Electronic Colloquium on Computational Complexity - Reports Series 1996 - available via:
ECCC
TR96-011
FTP: ftp.eccc.uni-trier.de:/pub/eccc/
WWW: http://www.eccc.uni-trier.de/eccc/
Email: ftpmail@ftp.eccc.uni-trier.de with subject 'help eccc'

Sharply Bounded Alternation within P

Stephen A. Bloch[*§] Jonathan F. Buss[†] Judy Goldsmith[‡§]

October 31, 1995

Abstract

We define the *sharply bounded hierarchy*, $SBH(QL)$, a hierarchy of classes within P, using quasilinear-time computation and quantification over values of length $\log n$. It generalizes the limited nondeterminism hierarchy introduced by Buss and Goldsmith, while retaining the invariance properties. The new hierarchy has several alternative characterizations.

We define both $SBH(QL)$ and its corresponding hierarchy of function classes, $FSBH(QL)$, and present a variety of problems in these classes, including $\leq_m^{ql}$-complete problems for each class in $SBH(QL)$. We discuss the structure of the hierarchy, and show that certain simple structural conditions on it would imply $P \neq PSPACE$. We present characterizations of $SBH(QL)$ relations based on alternating Turing machines and on first-order definability, as well as recursion-theoretic characterizations of function classes corresponding to $SBH(QL)$.

AMS(MOS) Subject Classifications: 68Q15, 68Q05, 03D55, 03C13.

Key Words: nondeterminism, quasilinear time, alternation, polynomial time, PSPACE, finite models, function algebras, computational complexity.

*Department of Mathematics and Computer Science, Adelphi University, Garden City, NY 11530.

†Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. Work supported in part by grants from the Natural Sciences and Engineering Research Council (NSERC) of Canada and the Information Technology Research Centre (an Ontario Centre of Excellence).

‡Computer Science Department, University of Kentucky, Lexington, KY, 40506–0027. Supported in part by National Science Foundation grant CCR-9315354.

§The first and third authors were supported in part by NSERC operating grant OGP0121527 while at the University of Manitoba.

1 Introduction

Since the class P of polynomial-time-recognizable sets was formally introduced in the 1960's [8, 17, 22, 37] many complexity theorists have considered it a good formalization of the notion of feasible computation [19]. Arguments for this view include the robustness of P—it is largely independent of machine model, and is closed under composition and Boolean operations—and its many machine-independent, logical characterizations (*e.g.* [7, 13, 15, 28, 29, 31, 24, 33, 34, 40, 41, 45, 51]). Unfortunately, general polynomials are too large to be considered feasible. A program that runs in time $\Theta(n^{17})$ has little practical value.

Many classes within P have been studied, including those defined by circuit complexity, or those defined on Turing machines or RAM models by tightly bounding the polynomial time bound. But many of these classes lack robustness. For example, quadratic time is not closed under composition, and linear time, while closed under composition, is highly sensitive to details of the computational model such as number of tapes. Furthermore, linear time does not seem to suffice for such basic problems as multiplication and sorting. We draw particular attention to sorting, since many simple, natural problems can sort as a by-product of their operations, and sorting itself is an extremely useful technique for problems that do not explicitly require it. (For complexity-theoretic examples, see [12, 23, 30, 49] and section 2.2 below.)

Recently, several approaches have been considered for complexity classes slightly larger than linear time. Schnorr [49] and Gurevich and Shelah [30] have considered *quasilinear* and *nearly linear* time classes (QL and NLT), defined as $DTIME(\mathcal{O}(n\log^{\mathcal{O}(1)} n))$ on multi-tape Turing machines and on random-access machines, respectively. These classes are closed under composition, and are relatively machine-independent—QL is invariant under obliviousness assumptions [23] and the number of tapes of a multi-tape TM, while NQL is equivalent under five different random-access models. It is an open question whether $QL = NLT$. In the current paper, we focus primarily on the Turing-machine model (QL). We denote $\mathcal{O}(n\log^{\mathcal{O}(1)} n))$ by $Q(n)$.

If one takes quasilinear time (of any flavor) as a definition of efficient computation, it is natural to ask what problems have no efficient solutions. What makes a polynomial-time algorithm inefficient? Given a polynomial-time decision algorithm, we may distinguish between two possible sources of complexity: is it performing an inherently time-consuming test, or is it merely testing a large number of possible candidates for problem solution? For example, a naive algorithm to determine whether a graph contains a k-clique takes time $\mathcal{O}(n^k)$ (or slightly more, depending on the precise model). Most of the time cost comes from the loop over all possible k-vertex subgraphs, a cost we could eliminate with $k\log n$ bits of nondeterminism. This observation was developed by Buss and Goldsmith [12], who extended QL with *sharply bounded* existential quantifiers (of the form "$\exists y |y| \leq \log |x|$"). They developed a structure theory for these classes and showed that if the hierarchy contains all of P then $P \neq NP$.

Many natural polynomial-time problems seem to require more than just sharply bounded existential quantifiers and quasilinear-time predicates. In this paper we examine the implications of mixing sharply bounded universal ($\tilde{\forall}$) and existential ($\tilde{\exists}$) quantification over a quasilinear-time test. The resulting hierarchy of complexity classes, extending the Buss-Goldsmith hierarchy but still contained within P, we call the Sharply Bounded Hierarchy over Quasilinear Time and denote by $SBH(QL)$ or (when the time bound is clear from context) SBH.

The Sharply Bounded Hierarchy is partly inspired by Wrathall's *linear-time hierarchy* (LTH) [52]. Like her, we define a hierarchy based on bounded quantifiers over a small time-bounded class within P, and then give other characterizations of the hierarchy and the classes therein. There are several important differences, however. By allowing quasilinear-time predicates, we give our classes more machine independence. By sharply bounding our quantifiers, we keep our entire hierarchy within P. The sharp bounds apparently add to the intricacy of the hierarchy, because strings of the same quantifier do not obviously collapse (in fact, we give oracles relative to which they separate). Thus, the $SBH(QL)$ does not mirror the polynomial-time hierarchy as neatly as does the LTH. (The "quasilinear-time hierarchy" of Naik, Regan and Sivakumar [43] differs from $SBH(QL)$ by allowing larger quantifiers and mimics LTH more closely.)

Results of Grädel and McColm, showing that a quantifier-defined hierarchy within NL is proper [26], give us hope of showing that the $SBH(QL)$ does not collapse. Their classes are defined by alternations of $\exists^*$ and $\forall^*$ quantifiers, which model the transitive closure operator and its dual in Immerman's characterization of NL [34]. This connection also motivates our interest in logical characterizations of the $SBH(QL)$.

In the next section, we give definitions of the relevant complexity classes. We then present a variety of problems in the $SBH(QL)$, define a notion of completeness, and show that some of the problems considered yield quasilinear-complete sets for classes in the hierarchy. We also define function classes based on the sharply bounded hierarchies over linear and quasilinear time, and show that these function classes are appropriate generalizations of the relation classes. Later sections discuss the structure of the $SBH(QL)$, including upward collapse theorems, and show that if $SBH(QL)$ contains a non-trivial amount of P, then $P \neq PSPACE$. We present characterizations of both $SBH(QL)$ and *SBH(linear)* in terms of a variant of alternating Turing machines, and in terms of first-order definability. We also present machine-independent characterizations of both the relation and the function classes of both $SBH(QL)$ and *SBH(linear)*, using recursion-theoretic methods. We conclude with some open problems.

2 Definitions and Example Problems

2.1 The Basic Classes

Schnorr [49] introduced the classes QL and NQL, where $QL = \bigcup_k DTIME(n \log^k n)$ and $NQL = \bigcup_k NTIME(n \log^k n)$, with respect to multi-tape Turing machines. We will use the notation of Buss and Goldsmith [12] and refer to these classes as P_1 and NP_1, respectively. Similarly, we define $P_r = \bigcup_k DTIME(n^r \log^k n)$.

We define classes in the Sharply Bounded Hierarchy over Quasilinear Time, $SBH(QL)$, by means of *sharply bounded quantifiers* over quasilinear-time predicates. A sharply bounded quantifier is one of the form $\exists y\ |y| \leq \log|x|$ or $\forall y\ |y| \leq \log|x|$. We write these as $\tilde{\exists}$ and $\tilde{\forall}$. If S is a string of sharply bounded quantifiers (such as $\tilde{\forall}\tilde{\exists}\tilde{\forall}\tilde{\forall}$), then $A \in SP_1$ iff there is a P_1 relation R so that $x \in A \Leftrightarrow S\vec{y}[R(x,\vec{y})]$. The notation $SBH(QL)$, or SBH for short, can mean either the collection of classes $\{SP_1 : S \in \{\tilde{\exists},\tilde{\forall}\}^*\}$ or the single class which is the union of them; which we mean in a particular instance should be clear from context.

Note that the strings of quantifiers in the above may contain repetitions. Unlike the classes defined by *bounded* quantifiers, there is no evidence that iterations of a quantifier ($\tilde{\forall}\tilde{\forall}$, for example) can be collapsed to a single quantifier. Note also that, if S is a string of r sharply bounded quantifiers, then $SP_1 \subseteq P_{r+1}$. Hence $SBH \subseteq P \cap QLSPACE$.

We also consider function classes in Sections 2.4 and 5. We denote the functions btt-reducible in quasilinear time to SBH relations by $FQL^{SBH(QL)[1]}$. Furthermore, we consider analogous classes with quasilinear time replaced by linear time, in both the relation class and the reduction. These classes we denote by $FL^{SBH(linear)}$ and $FL^{SBH(linear)[1]}$.

2.2 Languages in the Classes

Buss and Goldsmith [12] have given various examples of problems in the classes $\tilde{\exists}^k P_1$. Many of the problems they considered are fixed-parameter variants of NP-complete problems. For instance, they consider CSAT(k), a restriction of the circuit satisfiability problem. A Boolean circuit C with $k \log n$ inputs and n total gates is in CSAT(k) if there is an input string that causes it to output a 1. CSAT(0), the Circuit Value Problem, was shown to be in P_1 by Pippenger [47]. Buss and Goldsmith showed that CSAT(k) is complete for $\tilde{\exists}^k P_1$ (for all k); we observe that there are variants of this problem in all classes of SBH. For example, a Boolean circuit C is in $\tilde{\exists}\tilde{\forall}$CSAT if it has n gates, $2\log n$ of which are input gates, and there is a string x_1 of length $\log n$ such that for any string x_2 of length $\log n$, C on input $x_1 x_2$ outputs a 1.

The following examples include some P-complete problems [27]. Some are also complete for classes in the SBH, but we do not immediately get $P \subseteq SBH$ because they are P-complete under apparently different reductions, such as $LOGSPACE$ or NC^1. We also mention a P-complete problem that seems inherently sequential, and appears not to be in $SBH(QL)$.

MATRIX MULTIPLICATION TESTING is in $\tilde{\forall}P_1$: given three $\sqrt{n} \times \sqrt{n}$ matrices A, B, and C, is $A \cdot B = C$?

