Theoretical Computer Science 3 (1977) 1–22.

THE POLYNOMIAL-TIME HIERARCHY*

Larry J. STOCKMEYER

Mathematical Sciences Department, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y. 10598, U.S.A.

Communicated by A. Meyer
Received April 1975

The polynomial-time hierarchy is that subrecursive analog of the Kleene arithmetical hierarchy in which deterministic (nondeterministic) polynomial time plays the role of recursive (recursively enumerable) time. Known properties of the polynomial-time hierarchy are summarized. A word problem which is complete in the second stage of the hierarchy is exhibited. In the analogy between the polynomial-time hierarchy and the arithmetical hierarchy, the first order theory of equality plays the role of elementary arithmetic (as the ω-jump of the hierarchy). The problem of deciding validity in the theory of equality is shown to be complete in polynomial-space, and close upper and lower bounds on the space complexity of this problem are established.

1. Introduction

One goal of complexity theory is to be able to characterize the amount of computational resource (for example, time or space) required to solve specific problems. For a number of well-known problems, there are presently large gaps between known upper and lower bounds on the requisite time and space. For example, it is not known if there exists an algorithm which recognizes the satisfiable Boolean formulas and which runs within time bounded by a polynomial in the length of the input.

Failing to find close explicit upper and lower bounds on the complexity of a given problem, one would nonetheless like to classify the problem as being *complete* in some larger class of problems; see, for example, [2, 6, 8, 12, 13, 15, 24, 25]. Such a result relates the complexity of the particular problem to that of the larger class as a whole. For example, since Cook [6] has shown that the set of satisfiable Boolean formulas is complete in $\mathcal{NP}$ with respect to polynomial-time reducibility, it follows that the set of satisfiable formulas can be recognized in deterministic polynomial time if and only if $\mathcal{P} = \mathcal{NP}$ (where $\mathcal{P}$ ($\mathcal{NP}$) is the class of languages recognizable by deterministic (nondeterministic) Turing machines within time polynomial in the

*Portions of this work were reported previously in [17, 25]. This work was supported in part by a National Science Foundation Graduate Fellowship and by NSF Grant GJ-34671.

length of the input). The large number of common computational problems which are complete in $\mathcal{NP}$ testify to the importance of this question of whether $\mathcal{P} = \mathcal{NP}$.

One purpose of the current work is to describe and summarize several basic properties of a "hierarchy" of classes of sets called the *polynomial-time hierarchy* (*$\mathcal{P}$-hierarchy*). Briefly, the $\mathcal{P}$-hierarchy is that subrecursive analog of the Kleene arithmetical hierarchy (see [18]) in which deterministic polynomial time plays the role of recursive time. We are interested in studying the $\mathcal{P}$-hierarchy mainly because we feel that it will prove technically useful in classifying (by completeness results) certain recursive problems just as the arithmetical hierarchy has proven useful in classifying certain nonrecursive problems.

The classes of the $\mathcal{P}$-hierarchy, $\Sigma_k^{\mathcal{P}}$, $\Pi_k^{\mathcal{P}}$ and $\Delta_k^{\mathcal{P}}$ for integer $k \geq 0$, can be defined as follows (complete definitions appear in Section 3): $\Sigma_0^{\mathcal{P}} = \Pi_0^{\mathcal{P}} = \Delta_0^{\mathcal{P}} = \mathcal{P}$; $\Sigma_{k+1}^{\mathcal{P}}$ ($\Delta_{k+1}^{\mathcal{P}}$) is the class of sets accepted within polynomial time by nondeterministic (deterministic) Turing machines with oracles (cf. [4, 6]) for sets in $\Sigma_k^{\mathcal{P}}$; and $\Pi_{k+1}^{\mathcal{P}}$ is the class of sets whose complements are in $\Sigma_{k+1}^{\mathcal{P}}$. In particular, note that $\Sigma_1^{\mathcal{P}} = \mathcal{NP}$, and $\Sigma_k^{\mathcal{P}} \cup \Pi_k^{\mathcal{P}} \subseteq \Delta_{k+1}^{\mathcal{P}} \subseteq \Sigma_{k+1}^{\mathcal{P}} \cap \Pi_{k+1}^{\mathcal{P}}$. The relation "set A is accepted within polynomial time by a nondeterministic (deterministic) Turing machine with oracle B" is the analog of the relation "A is r.e. (recursive) in B" in the arithmetical hierarchy.

For example, one problem for which the $\mathcal{P}$-hierarchy might provide a precise classification is the classical minimization problem for Boolean formulas (see [10]). This problem can be cast as a language recognition problem by defining MIN to be the set of all pairs (F, k) such that F is (the encoding of) a Boolean formula in disjunctive normal form, k is (the binary representation of) an integer, and there is a disjunctive form formula G equivalent to F, such that G contains k or fewer occurrences of variables. It is not hard to see that $\text{MIN} \in \Sigma_2^{\mathcal{P}}$. Somewhat informally, this is true because, given an input (F, k), a nondeterministic Turing machine can "guess" a Boolean formula G containing at most k occurrences of variables, and consult an oracle to determine whether or not the formula $(F \leftrightarrow G)$ is a tautology. It is easy to see that the set of nontautologous formulas belongs to $\mathcal{NP} = \Sigma_1^{\mathcal{P}}$ (see [6]) and therefore MIN belongs to $\Sigma_2^{\mathcal{P}}$.

It is not known however if $\text{MIN} \in \Sigma_1^{\mathcal{P}} \cup \Pi_1^{\mathcal{P}}$. Indeed, it is not known if $\Sigma_1^{\mathcal{P}} \neq \Sigma_2^{\mathcal{P}}$, or if $\Sigma_k^{\mathcal{P}} \neq \Sigma_{k+1}^{\mathcal{P}}$ for any $k \geq 0$. However, one result (Theorem 3.2) states that if the hierarchy collapses at the kth stage (i.e., $\Sigma_k^{\mathcal{P}} = \Sigma_{k+1}^{\mathcal{P}}$), then it collapses entirely above the kth stage (i.e., $\Sigma_k^{\mathcal{P}} = \Sigma_j^{\mathcal{P}}$ for all $j \geq k$). In particular, if $\Sigma_k^{\mathcal{P}} \neq \mathcal{P}$ for some $k \geq 1$, then $\mathcal{P} \neq \mathcal{NP}$. By ostensibly enlarging the class of sets B for which $B \notin \mathcal{P}$ implies $\mathcal{P} \neq \mathcal{NP}$, the $\mathcal{P}$-hierarchy may be useful in settling this question.

In Section 4, we consider the existence of complete sets in the $\mathcal{P}$-hierarchy. Although we do not know if MIN is complete in $\Sigma_2^{\mathcal{P}}$, we do exhibit a simple explicit word problem (the inequivalence problem for integer expressions) for which the $\mathcal{P}$-hierarchy provides a precise classification; this problem is shown to be complete in $\Sigma_2^{\mathcal{P}}$ (with respect to *logspace*-reducibility [12, 25]).

The final two sections, 5 and 6, are devoted to the second main purpose of this

paper, to provide new examples of sets which are complete in $\mathcal{P}$-SPACE. ($\mathcal{P}$-SPACE is the class of languages recognizable by deterministic Turing machines within polynomial space.) In fact, these results reveal an interesting relationship between the $\mathcal{P}$-hierarchy and polynomial space, since these complete sets can be viewed as the natural analog for the $\mathcal{P}$-hierarchy of $\emptyset^{(\omega)}$ (the ω-jump of $\emptyset$) for the arithmetical hierarchy. One of these sets, denoted 1EQ, is the set of sentences which are valid in the first-order theory of equality. We show that 1EQ is complete in $\mathcal{P}$-SPACE with respect to *logspace*-reducibility. Although it is not known if 1EQ can be recognized in deterministic polynomial time, a consequence of completeness in $\mathcal{P}$-SPACE is that $1EQ \in \mathcal{P}$ iff $\mathcal{P} = \mathcal{P}$-SPACE. We also establish a lower bound of $cn^{1/2}$ on the space required (infinitely often) to recognize 1EQ.

2. Preliminaries

Familiarity with the basic concepts of formal language theory and automata theory is assumed; see, for example, [11]. A more detailed discussion of this preliminary material can be found in [24, Chaps. 2, 3]. We only outline the necessary concepts here.

If Θ is a finite alphabet, Θ^* denotes the set of all words over Θ including the empty word λ. $\Theta^+ = \Theta^* - \{\lambda\}$. $|\omega|$ denotes the length of the word ω. N denotes the nonnegative integers. If $k \in \mathbf{N}$, $\Theta^k = \{\omega \in \Theta^* : |\omega| = k\}$. Most of the computational problems considered here concern the recognition of a set of words (or *language*) $A \subseteq \Theta^+$, for some finite alphabet Θ. If $A \subseteq \Theta^+$, then $\bar{A} = \Theta^+ - A$, the alphabet Θ being clear from context. If $\mathcal{C}$ is a class of languages, then co-$\mathcal{C} = \{A : \bar{A} \in \mathcal{C}\}$.

