Subrecursive Predicates and Automata

Celia Wrathall

Research Report #56

October 1975

This work was presented to Harvard University in candidacy for the degree of Doctor of Philosophy. The research was supported in part by the National Science Foundation under Grants GJ-30409 and DCR-75-15945.

PREFACE

I am indebted to my advisor, Professor Ronald V. Book, for suggesting this problem area and for his continued support and encouragement. Our discussions of this dissertation were invaluable for its development and organization.

The courses I took from Professor Forbes D. Lewis were essential preparation for this research. I wish to thank him also for his willingness to discuss these problems and his helpful criticism of the style. I am grateful to Mr. Richard Barakat and Professors Emily P. Friedman and Charles J. Prenner for their efforts as morale boosters.

The credit for the excellent preparation of the manuscript goes to Ms. Mary-Claire van Leunen and Ms. Gail Beyer Ohlson.

This research was supported in part by the National Science Foundation under Grants GJ-30409 and DCR-75-15945 and by a John Parker Fellowship.

TABLE OF CONTENTS

INDEX OF DEFINITIONS AND SYMBOLS

Some of the terms and special symbols used in this dissertation are listed here, with the pages on which they appear. Standard notation from automata and formal language theory is reviewed in Section 1.3.

SYNOPSIS

This work addresses questions from automata-based computational complexity, using techniques and conceptual tools from formal language theory and the study of subrecursive functions as well as from automata theory. The goal is to obtain information on the recognitional power of resource-bounded automata, especially as compared to the expressive power of string-theoretic predicates used in defining sets.

Chapter 1 introduces the topics considered in this dissertation and contains an overview of the results.

"Query machines" and "oracle machines," models for resource-bounded relative computation, have been studied in [3,16,37,38,39,40,52]. The definition of oracle machine from [39] is used here. Chapter 2 gives some basic results for time-bounded oracle machines, especially in the context of linear and polynomial time bounds. Properties considered are closure under language operations (Theorem 2.2.4), the hierarchy induced by increasing time bounds (Theorem 2.4.1), relationship to classes defined by time-bounded Turing acceptors (Propositions 2.1.5, 2.2.2), and the comparative power of deterministic and nondeterministic operation (Proposition 2.2.3). It is also shown (Theorem 2.3.1) how the language accepted by an oracle machine can be represented using language-theoretic operations applied to the oracle set and a simpler language. With the exception of Propositions 2.1.5 and 2.2.3, which appeared in [38] and [3], respectively, for polynomial time bounds, all results in this chapter are new. Many of the results are extensions to oracle machines of known

properties of Turing acceptors. The representation theorem (2.3.1) was suggested by the definition of "r.e. in" given in [47].

In Chapter 3, the rudimentary relations [51] are investigated by applying the connections established there between definition of languages using oracle machines and definition using language operations or string predicates. Two new characterizations of the class of rudimentary relations are given (Theorem 3.2.7 and Corollary 3.3.5(2)), which allow different proofs of known properties of the class (Corollary 3.2.8, Corollary 3.3.5(1)). Some basic results on the rudimentary relations from [33,34,42] are given in Proposition 3.2.4(1)-(3) and are used in proving the characterizations. Corollary 3.2.8 is proved directly in [41,45] and Corollary 3.3.5(1), in [57]. Proposition 3.2.2, the version of the Chomsky-Schutzenberger Theorem [14] used here, is taken from [8], and Proposition 3.2.3 is from [9]. All other results in this chapter are new.

The linear hierarchy, a decomposition of the rudimentary relations into a structure of classes defined using linear-time oracle machines, is the topic of Chapter 4. Closure properties (Propositions 4.1.2, 4.3.2) and characterizations (Theorems 4.1.4, 4.2.2) of the classes in the linear hierarchy are established, as well as equivalences among the questions remaining open (Proposition 4.1.3, Corollaries 4.3.8, 4.3.10). A generator ("complete set") for each class is constructed (Theorem 4.3.5). All results in this chapter are new; some are generalizations of results previously known for the quasi-realtime languages, the first class in the linear hierarchy.

The polynomial hierarchy [40,52,53] is considered in Chapter 5. The technique of "polynomial translation" (as used in [7]) is combined with results from Chapters 3 and 4 to yield simple proofs of properties of the polynomial hierarchy. Proposition 5.2.3 appears in [38]. Proposition 5.3.10 was announced in [53] but was obtained independently. As with the linear hierarchy, many results on the polynomial hierarchy are generalizations of results known for the first class in the hierarchy.

The appendixes contain proofs, deferred from the text, of two new results. Theorem 2.4.1 is proved in Appendix A; it is a relativized version of theorems from [28,49]. Appendix B contains the proofs of Proposition 3.3.3 and some related results. The development in Appendix B leads to simplified proofs of Corollary B.4(1) and (2), which appear originally in [46] and [31], respectively.

Chapter 1: INTRODUCTION

1.1. SETTING

This work addresses questions from automata-based computational complexity, using techniques and conceptual tools from formal language theory and the study of subrecursive functions as well as from automata theory. The goal is to obtain information on the recognitional power of resource-bounded automata, especially as compared to the expressive power of string-theoretic predicates used in defining sets. Only strictly subelementary families of languages are considered; in particular, the languages they contain have membership problems of at most exponential time complexity by means of Turing acceptors. Since families of languages are the objects for analysis, the only specific languages that are of interest are generators for the families.

The study of the finite specification of functions, and, in particular, of characteristic functions for infinite sets, is central to the development of the mathematical foundations of computer science. Specification of a set in some system gives a view of the complexity of its characteristic function relative to the basic operations or capabilities of the system. On the one hand, research in computational complexity focuses on finding specifications that are "concise" or "simple" i.e., that are of minimal complexity. On the other hand, properties of the sets that possess simple descriptions are investigated, leading to characterization of the expressive power of the system.

Models for computing functions or recognizing sets that are based on Turing machines are useful for studying problems of computation and of complexity. While unrestricted Turing machines are equivalent to many other formulations of the idea of "effective computability," it is more important for the study of computational complexity that correspondences are known between complexity as measured using Turing machines and as measured using more realistic models of computation. Thus although the basic operations allowed in the model are highly restricted, questions about the actual complexity of functions may be answered by considering and comparing the power of resource-bounded Turing machines. The restricted nature of the model simplifies manipulations and constructions involving the machines and the sets they accept. This is particularly important for arithmetization and other representations of languages accepted by machines with small bounds on computational measures.

