $w01^{l-|w|-1} \subseteq_{M(x)} M^B(x)$ follows from M's monotonicity by Proposition 3.6. Since $acc(x, w01^{l-|w|-1})$ or $acc(x, w10^{l-|w|-1})$, it follows from the positiveness of M that $acc(x, M^B(x))$, and therefore $x \in A$. Second, consider $f(x,w) \notin B$. Then $M^B(x) \subseteq_{M(x)} w01^{l-|w|-1}$, and $M^B(x) \subseteq_{M(x)} w10^{l-|w|-1}$ follows from M's monotonicity by Proposition 3.6. Since either $\neg acc(x, w01^{l-|w|-1})$ or $\neg acc(x, w10^{l-|w|-1})$, it follows from the positiveness of M that $\neg acc(x, M^B(x))$, and therefore $x \notin A$.

   Concluding, we have $x \in A$ if and only if $f(x,w) \in B$.

2. Assume w is of type 2. Then $acc(x, w10^{l-|w|-1})$. If $f(x,w) \in B$, then $w1 \sqsubseteq M^B(x)$, and thus $w10^{l-|w|-1} \subseteq_{M(x)} M^B(x)$. Since M is positive, $acc(x, w10^{l-|w|-1})$ and Proposition 4.3 imply $acc(x, M^B(x))$. Hence $x \in A$.

3. Assume w is of type 3. Then $\neg acc(x, w01^{l-|w|-1})$. If $f(x,w) \notin B$, then $w0 \sqsubseteq M^B(x)$, and thus $M^B(x) \subseteq_{M(x)} w01^{l-|w|-1}$. Since M is positive, from $\neg acc(x, w01^{l-|w|-1})$ and Proposition 4.3 therefore follows $\neg acc(x, M^B(x))$. Hence $x \notin A$.

□

Now we can define a machine $\hat{M}$ (Figure 2) which collects the queries needed to decide $x \in A$. It searches through $M(x)$ constructing a path of queries using the above Claim.

It follows straightforward from the description of $\hat{M}$ that

input x
$w := \varepsilon$
for $i := 1$ **to** $q(|x|)$ **do**
  if $acc_{\hat{M}}(x, w0)$ is undefined **then** $w := w1$
  else if $acc_{\hat{M}}(x, w1)$ is undefined **then** $w := w0$
  end
end (* of the **for** loop *)
let $v_0, v_1, \ldots, v_{|w|-1}$ be the set of prefixes of w ordered w.r.t. inorder $\prec_{in}$
$T := \emptyset$
for $i := 0$ **to** $|w| - 1$ **do**
  if $f(x, v_i) \in B$ **then** $T := T \cup \{f(x, v_i)\}$
end
simulate $\hat{M}$ with oracle T on input x

Figure 3: Oracle machine N

Claim 3 *For every x and w, $acc_{\hat{M}}(x, w0)$ is defined or $acc_{\hat{M}}(x, w1)$ is defined.*

Claim 4 $L(\hat{M}, B) = A$.

Proof Take some x and consider $\hat{M}^B(x)$. At first, we show that $\hat{M}^B(x) \sqsubseteq M^B(x)$. We proceed by induction showing that every prefix of $\hat{M}^B(x)$ is a prefix of $M^B(x)$. For ε the case is clear. Assume as induction hypothesis that the statement holds for w, and consider the length $|w| + 1$ prefix w' of $\hat{M}^B(x)$. If w is of type 1, then $w = \hat{M}^B(x)$ and thus the statement holds. If w is of type 2 or 3, then clearly $w' \sqsubseteq M^B(x)$.