TOTAL UNARY ORDERED GENERATOR is complete for $\tilde{\forall}P_1$: given an ordered context-free grammar G and a set T of terminal symbols, does G generate a string in τ^* for each $\tau \in T$? (A grammar is *ordered* if, for each nonterminal A in G, each occurrence of A in the lefthand side of a production occurs before any occurrence of A on the righthand side of a production.) Buss and Goldsmith [12] show that the set of ordered unary generators is complete for $\tilde{\exists}P_1$; this variant follows easily from their proof.

The following five examples come from Fellows and Downey's work on fixed-parameter intractibility [21].

PERFECT CODE(k) is in $\tilde{\exists}^{k-1}P_1$: given a graph G, does G have a k-element perfect code; i.e., is there a set V' of k vertices such that for all vertices $v \in G$, v has a unique neighbor in V'? The algorithm nondeterministically chooses all but one of the vertices; the last must have a neighborhood equal to the vertices not in the neighborhoods of the chosen vertices.

WEIGHT(k) SAT is in $\tilde{\exists}^kP_1$: given a Boolean formula F in CNF, is there a weight-k satisfying assignment, i.e., a satisfying truth assignment with exactly k 1's?

WEIGHT(k) EXACT SAT is in $\tilde{\exists}^kP_1$: given a Boolean formula F in CNF, is there a weight-k satisfying assignment that assigns exactly one "true" to each clause?

UNIQUE WEIGHT(k) SAT is in $\tilde{\exists}^k\tilde{\forall}^kP_1$: given a Boolean formula F in CNF, is there exactly one weight-k satisfying assignment for F?

LEXICOGRAPHICALLY FIRST CSAT(k) is in $\tilde{\exists}^{k-1}\tilde{\forall}^kP_1$: given a topologically sorted circuit description of size n, with $k \log n$ input gates, and a string y of length $\log n$, does the lexicographically first satisfying assignment to the circuit include y as one of the k blocks of size $\log n$?

The $AW[*]$-complete problems SHORT GEOGRAPHY and SHORT NODE KAYLES [1] give examples of alternating quantifiers. Both model two-player games on a graph. In the geography game on a graph G, the players alternately pick nodes of a simple path; the first player unable to extend the path to a new node loses. A graph is in SHORT GEOGRAPHY(k) if the player moving first has a winning strategy of k moves or less. One can easily show that SHORT GEOGRAPHY($2k+1$) is in $(\tilde{\exists}\tilde{\forall})^kP_1$. Likewise, SHORT NODE KAYLES($2k$) is in $(\tilde{\exists}\tilde{\forall})^kP_1$. We refer the reader to Abrahamson, *et al.* [1] for more details on these problems and the class $AW[*]$.

In many cases, quantifiers in a problem specification can be eliminated by enhancing the quasilinear-time test. One such example is testing the Vapnik-Chervonenkis dimension of a set, which is defined as a Σ_3 property.

V-C DIMENSION(k) [50, 42]: given a family of subsets $\mathcal{C} = \{C_1, \dots, C_r\}$, drawn from a finite universe $U = \{x_1, \dots, x_m\}$, is the Vapnik-Chervonenkis (V-C) dimension of $\mathcal{C}$ at least k? That is, is there a set $S \subseteq U$, with $|S| = k$, such that for all subsets T of S, there is a $C_i \in \mathcal{C}$ such that $S \cap C_i = T$?

We assume that each subset C_i is specified in the input by an m-bit binary string, so that the total input size is $n = rm$. The V-C dimension of family $\mathcal{C}$ can be at most $\log r$.

A simple algorithm puts V-C DIMENSION(k) in $\tilde{\exists}^k P_1$: given $\mathcal{C}$ (of size rm), guess S using $k \log m \leq k \log n$ bits. Compute for each $i \leq r$ the set $D_i = C_i \cap S$. Sort the strings representing D_i and verify that there are 2^k distinct values.

To show that V-C DIMENSION(k) $\in \tilde{\exists}^{k-1} P_1$, we guess a set R of $k-1$ elements, and determine in quasilinear time if there is some x such that $S = R \cup \{x\}$ meets the conditions of the problem. In order for such an x to exist, there must exist 2^{k-1} distinct sets of the form $R \cap C_i$. In addition, each such intersection must be realizable in two (or more) different ways; there must be two subsets C_i and C_j of $\mathcal{C}$ such that $R \cap C_i = R \cap C_j$ and x is a member of exactly one of C_i and C_j.

To determine the subsets of $\mathcal{C}$, on a new tape write for each i a $2m$-bit string, the first m bits representing $D_i = C_i \cap R$ and the second m representing C_i. Sort these r strings by their first halves, using time $Q(n)$. Each group of strings with identical first halves represents all ways of forming one intersection set. For each group p, compute a string s_p of length m with a 1 in position j if x_j appears in some but not all the sets C_i in group p. The required time for a group is proportional to the number of bits in the group strings. Finally, in time $O(m2^{k-1}) = O(mr)$, compute the bitwise AND of these s_p. Each 1 in the final string indicates a possible x.

The entire algorithm uses time quasilinear in the size of $\mathcal{C}$.

Some P-complete problems do not appear to be in SBH. One example is the fundamental problem UNIT RESOLUTION [36], which is clearly in P_2, but not apparently in quasilinear time, even with the help of sharply bounded quantifiers.

2.3 Complete Sets

The notion of quasilinear-time computation can also be applied to transducer Turing machines. Since all of the classes in *SBH* are closed under quasilinear-time functions, it is natural to define a notion of quasilinear-time, many-one complete sets for these classes [12, 30, 49]. Schnorr [49] showed that SAT is $\leq_m^{ql}$-complete for nondeterministic quasilinear time (NQL, a.k.a. NP_1). The same techniques apply to alternating quantifiers.

Theorem 1 QBF, *the set of true quantified Boolean formulas, is $\leq_m^{ql}$-hard for any class in SBH.*

Buss and Goldsmith gave various problems that are complete for classes of the form $\tilde{\exists}^k P_1$, and Cai and Chen [14] have recently shown that the "weight $\leq k$ CSAT" problem is also complete for $\tilde{\exists}^k P_1$. Variants of some of these problems can be shown to be $\leq_m^{ql}$-complete for classes higher in *SBH*. For all S in $\{\tilde{\exists}, \tilde{\forall}\}^*$, we define the following two problems.

S-CSAT: Given a Boolean circuit C with $|S| \log |C|$ inputs, does $S\vec{x}[C$ on input $\vec{x}$ outputs a 1] hold?

S-ShortSat: Given a Boolean formula B with at least $|S| \log |C|$ variables, does $S\vec{x}$[the assignment of $\vec{x}$ to the first $|S| \log |C|$ variables causes B to unravel] hold? (A partial assignment of truth values to variables in the formula may induce values on other variables if the formula is to be true: if B is in 3CNF, and $(a_1 \vee a_2 \vee a_3)$ is a clause, and all but one of the a_i's are false, then the remaining variable is induced to assume the value true; if one variable a_i is true, then the clause is true, and no additional assignment is induced. If this procedure can be applied iteratively until all clauses evaluate to true, then we say the original partial assignment caused the formula to *unravel*.)

Theorem 2 *For all S in $\{\tilde{\exists}, \tilde{\forall}\}^*$, S-CSAT and S-ShortSat are complete for SP_1.*

2.4 Function Classes

One can define function classes that correspond in a natural way to the levels of the *SBH*, or to the union of the whole hierarchy. In Section 2.1, we defined classes that correspond to the union of the hierarchies (over *QL* and *linear*). We demonstrate here that the recursion-theoretic characterizations give very closely related classes. The theorems in Section 5 reinforce this.

Let *FL* and *FQL* denote respectively the functions computable in linear and quasilinear deterministic time. Let $B(\cdot)$ denote the closure of a class under Boolean operations.

Theorem 3 *For any* $S \in \{\tilde{\exists}, \tilde{\forall}\}^*$, *a* 0/1*-valued function is in* $FQL^{SP_1[1]}$ *if and only if it is the characteristic function of a* $B(SP_1)$ *relation.*

Since $B(SBH) = SBH$, it follows immediately that

Corollary 4 *A* 0/1*-valued function is in* $FQL^{SBH(QL)[1]}$ *if and only if it is the characteristic function of an SBH relation.*

Proof: Suppose $f \in FQL^{SP_1[1]}$ and is 0/1-valued. Then f can be computed by generating some constant number c of quasilinear-length queries to SP_1, and then doing *QL* work on the results. By standard techniques, c adaptive queries can be simulated by 2^c nonadaptive queries. The work after the queries can equally well be done before them, at the cost of a 2^{2^c} factor of time to handle each possible outcome of the queries. Thus f can be computed by doing *QL* work, asking constantly many queries in parallel, and computing a Boolean function of the results; *i.e.*, f is the characteristic function of a $B(SP_1)$ relation.

Conversely, suppose f is the characteristic function of a $B(SP_1)$ relation; *i.e.*, f is Boolean function applied to constantly many SP_1 queries, each generated in *QL* time. It is then straightforward to compute f in $FQL^{SP_1[1]}$.

The proof for *SBH* is similar. ∎

An analogous theorem and corollary hold with quasilinear time replaced by linear time throughout.

3 The Structure of the Classes

Like most hierarchies of complexity classes, *SBH* exhibits upward collapse/downward separation. Some of the collapses presented in this section concern the compression of like quantifiers (from $\tilde{\exists}\tilde{\exists}$ to $\tilde{\exists}$, for instance), as well as (or instead of) the quantifier-swapping we expect of collapsing results.

Theorem 5 *If C is a class in SBH, and $\tilde{\exists} C \subseteq C$, then for all $m \geq 0$, $\tilde{\exists}^m C \subseteq C$.*

The proof is a simple induction on m.

Theorem 6 *If C is a class in SBH, closed under complement, and $k > k'$, and $\tilde{\exists}^k C \subseteq \tilde{\forall}^{k'} C$, then for all $m > k$, $\tilde{\exists}^m C \subseteq \tilde{\exists}^{k'} C$.*

Proof: If $\tilde{\exists}^k C \subseteq \tilde{\forall}^{k'} C$, then dually $\tilde{\forall}^k \overline{C} \subseteq \tilde{\exists}^{k'} \overline{C}$. Since C is closed under complement by assumption, all four classes are equal. The result follows by Theorem 5. ∎

Note that Theorem 6 and its proof do not allow the case $k = k'$; that is, we derive no consequences from the closure of $\tilde{\exists}^k C$ under complement. Unlike the case of unbounded quantifiers, closure under complement does not translate easily to a larger number of quantifiers.

If the classes $\tilde{\exists}^k P_1$ are all closed under complement, then *SBH* is contained in the Buss-Goldsmith existential hierarchy. This would certainly simplify *SBH*, but would it simplify it right down to *QL*? In the presence of an oracle, not necessarily. The following result is analogous to recent work on the β hierarchy [4]. We assume that a query to the oracle erases the query string from the tape.