Variations on the Turing machine model have been defined by varying the mode of reading input (e.g., one- or two-way, one symbol per step or with delay), by allowing nondeterministic operation, and by extending or restricting access to the storage tapes or the actions performed on them (e.g., allowing multiple heads on one tape or restricting a tape to serve as a stack or a counter). To derive quantitative information on the complexity of recognition problems relative to the chosen model, measures of computational resource are assigned, most commonly the time or space used during a computation as a function of the input. Certain questions can be posed concerning the power of the resource-bounded automata thus defined and the properties of the classes of languages they accept. For

example:

(1) What correspondence exists among the measured resources used in computations and among the classes of languages determined by different measures of resource?

(2) What increase in power results from the extension to nondeterministic operation?

(3) What increase in allowed resources will result in increased recognitional capacity?

(4) How do classes defined by various resource bounds relate to classes of languages arising from other machines models or in other contexts?

Only partial answers to these questions are known. With respect to the second question, for instance, nondeterminism is known to add recognitional capacity for certain classes of (time-bounded) machines with restricted storage, but does not add power for finite-state machines, the most restricted model. Due to its importance for establishing whether practical procedures exist for solving certain persistent recognition problems, the question of the power of nondeterminism in computations of length bounded by a polynomial in the length of the input ("P vs. NP") is currently receiving much attention. This question need not be attacked directly: An answer to it can be derived from answers to other questions, about the complexity of specific languages and about the closure properties of the classes involved and their relationship to other classes of languages. As indicated by this example, these four general questions are not independent. In particular, instances of the first three questions can often be stated in terms of topics contained in the fourth.

Many approaches other than acceptance by automata can be taken for arriving at definitions of classes of languages. A class might be defined as consisting of the languages generated by some class of formal grammars (e.g., the context-free languages, generated by the context-free grammars), or as consisting of the languages associated with a class of trees or of functions (e.g., the class of languages whose characteristic functions are elementary functions). A class might also be defined by requiring the strings in the languages it contains to satisfy some structural property (e.g., the class of languages with the semilinear property). Two other approaches are based on defining classes by using operations on languages. In the first, a class of languages is defined inductively from another, that is, is defined to be the smallest class containing some basis languages and closed under given operations. For instance, the regular sets can be defined inductively from the finite sets using the operations of union, product, and Kleene closure. In the second approach, the defined class consists of those languages resulting from one application of some operation (e.g., complementation) to the basis languages or from one application of some function in a class of string mappings (e.g., inverse homomorphisms). As is the case for the examples given, the classes resulting from these forms of definition can sometimes be shown to equal or to contain the class of languages accepted by some type of automaton. Such connections give a different view of the properties of the computing devices and of the languages they are

capable of accepting, giving rise both to new questions and to new techniques for studying the automata. Characterizations can be used to restate open problems into possibly more tractable forms, and often allow for simpler proofs than the original definition.

Any machine derived from Turing machines operates using symbol-by-symbol scanning and manipulation of strings, and the basic operation on strings is concatenation. A natural candidate, therefore, for a class of languages to compare to automata-based complexity classes is one defined inductively from a language representing concatenation. Quine [44] investigated the form in which concatenation could serve as a basis for arithmetic (and hence for computation with numerals), showing that addition and multiplication are first-order definable from concatenation. As part of a study of recursive function theory, Smullyan [51] defined and used the class of rudimentary relations, consisting of those sets that are "constructively" definable from concatenation (restricting the quantifications allowed to a bounded form). Bennett [4] subsequently showed that addition and multiplication are constructively definable from concatenation, and defined some further classes of rudimentary sets. Connections between these classes and classes arising in automata and formal language theory have been recognized and studied [41,15,33,34,42,57].

In this dissertation, a machine model ("oracle machines") that extends Turing machines by allowing relative computation is studied. Since the only measure of resource assigned is the number of steps taken in a computation, the first of the questions cited above for resource-bounded automata is not considered for oracle machines; partial answers to the last three questions are developed. Primary attention is given to the final topic, of the relationships of classes of languages accepted by time-bounded oracle machines to other classes of languages. We use oracle machines to derive new characterizations of the rudimentary relations, which yield further knowledge of the relationship of this class to classes from automata theory as well as information on open problems.

1.2. OVERVIEW OF CHAPTERS

This section contains a survey of the topics considered and results presented in each chapter of this dissertation. In the final section of this chapter some terminology from automata and formal language theory is reviewed.

An oracle machine is a multitape Turing acceptor with the added ability to determine membership in a (variable) language. During a computation such a machine may write a string on a distinguished tape and ask for information on the string; the next state the machine enters is determined by whether that string is a member of the "oracle set" or not. Thus computations of the machine proceed relative to the information it receives, and it may accept different languages relative to different oracle sets.

Unrestricted oracle machines can be used to define Turing reducibility and hence serve in expressing the notion of degrees of unsolvability [54,47]. Cook [16] introduced time-bounded relative computation to define a restricted Turing reducibility and applied it fruitfully to the question of the power of nondeterminism in polynomially time-bounded computations by Turing acceptors. Further studies of these (and other) time-bounded reducibilities and of the structure they place on the recursive sets have since been made [3,37,38,39,52].

The model of oracle machine used here is a technical variant of Cook's "query machines." We view relative computation as a way to define classes of languages from others and use the tools of automata and formal language theory to study the languages accepted by oracle machines. Time-bounded oracle machines share many of the properties of other resource-bounded abstract automata and proofs that have become standard in automata theory can be easily extended to apply to them. Chapter 2 gives some basic results for time-bounded oracle machines, especially in the context of linear and polynomial bounds, which are used in the remainder of the dissertation. Properties considered are closure under language operations (Theorem 2.2.4), the hierarchy induced by increasing time bounds (Theorem 2.4.1), relationship to classes defined by time-bounded Turing acceptors (Propositions 2.1.5, 2.2.2), and the comparative power of deterministic and nondeterministic operation (Proposition 2.2.3). It is also shown (Theorem 2.3.1) how the language accepted by an oracle machine can be represented using language-theoretic operations applied to the oracle set and a simpler language. This representation theorem, in the spirit of those in [14,9,12], is used extensively in the following chapters.

The class of rudimentary relations of Smullyan [51] can be defined as the smallest class of string relations containing the concatenation relations and closed under the Boolean operations, explicit transformation and a form of bounded (existential and universal) quantification. It is known that the addition and multiplication relations are rudimentary [4] (identifying a number with its dyadic notation), and since, further, this class is closed under many of the operations studied in automata and formal language theory, it is of interest for the formal analysis of computation (as well as for its role in logic [4,51]).