We now show that $acc(x, \hat{M}^B(x)) \Leftrightarrow x \in A$. If $\hat{M}^B(x) = M^B(x)$, then $acc_{\hat{M}}(x, \hat{M}^B(x)) \Leftrightarrow acc(x, M^B(x))$ by the definition of $\hat{M}$. Now assume that $\hat{M}^B(x)$ is a proper prefix of $M^B(x)$. Assume $acc(x, \hat{M}^B(x))$. Then $\hat{M}^B(x) = w1$. If w has type 1, then it follows from Claim 2 case 1 that $x \in A$, since $w1 \sqsubseteq M^B(x)$. If w has type 2, then we conclude $x \in A$ from Claim 2 case 2. w cannot be of type 3. Finally assume $\neg acc(x, \hat{M}^B(x))$. Then $\hat{M}^B(x) = w0$. If w has type 1, then it follows from Claim 2 case 1 that $x \notin A$, since $w0 \sqsubseteq M^B(x)$. w cannot be of type 2. If w has type 3, then we conclude $x \notin A$ from Claim 2 case 3. □

Claim 5 $\hat{M}$ *is positive.*

Proof Consider u and v such that $acc_{\hat{M}}(x, u)$ and $acc_{\hat{M}}(x, v)$ are defined. If $u \neq v$ and $u \subseteq_{\hat{M}(x)} v$, then $u = y0u'$ and $v = y1v'$. If y is of type 1, then $\neg acc(x, y0)$ and $acc(x, y1)$. If y is of type 2, then $acc(x, y1)$, and if y if of type 3, then $\neg acc(x, y0)$. Thus, $acc(x, u) \Rightarrow acc(x, v)$. □

Concluding we have that $\hat{M}$ is a positive t-in machine reducing A to B. By Claim 3, one can compute for every x all the queries in the query-tree $\hat{M}(x)$ in polynomial time, order them w.r.t. inorder, compute the intersection of the queries and the oracle, and then simulate $\hat{M}$ with that set. This is done by machine N given in Figure 3.

Since M is t-in w.r.t. B and the path of $\hat{M}$ is in fact a path of M, it follows that N is t-pre w.r.t. B. Since N finally simulates $\hat{M}$, N is positive by Claim 5. It is straightforward to see that $L(N, B) = L(\hat{M}, B)$, and thus N accepts A using oracle B. ■

In Proposition 3.11 we showed the relations between preorder monotonic and Hausdorff reducibilities. We now show that more than one monotonic query to the oracle is not needed if the

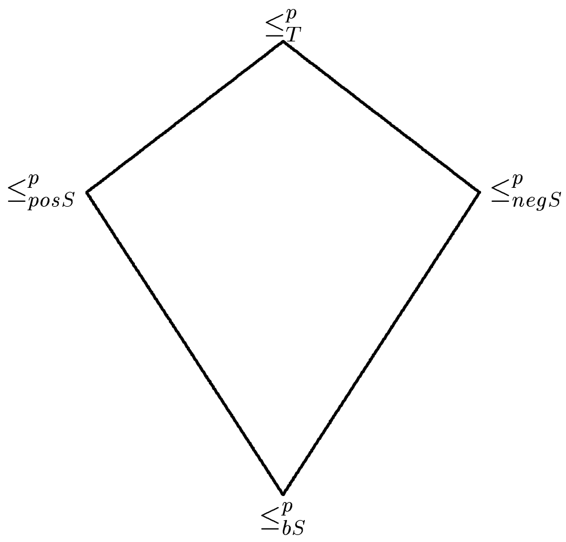

Figure 4: Positive monotonic reducibilities

oracle machine is preorder monotonic *and* positive. Informally, the qualitative restrictions of positiveness and monotonicity yield a quantitative restriction on the number of queries to the oracle. This intends us to argue that these properties are fundamental properties in the study of restricted oracle machines.

Proposition 4.5 *1. $A \leq_{posS}^p B$ via a positive oracle machine if and only if $A \leq_c^p B$.*

2. $A \leq_{negS}^p B$ via a positive oracle machine if and only if $A \leq_d^p B$.

3. $A \leq_{fbS}^p B$ via a positive oracle machine if and only if $A \leq_m^p B$.

The proof follows from Proposition 3.11 and results from [AKM96a]. Theorem 4.4 together with part 3 of Proposition 4.5 yields

Theorem 4.6 *$A \leq_{bS}^p B$ via a positive oracle machine if and only if $A \leq_m^p B$.*

Thus we get the following relations of positive monotonic reducibilities. The properness of the inclusions follows from [LLS75].