Theorem 7 *There is an oracle A such that for all $k \geq 0$ and $h > 0$,*

- $\tilde{\exists}^k P_h^A \neq \tilde{\exists}^{k+1} P_h^A$ *and*
- $\tilde{\exists}^k P_h^A = \tilde{\forall}^k P_h^A$.

Proof: Let $\{E_{h,i,k}^{()}\}_{h,i,k}$ enumerate the $\tilde{\exists}^k P_h^{()}$ machines. Assume the (h, i, k)-th machine is explicitly clocked to run for at most $p_{h,i}(n) = n^h \log^i n$ steps for all input lengths $n > 1$. We design the oracle A to satisfy the following conditions.

Coding: For every h, i, k, and x, machine $E_{h,i,k}^A$ accepts x if and only if A does not contain a code for $E_{h,i,k}^A(x)$, namely a string of the form $0^{p_{h,i}(|x|)\log(p_{h,i}(|x|))}\#h\#i\#k\#x\#y$ with $|y| = k \log |x|$.

Diagonalization: For every h, i and k, the language $D^A_{h,k+1} = \{ x \mid (\tilde{\exists}^{k+1} y) 1^{|x|^h} xy \in A \}$ is not accepted by $E^A_{h,i,k}$.

Let $\langle \cdot, \cdot, \cdot \rangle$ be a bijection from triples of positive integers to positive integers. We will write $E^A_{\langle h,i,k \rangle}$ to mean $E^A_{h,i,k}$.

The only important strings are those mentioned above—codes for a computation and witnesses for a diagonalization language. We fix all other strings out of A. We consider the important strings in stages; at stage s we determine oracle membership for all strings of length s, and perhaps some longer strings. Set $s = d = b = 1$; s is the stage number, E^A_d is the next machine we will diagonalize against, and b is a lower bound on the lengths of strings eligible for diagonalization.

Stage s:

Stage s begins by fixing the codes of length s in or out of A. For each w of length s, if w is a code for some $E^A_{h,i,k}(x)$, then $E^A_{h,i,k}(x)$ queries only strings of lengths less than s. Thus, at the beginning of stage s, the value of $E^A_{h,i,k}(x)$ is already determined. If $A(w)$ was fixed in a previous stage, it has a suitable value; otherwise, set $A(w) = 1$ if and only if $E^A_{h,i,k}(x) = 0$.

Let $l = s^{1/h}$. If l is an integer such that $l > b$ and $\log^i l < l$, then do the following diagonalization. Otherwise, proceed immediately to the next stage by incrementing s.

The current diagonalization condition is $d = \langle h, i, k \rangle$. We wish to guarantee that $E^A_{h,i,k}(1^l) \neq D^A_{h,k+1}(1^l)$. Note that $D^A_{h,k+1}(1^l)$ has l^{k+1} witness strings, each of length greater than l^h. We describe how to simulate the computation of $E^A_{h,i,k}(1^l)$ (and other computations in the process) to meet the following condition: over all queries of witness strings during the simulation, the total number of bits is less than l^{h+k+1}. Hence some witness string for $D^A_{h,k+1}(1^l)$ is not queried during the simulation and remains available to complete the diagonalizaton.

The computation $E^A_{h,i,k}(1^l)$ has l^k branches, each running for at most $l^h \log^i l$ steps. Thus it can directly query at most $l^{k+h} \log^i l$ bits. Furthermore, each query has length at most $l^h \log^i l$. If the machine queries a witness string for $D^A_{h,k+1}(1^l)$, we restrain that string from A; this direct computation won't force $D^A_{h,k+1}(1^l) = 0$. However, $E^A_{h,i,k}(1^l)$ may query codes whose membership in A is not yet determined; we must fix them suitably in order to complete the diagonalization. In most cases, the queried string is one of many unfixed codes for the corresponding computation, and we simply fix the string out of A without harm. If, however, the queried string is the last remaining code for its computation, we must ensure that it codes the computation correctly. We therefore simulate the computation to determine its acceptance or rejection; when we know the correct answer, we continue the previous computation. The new computation may itself query codes, creating a recursive tree of codes and their computations. We will term the codes which require recursion *crucial* codes. For a crucial code p, we denote by C_p the computation coded by p.

To count the number of bits queried in the entire recursion, we consider a directed graph whose nodes are the crucial codes. The graph has an edge from p to q if the computation C_p queries any code of C_q (crucial or not). The graph has no cycles, and the longest path has length at most i.

Suppose that crucial code q has computation $C_q = E^A_{s,t,u}(x)$. C_q has $|x|^u$ codes, each of length more than $s|x|^s \log^{t+1}|x|$. Thus the predecessors of q used $s|x|^{s+u}\log^{t+1}|x|$ bits querying codes of C_q. Computation C_q itself has $|x|^u$ branches of length at most $|x|^s \log^t |x|$ and thus queries strings totalling at most $|x|^{s+u}\log^t |x|$ bits.

Since creating a crucial query requires more query bits than the coded computation itself queries, the number of bits in queries of witness strings is at most the original number $l^{k+h}\log^i l$. We only do a diagonalization when $\log^i l < l$; hence some witness string for $D^A_{h,k+1}(1^l)$ remains unfixed after the simulation of $E^A_{h,i,k}(1^l)$. We fix that string in A if $E^A_{h,i,k}(1^l) = 0$ and out otherwise.

Finally, set b to the length of the longest witness string fixed during the simulation, and increment d and s.

End of stage s.

The union of all the stages defines a consistent oracle A that satifies all the required conditions. ∎

Placing the *SBH* with respect to deterministic computation would have significant consequences.

Theorem 8 *If for all l, $SBH \not\subseteq P_l$, then $P \neq PSPACE$.*

Proof: Suppose $P = PSPACE$. Then QBF $\in P$, and hence QBF $\in P_l$ for some $l \geq 1$. But then $SBH \subseteq P_l$. ∎

Theorem 9 *If for some rational $r > 1$, $P_r \subseteq SBH$, then $P \neq PSPACE$.*

Proof: Since P_r has a complete set [12], the inclusion $P_r \subseteq SBH$ implies that $P_r \subseteq SP_1$, for some string S of sharply bounded quantifiers.

Lemma 10 *If $P_r \subseteq SP_1$ for some string S of sharply-bounded quantifiers and some $r > 1$, then $P_{r^2} \subseteq SSP_1$.*

Let $A \in P_{r^2}$ and $B = \{\, y \mid y = x10^{\lfloor |x|^r \rfloor - |x| - 1} \wedge x \in A \,\}$. Then $B \in P_r$, and thus in SP_1 by assumption. To decide $A(x)$, in time $Q(n^r)$, write $y = x10^{\lfloor |x| \rfloor^r - |x| - 1}$, and decide (in time $Q(|x|^r)$, with quantifiers S), if $y \in B$. In other words, $A \in SP_r \subseteq SSP_1$.

Iterating the lemma, we get that $P_r \subseteq SBH$ implies $P \subseteq SBH \subseteq SPACE(Q(n))$ and hence $P \neq PSPACE$. ∎

4 Characterizations of the Classes

Many people have argued that the class P is *natural,* based on the great variety of apparently different characterizations of the class. We present several characterizations of the *SBH* here. We begin with what is perhaps the most obvious characterization, namely by alternating Turing machines. We then show that, like P, the *SBH* can be characterized in terms of first order definability, or in terms of function algebras. Finally, we consider *functions* as well as relations within the *SBH*.

The definition of *SBH* in section 2.1 uses the apparently arbitrary bound of $\log n$ on the length of a quantified variable. In this section and the next, we show several equivalent definitions that do not depend on an arbitrary bound on nondeterminism.

4.1 An Alternating Machine Model

Our first characterization comes from a modified alternating machine, which allows "blocks" of nondeterminism in a simple, natural way. The standard model of an alternating Turing machine [15] chooses (existentially or universally) one of two possible "next states" at each step. However, alternation can also be achieved by existentially or universally choosing a *head position.* This provides the desired $(\log n)$-bit blocks of nondeterminism.

Definition 11 A *length-alternating Turing machine* is a multi-tape Turing machine whose states are partitioned into universal, existential, and deterministic states. (We shall refer to universal and existential states collectively as indeterministic states.) Each state, whether deterministic or not, has a single next state. Each indeterministic state s has an associated tape k_s; when the machine reaches state s, it jumps the head on tape k_s to an arbitrary nonblank square, and changes to the next state. This transition counts as a single step of the computation.

Acceptance and rejection are determined for each possible configuration of a length-alternating Turing machine in the same way as for standard alternating machines [15].

The *indeterminism depth* of a configuration in an LATM computation tree is the number of its ancestor configurations in the tree which are indeterministic. The indeterminism depth of an entire LATM computation tree is the maximum indeterminism depth of all its configurations.

The *time bound* of an LATM computation tree is the depth of the tree, counting deterministic as well as indeterministic configurations.

Theorem 12 *A relation $R(\vec{x}) \subseteq (\Sigma^*)^k$ is in SBH(QL) if and only if there are a constant c, a quasilinear function q, and a length-alternating Turing machine such that for all $\vec{x} \in (\Sigma^*)^k$,*

- *the machine accepts on input $\vec{x}$ if and only if $R(\vec{x})$ holds,*
- *the indeterminism depth of the computation tree is at most c, and*
- *the time bound of the computation tree is at most $q(\max(|\vec{x}|))$.*

Proof: Suppose $R(\vec{x})$ is in $SBH(QL)$. A machine with the required properties can compute R as follows. Assume that each input x_i is written on the i-th tape. For each quantifier $(Qy_i < q_i(|\vec{x}|))$ in turn, construct a string of length $q_i(|\vec{x}|)$ on an additional tape (one extra tape for each of constantly many quantifiers), indeterministically choose a head position on that tape, and in time $O(q_i(|\vec{x}|))$ count how far the head is from the right-hand end, writing the result in binary on another tape. Subsequent computation then treats the contents of this latter tape as y_i.

Conversely, suppose relation $R(\vec{x})$ is computed by a machine M satisfying the conditions of the theorem. We show by induction on c that R is in $SBH(QL)$.

If $c = 0$, then M is a deterministic machine; hence we have $R \in QL \subseteq SBH(QL)$.

If $c > 0$, let the "deterministic frontier" of M on input $\vec{x}$ be the set of configurations reachable from the start state without passing through an indeterministic state. Let $q(|\vec{x}|)$ be the (quasilinear) maximum number of steps along any computation path in this frontier. (We assume without loss of generality that q is of the form $\max(|\vec{x}|) \cdot p(\max(||\vec{x}||)) + d$, for some polynomial p; $||\cdot||$ represents the iterated length operator.) Upon reaching any indeterministic configuration on the frontier, at most $q(|\vec{x}|)$ squares can have been written and so tape 1 has at most $|x_1| + q(|\vec{x}|)$ nonblank squares.