In a natural way the rudimentary relations give rise to a family of languages. Since (Proposition 3.2.4) a relation is rudimentary if and only if its associated language is rudimentary, the distinction between rudimentary relations and "rudimentary languages" is ignored. The class of rudimentary relations contains all context-free languages [34], all languages accepted in linear time by nondeterministic multitape Turing machines (Proposition 3.2.4), and all languages accepted in logarithmic space by nondeterministic Turing machines [42]; on the other hand, every rudimentary relation can be accepted in linear space by a deterministic Turing machine [41].

In Chapter 3, the "machinery" developed in Chapter 2 is used to provide two new characterizations of the rudimentary relations, one using oracle machines and the other as an inductively defined class, with a different basis and different operations than in the original definition.

These results will now be described.

Consider nondeterministic oracle machines that operate in linear time (i.e., the number of steps in any computation is bounded by a constant multiple of the length of the input string). If A is a language, then let NL({A}) denote the class of languages accepted by such machines when given an oracle for membership in A. For a class $\mathcal{C}$ of languages, let $NL(\mathcal{C}) = \cup\{NL(\{A\}): A \in \mathcal{C}\}$. Using NL( ) to denote this operator on a class of languages, let NL*( ) denote the closure of a class under applications of NL( ). This provides the notation necessary to state our first characterization of the rudimentary relations.

Theorem 3.2.7. The class of rudimentary relations is the smallest nonempty class closed under the operation of relative acceptance by nondeterministic linear-time oracle machines. That is, $NL^*(\{\phi\})$ is exactly the class of rudimentary relations.

The proof of Theorem 3.2.7 depends on a characterization of the operator NL( ) in terms of language-theoretic operations that follows from Theorem 2.3.1.

Because NL( ) and NL*( ) are defined by means of abstract automata, it is apparent from Theorem 3.2.7 that the class of rudimentary relations is closed under various operations studied in formal language theory, as well as under the Boolean operations given in its definition; for example, it is closed under product of languages, Kleene *, inverse homomorphism, non-erasing homomorphism, linear erasing, and reversal. It was shown by Yu [57] that the rudimentary relations are the smallest class containing the context-free languages and closed under the Boolean operations and non-erasing homomorphism. Here, a stronger characterization is established.

Corollary 3.3.5. The class of rudimentary relations is the smallest class of languages that contains the language $\{a^n b^n: n \geq 0\}$ and is closed under the Boolean operations, inverse homomorphism, and length-preserving homomorphism.

This characterization follows from a general statement concerning the effect of applying NL*( ) to families of languages (Theorem 3.3.4), combined with a result on the definability of context-free languages from the language $\{a^n b^n: n \geq 0\}$ (Proposition 3.3.3).

In Chapter 4 we turn to a decomposition of the rudimentary relations based on the characterization given in Theorem 3.2.7. Let σ_0 be the class containing only the empty set. For each k, let $\sigma_{k+1} = NL(\sigma_k)$. Thus $\cup\{\sigma_k: k \geq 0\}$ is the class of rudimentary relations. Since an oracle machine with an oracle that always replies "no" can be simulated by a Turing acceptor, σ_1 is the class of languages accepted in linear time by nondeterministic multitape Turing machines (the quasi-realtime languages of [9]). The linear hierarchy is the structure $\sigma_0 \subseteq \sigma_1 \subseteq \sigma_2 \subseteq \ldots$ (along with some related classes).

Whether the linear hierarchy is in fact an infinite hierarchy of classes (i.e., whether $\sigma_k \subsetneq \sigma_{k+1}$ for each k) is unknown; an affirmative answer to this question would also settle other previously studied questions: for example, whether the class of quasi-realtime languages is

closed under complementation, and whether the class of rudimentary relations is properly contained in the class of relations associated with $\mathcal{E}^2$.

While $\bigcup_k \sigma_k$ (i.e., the rudimentary relations) is closed under complementation, it is not known whether there exists a k such that σ_k is closed under complementation. This question and that of the finiteness of the linear hierarchy are closely connected, as is seen in the following result.

Proposition 4.1.3. The linear hierarchy is finite if and only if there exists some k such that σ_k is closed under complementation.

It is shown in Chapter 3 that for any $k \geq 1$, a language belongs to σ_{k+1} if and only if it is the image under a nonerasing homomorphism of the complement of a language in σ_k. From this fact we establish the following characterization of the classes in the linear hierarchy.

Let δ_1 denote the family of languages that can be accepted in linear time by deterministic multitape Turing acceptors.

Theorem 4.2.2. For each $k \geq 1$, the class σ_k consists of exactly those languages that can be obtained from languages in δ_1 by application of k (linearly) bounded quantifications that alternate between existential and universal quantification and end with an existential quantifier.

Thus the class in the linear hierarchy to which a rudimentary relation belongs is closely related to the syntactic form of its definition from concatenation relations.

In the last section of Chapter 4, we consider the internal structure of the classes σ_k, employing the concept of "efficient reducibility." Each class is shown to possess a complete set with respect to a simple-to-compute reducibility. This property allows some comparisons to be made between the classes in the linear hierarchy and other classes of languages, and questions about the classes σ_k and about the rudimentary relations can be reduced to questions about these generators.

Theorem 4.3.5. For all $k \geq 0$, there exists a language $A_k \in \sigma_k$ with the property that for every $L \in \sigma_k$, there is a homomorphism h such that $L - \{e\} = h^{-1}(A_k)$.

The sequence of languages $A_0, A_1, \ldots$ is defined uniformly from $A_0 = \phi$ by means of a "universal" nondeterministic linear-time oracle machine M_0 (which is constructed along the lines of the universal machines studied in [55,11]): A_{k+1} is the language accepted by M_0 when M_0 is given an oracle for membership in A_k.

For each k, the language A_k is a "hardest" language for σ_k with respect to deterministic time-bounded or space-bounded recognition, in the same sense that the language exhibited by Greibach [24] is a hardest context-free language. Thus, for example, A_k can be accepted by a deterministic Turing machine in polynomial time if and only if every language in σ_k can be so accepted.

Although the rudimentary relations form a subclass of the class of languages accepted in linear space by deterministic Turing machines (i.e., accepted by deterministic linear-bounded automata), it is not known

whether this inclusion is proper. However, from Theorem 4.3.5 and known properties of the class of languages accepted in deterministic linear space, we arrive at the following result.

Corollary 4.3.8. If the linear hierarchy is infinite, then there exists a language that is not rudimentary but can be accepted in linear space by a deterministic Turing machine.