Our models of computation are nondeterministic and deterministic Turing machines [11] which have, in addition to the read/write *work tapes*, an *input tape* and an *output tape*. The input tape is scanned by a 2-way read-only head, and the output tape by a 1-way write-only head. Call these models simply *Turing machines*. When used for language recognition, certain states of the machine are designated as *accepting* states. An *accepting computation* of a Turing machine M on an input word x is a computation of M which starts with the word x written on the input tape with all work tapes blank, and terminates in an accepting state. The *time* of a computation is its length. The *space* of a computation is the number of tape squares visited during the computation by heads on the work tapes.

Precise definitions are not given here since our results are invariant under the differences in the definitions of Turing machines and their computations found in the literature, for example, [2, 11, 24]. In fact, since we specify time bounds only to within a polynomial, and space bounds only to within a constant factor, our results remain true with respect to a wide variety of machine models, including random access register machines [2, 7]. See [2, 7] for relationships between Turing machines and other machine models.

Definition. Let Θ be a finite alphabet, $A \subseteq \Theta^+$, M be a Turing machine, and $F: \mathbf{N} \to \mathbf{N}$. *$M$ accepts A within time $F(n)$ (within space $F(n)$)* iff for all $x \in \Theta^+$:

(i) If $x \in A$ then there is an accepting computation of M on input x such that the time (space) of the computation does not exceed $F(|x|)$; and

(ii) if $x \notin A$ then there is no accepting computation of M on input x.

Similarly, for finite alphabets Θ and Δ, a deterministic Turing machine *computes* the function f, $f: \Theta^+ \to \Delta^+$, *within time (space) $F(n)$* iff for each input $x \in \Theta^+$: (i) the time (space) of the unique computation of M on input x does not exceed $F(|x|)$; and (ii) the computation produces $f(x)$ on the output tape and then halts.

Let NTIME($F(n)$) (DTIME($F(n)$)) be the class of languages accepted by nondeterministic (deterministic) Turing machines within time $F(n)$. Let NSPACE($F(n)$) (DSPACE($F(n)$)) be the class of languages accepted by nondeterministic (deterministic) Turing machines within space $F(n)$. In particular, let

$$\mathscr{P} = \bigcup_{k=1}^{\infty} \text{DTIME}(n^k); \qquad \mathscr{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k);$$

$$\mathscr{P}\text{-SPACE} = \bigcup_{k=1}^{\infty} \text{DSPACE}(n^k).$$

Let *logspace* be the class of functions computable by deterministic Turing machines within space $\log n$. The base of the logarithm is immaterial to our discussion; for definiteness it is convenient to define $\log n = \lceil \log_2 n \rceil$ if $n > 1$, and $\log 1 = 1$, where $\lceil r \rceil$ is the least integer not less than r.

The following type of efficient reducibility plays a key role in the sequel. Since this reducibility corresponds to the "many-one" reducibilities of recursive function theory, following a definitional suggestion of Knuth [14] we use the term *transformation* for functions which reduce one set to another.

Definition. Let $A \subseteq \Theta^+$ and $B \subseteq \Delta^+$ for finite alphabets Θ and Δ. *A transforms to B within logspace via f* ($A \leqslant_{\log} B$ *via* f) iff f is a transformation, $f: \Theta^+ \to \Delta^+$, such that $f \in$ *logspace* and $x \in A \leftrightarrow f(x) \in B$ for all $x \in \Theta^+$.

Remark. Since *logspace* is closed under composition [12, 25], $\leqslant_{\log}$ is a transitive relation on sets of words. It is well-known that a Turing machine which computes within space $\log n$ also computes within polynomial time; therefore $A \leqslant_{\log} B$ implies A is polynomial-time-transformable to B in the sense of Karp [13]. Lind [16] gives a machine independent characterization of *logspace* by which one can give rigorous proofs of membership in *logspace*. Although, for the transformations described in this paper, we at most sketch proofs of membership in *logspace*.

Definition. Let B be a set and $\mathscr{C}$ be a class of sets.

(1) $\mathscr{C} \leqslant_{\log} B$ iff $A \leqslant_{\log} B$ for all $A \in \mathscr{C}$.

(2) B is *log-complete* in $\mathscr{C}$ iff
  (i) $\mathscr{C} \leqslant_{\log} B$, and
  (ii) $B \in \mathscr{C}$.

3. The polynomial-time hierarchy

In this section, we define the $\mathscr{P}$-hierarchy, summarize several of its known properties, and state some open questions. As outlined in the introduction, the $\mathscr{P}$-hierarchy is defined in terms of polynomial-time bounded oracle machines. Our notion of oracle machine is the query machine of [4, 6]. A *query machine* is a (nondeterministic or deterministic) Turing machine with a distinguished work tape called the *query tape*, and three distinguished states, the *query state*, the *yes state*, and the *no state*. The computations of a query machine depend not only on the input, but also on a given set of words called the *oracle*. The actions of a query machine with oracle B are identical to those of Turing machines with one exception. If the machine enters its query state at some step, the machine next enters its *yes* state if the nonblank portion of the query tape contains a word in B; otherwise the machine next enters its *no* state. An oracle machine M *operates within time* $T(n)$ iff, for every input x, every computation of M (relative to any oracle) halts within $T(|x|)$ steps. Let $M(B)$ denote the language accepted by oracle machine M with oracle B. (The definition of acceptance for oracle machines is analogous to that for ordinary Turing machines.) The following notation of [27] defines the analogs in the $\mathscr{P}$-hierarchy of the relations "recursive in" and "r.e. in" in the arithmetical hierarchy.

Definition. (a) Let B be a set of words.

$$\mathscr{P}(B)\ (\mathscr{NP}(B)) = \{M(B)\colon M \text{ is a deterministic (nondeterministic) oracle machine which operates within time } p(n) \text{ for some polynomial } p(n)\}.$$

(b) For a class of sets $\mathscr{C}$,

$$\mathscr{P}(\mathscr{C}) = \bigcup_{B \in \mathscr{C}} \mathscr{P}(B), \qquad \mathscr{NP}(\mathscr{C}) = \bigcup_{B \in \mathscr{C}} \mathscr{NP}(B).$$

The polynomial-time hierarchy was defined previously by Meyer and the author in [17]; the notion was also known to Karp [13].

Definition. The *polynomial-time hierarchy* ($\mathscr{P}$-*hierarchy*) is $\{\Sigma_k^{\mathscr{P}}, \Pi_k^{\mathscr{P}}, \Delta_k^{\mathscr{P}}\colon k \geqslant 0\}$, where

$$\Sigma_0^{\mathscr{P}} = \Pi_0^{\mathscr{P}} = \Delta_0^{\mathscr{P}} = \mathscr{P};$$

and for $k \geq 0$,

$$\Sigma_{k+1}^p = \mathcal{NP}(\Sigma_k^p),$$

$$\Pi_{k+1}^p = \text{co-}\mathcal{NP}(\Sigma_k^p),$$

$$\Delta_{k+1}^p = \mathcal{P}(\Sigma_k^p).$$

Also define $\mathcal{PH} = \bigcup_{k=0}^{\infty} \Sigma_k^p$.

In particular, note that $\Sigma_1^p = \mathcal{NP}$ and $\Pi_1^p = \text{co-}\mathcal{NP}$. Since obviously $B \in \mathcal{P}(B)$ and $\mathcal{P}(B) \subseteq \mathcal{NP}(B) \cap \text{co-}\mathcal{NP}(B)$ for any set B, the $\mathcal{P}$-hierarchy possesses the following inclusion structure:

$$\Sigma_k^p \cup \Pi_k^p \subseteq \Delta_{k+1}^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p \quad \text{for all } k \geq 0.$$

It is not known however if these inclusions are proper. The following questions are open:

(i) Does $\Sigma_k^p \neq \Sigma_{k+1}^p$ for all $k \geq 0$?

(ii) Does $\Sigma_k^p \neq \Pi_k^p$ for all $k \geq 1$?

(iii) Does $\Delta_k^p \neq \Sigma_k^p \cap \Pi_k^p$ for all $k \geq 1$?

Considering the analogs of these questions for the arithmetical hierarchy [18], (i) and (ii) are true, while (iii) is false (taking the analog of Δ_k^p to be the class of sets recursive in B for some $B \in \Sigma_{k-1}$).

The $\mathcal{P}$-hierarchy can be characterized as those languages definable by polynomial bounded quantification over the variables of relations in $\mathcal{P}$. This characterization further illuminates the analogy between the $\mathcal{P}$-hierarchy and the arithmetical hierarchy, and facilitates the proofs of certain properties of the $\mathcal{P}$-hierarchy.

An *n-ary relation* on words is a subset of $\Theta^+ \times \Theta^+ \times \ldots \times \Theta^+$ (n times) for some finite alphabet Θ. If R is such a relation and $\mathcal{C}$ is a class of languages, then by $R \in \mathcal{C}$ we mean that the language $\{x_1 \# x_2 \# \ldots \# x_n : R(x_1, x_2, \ldots, x_n)\}$ belongs to $\mathcal{C}$ for some new symbol $\# \notin \Theta$.

Theorem 3.1 (see [25]). *Let Θ be a finite alphabet and $A \subseteq \Theta^+$. $A \in \Sigma_k^p$ iff there is a polynomial $p(n)$, an alphabet Γ, and a $(k+1)$-ary relation $R \in \mathcal{P}$ such that for all $x \in \Theta^+$,*

$$x \in A \quad \text{iff} \quad (\exists y_1)(\forall y_2)(\exists y_3)\ldots(Q_k y_k)[R(x, y_1, y_2, \ldots, y_k)],$$

where the quantifiers alternate (so Q_k is $\exists$ ($\forall$) if k is odd (even)), and $y_1, \ldots, y_k$ range over all words in Γ^+ of length not exceeding $p(|x|)$.