Define a machine M' with inputs $\vec{x}, z_e, z_u$ as follows. The new machine M' simulates M until it reaches its first indeterministic state. If this state is existential, it deterministically moves the head on tape 1 to the position whose binary representation is z_e; if universal, it uses z_u. It then checks the validity of z_e or z_u by testing whether the head is on a blank square; if so, it "abstains" by rejecting in the existential case and accepting in the universal case. Otherwise, it subsequently acts exactly like M after the indeterministic state.

Define relation $S(\vec{x}, z_e, z_u)$ to be the accept/reject decision of M'. Then $R(\vec{x})$ holds iff

$$(\exists z_e |z_e| \leq \log(|x_1| + q(|\vec{x}|)))(\forall z_u |z_u| \leq \log(|x_1| + q(|\vec{x}|)))\ S(\vec{x}, z_e, z_u).$$

Since machine M' reaches fewer than c indeterministic configurations on any given branch, the induction hypothesis implies $S \in SBH(QL)$. Since R is defined by sharply-bounded quantification from S, we have $R \in SBH(QL)$ as well. ∎

An exactly analogous characterization can be given of *SBH(linear)*, the closure of *linear* time (on Turing machines with fixed but arbitrary numbers of tapes) under sharply bounded quantifiers.

4.2 First-Order Definability

The *SBH* has several clear analogues, including the polynomial hierarchy. The quantifiers of *SBH* have the same length as those of the log-time hierarchy, which is equivalent to logtime-uniform AC^0. The most elegant characterization of uniform AC^0 is in terms of first-order definability, as discovered by Immerman [3, 34, 35]. The *SBH* can be characterized similarly, by generalizing Immerman's definition of first-order definability to treat an arbitrary class of relations as atomic. We show that when *QL* is used as the atomic class, the result is precisely *SBH*(*QL*).

Definition 13 *Let $\mathcal{C}$ be a class of relations on* $\mathbf{N}$. *A relation $R(x_1, \ldots x_k)$ is* first-order definable *from $\mathcal{C}$, denoted $R \in FO(\mathcal{C})$, if there is a relation $S(\vec{x}, \vec{y}) \in \mathcal{C}$ such that for all $\vec{x} \in \mathbf{N}$,*

$$R(\vec{x}) \iff (Q_1 y_1 < n)(Q_2 y_2 < n) \cdots (Q_m y_m < n) S(\vec{x}, \vec{y}),$$

where each Q is either $\exists$ or $\forall$, and $n = \max(|\vec{x}|)$.

Note that the quantifiers range over $y < n$ rather than $|y| \leq \log n$ as in the definition of *SBH*(*QL*). The latter criterion is slightly more restrictive, but the two differ by at most one bit of nondeterminism, which can be simulated in a constant factor of time.

The following lemma shows Definition 13 to be a generalization of Immerman's definition. Define the Boolean function $BIT(n, m)$ to equal the m^{th} bit of n.

Lemma 14 *Let $\mathcal{C}_0$ be the set of (the natural interpretations of) atomic formulas*

$$\{y_i = y_j, y_i \leq y_j, BIT(y_i, y_j), BIT(y_i, x) \mid i, j \in \mathbf{N}\},$$

and let $\mathcal{C}$ be their closure under binary Boolean operators. Then a unary relation $R(x)$ is in FO if and only if it is in $FO(\mathcal{C})$.

The proof follows Immerman's [34, 35] straightforwardly, and we omit it here.

A different special case of Definition 13 is obtained by letting $\mathcal{C}$ be the set of *QL* relations on $\vec{x}, \vec{y}$; the result can naturally be written FO(*QL*).

Theorem 15 *FO*(*QL*) = *SBH*(*QL*)

Proof: A *FO*(*QL*) formula $(\exists y < n)\phi(\vec{x}, y)$ can be rewritten with sharply-bounded quantifiers as

$$(\tilde{\exists} y)(\phi(\vec{x}, y) \vee \phi(\vec{x}, y + 2^{\lfloor \log n \rfloor}))$$

which is clearly in *SBH*(*QL*). (The two copies of ϕ simulate existentially guessing the aforementioned one additional bit.) Universal quantifiers are handled similarly.

For the reverse containment, any existential quantifier in an *SBH* definition, *e.g.* $(\exists y |y| \leq \log n) R(y)$, can be written in *FO* form as $(\exists y < n)(|y| \leq |n| - 1 \wedge R(y))$, because $|y| \leq \log n \Leftrightarrow |y| \leq \lfloor \log n \rfloor \Leftrightarrow |y| \leq |n| - 1$. The test $|y| \leq |n| - 1$ is in *QL*, and *QL* is closed under $\wedge$, so this is a legitimate *FO*(*QL*) formula. Universal quantifiers can be handled similarly. ∎

Thus *SBH*(*QL*) relations can be thought of as those relations definable by first-order formulas with atomic relations in *QL*, and in which the domain of quantification is $\{0, 1 \ldots n-1\}$. Alternatively, an *SBH*(*QL*) relation can be thought of as a uniform AC^0 relation composed with a *QL* relation (which takes an extra input provided by the AC^0 uniformity condition).

The characterization of *SBH(linear)* as the closure of linear time under linear quantification is analogous, and one can similarly characterize the *SBH* closure of other time-bounded classes.

5 Function-Algebraic Characterizations

Our next characterization of *SBH* is based on characterizations of P by function classes. Early function-algebraic characterizations of P, such as Cobham's [17], relied on explicit polynomial bounds on recursion; more recent work [5, 7, 10, 39] has used the notion of *safe*, or *tiered*, recursion to replace such artificial bounds with more foundationally justifiable limitations. In this section, we present the relevant notions of safe recursion, prove function-algebraic characterizations of both linear and quasilinear time, and modify these characterizations to characterize the *SBH* function classes defined earlier.

5.1 Safe Parameters

Bellantoni and Cook [7] define a scheme of "safe recursion on notation" that resembles Cobham's "bounded recursion on notation" [17] but replaces Cobham's growth-rate bounds by syntactic restrictions on the use of parameters. They consider each function to take two kinds of parameters, called "normal" and "safe"; the latter are subject to only a restricted set of operations. They distinguish the two kinds syntactically by listing a function's normal parameters first, separated from its safe parameters with a semicolon, *e.g.* $f(x_1, x_2; y, z_1, z_2)$. (A similar notion appeared, more or less independently, in Leivant [39].)

Safe and normal parameters differ in two important ways: first, a safe parameter may not be used to control the depth of a recursion; and second, a safe parameter to a function cannot be used as normal within the computation of that function, nor in any way affect a value used as normal within the computation of that function. Normal parameters, however, may be used as safe—there is a "one-way door" between the two.

To enforce these constraints, Bellantoni and Cook define function-constructor operations called "safe composition" and "safe recursion on notation", and show that closing a set of simple base functions under these constructors captures exactly polynomial time.

Characterizing linear or quasilinear time requires an additional technique. In particular, one cannot freely use as many copies of a parameter as one wishes; making copies of a linear-length value takes linear time and cannot be allowed inside any kind of loop. Accordingly, we impose a third distinction between safe and normal parameters, one reminiscent of linear logic: a function may use only one copy of each safe parameter, but may make unlimited copies of normal parameters.

We shall work with *tuple-valued* functions, which take j normal and k safe parameters and produce as output a tuple of m values. (The values of j, k, and m are fixed for any given function.) Such a tuple is thought of not as a distinct object with boundaries—tuple concatenation is associative—but rather as a sequence of values corresponding to the parameters of a subsequent composed function. Given a tuple, $\vec{x} = (x_1, \ldots, x_n)$, the *arity* of $\vec{x}$ is n; the *length-vector* of $\vec{x}$, written $|\vec{x}|$, is the tuple $(|x_1|, \ldots, |x_n|)$.

The tuples will allow us to define sufficiently powerful compositions without copying safe parameters. For example, if $f_1(;x)$, $f_2(;x)$ and $g(;x,y)$ are defined, one might wish to define $h(;x) = g(;f_1(;x), f_2(;x))$, but this entails copying x. If f_1 and f_2 happen to be such that both can easily be computed from a single copy of x (*e.g.* $f_1(;x) = x \bmod 4$, $f_2(;x) = \lfloor x/4 \rfloor$), then the tuple-valued function $f(;x) = \langle f_1(;x), f_2(;x) \rangle$ will be definable, and the composition $h(;x) = g(;f(;x))$ will be legal.

We start with the following set of *BASE* functions.

- The constant 0 function;
- The function $\mathit{Half}(;y) = \lfloor y/2 \rfloor$;
- The function $s_0(;y) = 2y$;
- The function $s_1(;y) = 2y + 1$; and
- Any fixed permutation of any fixed subset of j normal and k safe parameters, for all j and k.

The first four *BASE* functions return arity 1; only the last can return a nontrivial tuple.

Bellantoni and Cook also included a conditional function among their *BASE* functions. In practice, such a conditional function tends to be used in composition with other functions that take the same parameters, *e.g.* $f(;x,y) = \mathit{Cond}(;x, s_0(;y), s_1(;y))$. This example is syntactically illegal in our composition scheme, since it uses the safe parameter y more than once. Semantically, however, it is harmless, since only one copy of y is actually used in computing any particular instance of $f(;x,y)$. We therefore define the conditional not as a *BASE* function but as another constructor.

Definition 16 *The function* $f(\vec{x};\vec{y})$ *is* defined by cases on the i-th parameter *from functions* g_0*,* g_1*, and* g_2 *if*

$$f(\vec{x};\vec{y}) = \begin{cases} g_0(\vec{x};\vec{y}) & \text{if the } i\text{-th parameter of } \vec{x},\vec{y} \text{ is } 0 \\ g_1(\vec{x};\vec{y}) & \text{if the } i\text{-th parameter of } \vec{x},\vec{y} \text{ is odd} \\ g_2(\vec{x};\vec{y}) & \text{if the } i\text{-th parameter of } \vec{x},\vec{y} \text{ is positive and even.} \end{cases}$$

A function definable from the base functions using any fixed number of applications of definition by cases can be computed in constant time on a multi-tape Turing machine using an appropriate representation for input and output. The machine has one tape for each input string (and possibly others); each has its head at the right (low-order) end of the string. We use the states of the machine to store, among other things, a permutation of the constantly many tapes, so (for example) any fixed permutation can be computed in one step of the machine. In one step, the machine makes the necessary change to one input and changes state to record the output permutation. Our proofs will use this representation for intermediate computations; any reasonable input-output convention can be converted to and from this one in linear time at the start and end of a computation.