The polynomial hierarchy of Meyer and Stockmeyer [40,52,53] is (like the linear hierarchy) a structure of classes analogous to the arithmetic hierarchy and is potentially useful for classifying languages. It can be defined using nondeterministic oracle machines that operate in time bounded by some polynomial of the length of the input: Each class in the polynomial hierarchy consists of the languages accepted by such machines relative to a language in the previous class.

In Chapter 5 an investigation of the polynomial hierarchy is made based on a strong connection that exists between it and the linear hierarchy. It is shown (Theorem 5.2.7) that a class in the linear hierarchy forms a basis for the corresponding class in the polynomial hierarchy under "polynomial translation" [7]. Thus the linear hierarchy embodies in a simplified form the properties of the polynomial hierarchy. The same questions remain open for both structures and certain solutions in the context of the linear hierarchy will supply solutions for the polynomial hierarchy; for example, if the linear hierarchy is not infinite (i.e., collapses at some class) then the polynomial hierarchy must collapse as well (Corollary 5.3.3).

Under certain conditions established in Chapter 2, an increase in the time allowed an oracle machine yields increased computational power. From this it can be concluded that no class in the linear hierarchy can be equal to any in the polynomial hierarchy. Furthermore, each class in the polynomial hierarchy is decomposed into an infinite union of classes and hence cannot have generators under some operations (in particular, under the operations used for Theorem 4.3.5). A representation of the linear hierarchy such as that given in Theorem 4.3.5 is necessary for these conclusions to be drawn. The generators constructed for the classes in the linear hierarchy lift to become complete sets for the polynomial hierarchy, necessarily under an extended class of functions.

1.3. PRELIMINARY DEFINITIONS AND NOTATION

Some basic definitions from automata and formal language theory, used throughout this dissertation, are collected here. All should be familiar to the reader, with the possible exception of the function θ used to encode tuples of strings as strings.

For a set A and an integer $m \geq 1$, $[A]^m$ denotes the cross product of A with itself m times. The cardinality of a set A is denoted $\#(A)$.

If S is a finite set of symbols, called an alphabet, then S* denotes the free monoid generated by the symbols in S. The elements of S* are strings (finite sequences) of symbols from S; the operation in S* is termed "(string) concatenation" and is denoted by juxtapostion of the strings. The identity element of S* is the "empty word" or "empty string," denoted by e. Thus $S^* = \{e\} \cup \{s_1 \ldots s_n : n \geq 1, s_1, \ldots, s_n \in S\}$. If $n \geq 1$ and $x = s_1 \ldots s_n$ is a string in S ($s_i \in S$ for $1 \leq i \leq n$) then the length of x, denoted $|x|$, is n; $|e| = 0$.

If for some alphabet S, a set L is a subset of S* then L is a language. An m-ary string relation is a subset of $[S^*]^m$ for some alphabet S.

In order to use the tools of formal language theory to investigate classes of string relations, we combine a tuple of strings into a single string, as follows. Suppose S is an alphabet and # is a symbol not in S.

Let $S_{\#} = S\cup\{\#\}$. For $n\geq 1$, $\theta_n^{\#}:[S^*]^n \to ([S_{\#}]^n)^*$ is defined as follows:

(1) For all $x\in S^*$, $\theta_1^{\#}(x) = x$;

(2) For $n\geq 2$, if $x_1,\ldots,x_n\in S^*$ then $\theta_n^{\#}(x_1,\ldots,x_n) = z_1\ldots z_m$ where $m = \max\{|x_i| : 1\leq i\leq n\}$; for $1\leq j\leq m$, $z_j = [z_j^1,\ldots,z_j^n]\in[S_{\#}]^n$; and for $1\leq i\leq n$, $z_1^i z_2^i\ldots z_m^i = x_i \#^{m-|x_i|}$.

The mapping $\theta_n^{\#}$ is extended to subsets of $[S^*]^n$ by:

$\theta_n^{\#}(R) = \{\theta_n^{\#}(x_1,\ldots,x_n) : (x_1,\ldots,x_n)\in R\}$.

Hereafter, "θ" will be used ambiguously for any $\theta_n^{\#}$: n will be clear from the context and it will be assumed that $\#\notin S$. For an example, suppose $S = \{0,1\}$ and $n = 3$. Then $\theta(e,101,1) =$ $[\#,1,1][\#,0,\#][\#,1,\#]$ and $\theta(000,101,11) = [0,1,1][0,0,1][0,1,\#]$. The intention of θ is to describe writing n strings on n "tracks" of a Turing tape; thus the second example should be read as

$$\theta(000,101,11) = \begin{array}{ccc} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & \# \end{array}.$$

This "parallel" encoding is due to Myhill [41]. The notation "θ" is from [57].

Let S and T be alphabets and $h:S^*\to T^*$ be a (monoid) homomorphism, so that for any $x,y\in S^*$, $h(xy) = h(x)h(y)$. If $L_1\subseteq S^*$ then $h(L_1) =$ $\{h(x) : x\in L_1\}$ is the image of the language L_1 under the homomorphism h. If $L_2\subseteq T^*$ then $h^{-1}(L_2) = \{y\in S^* : h(y)\in L_2\}$; h^{-1} is a mapping from subsets of S^* to subsets of T^*, called an inverse homomorphism. Definitions of homomorphisms need only be given for the symbols in S; they are then extended to S^* by $h(s_1\ldots s_n) = h(s_1)\ \ldots\ h(s_n)$. Note that $h(e) = e$.

Suppose $h:S^*\to T^*$ is a homomorphism. Then h is <u>length-preserving</u> if $|h(s)| = 1$ for all $s\in S$. It is <u>nonerasing</u> if $h(s) \neq e$ for all $s\in S$ (i.e., $|h(s)|\geq 1$). For a language $L\subseteq S^*$, the homomorphism h is <u>e-limited on L</u> if there exists an integer k such that for any string $w\in L$, if $w = xyz$ for some $x,y,z\in S^*$ such that $h(y) = e$ then $|y|\leq k$. (That is, h can erase at most k consecutive symbols from a string in L.) The homomorphism h is said to perform <u>linear erasing on</u> $L\subseteq S^*$ if there is an integer k such that for any $w\in L$, $|w|\leq k.\max\{|h(w)|,1\}$. A family of languages $\mathcal{L}$ is closed under linear erasing if whenever $L\in\mathcal{L}$ and h is a homomorphism that performs linear erasing on L, also $h(L)\in\mathcal{L}$.