Similarly, $A \in \Pi_k^p$ iff for all x,

$$x \in A \quad \text{iff} \quad (\forall y_1)(\exists y_2)(\forall y_3)\ldots(Q_k' y_k)[R(x, y_1, y_2, \ldots, y_k)],$$

(so Q_k' is $\forall$ ($\exists$) if k is odd (even)).

See [27] for a proof of Theorem 3.1.

As mentioned above, we have not been able to prove proper inclusion between successive classes of the $\mathcal{P}$-hierarchy. However, proper inclusion between *some* pair of successive classes implies $\mathcal{P} \neq \mathcal{NP}$ as the following shows.

Theorem 3.2. *Assume $\Sigma_k^p = \Pi_k^p$ for some $k \geq 1$. Then $\Sigma_j^p = \Pi_j^p = \Sigma_k^p$ for all $j \geq k$.*

Proof. Assume $\Sigma_k^p = \Pi_k^p$ for some $k \geq 1$. The proof is by induction on j. The basis $j = k$ is immediate. Assume for induction that $\Sigma_{j-1}^p = \Pi_{j-1}^p = \Sigma_k^p$ for some $j > k$. We show that $\Sigma_j^p \subseteq \Sigma_k^p$ and thus $\Sigma_j^p = \Sigma_k^p$. Let $A \in \Sigma_j^p$. The following is an easy consequence of Theorem 3.1: There is a 2-ary relation $R \in \Pi_{j-1}^p$ and a polynomial p such that for all words x,

$$x \in A \quad \text{iff} \quad (\exists y)[|y| \leq p(|x|) \text{ and } R(x, y)].$$

By induction, we have $R \in \Sigma_k^p$. Now $A \in \Sigma_k^p$ because, for $k \geq 1$, Σ_k^p is closed under the operation of polynomial-bounded existential quantification over variables of relations [27, Proposition 2]. This establishes $\Sigma_j^p = \Sigma_k^p$; thus also $\Pi_j^p = \Pi_k^p$ by definition. □

Corollary 3.3. (i) *If $\Sigma_0^p \neq \Sigma_k^p$ for some $k \geq 1$, then $\mathcal{P} \neq \mathcal{NP}$.*

(ii) *If $\{\Sigma_k^p: k \geq 0\}$ contains infinitely many distinct classes, then $\Sigma_k^p \neq \Sigma_{k+1}^p$ for all $k \geq 0$.*

An interesting open question is whether $\mathcal{P} \neq \mathcal{NP}$ implies $\Sigma_k^p \neq \Sigma_{k+1}^p$ for all k. Theorem 3.2 does not contradict the possibility that $\mathcal{P} \neq \mathcal{NP}$ but the $\mathcal{P}$-hierarchy contains only a finite number of distinct classes.

Regarding these questions, it is important to note the work of Baker et al [4]. Following a suggestion of Meyer, they consider the $\mathcal{P}$-hierarchy "relativized" to arbitrary sets. If X is a set of words, they define

$$\Sigma_0^{p,X} = \Pi_0^{p,X} = \mathcal{P}(X), \qquad \Sigma_{k+1}^{p,X} = \mathcal{NP}(\Sigma_k^{p,X}), \qquad \Pi_{k+1}^{p,X} = \text{co-}\mathcal{NP}(\Sigma_k^{p,X}).$$

For each X, Theorem 3.2 holds for the $\mathcal{P}$-hierarchy relativized to X. The relativized $\mathcal{P}$-hierarchy may extend either zero levels, exactly one level, or at least two levels above $\Sigma_0^{p,X}$ depending on which X is chosen [4]; that is, there exist recursive sets A, B and C such that

$$\Sigma_0^{p,A} = \Sigma_1^{p,A}, \qquad \Sigma_0^{p,B} \neq \Sigma_1^{p,B} = \Pi_1^{p,B}, \qquad \Sigma_0^{p,C} \neq \Sigma_1^{p,C} \neq \Sigma_2^{p,C}.$$

However, Baker et al. [4] leave open the following question: Is there a set X such that $\Sigma_k^{p,X} \neq \Sigma_{k+1}^{p,X}$ for all $k \geq 0$?

Remark. There is a natural correspondence between $\mathcal{PH}$ and the sets of finite structures of formulas written in second-order predicate calculus. This extends the correspondence shown by Fagin [9] between $\mathcal{NP}$ and *generalized spectra*. For

$k \geq 1$, define an *open second-order k-formula* to be a formula σ written in second-order predicate calculus such that σ is in prenex normal form with all second-order quantifiers preceding all first-order quantifiers, σ has $k-1$ alternations of second-order quantifiers with leading quantifier existential, and σ contains at least one free predicate variable and contains no free first-order variable. For $k \geq 1$, let GS_k be the set of $\mathcal{A}$ such that, for some open second-order k-formula σ, $\mathcal{A}$ is the set of finite structures in which σ is true. Fagin [9] describes an encoding E of sets of finite structures to languages, and proves that $\mathcal{A} \in GS_1$ iff $E(\mathcal{A}) \in \mathcal{NP}$. Combining this result and Theorem 3.1, it is easy to show that, for each k, $\mathcal{A} \in GS_k$ iff $E(\mathcal{A}) \in \Sigma_k^p$. It is also true that, for each $k \geq 1$, there is an $\mathcal{A} \in GS_k$ such that $E(\mathcal{A})$ is log-complete in Σ_k^p. In particular, it then follows that, for each k, $\Sigma_k^p \neq \Sigma_{k+1}^p$ iff $GS_k \neq GS_{k+1}$.

We close this section by noting the following upper bound on the complexity of sets in the $\mathcal{P}$-hierarchy.

Theorem 3.4. $\mathcal{PH} \subseteq \mathcal{P}$-SPACE.

Proof. As Baker et al. [4] point out, $\mathcal{NP}(\mathcal{P}\text{-SPACE}) = \mathcal{P}$-SPACE is an immediate consequence of the result of Savitch [19] that $NSPACE(S(n)) \subseteq DSPACE((S(n))^2)$. It then follows by induction on k that $\Sigma_k^p \subseteq \mathcal{P}$-SPACE for all k. □

It is not known if the containment of Theorem 3.4 is proper. However, Wrathall [27] observes that if $\mathcal{PH} = \mathcal{P}$-SPACE then the $\mathcal{P}$-hierarchy is finite.

4. Complete sets

A variety of well-known combinatorial optimization problems and decision problems from automata theory and logic have been classified as being complete in certain complexity classes such as $\mathcal{NP}$ and $\mathcal{P}$-SPACE with respect to efficient transforms such as $\leq_{log}$ [6, 8, 13, 15, 24, 25]. Even though it is not known whether $\Sigma_k^p \neq \Sigma_{k+1}^p$, we feel that the $\mathcal{P}$-hierarchy will prove technically useful in the classification of other problems. For example, as mentioned in the introduction, the minimization problem for Boolean formulas (viewed as a recognition problem) belongs to Σ_2^p, but is not known to belong to $\mathcal{NP} \cup$ co-$\mathcal{NP}$. Possibly the minimization problem is log-complete in Σ_2^p, and this remains an intriguing open question. The main purpose of this section is to exhibit a simply defined word problem, the inequivalence problem for integer expressions (defined below), which is log-complete in Σ_2^p.

First, we note that for each k there is a set which is log-complete in Σ_k^p. Viewing

the satisfiability problem for Boolean formulas as a question involving existential quantification over the variables in formulas, these complete sets are the extensions of the satisfiability problem to alternating quantifiers.

We are concerned with Boolean formulas involving doubly subscripted *variable symbols* x_{ij} for $i, j \geq 1$, *constant symbols* 0 and 1, *operation symbols* $\sim$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and parentheses. Assume these symbols are distinct.

Definition. (i) 0, 1, and x_{ij} for $i, j \geq 1$ are *Boolean formulas.*

(ii) If F and G are Boolean formulas, and $\varrho \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, then $\sim F$ and $(F \varrho G)$ are *Boolean formulas.*

(iii) Nothing is a Boolean formula unless implied by (i) or (ii).

If V is a set of variable symbols, a *V-assignment* is a mapping from V to $\{0, 1\}$. If F is a Boolean formula containing the set V of variable symbols, then F defines, in the obvious way, a function mapping V-assignments to $\{0, 1\}$; $\sim$, $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$ are interpreted as the Boolean operations complementation, conjunction, disjunction, implication and equivalence, respectively; and 0 and 1 are interpreted as Boolean constants. A *literal* is either x or $\sim x$ where x denotes a variable symbol. A Boolean formula F is in *conjunctive normal form* iff F is a conjunction of disjunctions of literals. F is in *disjunctive normal form* iff F is a disjunction of conjunctions of literals. Let CNF (DNF) denote the set of Boolean formulas in conjunctive (disjunctive) normal form. Let 3CNF denote the set of $F \in$ CNF such that F is $C_1 \wedge C_2 \wedge \ldots \wedge C_m$ where, for $1 \leq i \leq m$, C_i is a disjunction of at most three literals. 3DNF is defined dually. (When writing formulas within the text, parentheses are deleted when not needed to determine the precedence of operations.) In describing Boolean formulas, we write X_i for the sequence of variable symbols $x_{i1}, x_{i2}, x_{i3}, \ldots$, $\exists X_i$ for $\exists x_{i1} \exists x_{i2} \ldots$; etc. These abbreviations do not appear in the Boolean formulas themselves. We use the notation $F(X_1, \ldots, X_k)$, $G(X_1, \ldots, X_k)$, etc. to denote a Boolean formula containing no variable symbol x_{ij} with $i > k$.