Under certain circumstances there is some trade-off between the strength of a monotonic reduction and positiveness. One can replace a fast binary search reduction to a set A by a positive path monotonic reduction to $\langle A, \overline{A} \rangle$, where for sets A, B, $\langle A, B \rangle$ denotes $\{\langle x, y \rangle \mid x \in A, y \in B\}$ for a polynomial-time computable and invertible pairing function $\langle \cdot, \cdot \rangle$ on strings.

Theorem 4.7 *For every set B, if $A \leq_{bS}^p B$, then $A \leq_{posS}^p \langle B, \overline{B} \langle$ via a positive oracle machine.*

Proofsketch By Proposition 3.11, it follows for $B \leq_{fbS}^p A$, that for a function $f \in$ FP and every x it holds that $x \in B \iff \bigvee_{1 \leq i \leq m} (y_{2i-1} \in A \wedge y_{2i} \notin A)$, for $f(x) = \langle y_1, \ldots, y_{2m} \rangle$. Thus, $x \in B \iff \{\langle y_{2i-1}, y_{2i} \rangle \mid 1 \leq i \leq m\} \cap \langle A, \overline{A} \rangle \neq \emptyset$, i.e. $B \in R_d^p(\langle A, \overline{A} \rangle)$. Using Proposition 4.5 the claim follows. ■

5 Monotonic reductions to NP and to sparse sets

We now compare the closures of NP resp. the classes of sparse or tally sets under monotonic reducibilities. We show that the power of a monotonic machine with an NP oracle does not depend on if it is path or tree monotonous; it only depends on the order. For sparse or tally oracles it behaves the other way round: all the tree monotonic reducibilities coincide and are weaker than the path monotonic reducibilities.

Monotonic reductions to sets in NP completely characterize the classes Θ_2^p and Δ_2^p of the Polynomial-Time Hierarchy. For reductions to NP there is no difference between tree and path monotonicity.

Theorem 5.1 *1.* $R^p_{posS}(\mathrm{NP}) = R^p_{fbS}(\mathrm{NP}) = \mathrm{P}^{\mathrm{NP}}[O(\log n)](= \Theta_2^p)$.

2. $R^p_T(\mathrm{NP}) = R^p_{bS}(\mathrm{NP}) = \mathrm{P}^{\mathrm{NP}}(= \Delta_2^p)$.

The proof of the first part of the Theorem is quite similar to a proof of Wagner [Wag90], who considered the closure of NP under a Hausdorff reducibility allowing an exponential number of queries. It is straightfroward to see that this type of reducibility coincides with binary search reducibility. The proof of the second part follows from Proposition 3.11 and a proof in [Wag87].

Positive and negative search are more powerful than binary search w.r.t. sparse oracles. Whereas it is still open, if the former differ on sparse oracles, we can show that binary search and fast binary search reductions are equivalent on sparse sets. To prove this, we show how to pick out from the exponentially big query tree of an oracle machine being monotonic w.r.t. some sparse set some polynomially bounded number of queries which suffice to simulate the computations of the whole tree. Moreover, for tally sets we show that binary search reductions can be replaced by positive reductions.

Theorem 5.2 *Let S be a sparse set. $A \leq^p_{bS} S$ if and only if $A \leq^p_{fbS} S$.*

Proof The implication from right to left follows from Proposition 3.10. For the other direction, let $A = L(M, S)$ for a sparse set S and a t-in oracle machine M (w.r.t. S) time bounded by the polynomial q. Let f denote the query-function associated to M. Since all labels of nodes in $M(x)$ being in S are in $S^{\leq q(|x|)}$, there exists a polynomial p bounding the number of elements in the query-tree, i.e. $|\bigcup_{|w| \leq q(|x|)} \{f(x, w)\} \cap S| \leq p(|x|)$. We will show, how from the exponentially big query-tree of any x we can choose polynomially many elements which suffice as oracle queries to decide x.