Definition 17 *The function* $f(\vec{x};\vec{y})$ *is* defined by safe composition *from functions* g, $u_1,\ldots,u_j$, *and* $v_1,\ldots,v_k$ *if for some fixed partition of the safe parameters* $\vec{y}$ *into* $\vec{y}_1,\vec{y}_2,\ldots,\vec{y}_k$,

$$f(\vec{x};\vec{y}) = g(u_1(\vec{x};),\ldots,u_j(\vec{x};);v_1(\vec{x};\vec{y}_1),\ldots,v_k(\vec{x};\vec{y}_k)).$$

Partitioning the safe parameters avoids implicit copying. Normal parameters, however, may be copied freely: each of the constantly many u and v functions gets its own copy of the normal parameters $\vec{x}$. Note that, in order for this definition to make sense, f must return a tuple of the same arity as g, g must take a number of normal parameters equal to the sum of the arities returned by $u_1,\ldots,u_j$, and g must likewise take a number of safe parameters equal to the sum of the arities returned by $v_1,\ldots,v_k$.

Functions defined from *BASE* using only safe composition and cases are still very simple. If such a definition doesn't use multiple copies of its normal parameters (*e.g.* if it takes no normal parameters at all), then it can likewise be computed in constant time using the above input/output convention.

Most interesting functions require some form of iteration or recursion. We define a restricted form of recursion as follows.

Definition 18 *The function* $f(z,\vec{x};\vec{y})$ *is defined by* very safe recursion on notation *from functions* $h(;\vec{w})$ *and* $g(\vec{x};\vec{y})$ *if*

$$f(z,\vec{x};\vec{y}) = \begin{cases} g(\vec{x};\vec{y}) & \text{if } z=0 \\ h\left(;f\left(\lfloor z/2\rfloor,\vec{x};\vec{y}\right)\right) & \text{if } z>0. \end{cases}$$

The variable of recursion is always a normal parameter to f, while the iterated function h takes no normal parameters.

Using very safe recursion and safe composition, one can define functions of any linear growth rate (measured as usual in terms of the length of the arguments and output). For example, let $Succ_k(;y)$ be the k-th iterate of $s_1(;y)$; then the function

$$f(x;) = \begin{cases} 0 & \text{if } x=0 \\ Succ_k(;f(\lfloor x/2\rfloor;)) & \text{if } x>0 \end{cases}$$

satisfies $|f(x;)| = k\cdot|x|$. We shall show in the next section that no function with superlinear growth can be defined.

Bellantoni has speculated on the effects of loosening in various ways the prohibition that no safe parameter may in any way affect a normal parameter. One such change allows the *length* of a safe parameter to be used as normal; Bellantoni [5] shows that this makes no difference to his characterization of *FP*. In our setting, however, it makes the difference between linear and quasilinear time, as we shall see in Theorem 24. To formalize the change we will replace the above schema of safe composition by the following variant.

Definition 19 *The function* $f(\vec{x};\vec{y})$ *is* defined by length composition *from functions* g, $u_1,\ldots u_j$, *and* $v_1,\ldots,v_k$ *if for some fixed partition of the safe parameters* $\vec{y}$ *into* $\vec{y}_1,\vec{y}_2,\ldots,\vec{y}_k$,

$$f(\vec{x};\vec{y}) = g(u_1(\vec{x},|\vec{y}|;),\ldots,u_j(\vec{x},|\vec{y}|;);v_1(\vec{x},|\vec{y}|;\vec{y}_1),\ldots,v_k(\vec{x},|\vec{y}|;\vec{y}_k)).$$

Combining length recursion with very safe recursion can lead to superlinear growth rates. Let

$$q_0(z;y) = \begin{cases} y & \text{if } z=0 \\ s_0(;q_0(\lfloor z/2\rfloor;y)) & \text{if } z>0, \end{cases}$$

and for all $k \geq 1$,

$$r_k(;y) = q_{k-1}(|y|;y)$$

and

$$q_k(z;y) = \begin{cases} y & \text{if } z=0 \\ r_k(;q_k(\lfloor z/2\rfloor;y)) & \text{if } z>0. \end{cases}$$

For each k, the value of $q_k(z;z)$ has length $\Theta(|z|\log^{k-1}|z|)$.

5.2 Characterizing Linear and Quasilinear Time

In this section, we characterize linear and quasilinear time on multi-tape Turing machines by the use of safe parameters. Versions of Theorem 21 were conjectured simultaneously and independently by Bloch, by Bellantoni, and by Leivant, but to our knowledge neither Theorem 24 nor a precise statement or proof of Theorem 21 has appeared in the literature.[1]

Definition 20 *The function class* $\mathcal{D}$ *is the closure of the set BASE under the operations of definition by cases, safe composition and very safe recursion on notation.*

Theorem 21 *A function* $f(\vec{x})$ *is computable by a linear-time multi-tape Turing machine if and only if* $f(\vec{x};) \in \mathcal{D}$.

[1] Otto [44] has given a category-theoretic characterization of these classes inspired by an earlier version of this section.

Proof: For any function and inputs $f(\vec{x};\vec{y})$, we denote by $Tf(\vec{x};\vec{y})$ the time required to compute f at $\vec{x};\vec{y}$ on a multi-tape Turing machine using the input-output convention defined in the previous section. If $f(\vec{x};) \in \mathcal{D}$, we bound its growth rate and required computation time by induction on its definition.

Lemma 22 *For all f in $\mathcal{D}$, there are constants l_f and c_f such that for all $\vec{x}$ and $\vec{y}$,*

$$|f(\vec{x};\vec{y})| \leq l_f \cdot (1 + \max(|\vec{x}|)) + \max(|\vec{y}|)$$

and

$$Tf(\vec{x};\vec{y}) \leq c_f \cdot (1 + \max(|\vec{x}|)).$$

(In particular, seemingly simple functions like $f(;x) = x + 1$ cannot be defined with only safe parameters.)

We prove the lemma by induction on the $\mathcal{D}$ definition of f.

If f is in *BASE*, we can take $l_f = 1$ and $c_f = 2$.

If $f(\vec{x};\vec{y})$ is defined by cases from $g_0(\vec{x};\vec{y})$, $g_1(\vec{x};\vec{y})$, and $g_2(\vec{x};\vec{y})$, we can take $l_f = \max_i(l_{g_i})$ and $c_f = \max_i\{c_{g_i}\} + 2$.

If $f(\vec{x};\vec{y})$ is defined by safe composition, *i.e.*

$$f(\vec{x};\vec{y}) = g(u_1(\vec{x};), \ldots, u_j(\vec{x};); v_1(\vec{x};\vec{y}_1), \ldots, v_k(\vec{x};\vec{y}_k)),$$

then we have

$$\begin{aligned} |f(\vec{x};\vec{y})| &\leq l_g \cdot (1 + \max_i(|u_i(\vec{x};)|)) + \max_i(|v_i(\vec{x};\vec{y}_i)|) \\ &\leq (l_g \cdot (1 + \max_i(l_{u_i}) + \max_i(l_{v_i})) \cdot (1 + \max(|\vec{x}|)) + \max(|\vec{y}|), \end{aligned}$$

so we may take $l_f = l_g \cdot (1 + \max_i(l_{u_i}) + \max_i(l_{v_i}))$.

To compute $f(\vec{x};\vec{y})$, make $j+k$ copies of $\vec{x}$ (on new tapes), then compute $\vec{u}(\vec{x};)$, $\vec{v}(\vec{x};\vec{y})$ and finally $g(\vec{u}(\vec{x};);\vec{v}(\vec{x};\vec{y}))$. Thus

$$\begin{aligned} Tf(\vec{x};\vec{y}) &\leq \max(|\vec{x}|) + \max_i Tu_i(\vec{x};) + \max_i Tv_i(\vec{x};\vec{y}_i) + Tg(\vec{u}(\vec{x};);\vec{v}(\vec{x};\vec{y})) \\ &\leq \max(|\vec{x}|) + (1 + \max|\vec{x}|) \cdot \left(\max_i c_{u_i} + \max_i c_{v_i}\right) + (1 + \max_i |u_i(\vec{x};)|) \cdot c_g \\ &\leq (1 + \max|\vec{x}|) \cdot \left(1 + \left(\max_i c_{u_i} + \max_i c_{v_i} + (1 + \max_i(l_{u_i})) \cdot c_g\right)\right). \end{aligned}$$

Hence we may take $c_f = 1 + (\max_i c_{u_i} + \max_i c_{v_i} + (1 + \max_i(l_{u_i})) \cdot c_g)$.

If $f(\vec{x};\vec{y})$ is defined by very safe recursion on notation, *i.e.*

$$f(z,\vec{x};\vec{y}) = \begin{cases} g(\vec{x};\vec{y}) & \text{if } z = 0 \\ h(; f(\lfloor z/2 \rfloor, \vec{x};\vec{y})) & \text{if } z > 0, \end{cases}$$

then $Th(\vec{w})$ is constant and $|h(;\vec{w})| \leq |\vec{w}| + l_h$. Thus

$$\begin{aligned} |f(z,\vec{x};\vec{y})| &\leq |g(\vec{x};\vec{y})| + |z| \cdot l_h \\ &\leq (l_g + l_h) \cdot (1 + \max(|z|, |\vec{x}|)) + \max(|\vec{y}|). \end{aligned}$$

and

$$\begin{aligned} Tf(z,\vec{x};\vec{y}) &\leq Tg(\vec{x};\vec{y}) + |z| \cdot Th(;f(\lfloor z/2 \rfloor, \vec{x};\vec{y})) \\ &\leq c_g \cdot (1 + \max|\vec{x}|) + |z| \cdot c_h. \end{aligned}$$

Hence we may take $l_f = l_g + l_h$ and $c_f = c_g + c_h$.

The lemma follows by induction. Thus functions in $\mathcal{D}$ with only normal parameters are computable in linear time with any reasonable input-output convention.

For the other direction of Theorem 21, let M be a k-tape Turing machine with a binary tape alphabet, using the standard input-output convention. We represent a configuration of M by a $(2k+1)$-tuple of the form $\langle q, L_1, R_1, \ldots L_k, R_k \rangle$, comprising the state and the left and right halves of each tape. We define a function *NEXT* which takes $2k+1$ safe parameters and outputs a $(2k+1)$-tuple representing the next configuration of the machine. The *NEXT* function can be defined by composition and cases alone from *BASE* functions, and hence needs no normal parameters.

As demonstrated in the previous section, $\mathcal{D}$ contains a function $f(\vec{x};)$ whose length bounds the run time of $M(x)$. The function $\hat{f}(\vec{x};) = \langle f(\vec{x}), \vec{x} \rangle$, defined from f by safe composition, computes the run-time bound while preserving the input for simulating M.

Once the run-time bound is computed, we apply very safe recursion on notation, using the bound and the *NEXT* function above, to define the value of the state and of each tape at the end of the computation of M. By constantly many steps of definition by cases, we can then determine which tape contains the desired function value, and extract it by projection. ∎

Definition 23 *The function class $\mathcal{D}'$ is the closure of the set BASE under the operations of definition by cases, length composition and very safe recursion on notation.*

Theorem 24 *A function $f(\vec{x})$ is computable by a quasilinear-time multi-tape Turing machine if and only if $f(\vec{x};) \in \mathcal{D}'$.*

Proof: This proof is similar to that of Theorem 21, but the growth rates become more complicated. To state the analogue of Lemma 22 cleanly, we enhance the Turing machine model with a unit-time *length-replacement* operation. In one step, the length-replacement operation replaces the current contents of any one tape with the length of the non-blank string on another tape, written in binary. For the duration of the following lemma, the notation Tf represents time on this model.