Boolean formulas are encoded as words over a finite alphabet by encoding variable symbols as words over the alphabet $\{x, 0, 1, \#\}$; x_{ij} is encoded as $x\omega_i \# \omega_j$ where $\omega_i, \omega_j \in \{0, 1\}^+$ are the binary representations of i and j. Where no confusion can arise, we shall identify a Boolean formula with its encoding, and with the Boolean function it defines. The following definition of B_k as a set of words should thus be clear.

Definition. Let $k \geq 1$.

$$B_k = \{F(X_1, \ldots, X_k)\colon F(X_1, \ldots, X_k) \text{ is a Boolean formula, and } (\exists X_1)(\forall X_2)(\exists X_3) \ldots (Q_k X_k)[F(X_1, \ldots, X_k) = 1]\}.$$

Note that B_1 is the set of satisfiable formulas.

Theorem 4.1. *Let $k \geq 1$.*

(1) *B_k is log-complete in Σ_k^p.*

(2) *If k is odd (even), then $B_k \cap$ 3CNF (resp., $B_k \cap$ 3DNF) is log-complete in Σ_k^p.*

Theorem 4.1 was first noted in [17, 25]. A proof of part (1), by a direct "arithmetization" of query machines, is given in [17]. Wrathall [27] gives a simpler proof of Theorem 4.1, using Theorem 3.1.

One value of the sets B_k is to serve as starting points for further transformations. As the work of Karp [13] demonstrates, sets which are known to be complete in a class often facilitate proofs that other sets are complete in that class. For example, to show that $\Sigma_2^p \leq_{\log} A$ for some set A, it suffices to show $B_2 \cap \text{DNF} \leq_{\log} A$ (since $\leq_{\log}$ is transitive). Using $B_2 \cap$ DNF in this way, we prove that the inequivalence problem for integer expressions is log-complete in Σ_2^p. Even though this decision problem may seem somewhat contrived, it does serve to illustrate how one might so utilize $B_2 \cap$ DNF. Possibly this will suggest further examples of problems which are complete in Σ_k^p or Π_k^p for some $k \geq 2$.

Integer expressions are syntactically similar to the familiar regular expressions of finite automata theory [2], but they define sets of nonnegative integers rather than sets of words.

Definition. If $z \in \mathbf{N}$, let $\text{bin}(z) \in \{0,1\}^+$ denote the binary representation of z.

We define the class of integer expressions and simultaneously define the map L which maps integer expressions to subsets of $\mathbf{N}$.

(i) If $z \in \mathbf{N}$, then $\text{bin}(z)$ is an *integer expression*, and $L(\text{bin}(z)) = \{z\}$.

(ii) If E and F are integer expressions, then $(E \cup F)$ and $(E + F)$ are *integer expressions*, and

$$L((E \cup F)) = L(E) \cup L(F),$$

$$L((E + F)) = \{m + n : m \in L(E) \text{ and } n \in L(F)\}.$$

Let N-INEQ $= \{(E, F) : E$ and F are integer expressions and $L(E) \neq L(F)\}$.

Theorem 4.2. N-INEQ *is log-complete in Σ_2^p.*

Proof. (1) N-INEQ $\in \Sigma_2^p$. Given $z \in \mathbf{N}$ and integer expression E, we first define a "proof" that $z \in L(E)$ recursively as follows: $\text{bin}(z)$ is a proof of $(z, \text{bin}(z))$; if P_1 is a proof of (z_1, E_1), P_2 is a proof of (z_2, E_2), and $z = z_1 + z_2$, then $(P_1 + P_2)$ is a proof of $(z, (E_1 + E_2))$; if P_1 is a proof of (z, E), then P_1 is a proof of $(z, (E \cup F))$ and of $(z, (F \cup E))$ for any integer expression F. Let Q be the relation $Q(x, E, P)$ iff $x = \text{bin}(z)$ for some $z \in \mathbf{N}$ and P is a proof of (z, E). It is not hard to see that $Q \in \mathcal{P}$.

By induction on the structure of E, one verifies that: (i) if $z \in L(E)$ then

$|\text{bin}(z)| \leq |E|$; (ii) if P is a proof of (z, E), then $|P| \leq |E|$. Therefore, for all integer expressions E_1 and E_2,

$$(E_1, E_2) \in \text{N-INEQ} \quad \text{iff}$$

$$(\exists x)[(\exists P_1)[Q(x, E_1, P_1)] \leftrightarrow \sim (\exists P_2)[Q(x, E_2, P_2)]],$$

where quantifiers range over words of length $\leq |(E_1, E_2)|$. Standard manipulation of quantifiers and Theorem 3.1 now imply that N-INEQ $\in \Sigma_2^p$.

(2) $\Sigma_2^p \leq_{\log}$ N-INEQ. By Theorem 4.1, it suffices to show that $B_2 \cap \text{DNF} \leq_{\log}$ N-INEQ via some transformation g. We describe a g which accomplishes this.

Let G be a given input. If $G \notin$ DNF or G contains a variable x_{ij} with $i > 2$, then $G \notin B_2 \cap$ DNF, and we can take $g(G) = (0, 0)$. (The test that G is a well-formed Boolean formula can be performed within space $\log(|G|)$ by well-known methods; see [3].)

Thus we may assume $G = G(X_1, X_2) \in$ DNF. For simplicity (and without loss of generality) assume that, for $i = 1, 2$ and $j > 1$, if G contains x_{ij} then G contains $x_{i,j-1}$. Let $n = \max\{j: x_{1j} \text{ or } x_{2j} \text{ appears in } G\}$. Say $G = C_1 \vee C_2 \vee \ldots \vee C_m$, where, for $1 \leq i \leq m$, C_i is a conjunction of literals. We may also assume that, for all i, j and k, C_k does not contain both x_{ij} and $\sim x_{ij}$, and C_k does not contain a constant symbol 0 or 1. If α is a literal, define $[\alpha \in C_k]$ to be 1 if α appears in C_k; and $[\alpha \in C_k]$ to be 0 if α does not appear in C_k. For each literal α define $I(\alpha) \in \mathbf{N}$ to be

$$I(\alpha) = \sum_{k=1}^{m} [\alpha \in C_k] \cdot b^k, \quad \text{where } b = 2^{2+\log n}.$$

Let $a = \Sigma_{k=1}^{m} b^k$, and for $1 \leq k \leq m$ let F_k be the integer expression

$$((\text{bin}(b^k) \cup 0) + (\text{bin}(b^k) \cup 0) + \ldots + (\text{bin}(b^k) \cup 0)),$$

where the term $(\text{bin}(b^k) \cup 0)$ appears $2n - 1$ times. (Unnecessary parentheses have been omitted.)

Finally define integer expressions

$$E_1 = \left(\sum_{j=1}^{n} (\text{bin}(a - I(x_{1j})) \cup \text{bin}(a - I(\sim x_{1j}))) + \text{bin}(na)\right),$$

$$E_2 = \left(\sum_{j=1}^{n} (\text{bin}(I(x_{2j})) \cup \text{bin}(I(\sim x_{2j}))) + \sum_{k=1}^{m} F_k\right).$$

Now note four facts about E_1 and E_2. In (a)–(d) below, corresponding to each $y < b^{m+1}$, let $a_1, a_2, \ldots, a_m$ denote the unique integers such that $y = \Sigma_{k=1}^{m} a_k b^k$ and $0 \leq a_k < b$. For $i = 1, 2$ and $1 \leq k \leq m$, we say that an X_i-assignment *kills* C_k iff for some j either (x_{ij} appears in C_k and x_{ij} is assigned value 0) or ($\sim x_{ij}$ appears in C_k and x_{ij} is assigned value 1).

(a) For each $y \in L(E_1)$, $n \leq a_k \leq 2n$ for $1 \leq k \leq m$; and there is an X_1-assignment such that, for $1 \leq k \leq m$, $a_k = 2n$ iff the assignment does not kill C_k.

(b) For each X_1-assignment there is a $y \in L(E_1)$ such that, for $1 \leq k \leq m$, $a_k = 2n$ iff the assignment does not kill C_k.

(c) For each $y \in L(E_2)$ there is an X_2-assignment such that, for $1 \leq k \leq m$, if $a_k = 2n$ then the assignment kills C_k.

(d) Let A_2 be an X_2-assignment and y be an integer such that $n \leq a_k \leq 2n$, and $a_k = 2n$ implies that A_2 kills C_k, for $1 \leq k \leq m$. Then $y \in L(E_2)$.

For example, to prove (a), choose any "parse" of E_1 as a sum of integers such that the sum equals y. Assign $x_{1j} = 0$ if the integer $a - I(x_{1j})$ is chosen in the parse, or assign $x_{1j} = 1$ if the integer $a - I(\sim x_{1j})$ is chosen in the parse. The proofs of (b), (c) and (d) are similar, and are left to the reader.