The proof idea is as follows. Fix some input x and consider the query-tree $M(x)$. Let $w = M^S(x)$ be the path consistent with S in $M(x)$. Since M is tree-inorder monotonic, one can see that w divides $M(x)$ into two parts: the part to the left of w consists of nodes labeled with queries in S, and the part to the right of w consists of nodes labeled with queries out of S. The labels of the nodes on w may be in S or out of S. Since S is sparse, the nodes left of w are labeled with at most $p(|x|)$ different queries. Consider now the k-th level of the query-tree $M(x)$, and let $w_{[k]}$ be the length k prefix of w. If the label $f(x, w_{[k]})$ of the node $w_{[k]}$ is in S, then (1) no label of a node at the k-th level to the right of $w_{[k]}$ equals $f(x, w_{[k]})$ – since they are all out of S – and (2) to the left of it there are at most $p(|x|)$ many different labels. Formally: let T_k be any subset of Σ^k, and consider the set U_k of all strings $v \in T_k$ fulfilling the two conditions

1. $f(x, v) \notin \{f(x, u) \mid u \in T_k, v \prec_{in} u\}$, and

input x
$T'_0 = \{\varepsilon\}$
for $i := 1$ **to** $q(|x|)$ **do**
  $T_i := \{w0, w1 \mid w \in T'_{i-1}\}$ (* expand to the i-th level *)
  (* compute $V \subseteq T_i$ of candidates containing $w_{[i]}$ with $f(x, w_{[i]}) \notin S$ *)
  $V := \emptyset$
  for all $v \in T_i$ in lexicographic order **do**
    if $f(x, v) \notin \{f(x, w) \mid w \in T_i, w < v\}$ and $|V| \le p(|x|)$ **then** $V := V \cup \{v\}$
  end
  (* compute $U \subseteq T_i$ of candidates containing $w_{[i]}$ with $f(x, w_{[i]}) \in S$*)
  $U := \emptyset$
  for all $v \in T_i$ in lexicographic order **do**
    if $f(x, v) \notin \{f(x, w) \mid w \in T_i, v < w\}$ and $|U| < p(|x|)$ **then** $U := U \cup \{v\}$
  end
  $T'_i := U \cup V$ (* T'_i contains $w_{[i]}$, if T_i contained it*)
end
(* compute a set T containing all the $f(x, w_{[i]})$ *)
$T := \emptyset$
for every $w \in \bigcup_i T'_i$ in order $\prec_{in}$ **do**
  if $f(x, w) \in S$ **then** $T := T \cup \{f(x, w)\}$ **end**
end
simulate M on input x using oracle T

Figure 5: Oracle machine M' using oracle S

2. $|\{f(x, u) \mid u \in T_k, u \prec_{in} v\}| < p(|x|)$.

If $f(x, w_{[k]}) \in S$, then $v = w_{[k]}$ fulfills both conditions. To prove it, assume that condition (1) is not fulfilled. Then for some $u \in T_k$, $w_{[k]} \prec_{in} u$, it holds that $f(x, u) \in S$. Let z be the longest common prefix of $w_{[k]}$ and u. Then $w_{[k]} \prec_{in} u$ implies $z0 \sqsubseteq w_{[k]}$ and $z1 \sqsubseteq u$. From the first follows $f(x, z) \notin S$ since $z0$ is a prefix of w, and the latter implies $z \prec_{in} u$ and thus $f(x, z) \in S$, a contradiction. The second condition follows straightforward by M's monotonicity and S's sparseness. Condition (2) also guarantees that $|U_k| \le p(|x|)$.

Similarly, let V_k be the set of all strings $v \in T_k$ fulfilling the conditions

3. $f(x, v) \notin \{f(x, u) \mid u \in T_k, u \prec_{in} v\}$, and
4. $|\{f(x, u) \mid u \in T_k, u \prec_{in} v\}| \le p(|x|)$.

As above we can show that $w_{[k]}$ fulfills both these properties, if $f(x, w_{[k]}) \notin S$. Also $|V_k| \le p(|x|)+1$. Thus, if $w_{[k]} \in T_k$, then $w_{[k]}$ fulfills either conditions (1) and (2), or conditions (3) and (4); i.e. $w_{[k]} \in U_k \cup V_k$ and $|U_k \cup V_k| \le 2p(|x|) + 1$.