A standard Turing machine can easily simulate a Turing machine with length replacement by doubling the number of tapes. Initially, the length of each input tape is computed and written in binary on a new tape. At each step, the lengths are updated to reflect head motion of the original machine. A length replacement copies the appropriate counter and computes its length. Each original step requires at most logarithmic time in the simulation; hence, quasilinear time is preserved.

Lemma 25 *For all f in $\mathcal{D}'$, there are polynomials l_f and p_f such that for all $\vec{x}$ and $\vec{y}$,*

$$|f(\vec{x};\vec{y})| \leq (1+\max(|\vec{x}|)) \cdot l_f(||\vec{x}||, ||\vec{y}||) + \max(|\vec{y}|)$$

and

$$Tf(\vec{x};\vec{y}) \leq (\max(|\vec{x}|)+1) \cdot p_f(||\vec{x}||, ||\vec{y}||)$$

on Turing machines with length replacement.

The proof is again an induction of the definition of a function. The *BASE* functions all clearly satisfy the lemma. Definition by cases also preserves the conditions, since the set of growth rates defined in the lemma is closed under max.

Suppose f is defined by length composition, *i.e.,*

$$f(\vec{x};\vec{y}) = g(u_1(\vec{x},|\vec{y}|;),\ldots,u_j(\vec{x},|\vec{y}|;);v_1(\vec{x},|\vec{y}|;\vec{y}_1),\ldots,v_k(\vec{x},|\vec{y}|;\vec{y}_k)),$$

By the induction hypothesis, there are polynomials l_g, l_{u_i}, and l_{v_i} such that

$$\begin{aligned} |g(\vec{u};\vec{v})| &\leq (1+\max(|\vec{u}|)) \cdot l_g(||\vec{u}||, ||\vec{v}||) + \max(|\vec{v}|) \\ |u_i(\vec{x},\vec{Y};)| &\leq (1+\max(|\vec{x}|,|\vec{Y}|)) \cdot l_{u_i}(||\vec{x}||, ||\vec{Y}||) \end{aligned}$$

and

$$|v_i(\vec{x},\vec{Y};\vec{y}_i)| \leq (1+\max(|\vec{x}|,|\vec{Y}|)) \cdot l_{v_i}(||\vec{x}||, ||\vec{Y}||, ||\vec{y}_i||) + \max(|\vec{y}|)$$

Then

$$\begin{aligned} |f(\vec{x};\vec{y})| &\leq |g(\vec{u}(\vec{x},|\vec{y}|;);\vec{v}(\vec{x},|\vec{y}|;\vec{y}))| \\ &\leq (\max(|\vec{x}|,||\vec{y}||)+1) \cdot \max_i(l_{u_i}(||\vec{x}||,|||\vec{y}|||))) \cdot l_g(||\vec{u}(\vec{x},|\vec{y}|;)||, ||\vec{v}(\vec{x},|\vec{y}|;\vec{y})||) \\ &\quad + \max_i(|v_i(\vec{x},|\vec{y}|;\vec{y}_i)|) \\ &\leq (\max(|\vec{x}|)+1) \cdot l_f(||\vec{x}||, ||\vec{y}||) + \max(|\vec{y}|) \end{aligned}$$

for some polynomial l_f, as desired.

To compute the composition, first compute the needed lengths of safe parameters in a single length-replacement step, then make the needed copies of the normal parameters and lengths, compute $\vec{u}$ and $\vec{v}$ (in parallel, on constantly many tapes) and finally compute g. The time satisfies

$$\begin{aligned}
Tf(\vec{x};\vec{y}) &\leq \max|\vec{x}| + \max||\vec{y}|| + \max_{1\leq i\leq j}(Tu_i(\vec{x},|\vec{y}|;)) + \max_{1\leq i\leq k}(Tv_i(\vec{x},|\vec{y}|;\vec{y}_i)) \\
&\quad + Tg(\vec{u}(\vec{x},|\vec{y}|;);\vec{v}(\vec{x},|\vec{y}|;\vec{y})) \\
&\leq (\max(|\vec{x}|,||\vec{y}||)+1)\cdot\big(1+\max_{1\leq i\leq j}(p_{u_i}(||\vec{x}||,|||\vec{y}|||)) + \max_{1\leq i\leq k}(p_{v_i}(||\vec{x}||,|||\vec{y}|||,||\vec{y}_i||))\big) \\
&\quad + (\max|\vec{u}(\vec{x},|\vec{y}|;)|+1)\cdot p_g(||\vec{u}(\vec{x},|\vec{y}|;)||,||\vec{v}(\vec{x},|\vec{y}|;\vec{y})||) \\
&\leq (\max(|\vec{x}|,||\vec{y}||)+1)\cdot p_f(||\vec{x}||,||\vec{y}||)
\end{aligned}$$

for a polynomial p_f depending on l_u, l_v, p_u, p_v and p_g.

Suppose now that f is defined by very safe recursion on notation, i.e.

$$f(z,\vec{x};\vec{y}) = \begin{cases} g(\vec{x};\vec{y}) & \text{if } z=0 \\ h(;f(\lfloor z/2\rfloor,\vec{x};\vec{y})) & \text{if } z>0. \end{cases}$$

By the induction hypothesis, there are polynomials l_g and l_h such that

$$|g(\vec{x};\vec{y})| \leq (\max(|\vec{x}|)+1)\cdot l_g(||\vec{x}||,||\vec{y}||)+\max(|\vec{y}|)$$

and

$$|h(;\vec{u})| \leq l_h(||\vec{u}||)+\max(|\vec{u}|).$$

Assuming without loss of generality that $|h(;u)|$ is nondecreasing in $|u|$, we have for all $z>0$,

$$\begin{aligned}
|f(z,\vec{x};\vec{y})| &\leq l_h(||f(\lfloor z/2\rfloor,\vec{x};\vec{y})||)+\max(|f(\lfloor z/2\rfloor,\vec{x};\vec{y})|) \\
&\leq l_h(||f(\lfloor z/2\rfloor,\vec{x};\vec{y})||)\cdot|z|+\max(|f(0,\vec{x};\vec{y})|)
\end{aligned}$$

(If the notation "$\max(|f(0,\vec{x};\vec{y})|)$" seems odd, remember that f is tuple-valued.)

We now prove the existence of a polynomial l_f such that for all $z,\vec{x},\vec{y}$,

$$|f(z,\vec{x};\vec{y})| \leq l_f(||z||,||\vec{x}||,||\vec{y}||)\cdot(\max(|z|,|\vec{x}|)+1)+\max(|\vec{y}|).$$

Since $f(0,\vec{x};\vec{y}) = g(\vec{x};\vec{y})$ for all i, the polynomial l_g works for all $\vec{x},\vec{y}$ so long as $z=0$. Now assume inductively that some polynomial l_f satisfies the lemma for all $\vec{x},\vec{y}$ and all $z<z_0$. We shall find sufficient conditions on l_f to still satisfy the lemma at z_0; then if a

particular polynomial l_f meets the conditions for all z_0, the induction proof goes through and l_f satisfies the requirements of the lemma. We have

$$\begin{aligned}
|f(z_0, \vec{x}; \vec{y})| &\leq l_h\left(||f\left(\lfloor z_0/2\rfloor, \vec{x}; \vec{y}\right)||\right) \cdot |z_0| + \max(|f(0, \vec{x}; \vec{y})|) \\
&\leq l_h\left(|l_f\left(||\lfloor z_0/2\rfloor||, ||\vec{x}||, ||\vec{y}||\right) \cdot \max(|z_0|, |\vec{x}|) + \max(|\vec{y}|)|\right) \cdot |z_0| \\
&\qquad + \max(|f(0, \vec{x}; \vec{y})|) \\
&\leq l_h\left(\left|\mathrm{tr}(l_f) \cdot ||\max(z_0, \vec{x}, \vec{y})||^{\deg(l_f)} \cdot \max(|z_0|, |\vec{x}|) + \max(|\vec{y}|)\right|\right) \cdot |z_0| \\
&\qquad + \max(|f(0, \vec{x}; \vec{y})|) \\
&\leq l_h(|\mathrm{tr}(l_f)| + \deg(l_f) \cdot |||\max(z_0, \vec{x}, \vec{y})||| + ||\max(z_0, \vec{x})|| + ||\max(\vec{y})||) \cdot |z_0| \\
&\qquad + \max(|f(0, \vec{x}; \vec{y})|) \\
&\leq l_h(|\mathrm{tr}(l_f)| + (2 + \deg(l_f)) \cdot ||\max(z_0, \vec{x}, \vec{y})||) \cdot |z_0| \\
&\qquad + \max((\max(|\vec{x}|) + 1) \cdot l_g(||\vec{x}||, ||\vec{y}||) + \max(|\vec{y}|)) \\
&\leq (l_h(|\mathrm{tr}(l_f)| + (2 + \deg(l_f)) \cdot ||\max(z_0, \vec{x}, \vec{y})||) \cdot |z_0| + l_g(||\vec{x}||, ||\vec{y}||)) \\
&\qquad \cdot (\max(|z_0|, |\vec{x}|) + 1) \\
&\qquad + \max(|\vec{y}|).
\end{aligned}$$

It would suffice, therefore, to find a l_f such that, for all z_0, $\vec{x}$, $\vec{y}$,

$$l_f(||z_0||, ||\vec{x}||, ||\vec{y}||) \geq l_h(|\mathrm{tr}(l_f)| + (2 + \deg(l_f)) \cdot ||\max(z_0, \vec{x}, \vec{y})||) + l_g(||\vec{x}||, ||\vec{y}||).$$

Such a polynomial can be constructed as follows. Assume that all the coefficients of l_g and l_h are nonnegative, and let $l_0 = l_g + l_h$. If the inequality holds with l_0 plugged in for l_f, we're done. Otherwise, consider the effect of plugging in $2^m \cdot l_0$ for l_f. Multiplying l_0 by 2^m multiplies all the coefficients on the left by 2^m, while on the right hand side it merely adds m to $|tr(l_0)|$, and therefore adds a polynomial whose coefficients are polynomial in m. So for all sufficiently large m, the inequality is formally true (*i.e.*, each coefficient on the left is greater than or equal to the corresponding coefficient on the right), and hence true for all positive values of $|z_0|$. Choose $l_f = 2^m \cdot l_0$ for some such m, and the induction goes through.