We now claim that

$$(\forall X_1)(\exists X_2)[G(X_1, X_2) = 0] \quad \text{iff} \quad L(E_1) \subseteq L(E_2).$$

To verify this, first note that for any $(X_1 \cup X_2)$-assignment, $G(X_1, X_2) = 0$ iff for all k, $1 \leq k \leq m$, the assignment kills C_k. Now the "only if" direction of the claim is easily proved from (a) and (d), while "if" follows from (b) and (c).

Therefore, we finally have

$$\begin{aligned} G(X_1, X_2) \notin B_2 &\leftrightarrow (\forall X_1)(\exists X_2)[G(X_1, X_2) = 0] \\ &\leftrightarrow L(E_1) \subseteq L(E_2) \\ &\leftrightarrow ((E_1 \cup E_2), E_2) \notin \text{N-INEQ}. \end{aligned}$$

The required transformation g maps $G(X_1, X_2)$ to $((E_1 \cup E_2), E_2)$. The reader can verify that $g \in$ *logspace*, which completes the proof. (Cf. the proof of Lemma 6.3 where we argue that a certain transformation belongs to *logspace*. Essentially the same argument applies to this g.) □

5. The ω-jump of the $\mathcal{P}$-hierarchy

Define

$$B_\omega = \bigcup_{k=1}^{\infty} B_k.$$

Since B_k is complete in Σ_k^p, B_ω is the natural analog of $\emptyset^{(\omega)}$ in the arithmetical hierarchy [18]. The next result reveals an interesting relationship between the $\mathcal{P}$-hierarchy and polynomial space.

Theorem 5.1. *B_ω and $B_\omega \cap$ 3CNF are log-complete in $\mathcal{P}$-SPACE.*

Theorem 5.1 is established by Lemmas 5.2, 6.3 and 6.4 proved below.

Apart from its own interest, Theorem 5.1 is useful in showing that other sets are complete in $\mathcal{P}$-SPACE. Since $\leq_{\log}$ is transitive, to show that $\mathcal{P}$-SPACE $\leq_{\log} A$, it

suffices to show that $B_\omega \leqslant_{\log} A$. Thus one can avoid carrying out for each such A the "arithmetization" of Turing machines by which we here show that $\mathcal{P}$-SPACE $\leqslant_{\log} B_\omega$. The sets B_ω and $B_\omega \cap 3\text{CNF}$ have already proven useful in two such applications: Ladner [15] has shown that the validity problem for certain systems of modal logic is log-complete in $\mathcal{P}$-SPACE; and Even and Tarjan [8] have shown that the set of graphs for which the first player has a winning strategy in a "Shannon switching game on vertices" is log-complete in $\mathcal{P}$-SPACE.

It is convenient to prove $\mathcal{P}$-SPACE $\leqslant_{\log} B_\omega$ by modifications to the proof, given in Section 6, that $\mathcal{P}$-SPACE is transformable to the first order theory of equality (Lemma 6.3). For the present, we only observe the following.

Lemma 5.2. $B_\omega \in \text{DSPACE}(n)$.

Proof (sketch). Linear space is obviously sufficient to determine if a given input is (the encoding of) a well-formed Boolean formula. Assume then that $F(X_1, \ldots, X_k)$, some $k \geqslant 1$, is the input of length n. The deterministic Turing machine M checks if $\exists X_1 \forall X_2 \ldots Q_k X_k\, [F(X_1, \ldots, X_k) = 1]$ simply by cycling, in the proper order, through all possible assignments of Boolean values to the variables of F. M operates within space proportional to the space n sufficient to record the assignment. The classical constant factor "speedup" result [23, cf. 11] then gives the conclusion. □

6. The first order theory of equality

The problem of accepting B_ω is computationally very similar to the problem of deciding validity in the first order theory of equality. (In fact, as implied by the results below, these two problems are equivalent with respect to $\leqslant_{\log}$.)

Let $\mathcal{L}_{1\text{EQ}}$ denote the set of formulas written in the first order predicate calculus using only the binary relational symbol $=$, together with the usual logical connectives $\wedge$, $\vee$, $\sim$, $\rightarrow$ and $\leftrightarrow$, quantifiers $\exists$ and $\forall$, variables, and parentheses. We use (multiply) subscripted letters u, v, w, y, z to denote variables. Similar to Section 4, these variables are coded into words over the alphabet $\{u, v, w, y, z, 0, 1, \#\}$; $u_{2,6}$ becoming $u\,10\,\#\,110$, etc. Let $\mathcal{L}_{\text{Pr1EQ}}$ denote the formulas in $\mathcal{L}_{1\text{EQ}}$ which are in prenex normal form. A *sentence* is a formula containing no free variables. Let 1EQ (Pr1EQ) denote the set of sentences in $\mathcal{L}_{1\text{EQ}}$ (resp., $\mathcal{L}_{\text{Pr1EQ}}$) which are valid in [i.e., true in every model for] the theory of equality. Obviously, a model for the theory of equality is completely specified by the cardinality of its domain. See, for example, [1, 20] for further discussion of these terms.

The main result of this section is the following.

Theorem 6.1. 1EQ *and* Pr1EQ *are log-complete in* $\mathcal{P}$-SPACE.

The proof is immediate from Lemmas 6.2 and 6.3 to follow.

It is well-known that 1EQ is decidable because a sentence σ in $\mathcal{L}_{1EQ}$ containing m quantifiers belongs to 1EQ iff σ is true in the models of cardinalities 1 through m (see [1]). (We note that even though this fact is stated in [1] only for prenex sentences, the proof applies to arbitrary sentences as well.) This fact is used in the following.

Lemma 6.2. (1) $1EQ \in DSPACE(n \log n)$;
(2). $Pr1EQ \in DSPACE(n)$.

Proof. (1) Given a sentence $\sigma \in \mathcal{L}_{1EQ}$ such that σ contains m quantifiers, the outer loop of the Turing machine M cycles through the integers $1, 2, 3, \ldots, m$. For a given integer $k \leq m$, M decides whether σ is true in the model of cardinality k. This is done by a straightforward recursive procedure, the details of which are left to the reader. For example, similar to Lemma 5.2, quantifiers (e.g., $(\exists u)F$) are handled by recursively determining the truth of F for each possible assignment of domain element to the variable u. M can be designed to use space proportional to the space sufficient to record an assignment of domain elements to the variables of σ. By encoding domain elements as binary words of length $O(\log m)$, this space is $O(m \log m)$. Certainly $m \leq n$, where n is the length of σ, which, together with "speedup" [23], proves (1).

(2) M also accepts Pr1EQ. However, if σ is in prenex normal form with m quantifiers, we can assume without loss of generality that distinct variables occur in the prefix of σ. Due to the combined lengths of the encodings of these variables, we have $n \geq \sum_{i=1}^{m} \log(ci)$, for some constant $c > 0$, which implies $m = O(n/\log n)$. The space $O(m \log m)$ used by M now becomes $O(n)$. □

We now turn to the main result of this section, that $\mathcal{P}$-SPACE $\leq_{\log}$ Pr1EQ (Lemma 6.3). Within the proof of this result, it is convenient to assume that Turing machines are of a particular simple form described next. A *simple Turing machine* (STM) has one head and one tape. The single tape is one-way infinite to the right. An STM is given input x by writing x left-justified on the otherwise blank tape with the head scanning the leftmost tape square, and the control placed in a unique designated initial state. An STM can accept an input only by entering a unique designated accepting state with the tape entirely blank and the head scanning the leftmost tape square. It is convenient to assume that whenever this unique accepting configuration is entered, the STM continues "running" in this configuration.

The proof of Lemma 6.3 requires a means of formally describing the computations of STM's. If M is an STM with states Q and tape alphabet Γ, then an instantaneous description (i.d.) of M is any word in $\Gamma^* \cdot Q \cdot \Gamma^*$. Informally, if δ is an i.d. of M, say $\delta = yqsz$, where $y, z \in \Gamma^*$, $s \in \Gamma$ and $q \in Q$, we treat δ as

describing the symbols on the tape squares in an interval around the head, with q being the state of the control, and q being positioned in δ immediately to the left of the symbol s being scanned. We associate with M a function

$$\text{Next}_M: \Gamma^* \cdot Q \cdot \Gamma^* \to 2^{\Gamma^* \cdot Q \cdot \Gamma^*}.$$

(2^S denotes the set of subsets of set S.) $\text{Next}_M(\delta)$ is the set of i.d.'s δ' such that $|\delta'| = |\delta|$ and δ can reach δ' in one step of M. (The definition of Next_M can easily be made formal; see [24].) Also define $\text{Next}_M(\delta, 0) = \{\delta\}$, and $\text{Next}_M(\delta, i+1) = \text{Next}_M(\text{Next}_M(\delta, i))$ for $i \in \mathbf{N}$.

The following technical lemma formalizes the assertion that one can determine whether $\delta_2 \in \text{Next}_M(\delta_1)$ by making "local checks" within δ_1 and δ_2. This should be obvious since, in one step, only a few symbols around the state symbol can possibly change.