Consider the Turing machine M' described in Figure 5 run on some input x. Let $w = M^S(x)$. Since $\varepsilon \sqsubseteq w$, and since $w_{[k+1]} \in \{w_{[k]}0, w_{[k]}1\}$, it follows from the above discussion that the set T is computed in polynomial-time, and T contains $w_{[0]}, w_{[1]}, \ldots, w_{[q(|x|)]}$. It follows that the path $M^S(x)$ equals path $M^{T \cap S}(x)$. Thus $x \in A \iff x \in L(M', S)$. From the inorder-monotonicity of M it follows that the sequence of queries asked to the oracle S by M' becomes answered monotonically; thus M' is tree-preorder monotonic w.r.t. S. ■

Corollary 5.3 $R^p_{fbS}(\mathit{SPARSE}) = R^p_{bS}(\mathit{SPARSE})$.

Since $R^p_c(\text{SPARSE}) \subseteq R^p_{posS}(\text{SPARSE})$ by Proposition 4.5, $R^p_{bS}(\text{SPARSE}) \subseteq R^p_d(\text{SPARSE})$ (cf. [AKM96a]), and $R^p_c(\text{SPARSE})$ and $R^p_d(\text{SPARSE})$ are incomparable [GW94], it follows

Proposition 5.4 $R^p_{bS}(\mathit{SPARSE}) \subset R^p_{posS}(\mathit{SPARSE})$.

Using SPARSE $\subseteq R^p_c(\text{TALLY})$ [BLS93], we conclude $R^p_c(\text{SPARSE}) \subseteq R^p_c(\text{co-SPARSE})$. Using Propositions 3.9 and 3.5 we get the inclusions

Proposition 5.5 $R^p_{posS}(\mathit{SPARSE}) \subseteq R^p_{negS}(\mathit{SPARSE}) \subseteq R^p_T(\mathit{SPARSE})$.

None of these inclusions is known to be proper.

co-SPARSE $\subseteq R^p_d(\text{SPARSE})$ was shown in [AHOW92] (follows also from the result above cited from [BLS93]). Since SPARSE is additionally closed under pairing, we get from Proposition 4.7 and 4.5

Proposition 5.6 *1.* $R^p_{bS}(\mathit{SPARSE}) \subseteq R^p_c(\text{co-}\mathit{SPARSE}) \cap R^p_d(\mathit{SPARSE})$

2. $R^p_{posS}(\mathit{SPARSE}) \subseteq R^p_c(R^p_d(\text{co-}\mathit{SPARSE})) \cap R^p_d(R^p_c(\mathit{SPARSE}))$

3. $R^p_{negS}(\mathit{SPARSE}) \subseteq R^p_c(R^p_d(\mathit{SPARSE})) \cap R^p_d(R^p_c(\text{co-}\mathit{SPARSE}))$

For monotonic reductions to tally sets, the situation is easier since TALLY is closed under complementation. Thus we obtain

Proposition 5.7 *1.* $R^p_{fbS}(\mathit{TALLY}) = R^p_{bS}(\mathit{TALLY})$.

2. $R^p_{posS}(\mathit{TALLY}) = R^p_{negS}(\mathit{TALLY}) \subseteq R^p_T(\mathit{TALLY})$.

3. $R^p_{bS}(\mathit{TALLY}) \subseteq R^p_c(\mathit{TALLY}) \cap R^p_d(\mathit{TALLY})$

Finally we can separate one-query reducibilities from monotonic reducibilities by tally and sparse sets.

Proposition 5.8 *1.* $R^p_{1\text{-}tt}(\mathit{SPARSE}) \subset R^p_{bS}(\mathit{SPARSE})$.

2. $R^p_{1\text{-}tt}(\mathit{TALLY}) \subset R^p_{bS}(\mathit{TALLY})$.

Figure 6 summarizes the above results. For each pair of classes connected by some line, the class which is closer to the top of the figure includes the "lower" class. Dotted lines indicate that it is not known if the inclusion is proper, straight lines indicate proper inclusions (cf. [Ko89, BLS93, GW94]).

6 Sparse hard sets

We get the following collapses of the Polynomial-Time Hierarchy under the assumption that sparse or tally hard sets under monotonic reductions exist for NP.

Theorem 6.1 *1. If* SAT $\leq^p_{bS}$ *or* $\leq^p_{posS}$ *reduces to a sparse set, then the Polynomial-Time Hierarchy collapses to* P^{NP}.