Some further operations on languages are defined as follows:

(1) If $L_1\subseteq S^*$, $L_2\subseteq T^*$ are the languages then the product of L_1 and L_2 is the language $L_1L_2 = \{xy: x\in L_1, y\in L_2\}\subseteq(S\cup T)^*$. If ¢ is a new symbol, $¢\notin(S\cup T)$, then $L_1¢L_2 = \{x¢y : x\in L_1, y\in L_2\}$ is a marked product of L_1 and L_2.

(2) If $L\subseteq S^*$ is a language then $L^+ = \{y_1\ \ldots\ y_n : n\geq 1, y_i\in L \text{ for } 1\leq i\leq n\}$ and $L^* = L^+\cup\{e\}$. Thus for an alphabet T, $T^+ = T^* - \{e\} = \{x\in T^* : |x|\geq 1\}$. The operation taking L to L^* (L^+) is Kleene $*$ (Kleene +). If $¢\notin S$ then $(L¢)^+ = \{y_1¢\ldots y_n¢ : n\geq 1, y_i\in L\}$ is a marked + of L and $(L¢)^* = (L¢)^+\cup\{e\}$ is a marked $*$ of L.

(3) The Boolean operations are the operations of set union, intersection and difference. If $L_1,L_2\subseteq S^*$ then a marked union of L_1 and L_2, denoted here $L_1 \oplus L_2$, is any language of the form $\{¢\}L_1\cup\{\$\}L_2$ where $¢,\$\notin S$ are

two distinct symbols.

(4) A family of languages $\mathcal{L}$ is closed under complementation if whenever $L \in \mathcal{L}$ and S is any alphabet such that $L \subseteq S^*$ also $S^* - L = \{x \in S^* : x \notin L\}$ is in $\mathcal{L}$. If $\mathcal{L}$ is a family of languages then co-$\mathcal{L}$ denotes the family containing exactly the complements of languages in $\mathcal{L}$; that is, co-$\mathcal{L} = \{S^* - L : L \subseteq S^* \text{ in } \mathcal{L}\}$. Notice that $\mathcal{L}$ = co-$\mathcal{L}$ if and only if co-$\mathcal{L} \subseteq \mathcal{L}$ if and only if $\mathcal{L}$ is closed under complementation.

The family of regular sets is the smallest family of languages containing the finite languages and closed under the operations of union, product and Kleene *. It is well known that a language is regular if and only if it is acceptecd by some finite-state machine, and that the regular sets are closed under all the operations described in (1)-(4) above, as well as under homomorphic and inverse homomorphic mappings.

The model for Turing acceptor used here has a (one- or two-way) read-only input tape and multiple work tapes (see, e.g., [30]). It may be deterministic or nondeterministic. The language accepted by a Turing machine M is denoted by L(M). When Turing machines are used as transducers, i.e., to compute string functions, one of the work tapes becomes a one-way write-only output tape.

Let $t: \mathbb{N} \to \mathbb{N}$ and $s: \mathbb{N} \to \mathbb{N}$ be nondecreasing functions, with $t(n) \geq n$ for all $n \in \mathbb{N}$. ($\mathbb{N}$ denotes the natural numbers.) A Turing acceptor is said to operate in time t(n) if for every input string x every computation of the machine on x takes at most $t(|x|)$ steps. A Turing acceptor is said to operate in space s(n) if for any input x, no more than $s(|x|)$ tape squares are visited on any one of the work tapes during any computation on x. We use lg(n) to denote the function whose value at $n \in \mathbb{N}$ is the length of the binary representation of n, so $\lg(0) = 1$ and for $n>0$, $\log_2(n) < \lg(n) \leq \log_2(n)+1$. Following [49], define a function $t: \mathbb{N} \to \mathbb{N}$ to be a "running time" if there is a deterministic Turing acceptor M such that on any input x, M takes exactly $t(|x|)$ steps and halts.

The notation used here for families of languages defined by resource-bounded Turing acceptors follows.

For a time bounding function t(n):

DTIME(t(n)) = {L(M) | M is a deterministic Turing acceptor that operates in time bound t(n)};

NTIME(t(n)) = {L(M) | M is a nondeterministic Turing acceptor that operates in time bound t(n)}.

It is known [9] that for any constant c and any Turing acceptor that operates in time cn there is a nondeterministic Turing acceptor that accepts the same language and operates in time n(i.e., in real-time); hence

NTIME(n) = $\cup$\{NTIME(cn) : $c \geq 1$\}.

Also,

DTIME(lin) = $\cup$\{DTIME(cn) | $c \geq 1$\};

DTIME(poly) = $\cup$\{DTIME(p(n)) | p a polynomial\};

$DTIME(2^{lin}) = \cup\{DTIME(2^{cn}) \mid c > 0\}$.

NTIME(poly) and $NTIME(2^{lin})$ are defined similarly.

For a space bounding function s(n):

DSPACE(s(n)) = {L(M) | M is a deterministic Turing acceptor that operates in space bound s(n)};

NSPACE(s(n)) = {L(M) | M is a nondeterministic Turing acceptor that operates in space bound s(n)}.

Using standard tape-compression techniques, a Turing machine that operates in space $c \cdot s(n)$ for some $c \geq 1$ can be converted to an equivalent Turing machine that operates in space s(n). In particular,

$DSPACE(n) = \cup\{DSPACE(cn) : c \geq 1\}$ and

$NSPACE(n) = \cup\{NSPACE(cn) : c \geq 1\}$.

The class DSPACE(n) is the family of languages accepted by deterministic linear-bounded automata (LBA's); NSPACE(n) is the family of context-sensitive languages [41].

In this notational scheme, the class $\cup\{DSPACE(p(n)) \mid p$ a polynomial$\}$ is denoted by DSPACE(poly). However, since it is known [48] that $\cup\{DSPACE(p(n)) \mid p$ a polynomial$\}$ = $\cup\{NSPACE(p(n)) \mid p$ a polynomial$\}$, this class will be denoted here by PSPACE.

A push-down store is a Turing tape that is one-way infinite to the right and the action of which is restricted in the following ways: (i) only the rightmost symbol on the store may be read; and (ii) if the head moves left into the string written on the store then all symbols to the right of the head must be erased. Thus a machine with a push-down store can test it for emptiness and manipulate it by "pushing" symbols

onto the store (printing and moving right) or by "popping" symbols from the store, if it is nonempty (erasing and moving left). The family of languages accepted by nondeterministic (deterministic) Turing acceptors with one-way input and one push-down store as auxilliary storage is the family of context-free languages (deterministic context-free languages). The family of context-free languages is also that generated by the context-free grammars. See [18,30] for discussion of context-free languages and their properties.