Lemma 6.3.1 (see [24]). *Let M be a nondeterministic STM with states Q and tape alphabet Γ. Assume $\$ \notin Q \cup \Gamma$. Let $\Sigma = Q \cup \Gamma \cup \{\$\}$. There is a set $X_M \subseteq \Sigma^6$ with the following properties.*

(1) *Let δ_1 be any i.d. of M, let $k = |\delta_1|$, and write $\$\delta_1\$ = \delta_{1,0}\delta_{1,1}\ldots\delta_{1,k}\delta_{1,k+1}$ where $\delta_{1,j} \in \Sigma$ for $0 \leq j \leq k+1$. Let $\$\delta_2\$ = \delta_{2,0}\delta_{2,1}\ldots\delta_{2,k}\delta_{2,k+1}$ where $\delta_{2,j} \in \Sigma$ for $0 \leq j \leq k+1$. Then $\delta_2 \in \text{Next}_M(\delta_1)$ iff $\delta_{1,j-1}\delta_{1,j}\delta_{1,j+1}\delta_{2,j-1}\delta_{2,j}\delta_{2,j+1} \in X_M$ for all $j, 1 \leq j \leq k$.*

(2) *If $\sigma_1\sigma_2\sigma_3\sigma_1'\sigma_2'\sigma_3' \in X_M$, then $\sigma_i = \$$ iff $\sigma_i' = \$$ for $i = 1, 2, 3$.*

See [24] for a proof of Lemma 6.3.1 and further discussion of STM's. It is convenient to summarize our conventions concerning STM's in the following lemma.

Lemma 6.3.2. *Let $A \in \text{NSPACE}(S(n))$. There is a nondeterministic STM M and an $a \in \mathbf{N}$ such that for all x,*

$$x \in A \quad \text{iff} \quad q_1 \#^s \in \text{Next}_M(q_0 x \#^{s-n}, 2^{as}),$$

where $n = |x|$, $s = \max(S(n), n+1)$, $\#$ denotes the blank tape symbol, and q_0 (q_1) denotes the unique initial (accepting) state of M.

Proof (sketch). One readily verifies that there is a nondeterministic STM M which accepts A within space $\max(S(n), n+1)$. Choose the constant a so that, for all l, 2^{al} bounds the number of i.d.'s of M of length $l+1$. The conclusion follows since, for each accepted input x, some computation of M on x must enter $q_1 \#^s$ before looping. □

To obtain Corollary 6.6 below, we must know not only that each A in $\mathcal{P}$-SPACE is transformable to PrIEQ via some $f \in$ *logspace*, but also how $|f(x)|$ grows as a function of $|x|$. The following definitions are therefore useful.

Definition. Let $f: \Theta^+ \to \Delta^+$ and $L: \mathbf{N} \to \mathbf{N}$. f is *length $L(n)$ bounded* iff $|f(x)| \leq L(|x|)$ for all $x \in \Theta^+$.

Let B be a set and $\mathscr{C}$ be a class of sets. $\mathscr{C} \leq_{\log} B$ *via length order* $L(n)$ iff for each $A \in \mathscr{C}$ there is a constant c and an $f \in$ *logspace* such that $A \leq_{\log} B$ via f, and f is length $c \cdot L(n)$ bounded.

Lemma 6.3. *Let d be an integer, $d \geq 1$.*

(1) $\mathrm{NSPACE}(n^d) \leq_{\log} \mathrm{Pr1EQ}$ *via length order* $n^{2d} \log n$.

(2) $\mathrm{NSPACE}(n^d) \leq_{\log} B_\omega$ *via length order* $n^{2d} \log n$.

Proof. (1) Let $A \in \mathrm{NSPACE}(n^d)$ and $A \subseteq \Theta^+$ for an alphabet Θ. For each $x \in \Theta^+$ we describe a sentence $F_x \in \mathscr{L}_{\mathrm{Pr1EQ}}$ such that $x \in A$ iff $F_x \in \mathrm{Pr1EQ}$. Letting f be the function mapping x to F_x for all x, we shall see $f \in$ *logspace* and f is length $cn^{2d} \log n$ bounded for some constant c, thus proving part (1) of the lemma.

Let M be the nondeterministic STM of Lemma 6.3.2. Let Q, Γ and Σ be as in Lemma 6.3.1 for this M. Let $x \in \Theta^+$, $n = |x|$, and $s = \max(n^d, n+1)$.

The sentence F_x involves variables $u_{k,l,\sigma}$, $v_{k,l,\sigma}$, $w_{k,l,\sigma}$, y for $0 \leq k \leq t, 0 \leq l \leq s+2$ and $\sigma \in \Sigma$, where the integer t is specified below. Let U_k abbreviate $\{u_{k,l,\sigma}: \sigma \in \Sigma, 0 \leq l \leq s+2\}$. Similarly, $\exists U_k$ abbreviates the sequence $\{\exists u_{k,l,\sigma}\}$. The notations $\forall U_k$, V_k, $\exists W_k$, etc. are analogous.

A fixed interpretation of a sequence of variables (i.e., U_k, V_k, or W_k for some k) encodes an i.d. of M in a way very similar to [6, cf. 2]. Let D be some domain, and let $u^l_{k,l,\sigma}$, $y^l \in D$ for $\sigma \in \Sigma$ and $0 \leq l \leq s+2$ and k fixed. Let U^l_k denote $\{u^l_{k,l,\sigma}\}$. Let δ be an i.d. of M with $|\delta| = s+1$. (By Lemma 6.3.2, it is sufficient to consider only i.d.'s of length $s+1$.) We say that (U^l_k, y^l) *encodes* δ iff (i) for each l there is exactly one $\sigma \in \Sigma$ such that $u^l_{k,l,\sigma} = y^l$ (let σ_l denote this unique element of Σ); and (ii) $\sigma_0 \sigma_1 \sigma_2 \ldots \sigma_{s+2} = \$\delta\$$.

We construct, for $0 \leq k \leq t$, a formula $F_k(U_k, V_k, y)$ with free variables U_k, V_k and y. Informally, F_k asserts that the i.d. encoded by (V_k, y) follows, in 2^k steps of M, from the i.d. encoded by (U_k, y). More formally, F_k satisfies the following *property* P_k: Let D be a domain of cardinality ≥ 2. Let (U^l_k, y^l) encode an i.d. δ as above. Let $v^l_{k,l,\sigma} \in D$ for $0 \leq l \leq s+2$ and $\sigma \in \Sigma$; let V^l_k denote $\{v^l_{k,l,\sigma}\}$. Then $F_k(U^l_k, V^l_k, y^l)$ is true over D iff (V^l_k, y^l) encodes an i.d. $\delta' \in \mathrm{Next}_M(\delta, 2^k)$. (Here we let $F_k(U^l_k, V^l_k, y^l)$ denote an instance of F_k, with $u^l_{k,l,\sigma}$ replacing occurrences of $u_{k,l,\sigma}$, $v^l_{k,l,\sigma}$ replacing $v_{k,l,\sigma}$, etc.)

The formulas F_k are constructed inductively. First let

$$EU\langle k, l, \sigma\rangle \quad \text{denote} \quad \Big((u_{k,l,\sigma} = y) \wedge \bigwedge_{\tau \neq \sigma} \sim (u_{k,l,\tau} = y)\Big);$$

$$EV\langle k, l, \sigma\rangle \quad \text{denote} \quad \Big((v_{k,l,\sigma} = y) \wedge \bigwedge_{\tau \neq \sigma} \sim (v_{k,l,\tau} = y)\Big).$$

Let X_M be the set of Lemma 6.3.1. Let σ denote $\sigma_1\sigma_2\sigma_3\sigma_1'\sigma_2'\sigma_3'$. Now $F_0(U_0, V_0, y)$ is

$$\bigwedge_{l=1}^{s+1} \bigvee_{\sigma \in X_M} (EU\langle 0, l-1, \sigma_1\rangle \wedge EU\langle 0, l, \sigma_2\rangle \wedge EU\langle 0, l+1, \sigma_3\rangle \wedge EV\langle 0, l-1, \sigma_1'\rangle \wedge EV\langle 0, l, \sigma_2'\rangle \wedge EV\langle 0, l+1, \sigma_3'\rangle).$$

By Lemma 6.3.1, it is clear that F_0 satisfies property P_0.

We now write F_{k+1} using F_k as a subformula. The basic idea, used also by Savitch [19], is that $\delta' \in \mathrm{Next}_M(\delta, 2^{k+1})$ iff there exists an i.d. μ such that both $\mu \in \mathrm{Next}_M(\delta, 2^k)$ and $\delta' \in \mathrm{Next}_M(\mu, 2^k)$. Thus $F_{k+1}(U_{k+1}, V_{k+1}, y)$ could be written in the obvious way as

$$(\exists W_{k+1})(F_k(U_{k+1}, W_{k+1}, y) \wedge F_k(W_{k+1}, V_{k+1}, y)), \tag{1}$$

where $F_k(U_{k+1}, W_{k+1}, y)$ is identical to $F_k(U_k, V_k, y)$ with $w_{k+1,l,\sigma}$ replacing $v_{k,l,\sigma}$ etc. Clearly, the formula (1) satisfies property P_{k+1} if F_k satisfies property P_k. However, to obtain the desired bound on the length of F_x, it is essential that F_{k+1} contain only *one* occurrence of F_k. Therefore we write F_{k+1} as follows. Let

"$U_k = W_{k+1}$" abbreviate $\bigwedge_{l=0}^{s+2} \bigwedge_{\sigma \in \Sigma} (u_{k,l,\sigma} = w_{k+1,l,\sigma})$;

and similarly for "$U_k = U_{k+1}$", etc. $F_{k+1}(U_{k+1}, V_{k+1}, y)$ is the formula

$$(\exists W_{k+1})(\forall U_k)(\forall V_k)(((U_k = U_{k+1} \wedge V_k = W_{k+1}) \vee (U_k = W_{k+1} \wedge V_k = V_{k+1})) \rightarrow F_k(U_k, V_k, y)).$$

F_{k+1}, so written, is equivalent to (1) above. This completes the inductive construction of $F_0, \ldots, F_t$.