In computing f, the function h is iterated $|z|$ times on arguments of length at most $|f(z, \vec{x}; \vec{y})|$; hence

$$\begin{aligned}
Tf(z, \vec{x}; \vec{y}) &\leq Tg(\vec{x}; \vec{y}) + |z| \cdot p_h\left(\left|(1 + \max(|z|, |\vec{x}|)) \cdot l_f\left(||z||, ||\vec{x}||, ||\vec{y}||\right) + \max|\vec{y}|\right|\right) \\
&\leq (1 + \max(|z|, |\vec{x}|)) \cdot p_g\left(||z||, ||\vec{x}||, ||\vec{y}||\right) \\
&\qquad + |z| \cdot p_h\left(|1 + \max(|z|, |\vec{x}|)| + |l_f(||z||, ||\vec{x}||, ||\vec{y}||)| + \max||\vec{y}||\right) \\
&\leq (1 + \max(|z|, |\vec{x}|)) \cdot p_f\left(||z||, ||\vec{x}||, ||\vec{y}||\right)
\end{aligned}$$

for some polynomial p_f.

The lemma follows by induction, and the first half of the theorem is proved.

For the reverse direction of Theorem 24, the machine encoding proceeds as in the linear case, but is iterated over a quasilinear time bound rather than a linear one, using the functions q_k of the previous section. ∎

5.3 Characterizing *SBH*

We now characterize *SBH(QL)* and *SBH(linear)* by adding one additional schema to the tiered function algebras of Section 5.2. The 0/1-valued functions definable with the new schema will be the characteristic functions of *SBH(QL)* and *SBH(linear)* relations. Arbitrary (0/1-valued or not) definable functions are the functions in $FQL^{SBH(QL)[1]}$ and $FL^{SBH(linear)[1]}$.

Definition 26 *A function f is defined by* normal-bounded zero-detection *from g if*

$$f(z,\vec{x};\vec{y}) = \begin{cases} 0 & \textit{if } (\forall w < |z|)(g(w,\vec{x};\vec{y}) = 0) \\ 1 & \textit{if } (\exists w < |z|)(g(w,\vec{x};\vec{y}) > 0) \end{cases}.$$

Definition 27

- *The function class $\mathcal{D}^{nzd}$ is the closure of the BASE functions under safe composition, very safe recursion on notation, and normal-bounded zero-detection.*
- *The function class $\mathcal{D}'^{nzd}$ is the closure of the BASE functions under length composition, very safe recursion on notation, and normal-bounded zero-detection.*

In the spirit of the previous sections, we select a machine model suitable for reasoning about the definable functions. The natural model for $FQL^{SBH(QL)[1]}$ is an oracle machine: a quasilinear-time deterministic computation with a constant number of queries to an oracle in *SBH*. By Theorem 12, the oracle set is computable by an LATM in quasilinear time with constant indeterminism depth. In order to obtain clean time bounds in the inductive proofs below, we shall want to avoid copying strings to special oracle tapes. We therefore use a "self-query" machine, which combines the machine that queries the oracle with the machine that recognizes the oracle set.

A "self-query" state q designates three states q_t, q_y, and q_n, called "test", "yes", and "no" respectively. The test state q_t is the start of the computation of the oracle set. If the computation from q_t accepts, the answer to the query is yes, and the successor of query state q is q_y. If the computation from q_t rejects, then answer to the query is no, and the successor of q is q_n. Note that the self-query state has only one actual successor and is thus a deterministic state.

To treat this oracle machine and its query set as a single LATM with self-queries requires several restrictions. First, to keep the deterministic part that queries the oracle separate from the length-alternating part that decides the oracle, we require that query states may not appear as descendants of test states, while indeterministic states may *only* appear as descendants of test states. Second, the indeterminism depth of the whole computation tree must (as in Theorem 12) be constant.

Note that, since the time bound of an LATM is found by maximizing the length of *all* its branches, any linear or quasilinear time bound we impose will apply to both the deterministic and length-alternating parts.

With these restrictions, a self-querying LATM M using quasilinear time behaves essentially as a deterministic quasilinear-time oracle machine with an oracle in $SBH(QL)$, and thus computes a function in $FQL^{SBH(QL)[1]}$. The one difference is that M makes queries without copying its configuration to a special oracle tape. Because M makes only a constant number of queries, a standard oracle machine could compute the same result, with copying of query strings, also in quasilinear time.

Similarly, linear time bounds on self-querying LATMs characterize exactly $FL^{SBH(linear)[1]}$.

Theorem 28 *A function $f(\vec{x})$ is in $FL^{SBH(linear)[1]}$ if and only if $f(\vec{x};) \in \mathcal{D}^{nzd}$. Furthermore, if f is 0/1-valued,* i.e., *f is the characteristic function of a relation R_f, then $R_f \in SBH(linear)$ if and only if $f(\vec{x};) \in \mathcal{D}^{nzd}$.*

Proof: To show the required time bounds, we prove by induction on the definition of f that

- if $f(\vec{x};\vec{y}) \in \mathcal{D}^{nzd}$, then f is computable on an self-querying multi-tape Turing machine in time $O(|\vec{x}|)$ with constantly many queries to an $O(|\vec{x}|)$-time, constant-depth LATM, and
- if $f(\vec{x};\vec{y}) \in \mathcal{D}^{nzd}$ is 0/1-valued, then R_f is computable by an $O(|\vec{x}|)$-time, constant-depth LATM.

Corollary 4 implies that the former statement implies the latter.

The induction on the definition of f follows closely that of Theorem 21. The *BASE* functions, definition by cases, and safe composition work exactly as in Lemma 22; the property of using a constant number of self-queries is preserved by the computations.

If $f(z,\vec{x};\vec{y})$ is defined by very safe recursion on notation from $g(\vec{x};\vec{y})$ and $h(;\vec{u})$, a straightforward algorithm computes $g(\vec{x};\vec{y})$ and then applies h to the result $|z|$ successive times. Since h has no normal parameters, its definition cannot involve normal-bounded zero-detection; hence h is computable in constant time without queries by Theorem 21. Thus $f(z,\vec{x};\vec{y})$ requires no more queries than does $g(\vec{x};\vec{y})$.

Now suppose $f(z,\vec{x};\vec{y})$ is defined by normal-bounded zero-detection from $g(w,\vec{x};\vec{y})$. f may be equivalently defined from the 0/1-valued function $\hat{g} = \max(1,g)$. By the induction hypothesis, $R_{\hat{g}}$ is computable by a linear-time LATM. To compute f, immediately make a self-query. For the test computation, existentially guess a position w in z, and check if $R_{\hat{g}}(w,\vec{x};\vec{y})$. On a "yes" answer, output 1; on a "no" answer, output 0. The computation makes a single query, and adds a constant factor to the time bounds of R_g.

Hence all functions in $\mathcal{D}^{nzd}$ have the required time bounds, by induction.

Now suppose $R \in SBH(linear)$. Write $R(\vec{x})$ as $(\tilde{\exists} y_1)(\tilde{\forall} y_2)\cdots(\tilde{\exists} y_k)S(\vec{x},\vec{y})$, where S is in linear time. Then by Theorem 21, the characteristic function $\chi_S(\vec{x},\vec{y};)$ is in $\mathcal{D}$. A normal-bounded zero-detection on χ_S gives the characteristic function of $(\tilde{\exists} y_k)S(\vec{x},\vec{y})$. For $\tilde{\forall}$ quantifiers, simply rewrite them as $\neg\tilde{\exists}\neg$, negating a $0/1$-valued function by composing it with the function $x \mapsto \max(0, 1-x)$, which is easily definable in $\mathcal{D}$ by cases. By applying k steps of zero-detection, with negations in between as required, we get $\chi_R(\vec{x};) \in \mathcal{D}^{nzd}$.

Now if $f(\vec{x}) \in FL^{SBH(linear)[1]}$, we know f is computable in linear time with some constant number of queries to some fixed *SBH(linear)* oracle $R(\vec{y})$. By the previous paragraph, $\chi_R(\vec{y};) \in \mathcal{D}^{nzd}$. Using standard techniques, we can move all the queries to be consecutive and nonadaptive, at the cost of making exponentially more (but still constantly many) queries. Thus the values given to the queries are $g_1(\vec{x};)$, ..., $g_c(\vec{x};)$ for some c, where each function g_i is linear-time, and the value of f is a linear-time function F of $\vec{x}$ and the c responses to the queries. By Theorem 21, the functions g_1, ..., g_c, and F are all in $\mathcal{D}$; therefore, we can define $f(\vec{x};) \in \mathcal{D}^{nzd}$ as $F(\vec{x}, \chi_R(g_1(\vec{x};);), \ldots, \chi_R(g_c(\vec{x};);))$. ∎

Theorem 29 *A function $f(\vec{x})$ is in $FQL^{SBH(QL)[1]}$ if and only if $f(\vec{x};) \in \mathcal{D}'^{nzd}$. Furthermore, if f is 0/1-valued,* i.e., *f is the characteristic function of a relation R_f, then $R_f \in SBH(QL)$ if and only if $f(\vec{x};) \in \mathcal{D}'^{nzd}$.*

Proof Sketch: Again, we use induction on the definition of a function; however, we add an extra requirement. Using length-replacement operations, if $f(\vec{x};\vec{y}) \in \mathcal{D}'^{nzd}$, then

- f is computable in time $Q(|\vec{x}|, ||\vec{y}||)$ on a self-querying LATM,
- $f(\vec{x};\vec{y})$ is computable deterministically in time polynomial in $|\vec{x}|$ and $||\vec{y}||$, and
- if f is $0/1$-valued, then R_f is computable by an $Q(|\vec{x}|, ||\vec{y}||)$-time, constant-depth LATM.

By the same argument as Corollary 4, the first part implies the last.

The induction on the definition of f requires only a few modifications to the others above. For *BASE* functions, definition by cases, and length composition, the growth rates and time bounds combine exactly as in Lemma 25; the number of queries (if any) remains constant.

If f is defined by normal-bounded zero-detection, use the same computation as in Theorem 28 for the bounds on self-querying LATMs. For deterministic computation, simply test all possible zeroes, preserving polynomial time.

If f is defined from g and h by very safe recursion, then h, having no normal parameters, can be computed deterministically in polylogarithmic time by the inductive hypothesis. Use this computation to iterate h for both the self-querying and deterministic computations of f to obtain the required time and query bounds.

The required bounds hold for all functions in $\mathcal{D}'^{nzd}$ by induction. Simulating the length-replacement operations yields one direction of the theorem.

The reverse direction is almost identical to that in Theorem 28, replacing "linear" with "quasilinear", $\mathcal{D}$ with $\mathcal{D}'$, and Theorem 21 with Theorem 24. ∎

6 Further directions

We have characterized *SBH* in several ways; however, its relation to many other complexity classes remains unclear. Theorems 8 and 9 suggest limits on our ability to prove that *SBH* contains hard problems, but they allow some room to maneuver. Perhaps most significantly, we would like to know whether *SBH* contains all problems solvable in deterministic quasilinear time on a random-access machine (*NLT*). Gurevich and Shelah [30] show a complete problem for *NLT*; if this one problem is in *SBH*, then *SBH* is the same over random-access machines as over Turing machines. They show that all the machine models considered yield the same class with respect to unbounded nondeterminism (*i.e.*, $NQL = NNLT$).