Chapter 2: TIME-BOUNDED ORACLE MACHINES

In this chapter the definition of oracle machines is given and some basic properties of time-bounded oracle machines are developed. As a model for relative computation, oracle machines are a variant of the model used to explain the arithmetic hierarchy [47] and of the query machines used by Cook [16] to study efficient reductions between language recognition problems.

Constructions involving Turing acceptors generally apply with only slight modification to oracle machines, so we have, for example, the expected closure properties for families of languages defined by time-bounded oracle machines (Theorem 2.2.4). There is also a result (Theorem 2.4.1) corresponding to the "time hierarchy theorem" for Turing acceptors, giving conditions under which an increase in the time allowed an oracle machine yields an increase in its definitional power. The investigation to be made of the families of languages defined by time-bounded oracle machines is greatly aided by Theorem 2.3.1, in which the language accepted by an oracle machine is represented algebraically, in terms of language-theoretic operations applied to simpler languages. In subsequent chapters, we will consider only time bounds which are linear functions or polynomials; most of the results in this chapter are therefore stated for those cases, although they can be seen to hold more generally.

2.1. DEFINITION AND GENERAL PROPERTIES

We begin with an informal definition and discussion of oracle machines. The interested reader can make the connections to the formal definition that follows.

Definition 2.1.1. An oracle machine is a multitape Turing acceptor with an added dynamic capability. A computation of an oracle machine M depends on both an input string x and an oracle set A, which may be any language over the tape alphabet of M. The machine M has three distinguished states $q_?$, q_{yes} and q_{no}, along with its initial and final states, and one of its work tapes is distinguished as the oracle tape. At any point during a computation of M on x relative to A, there are two possibilities:

(i) The current state of M is not its query state $q_?$ (although it might be one of the response states q_{yes}, q_{no}). In this case the next step of the computation is determined by the transition function of M, as for an ordinary Turing acceptor. During such steps M can read from and write on its oracle tape, as well as its other work tapes.

(ii) M has entered its query state $q_?$, in order to make an oracle call. In this case the next step is determined by the string on the oracle tape and the oracle set: if the (nonblank) contents of the oracle tape is the string z, then the next state is q_{yes} if $z \in A$ and is q_{no} if $z \notin A$. During a step that is an oracle call, the oracle tape is erased (i.e., reset to blanks) but the configuration of the other work tapes and the input tape is unchanged.

The oracle machine M is deterministic if its transition function allows at most one move at any step, nondeterministic otherwise. The transition function is undefined for the query state, so moves from that state are uniquely determined by A. M is said to accept x relative to A if and only if some computation of M on x relative to A reaches an accepting state. Let M(A) denote the set of strings accepted by M relative to A.

This definition of oracle machines is essentially that used in [39] and differs from the model used in [3,16] in that the oracle tape is erased after an oracle call. This convention is made here to allow a simpler form for the representation of languages accepted by oracle machines in terms of language-theoretic operations (Theorem 2.3.1). There is no difference in computational power when the class of oracle machines that operate in arbitrary polynomial time bounds is considered. It should be apparent that a multitape Turing acceptor is (equivalent to) an oracle machine that never queries its oracle, and conversely.

More formally, a k-tape oracle machine is a (k+9)-tuple $M = (K, \Sigma, \Gamma_1, \ldots, \Gamma_k, \delta, q_0, q_?, q_{yes}, q_{no}, F, j)$ where the components have the following interpretations:

(1) $K, \Sigma, \Gamma_1, \ldots, \Gamma_k$ are finite sets, the state set, the input alphabet and the alphabets for tapes 1 through k, respectively. The set $\Gamma = \cup \{ \Gamma_i : 1 \le i \le k \}$ is the tape alphabet of M, and $\Gamma \cup \Sigma$ is the alphabet. Let B denote the blank tape symbol, $B \notin \Gamma$.

(2) $q_0 \in K$ is the initial state of M, $q_?$ is the query state, q_{yes}

and q_{no} are the response states, and $F \subseteq K$ is the set of final states.

(3) The j-th tape is the oracle tape, and $1 \le j \le k$. The set Γ_j will be termed the oracle tape alphabet of M.

(4) δ is the transition function of M, a function from

$(K - \{q_?\}) \times (\Sigma \cup \{e\}) \times (\Gamma_1 \cup \{B\}) \times \ldots \times (\Gamma_k \cup \{B\})$

into the finite subsets of

$K \times [(\Gamma_1 \cup \{B\}) \times \{0,1,-1\}] \times \ldots \times [(\Gamma_k \cup \{B\}) \times \{0,1,-1\}]$.

For simplicity, the dynamics of an oracle machine will be described only for the case of a two-tape machine $M = (K, \Sigma, \Gamma_1, \Gamma_2, \delta, q_0, q_?, q_{yes}, q_{no}, F, 2)$. Note that the second tape of M is the oracle tape.

An instantaneous description (ID) of M is a 6-tuple $(q, w, y_1, y_2, i_1, i_2)$ where $q \in K$, $w \in \Sigma^*$, $y_j \in (\Gamma_j \cup \{B\})^+$ and $1 \le i_j \le |y_j|$. The components of an ID have the usual interpretations: M is in state q, with w remaining on the input tape, y_1 on tape 1, and y_2 on tape 2 and is reading the i_j-th symbol (from the left) of y_j, $j = 1,2$.

If A is a subset of Γ_2^* then the yield relation $\vdash_A$ on IDs of M relative to A is defined as follows.

(1) Suppose $q \in K - \{q_?\}$, $a \in \Sigma \cup \{e\}$, $a_j \in \Gamma_j \cup \{B\}$ for $j = 1,2$, and $(q', b_1, d_1, b_2, d_2) \in \delta(q, a, a_1, a_2)$ for some $b_1 \in \Gamma_1 \cup \{B\}$, $b_2 \in \Gamma_2 \cup \{B\}$ and $d_1, d_2 \in \{0,1,-1\}$. Then for any $x_1, y_1 \in (\Gamma_1 \cup \{B\})^*$, $x_2, y_2 \in (\Gamma_2 \cup \{B\})^*$, $w \in \Sigma^*$,

$(q, aw, x_1a_1y_1, x_2a_2y_2, |x_1| + 1, |x_2| + 1) \vdash_A (q', w, z_1, z_2, i_1, i_2)$

where for $j = 1,2$

(i) if $d_j = 0$ then $z_j = x_jb_jy_j$ and $i_j = |x_j| + 1$.