Now let a be the constant of Lemma 6.3.2 and let $t = as$, so that M accepts x iff $q_1 \#^s \in \mathrm{Next}_M(q_0 x \#^{s-n}, 2^t)$.

Now construct a formula $\mathrm{Init}_x(U_t, y)$ which asserts that (U_t, y) encodes the initial i.d. $q_0 x \#^{s-n}$, and construct $\mathrm{Acc}(V_t, y)$ to assert that (V_t, y) encodes the accepting i.d. $q_1 \#^s$. Let $x = x_1 x_2 x_3 \ldots x_n$.

$\mathrm{Init}_x(U_t, y)$ is the formula

$$EU\langle t, 0, \$\rangle \wedge EU\langle t, 1, q_0\rangle \wedge \bigwedge_{l=2}^{n+1} EU\langle t, l, x_{l-1}\rangle \wedge \bigwedge_{l=n+2}^{s+1} EU\langle t, l, \#\rangle \wedge EU\langle t, s+2, \$\rangle.$$

$\mathrm{Acc}(V_t, y)$ is written similarly.

Let F_x'' be

$$(\exists y)(\exists U_t)(\exists V_t)(\mathrm{Init}_x(U_t, y) \wedge \mathrm{Acc}(V_t, y) \wedge F_t(U_t, V_t, y)).$$

So M accepts x iff F_x'' is true in all models of cardinality ≥ 2. Finally, if F_x' is $(\forall u)(\forall v)(u = v) \vee F_x''$, then $x \in A$ iff F_x' is valid.

Let F_x be the prenex form formula equivalent to F'_x obtained by standard manipulation of quantifiers. For example, replace occurrences of $(G \to (\exists z)H)$ by $(\exists z)(G \to H)$, and $(G \to (\forall z)H)$ by $(\forall z)(G \to H)$, where G and H denote subformulas of F_x and z denotes a variable. Note that each variable occurring in F'_x is bound by exactly one quantifier; and $\sim, \wedge, \vee$ and $\to$ are the only logical connectives appearing in F'_x. Therefore $|F_x| = |F'_x|$ (viewing these formulas as words over the alphabet of $\mathscr{L}_{1\text{EQ}}$).

We now bound the length of F_x. First note that, since $l \leq s+2 \leq n^d+3$ and $k \leq t \leq a(n^d+1)$, each variable occurring in F_x is encoded as a word of length $O(\log n)$. The following relations are now obvious by inspection. (Of course, the abbreviations $\bigwedge_{l=1}^{s+1}$ etc. must be fully expanded when bounding the lengths of these formulas.)

$$|F_0| \leq c' n^d \log n,$$

$$|F_{k+1}| \leq |F_k| + c' n^d \log n \quad \text{for} \quad 0 \leq k < t,$$

$$|F_x| = |F'_x| \leq |F_t| + c' n^d \log n,$$

where c' is a constant depending on M and d, but not on n. These relations imply $|F_x| \leq cn^{2d} \log n$ for some constant c and all x, where $n = |x|$. If $f: \Theta^+ \to \mathscr{L}_{\text{Pr1EQ}}$ is defined by $f(x) = F_x$, f is length $cn^{2d} \log n$ bounded.

It remains only to observe that $f \in$ *logspace*. The verification of this fact is not difficult; we only sketch the outline, leaving the details to the reader. First recall that all occurrences of k and l in the descriptions of $F_0, \ldots, F_t$ denote integers whose binary representations are of length $O(\log n)$. It can be seen that, given x, a Turing machine can produce the formula F_0 while using only space $O(\log n)$. In fact, our description of F_0 above can serve as a finite "program" for this machine. Now thinking of F_k and F'_x as words over the alphabet of $\mathscr{L}_{1\text{EQ}}$, write F_{k+1} as $\omega_k F_k \tau_k$ for $0 \leq k < t$, and write F'_x as $\omega_t F_t \tau_t$, where ω_k and τ_k are the appropriate words over this alphabet (cf. the descriptions of F_{k+1} and F'_x above). As for F_0, ω_k and τ_k can be produced within space $O(\log n)$, given x and the binary representation of k. Since also t is of "length" $O(\log n)$, and

$$F'_x = \omega_t \omega_{t-1} \ldots \omega_1 \omega_0 F_0 \tau_0 \tau_1 \ldots \tau_t,$$

F'_x can be produced using space $O(\log n)$. The transformation mapping F'_x to F_x belongs to *logspace* as the reader can verify. Thus $f \in$ *logspace* because *logspace* is closed under composition [12, 25]. This completes the proof of part (1).

(2) Let A be as in part (1). For each x, one can construct, by minor modifications to F_x, a Boolean formula E_x such that $x \in A$ iff $E_x \in B_\omega$. First replace each occurrence of an atomic formula $(u_{k,l,\sigma} = y)$ (resp., $(v_{k,l,\sigma} = y)$, $(w_{k,l,\sigma} = y)$) in F_x by the variable symbol $u_{k,l,\sigma}$ (resp., $v_{k,l,\sigma}$, $w_{k,l,\sigma}$). The remaining occurrences of the equality symbol (e.g., in the abbreviations "$U_k = W_{k+1}$" etc. used in describing F_{k+1}) are replaced by the logical connective $\leftrightarrow$. It should be evident how to rename

the variable symbols as the doubly subscripted variables required by our definition of Boolean formulas; and the naming is done such that E_x is the matrix (i.e., the non-prefix part) of the resulting formula. That is, the variables $(U_t \cup V_t \cup W_t)$ are renamed as variables in the set X_1, $(U_{t-j} \cup V_{t-j})$ are renamed as X_{2j}, and W_{t-j} as X_{2j+1} for $1 \leq j \leq t$. If $g(x) = E_x$, one verifies just as in part (1) that $g \in$ *logspace* and g is length $cn^{2d} \log n$ bounded. □

Lemmas 6.2 and 6.3(1) prove Theorem 6.1. Lemmas 5.2 and 6.3(2) prove that B_ω is log-complete in $\mathcal{P}$-SPACE. The following Lemma 6.4 completes the proof of Theorem 5.1.

Lemma 6.4. (1) *There is a transformation f and a $c \in \mathbf{N}$ such that $B_\omega \leq_{\log} B_\omega \cap$ 3CNF via f, and f is length cn bounded.*

(2) *For each $d \geq 1$, $\mathrm{NSPACE}(n^d) \leq_{\log} B_\omega \cap$ 3CNF via length order $n^{2d} \log n$.*

Proof. (1) The transformation f rests on a technique of Tseitin [26, cf. 5] for converting an arbitrary Boolean formula to a formula in 3CNF while preserving satisfiability. Let $G(X_1, \ldots, X_k)$ be a given Boolean formula for some $k \geq 1$. Tseitin's technique yields a Boolean formula $F(X_1, \ldots, X_k, X_{k+1})$ belonging to 3CNF such that for all assignments of Boolean values to the variables $X_1 \cup \ldots \cup X_k$,

$$G(X_1, \ldots, X_k) = 1 \quad \text{iff} \quad (\exists X_{k+1})[F(X_1, \ldots, X_k, X_{k+1}) = 1].$$

If f is the transformation mapping G to F, one verifies by inspection of [26, 5] that $f \in$ *logspace* and f is length cn bounded for some constant c. If k is even, f is the required transformation. If k is odd, construct F as above except rename the variables in X_{k+1} as variables in X_{k+2}.

(2) This is immediate from part (1) of this lemma, Lemma 6.3(2), and closure of *logspace* under composition. □

It is now straightforward to obtain corollaries concerning the complexity of the sets 1EQ and B_ω. The following lemma is needed.

Lemma 6.5 (Jones [12], Meyer [25]). *Let $T(n)$ and $S(n)$ be monotone nondecreasing functions from* **N** *to* **N**. *Say $A \leq_{\log} B$ via f, where f is length $L(n)$ bounded.*

(1) $$B \in \begin{Bmatrix}\mathrm{NSPACE}\\ \mathrm{DSPACE}\end{Bmatrix}(S(n)) \quad \textit{implies} \quad A \in \begin{Bmatrix}\mathrm{NSPACE}\\ \mathrm{DSPACE}\end{Bmatrix}(S(L(n)) + \log n).$$

(2) $$B \in \begin{Bmatrix}\mathrm{NTIME}\\ \mathrm{DTIME}\end{Bmatrix}(T(n)) \quad \textit{implies} \quad A \in \begin{Bmatrix}\mathrm{NTIME}\\ \mathrm{DTIME}\end{Bmatrix}(T(L(n)) + p(n))$$

for some polynomial $p(n)$.

Corollary 6.6. *Let B be one of the sets* 1EQ, Pr1EQ, B_ω *or* $B_\omega \cap$ 3CNF.