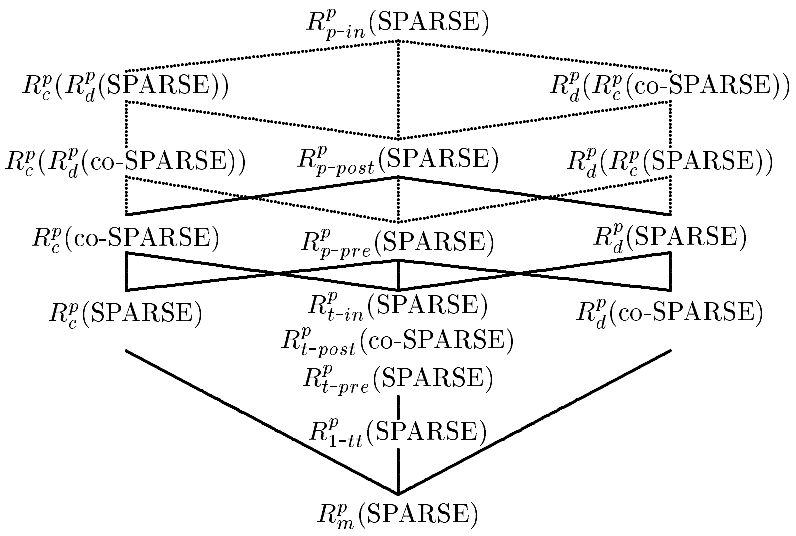

Figure 6: Monotonic reductions to sparse sets

---

2. If SAT $\leq^p_{bS}$ *reduces to a tally set, then* P = NP.

The first part of the theorem follows from Theorem 5.2 and results in [AKM96b], the second part from Proposition 5.7 and results in [AHH+93].

For ETIME we can prove that hard sparse sets do not exist w.r.t. positive search reducibility. This extends results of Watanabe [Wat87] who proved that ETIME cannot have sparse hard sets under conjunctive reducibility. Note that R^p_c(SPARSE) is properly contained in R^p_{posS}(SPARSE) (follows from [Ko89]).

Theorem 6.2 *No* $\leq^p_{posS}$ *or* $\leq^p_{bS}$ *hard set for* ETIME *is sparse.*

Proof If A is a sparse $\leq^p_{bS}$ hard set for ETIME, then by Theorem 5.2 and Proposition 3.9 it follows that A is also $\leq^p_{posS}$ hard for ETIME. We prove the Theorem by constructing a set C in ETIME which forces every set to which it $\leq^p_{posS}$-reduces to have a superpolynomial lower bound on its density. Let $M_0, M_1, \ldots$ be an enumeration of all machhines in 0-normal-form. C is defined to be the set accepted by the algorithm denoted in Figure 7.

Observing that for every input $\langle i, x\rangle$ the set variable Q contains at most $|Q| < 2^{|x|+1}$ elements, it is straightforward to see that $C \in$ ETIME.

Now consider a set A being $\leq^p_{posS}$ hard for ETIME. Then $C = L(M_k, A)$ for a p-pre oracle machine M_k time bounded by polynomial p. Let Q_x be the content of the set variable Q at the end of the execution of the above algorithm on input $\langle k, x\rangle$.

Claim 6 *For every x: $Q_x \subseteq A$.*

Proof The proof procedes by induction. For $x = \varepsilon$, $Q_\varepsilon = \emptyset \subseteq A$. Now assume as induction hypothesis that the claim holds up to a certain x and consider x', the successor of x. If $Q_{x'} = Q_x$, then $Q_{x'} \subseteq A$ by the induction hypothesis. If $Q_{x'} \neq Q_x$, then $Q_{x'} = Q_x \cup \{q\}$, where q is the first query not in Q_x asked by M_k on input $\langle k, x\rangle$, and M_k on that input can be simulated using not more than the allowed number of steps. Assume $q \notin A$. Then the induction hypothesis

```
input ⟨i,x⟩
Q := ∅
for all z < x in lexicographic order do
  if M_i on input ⟨i,z⟩ using oracle Q decides the input
  and computes a query q ∉ Q in at most 2^|⟨i,z⟩| steps
    then Q := Q ∪ {q} end
end (* of the for loop *)
if M_i^Q(⟨i,x⟩) rejects in at most 2^|⟨i,x⟩| steps of M_i
  then accept
  else reject
end
```