(ii) if $d_j = 1$ then $i_j = |x_j| + 2$. If $d_j = 1$ and $y_j = e$ then $z_j = x_jb_jB$; if $y_j \neq e$ then $z_j = x_jb_jy_j$.

(iii) if $d_j = -1$ and $x_j = e$ then $i_j = 1$ and $z_j = Bb_jy_j$; if $d_j = -1$ and $x_j \neq e$ then $i_j = |x_j|$ and $z_j = x_jb_jy_j$.

(2) Suppose $w \in \Sigma^*$, and for $j = 1, 2$, $y_j \in (\Gamma_j \cup \{B\})^+$ and $1 \leq i_j \leq |y_j|$. Then

$(q_?, w, y_1, y_2, i_1, i_2) \vdash_A (q, w, y_1, B, i_1, 1)$

if and only if $y_2 = B^m zB^j$ for some $z \in \Gamma_2^*$, $m,j \geq 0$; and either $z \in A$ and $q = q_{yes}$ or $z \notin A$ and $q = q_{no}$.

Let $\vdash_A^*$ denote the reflexive and transitive closure of $\vdash_A$. Then the language accepted by M relative to A is defined by

$M(A) = \{x \in \Sigma^*$: for some $q \in F$, $y_1 \in (\Gamma_1 \cup \{B\})^+$, $y_2 \in (\Gamma_2 \cup \{B\})^+$, $(q_0, x, B, B, 1, 1) \vdash_A^* (q, e, y_1, y_2, i_1, i_2)\}$.

In this dissertation only time-bounded oracle machines will be considered. The following definition establishes what will be meant by an oracle machine operating in a time bound: this property is to be independent of the selection of the oracle set. Time-bounding functions are assumed to be nondecreasing and at least as large as the identity function.

Definition 2.1.2. (1) Suppose $t: \mathbb{N} \to \mathbb{N}$ is a nondecreasing function which satisfies $t(n) \geq n$. An oracle machine M is said to operate in time $t(n)$ if for any input x and any oracle set A, every computation of M on x relative to A halts in at most $t(|x|)$ steps. (Recall that an oracle call costs one step.) Thus the time allowed M for a computation is a function of the length of the input string ($|x| = n$).

(2) For a language A and a function $t(n)$, define

DTIME($t(n)$, A) = {M(A): M is a deterministic multitape oracle machine that operates in time $t(n)$}; and

NTIME($t(n)$, A) = {M(A): M is a nondeterministic multitape oracle machine that operates in time $t(n)$}.

Thus for a language A, DTIME($t(n)$, A) (NTIME($t(n)$, A)) consists of languages that can be accepted (nondeterministically) relative to A in time $t(n)$. In particular, since an oracle machine need not consult its oracle, DTIME($t(n)$) $\subseteq$ DTIME($t(n)$, A) and NTIME($t(n)$) $\subseteq$ NTIME($t(n)$, A) for any function $t(n)$ and any language A.

By applying the standard construction for shortening computations of Turing acceptors by a constant factor [30], the following "speed-up" theorem for oracle machines can be derived. Since the work tapes of the machine constructed are "compacted" versions of the tapes of the original machine, the oracle set must be altered; the translation can be described as the application of an inverse homomorphism.

Proposition 2.1.3. Suppose M is an oracle machine which operates in time $t(n)$. Then for any $k \geq 1$ and any oracle set A for M, there exist an oracle machine M' and a homomorphism h such that

(i) $M(A) = M'(h^{-1}(A))$;

(ii) M' is deterministic if M is deterministic; and

(iii) M' operates in time $n + t(n)/k$. □

The next proposition uses a construction for "composing" oracle machines with deterministic Turing acceptors, in order to give an upper bound on the complexity of languages accepted by time-bounded oracle machines. This fact appears in [38] in the context of polynomial time bounds.

Definition 2.1.4. A function $t: \mathbb{N} \to \mathbb{N}$ is superadditive if for all n,m, $t(n) + t(m) \leq t(n+m)$. Note that polynomials are superadditive functions.

Proposition 2.1.5. Let $t_1(n)$, $t_2(n)$ be time-bounding functions, with $t_1(n)$ superadditive. If $A \in \text{DTIME}(t_1(n))$ then

$\text{DTIME}(t_2(n), A) \subseteq \text{DTIME}(2t_1(t_2(n)))$, and

$\text{NTIME}(t_2(n), A) \subseteq \text{NTIME}(2t_1(t_2(n)))$.

Proof. If A is a language in $\text{DTIME}(t_1(n))$, let M_1 be a deterministic Turing machine that accepts A and operates in time $t_1(n)$. Suppose M is an oracle machine that operates in time $t_2(n)$, and let $L_2 = M(A)$. The machines M_1 and M are combined to construct a Turing

machine M_2 to accept L_2. Given an input x, M_2 begins a computation of M on x; if M would query its oracle, M_2 instead uses M_1 to test whether the string on the oracle tape is in L_1 and then continues the computation of M from the appropriate state. (Note that we assume that M_1 always halts.) Clearly M_2 will accept precisely L_2; since M_1 is deterministic, M_2 will be deterministic if M is and nondeterministic otherwise. Suppose that on some input x of length n, M_2 follows a computation of M on x in which the oracle was queried about string $z_1, \ldots, z_m$ $(m \geq 0)$. The length of that computation of M_2 is then at most $t_2(n) + [t_1(|z_1|) + \ldots + t_1(|z_m|)]$.

Since the oracle tape of M is erased after every oracle call, $\sum_{i=1}^{m} |z_i| \leq t_2(n)$; hence since t_1 is a superadditive function, $\sum_{i=1}^{m} t_1(|z_i|) \leq t_1(t_2(n))$. Therefore M_2 takes at most $t_2(n) + t_1(t_2(n)) \leq 2t_1(t_2(n))$ steps in any computation on an input of length n. □

It is a simple matter to alter the argument for Proposition 2.1.5 when M_1 is a deterministic oracle machine rather than a Turing acceptor. Thus, for example, if $A \in \text{DTIME}(t_1(n), B)$ and $t_1(n)$ is superadditive, then $\text{NTIME}(t_2(n), A) \subseteq \text{NTIME}(2t_1(t_2(n)), B)$.

2.2. RELATIVE COMPUTATION IN LINEAR AND POLYNOMIAL TIME

The remaining chapters deal primarily with oracle machines that operate in time bounds which are linear functions or polynomials. We therefore establish the following notation for the families of languages accepted by such machines.