Similarly, *SBH* might contain Grädel's [25] class $DTIME(n^{1+})$, which is strictly larger than *QL*, but is not large enough to apply our Lemma 10. Further, one could easily define an analogous hierarchy over $DTIME(n^{1+})$ using the model of Section 4.1; would the inclusion $DTIME(n^{1+}) \in SBH$ imply the collapse of the $DTIME(n^{1+})$ hierarchy?

One extension of this work is to classes defined by a polylogarithmic number of sharply bounded quantifiers. These classes would be between *P* and *PSPACE* and might yield some more concrete insight into the $P =? PSPACE$ question. A strictly nondeterministic version of this hierarchy, known as the β-hierarchy, has been studied by various researchers [2, 4, 14, 38].

A different direction would consider $AQL[O(\log(n))]$, the class of relations computable in quasilinear time with $O(\log(n))$ bits of quantification but no bound on alternations. This class is still within *P* and contains *ALOGTIME*; thus if $AQL[O(\log(n))] \neq P$ then *a fortiori* $ALOGTIME \neq P$. On the other hand, if $AQL[O(\log(n))] = P$, then all polynomial-time computations can be parallelized down to quasilinear time using polynomially many processors. Such a collapse would answer a question of Condon [18] by showing that essentially all problems in *P* have polynomial parallel speedup.

Recent work of Cai and Chen [14] presents a useful general framework for discussing classes with limited amounts of nondeterminism. An obvious direction for further research is to extend their framework to handle alternation of universal and existential quantifiers.

Acknowledgements

The authors thank Peter Clote for suggesting that they work on this material, and making several helpful suggestions, and Michael Saks for pointing out that $\tilde{\forall}P_1$ contained interesting problems. Thanks to James Otto for pointing out an error in an earlier version of our safe-parameter definition.

References

[1] K.A. Abrahamson, R.G. Downey, and M.R. Fellows, "Fixed parameter tractability and completeness IV: on completeness for $W[P]$ and PSPACE analogues," Technical report DCS-216-IR, University of Victoria, 1993.

[2] C. Álvarez, J. Díaz, and J. Torán, "Complexity classes with complete problems between *P* and *NP*-complete," in *Foundations of Computation Theory,* Lecture Notes in Computer Science 380, Springer-Verlag, 1989, pp. 13-24.

[3] D. Mix Barrington, N. Immerman, and H. Straubing, "On uniformity in NC^1, *J. Comput. System Sci.* 41 (1990) 274–306.

[4] R. Beigel and J. Goldsmith, "Downward separation fails catastrophically for limited nondeterminism classes," in *Proceedings of the Ninth Annual Structure in Complexity Theory Conference*, IEEE Computer Society Press, 1994, pp. 134–138.

[5] S. Bellantoni, *Predicative Recursion and Computational Complexity.* Ph.D. thesis, University of Toronto, 1992.

[6] S. Bellantoni, "Predicative recursion and the polytime hierarchy," in P. Clote and J.B. Remmel, eds., *Feasible Mathematics II*, Birkhäuser, 1995, pp. 15–29.

[7] S. Bellantoni and S. Cook, "A new recursion-theoretic characterization of the polytime functions," *Computational Complexity* 2 (1992) 97–110.

[8] J.H. Bennett, *On Spectra*, Ph.D. thesis, Department of Mathematics, Princeton University, 1962.

[9] S.A. Bloch, "Alternating function classes within P," University of Manitoba Computer Science Dept. Technical Report 92–16, December 1992.

[10] S.A. Bloch, "Function-algebraic characterizations of log and polylog parallel time," *Computational Complexity* 4 (1994) 175–205. See also *Proceedings of the Seventh Annual Structure in Complexity Theory Conference*, IEEE Computer Society Press, 1992, pp. 193–206.

[11] J.F. Buss, "Relativized Alternation and Space-Bounded Computation," *J. Comput. System Sci.* 36 (1988) 351–378.

[12] J.F. Buss and J. Goldsmith, "Nondeterminism within P," *SIAM J. Comput.* 22 (1993) 560–572.

[13] S.R. Buss, *Bounded Arithmetic,* Bibliopolis, Naples, 1986.

[14] L. Cai and J. Chen, "On the amount of nondeterminism and the power of verifying," *SIAM J. Computing*, to appear. See also *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 711, Springer-Verlag, 1993, pp. 311–320.

[15] A. Chandra, D. Kozen, and L. Stockmeyer, "Alternation," *J. Assoc. Comput. Mach.* 28 (1981) 114-133.

[16] P. Clote, "Sequential, machine-independent characterizations of the parallel complexity classes *ALOGTIME*, AC^k, NC^k, and *NC*," in S. Buss and P. Scott, Eds., *Feasible Mathematics,* Perspectives in Computer Science, Birkhäuser, 1990.

[17] A. Cobham, "The intrinsic computational complexity of functions," in Y. Bar-Hillel, ed., *Logic, Methodology, and the Philosophy of Science: Proceedings of the 1964 International Congress*, North-Holland, 1965, pp. 24–30.

[18] A. Condon, "A theory of Strict *P*-completeness," *Computational Complexity* (1994).

[19] S.A. Cook, "Computational complexity of higher-type functions," in I. Satake, ed., *Proc. International Congrress of Mathematicians*, Springer-Verlag, 1990, pp. 55–69.

[20] J. Díaz and J. Torán, "Classes of bounded nondeterminism," *Math. Systems Theory* 23 (1990) 21–32.

[21] R.G. Downey and M.R. Fellows, "Fixed-parameter intractability," in *Seventh Annual IEEE Conference on Structure in Complexity Theory,* IEEE Computer Society Press, 1992, pp. 36–49.

[22] J. Edmonds, "Paths, trees, and flowers," *Canadian J. Math.* **17** (1965), 449–67.

[23] N. Pippenger and M.J. Fischer, "Relations among complexity measures", *J. Assoc. Comput. Mach.* 26 (1979) 361–381.

[24] J. Girard, A. Scedrov, and P. Scott, "Bounded linear logic: a modular approach to polynomial time computability," in S. Buss and P. Scott, eds., *Feasible Mathematics,* Perspectives in Computer Science, Birkhäuser, 1990, pp. 195–207.

[25] E. Grädel, "On the notion of linear-time computability," *Internat. J. Foundat. Comput. Sci.* 1 (1990) 295–307.

[26] E. Grädel and G.L. McColm, "Hierarchies in transitive-closure logic, stratified Datalog, and infinitary logic," in *33rd Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, 1992, pp. 167–176. Also *Ann. Pure Appl. Math.*, to appear.

[27] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo, *Limits to Parallel Computation: P-Completeness Theory,* Oxford University Press, 1995.

[28] Y. Gurevich, "Algebras of feasible functions," *24th Annual Symposium on Foundations of Computer Science,* IEEE Computer Society Press, 1983, pp. 210–214.

[29] Y. Gurevich, "Logic and the challenge of computer science," in E. Börger, ed., *Current Trends in Theoretical Computer Science,* Computer Science Press, 1987, pp. 1–57.

[30] Y. Gurevich and S. Shelah, "Nearly linear time," in A.R. Meyer and M.A. Taitslin, eds., *Logic at Botik '89,* Lecture Notes in Computer Science 363, Springer-Verlag, 1989, pp. 108–118.

[31] Y. Gurevich and S. Shelah, "Fixed-point extensions of first-order logic," *Annals of Pure and Applied Logic* **32** (1986), 265–280.

[32] L. Hemachandra and S. Jha, "Defying upward and downward separation," in *STACS 93: Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 665, Springer-Verlag, 1993, pp. 185–195.

[33] N. Immerman, "Relational queries computable in polynomial time," *Inform. and Control* 68 (1986) 86–104.

[34] N. Immerman, "Languages that capture complexity classes," *SIAM J. Comput.* 16 (1987) 760–778.

[35] N. Immerman, "Expressibility and parallel complexity," *SIAM J. Comput.* 18 (1989) 625–638.

[36] N.D. Jones and W.T. Laaser, "Complete problems for deterministic polynomial time," *Theoretical Computer Science* 3 (1976) 105–117.

[37] R.M. Karp, "Reducibility among combinatorial problems," in R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations,* Plenum Press, New York, 1972, pp. 85–104.

[38] C.M.R. Kintala and P.C. Fischer, "Refining nondeterminism in relativized polynomial-time bounded computations," *SIAM J. Comput.* 9 (1980) 46–53.

[39] D. Leivant, "Subrecursion and lambda representation over free algebras," in S.R. Buss and P. Scott, eds., *Feasible Mathematics*, Perspectives in Computer Science, Birkhäuser, 1990, pp. 281–291.

[40] D. Leivant, "A foundational delineation of computational feasibility," *Information and Control* 110 (1994).

[41] D. Leivant, "Inductive definitions over finite structures," *Information and Computation* 89 (1990) 95–108.

[42] N. Linial, Y. Mansour, and R.L. Rivest, "Results on learnability and the Vapnik-Chervonenkis dimension," *Information and Control* 90 (1991) 33–49.

[43] A.V. Naik, K.W. Regan, and D. Sivakumar, "Quasilinear-time complexity theory," *Theoretical Computer Science,* to appear.

[44] J. Otto, "Tiers, tensor, and the linear-time hierarchy," in *Proceedings of Workshop on Logic and Computational Complexity*, Lecture Notes in Computer Science, Springer-Verlag, 1995.

[45] C.H. Papadimitriou, "A note on the expressive power of PROLOG," *Bull. EATCS* 26 (June 1985) 21–23.

[46] C.H. Papadimitriou, A.A. Schäffer, and M. Yannakakis, "Simple local search problems that are hard to solve," *SIAM J. Computing* 20 (1991) 56–87.

[47] N. Pippenger, "Fast simulation of combinational logic circuits by machines without random-access storage," in F.P. Preparata and M.B. Pursley, eds., *Fifteenth Allerton Conference on Communication, Control, and Computing,* 1977, pp. 25–33.

[48] K. Regan, "Linear time and memory-efficient computation," technical report 92-28, Department of Computer Science, SUNY Buffalo, 1992.

[49] C.P. Schnorr, "Satisfiability is quasilinear complete in *NQL*," *J. Assoc. Comput. Mach.* 25 (1978) 136–145.

[50] V. Vapnik and A.Y. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and Its Applications* 16 (1971) 264–280.

[51] M. Vardi, "Complexity and relational query languages," in *Proc. Fourteenth Annual ACM Symposium on Theory of Computing,* 1982, pp. 137–146.

[52] C. Wrathall, "Rudimentary predicates and the linear-time hierarchy," *SIAM J. Comput.* 7 (1978) 194–209.