(1) $B \in \mathcal{P}$ *iff* $\mathcal{P} = \mathcal{P}$-SPACE.

(2) *If a nondeterministic Turing machine accepts B within space $S(n)$, then there is a constant $c > 0$ such that $S(n) > c(n/\log n)^{1/2}$ for infinitely many n.*

Proof. (1) This is immediate from Theorems 5.1 and 6.1 and Lemma 6.5(2).

(2) Suppose to the contrary that a nondeterministic Turing machine accepts B within space $S(n)$, where for all rational $c > 0$, $S(n) \leq c(n/\log n)^{1/2}$ for all but finitely many n. Let $S'(n) = \max\{S(m): m \leq n\}$. Then $B \in \text{NSPACE}(S'(n))$ and $S'(n)$ is nondecreasing.

From the hierarchy theorem for nondeterministic space, proved by Seiferas et al. [21, 22], we have the following fact: There is a set $A \in \text{NSPACE}(n)$ such that for all $S_1(n)$,

$$\lim_{n \to \infty} S_1(n+1)/n = 0 \quad \text{implies} \quad A \notin \text{NSPACE}(S_1(n)). \tag{2}$$

Now since $A \in \text{NSPACE}(n)$, Lemma 6.3 or 6.4 implies $A \leq_{\log} B$ via f, where f is length $bn^2 \log n$ bounded for some constant $b > 0$. Therefore $A \in \text{NSPACE}(S'(bn^2 \log n) + \log n)$ by Lemma 6.5(1). However, choosing $S_1(n) = S'(bn^2 \log n) + \log n$ now contradicts the condition (2). □

As a final remark, we note that this lower bound on space complexity can be improved in the case of 1EQ and Pr1EQ. In the proof of Lemma 6.3(1), the particular method of encoding i.d.'s as sequences of variables was chosen to simplify the argument and to be able to obtain part (2) of the lemma by modifications to part (1). However, Meyer has suggested a more efficient encoding which yields a slightly stronger lower bound.

Corollary 6.7. (1) $\text{NSPACE}(n^d) \leq_{\log} \text{Pr1EQ}$ *via length order* n^{2d}.

(2) *If a nondeterministic Turing machine accepts* Pr1EQ *within space $S(n)$, then $S(n) > cn^{1/2}$ for some $c > 0$ and infinitely many n.*

Proof. (1) The construction is very similar to that of Lemma 6.3(1). We only outline the differences. Notation is as in Lemma 6.3(1). Break the i.d. of length (roughly) n^d into blocks of length b, where b is chosen below. An i.d. is thus viewed as a word of length $m = n^d/b$ over the alphabet Σ^b of cardinality $p = r^b$, where $r = \text{card}(\Sigma)$ depends only on the machine M. In place of the variable y, introduce p variables $y_1, \ldots, y_p$ which are viewed as constants. A sequence of m variables (e.g. $U_k' = \{u_{k,l}': 1 \leq l \leq m\}$) can now encode an i.d.: $u_{k,l}' = y_j$ iff the lth block of the i.d. contains the jth symbol of Σ^b. Also let Y denote $\{y_1, \ldots, y_p\}$. Note that variables are encoded as words of length not exceeding $O(\log n + \log p)$.

Similar to Lemma 6.3.1, one can check whether $\delta' \in \text{Next}_M(\delta)$ by comparing two

consecutive blocks of δ against the corresponding two blocks of δ' (assuming $b \geq 2$). Similar to the formula F_0 in Lemma 6.3(1), the block comparisons can be expressed in a straightforward way as a formula $G_0(U_0', V_0', Y)$. Note that there are n places to choose the two consecutive blocks, and for each choice there are p^4 possible assignments of words in Σ^b to the relevant four blocks. Therefore G_0 can be written so that

$$|G_0| \leq c'mp^4(\log n + \log p) \quad \text{for some constant } c'.$$

For $1 \leq k \leq t$, the formulas $G_k(U_k', V_k', Y)$ are constructed inductively, just as the F_k are constructed in Lemma 6.3(1). Recalling that a sequence of m variables encodes an i.d., we have

$$|G_{k+1}| \leq |G_k| + c'm(\log n + \log p).$$

Now the formula G_x is constructed from G_t as before, where $t = O(n^d)$, so that $x \in A$ iff $G_x \in \text{Pr1EQ}$. Choosing $b = \lceil (\frac{1}{4}d) \cdot \log_p n \rceil$, one verifies by calculation that $|G_x| \leq cn^{2d}$.

(2) The proof is identical to that of Corollary 6.6(2), using Corollary 6.7(1) in place of Lemma 6.3(1). □

In comparing the upper bound n on the space complexity of Pr1EQ (Lemma 6.2) with the lower bound $cn^{1/2}$ (Corollary 6.7), one should note that the upper bound applies to deterministic space while the lower bound applies to nondeterministic space. Any success in tightening this gap, either by improving the upper bound of Lemma 6.2(2) or by improving the length bound n^{2d} in Corollary 6.7(1), would (by an application of Lemma 6.5(1)) improve the best presently known relationship between nondeterministic and deterministic space, proved by Savitch [19], that $\text{NSPACE}(n) \subseteq \text{DSPACE}(n^2)$.

Acknowledgements

The author is grateful to Ronald Fagin, Michael J. Fischer, Albert R. Meyer, and Arnold L. Rosenberg for helpful discussions and comments concerning this work.

References

[1] W. Ackermann, *Solvable Cases of the Decision Problem* (North-Holland, Amsterdam, 1954).

[2] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, Mass., 1974).

[3] A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation, and Compiling, Vol. I: Parsing* (Prentice-Hall, Englewood Cliffs, N.J., 1972).

[4] T. Baker, J. Gill and R. Solovay, Relativizations of the $\mathcal{P} = ?\mathcal{NP}$ question, *SIAM J. Comput.*, 4 (1975) 431–442.

[5] M. Bauer, D. Brand, M. Fischer, A. Meyer and M. Paterson, A note on disjunctive form tautologies, *SIGACT News* **5** (April 1973) 17–20.

[6] S.A. Cook, The complexity of theorem proving procedures, *Proc. Third Annual ACM Symposium on Theory of Computing* (1971) 151–158.

[7] S.A. Cook and R.A. Reckhow, Time bounded random access machines, *J. Comput. System Sci.* **7** (1973) 354–375.

[8] S. Even and R.E. Tarjan, A combinatorial problem which is complete in polynomial space, *Proc. Seventh Annual ACM Symposium on Theory of Computing* (1975) 66–71,

[9] R. Fagin, Generalized first-order spectra and polynomial-time recognizable sets, in: R. Karp, ed., *Complexity of Computation, SIAM-AMS Proc.* **7** (1974) 43–73.

[10] M.A. Harrison, *Introduction to Switching and Automata Theory* (McGraw-Hill, New York, 1965).

[11] J.E. Hopcroft and J.D. Ullman, *Formal Languages and their Relation to Automata* (Addison-Wesley, Reading, Mass., 1969).

[12] N.D. Jones, Space-bounded reducibility among combinatorial problems, *J. Comput. System Sci.* **11** (1975) 68–85.

[13] 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) 85–104.

[14] D.E. Knuth, Postscript about NP-hard problems, *SIGACT News* **6** (April 1974) 15–16.

[15] R.E. Ladner, The computational complexity of validity in T, S4, and S5, manuscript, University of Washington, Seattle, Wash. (1974).

[16] J.C. Lind, Computing in logarithmic space, Tech. Memo. 52, M.I.T., Project MAC, Cambridge, Mass. (1974).

[17] A.R. Meyer and L.J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, *Proc. Thirteenth Annual IEEE Symposium on Switching and Automata Theory* (1972) 125–129.

[18] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability* (McGraw-Hill, New York, 1967).

[19] W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4** (1970) 177–192.

[20] J.R. Shoenfield, *Mathematical Logic* (Addison-Wesley, Reading, Mass., 1967).

[21] J.I. Seiferas, Nondeterministic time and space complexity classes, Doctoral Thesis, Report TR-137, M.I.T., Project MAC, Cambridge, Mass. (1974).

[22] J.I. Seiferas, M.J. Fischer and A.R. Meyer, Refinements of the nondeterministic time and space hierarchies, *Proc. Fourteenth Annual IEEE Symposium on Switching and Automata Theory* (1973) 130–137.

[23] R.E. Stearns, J. Hartmanis and P.M. Lewis, Hierarchies of memory limited computations, *Sixth IEEE Symposium on Switching Circuit Theory and Logical Design* (1965) 179–190.

[24] L.J. Stockmeyer, The complexity of decision problems in automata theory and logic, Doctoral Thesis, Report TR-133, M.I.T., Project MAC, Cambridge, Mass. (1974).

[25] L.J. Stockmeyer and A.R. Meyer, Word problems requiring exponential time: preliminary report, *Proc. Fifth Annual ACM Symposium on Theory of Computing* (1973) 1–9.

[26] G.S. Tseitin, On the complexity of derivation in propositional calculus, in: A.O. Slisenko, ed., *Studies in Constructive Mathematics and Mathematical Logic*, Part II (Steklov Math. Institute, Leningrad, 1968); in Russian; English Transl.: Consultants Bureau, New York, 1970, 115–125.

[27] C. Wrathall, Complete sets and the polynomial hierarchy, *Theoret. Comp. Sci.* **3** (1976).