Figure 7: Algorithm accepting C

---

and the properties of p-pre machines imply $\langle k,x\rangle \in L(M_k,Q_x) \iff \langle k,x\rangle \in L(M_k,A)$. But $\langle k,x\rangle \in C \iff \langle k,x\rangle \notin L(M_k,Q_x)$ as defined in the above algorithm contradicts $C = L(M_k,A)$. $\square$

Since M_k runs in polynomial-time, there exists a string z_0 such that for all strings z lexicographically greater than z_0 and all oracles Q it holds that M_k on input $\langle k,z\rangle$ using oracle Q runs in time less than $2^{|\langle k,z\rangle|}$. For all these z holds $\langle k,z\rangle \in C \iff \langle k,z\rangle \notin L(M_k,Q_z)$, and since $C = L(M_k,A)$ it follows that M_k on input $\langle k,z\rangle$ using oracle Q_z ask a query not in Q_z. Thus we get for all these z that Q_z is a proper subset of $Q_{z'}$, where z' is the successor of z. We therefore get that there exists a constant $c \geq 0$ such that for all x: $|Q_x| \geq 2^{|x|} - 2^c$. Since $|A^{\leq p(|x|)}| \geq |Q_x|$ by Claim 6, A cannot be sparse. $\blacksquare$

As a consequence we obtain

Theorem 6.3 *If* SAT $\leq^p_{posS}$ *or* $\leq^p_{bS}$ *reduces to a sparse set, then the Polynomial-Time Hierarchy not equals* ETIME*.*

References

[AHH+93] V. Arvind, Y. Han, L. Hemachandra, J. Köbler, A. Lozano, M. Mundhenk, M. Ogiwara, U. Schöning, R. Silvestri, and T. Thierauf. Reductions to sets of low information content. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory, Current Research*, pages 1–45. Cambridge University Press, 1993.

[AHOW92] E. Allender, L. Hemachandra, M. Ogiwara, and O. Watanabe. Relating equivalence and reducibility to sparse sets. *SIAM Journal on Computing*, 21(3):529–539, 1992.

[AKM96a] V. Arvind, J. Köbler, and M. Mundhenk. Monotonous and randomized reductions to sparse sets. *RAIRO Journal on Computing*, 1996. To appear.

[AKM96b] V. Arvind, J. Köbler, and M. Mundhenk. Upper bounds for the complexity of sparse and tally descriptions. *Mathematical Systems Theory*, 29:63–94, 1996.

[BDG90] J.L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I/II*. EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1988/1990.

[BLS93] H. Buhrman, L. Longpré, and E. Spaan. SPARSE reduces conjunctively to TALLY. In *Proc. 8th Structure in Complexity Theory Conference*, pages 208–214. IEEE, 1993.

[GW94] R. Gavaldà and O. Watanabe. On the computational complexity of small descriptions. *SIAM Journal on Computing*, 1994. To appear.

[Har83] J. Hartmanis. Generalized Kolmogorov complexity and the structure of feasible computations. In *Proc. 24th IEEE Symp. on Foundations of Computer Science*, pages 439–445, 1983.

[HOW92] L. Hemachandra, M. Ogiwara, and O. Watanabe. How hard are sparse sets? In *Proc. 7th Structure in Complexity Theory Conference*, 1992.

[Ko89] K. Ko. Distinguishing conjunctive and disjunctive reducibilities by sparse sets. *Information and Computation*, 81(1):62–87, 1989.

[LLS75] R.E. Ladner, N.A. Lynch, and A.L. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

[Sel82] A.L. Selman. Reductions on NP and p-selective sets. *Theoretical Computer Science*, 19:287–304, 1982.

[Wag87] K.W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51:53–80, 1987.

[Wag90] K.W. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

[Wat87] O. Watanabe. Polynomial time reducibility to a set of small density. In *Proc. 1987 Structure in Complexity Theory Conference*, pages 138–146. Lecture Notes in Computer Science #223, Springer-Verlag, 1987.