Definition 2.2.1. (1) An oracle machine is termed a linear-time oracle machine if it operates in time $cn + d$ for some constants c, d. If an oracle machine M operates in time $t(n)$ and $t(n)$ is a polynomial (in n), then M is a polynomial-time oracle machine.
(2) If $\mathcal{C}$ is a class of languages, define

$$\mathrm{NL}(\mathcal{C}) = \{M(A)\colon A \in \mathcal{C},\ M \text{ a nondeterministic linear-time oracle machine}\}$$
$$= \cup\,\{\mathrm{NTIME}(cn + d\,,\ A)\colon\ A \in \mathcal{C},\ c,d \geq 0\}\ ;$$

$$\mathrm{NP}(\mathcal{C}) = \{M(A)\colon\ A \in \mathcal{C},\ M \text{ a nondeterministic polynomial-time oracle machine}\};\ \text{and}$$

$$\mathrm{P}(\mathcal{C}) = \{M(A)\colon\ A \in \mathcal{C},\ M \text{ a deterministic polynomial-time oracle machine}\}.$$

When no confusion can result, we will write, e.g., NL(A) for NL({A}). By analogy with the notational scheme used for families defined by Turing acceptors, let DTIME(lin, $\mathcal{C}$) denote $\cup\,\{\mathrm{DTIME}(cn + d,\ A)\colon\ A \in \mathcal{C},\ c,d \geq 0\}$.

The following proposition is immediate from Proposition 2.1.5 and previous remarks.

Proposition 2.2.2.

(1) $\mathrm{NL}(\{\emptyset\}) = \mathrm{NL}(\mathrm{DTIME}(\mathrm{lin})) = \mathrm{NTIME}(n)$.
$\mathrm{DTIME}(\mathrm{lin},\{\emptyset\}) = \mathrm{DTIME}(\mathrm{lin})$.

(2) $\mathrm{NP}(\{\emptyset\}) = \mathrm{NP}(\mathrm{DTIME}(\mathrm{poly})) = \mathrm{NTIME}(\mathrm{poly})$.
$\mathrm{P}(\{\emptyset\}) = \mathrm{P}(\mathrm{DTIME}(\mathrm{poly})) = \mathrm{DTIME}(\mathrm{poly})$. □

In the notation of [38] parts of Definition 2.2.1 can be restated as

$$B \in \mathrm{P}(\{A\}) \Leftrightarrow B \leq_T^P A \quad \text{and}$$
$$B \in \mathrm{NP}(\{A\}) \Leftrightarrow B \leq_T^{NP} A\ .$$

In [38] the symbol "≤" is used for reducibilities: in this case, the membership problem for B is reduced to that for A by means of a polynomial-time oracle machine. The superscript "P" ("NP") indicates that the reduction is performed deterministically (nondeterministically) in polynomial time and the subscript "T" denotes Turing reducibility. Thus, the relations "$B \in \mathrm{P}(\{A\})$" and "$B \in \mathrm{NP}(\{A\})$" are viewed as restricted Turing reducibilities [47,54]. The structural properties of these reducibilities on recursive sets have been studied recently [38,3,37]. In particular, in [3] it is shown that no general statement can be made about the inclusion (or equality) relations holding among the classes $\mathrm{P}(\mathcal{C})$, $\mathrm{NP}(\mathcal{C})$, co-$\mathrm{NP}(\mathcal{C})$.

Proposition 2.2.3. (1) [3,38] One can construct recursive sets A_1, A_2, A_3 such that $\mathrm{P}(A_1) = \mathrm{NP}(A_1)$, $\mathrm{P}(A_2) \subsetneq \mathrm{NP}(A_2) = \text{co-}\mathrm{NP}(A_2)$ and $\mathrm{NP}(A_3) \neq \text{co-}\mathrm{NP}(A_3)$.

(2) One can construct recursive sets B_1, B_2, B_3 such that

$DTIME(lin, B_1) = NL(B_1)$, $DTIME(lin, B_2) \subsetneq NL(B_2) = co\text{-}NL(B_2)$ and $NL(B_3) \neq co\text{-}NL(B_3)$. □

The second part of this proposition can be proved by simple modifications of the arguments for the polynomial case. Note that the existence of the languages A_3 and B_3 implies that neither of the relations "$B \in NP(A)$", "$B \in NL(A)$" is transitive.

Several positive closure properties are known to hold for a wide variety of classes of languages accepted by abstract automata, for example, closure under the operations corresponding to an "abstract family of languages" [20]: union, product, Kleene *, intersection with regular sets, inverse homomorphism and nonerasing homomorphism. For multitape devices the class of languages defined is usually closed under intersection. The following theorem gives these closure properties in the context of oracle machines; the statement of the theorem refers only to classes of languages $NL(\mathcal{C})$, but it will be clear which of the constructions apply to $P(\mathcal{C})$ and $NP(\mathcal{C})$ as well.

<u>Theorem 2.2.4</u>. (1) For any nonempty class of languages $\mathcal{C}$:

(i) $\mathcal{C} \cup co\text{-}\mathcal{C} \subseteq NL(\mathcal{C})$; and

(ii) $NL(\mathcal{C})$ is closed under marked +, Kleene *, linear-erasing homomorphism, inverse homomorphism, and union and intersection with languages in DTIME(lin).

(2) If either $\mathcal{C}$ is closed under marked union or consists of a single language, then $NL(\mathcal{C})$ is also closed under marked product, product, intersection, marked union and union.

The proof is for the most part applications of the standard constructions to oracle machines, and is given only for completeness. The operations of union, intersection and product differ from the others in that possibly two oracle sets are involved; use of two oracle sets is reduced to one by applying the operation of marked union.

<u>Proof</u>. (1) First suppose S is any alphabet. It is easy to construct deterministic oracle machines D_1 and D_2, both of which operate in time $n+1$, such that for any language $L \subseteq S^*$, $D_1(L) = L$ and $D_2(L) = S^* - L$; therefore if $L \in \mathcal{C}$ then L and $S^* - L$ are in $NL(\mathcal{C})$.

Now suppose $A \in \mathcal{C}$ and M is a nondeterministic linear-time oracle machine, so that $L = M(A)$ is a representative element of $NL(\mathcal{C})$. Let S be the input alphabet of M and suppose M operates in time $cn + d$.

(a) Suppose ¢ is a new symbol, $¢ \notin S$. Let M_1 be an oracle machine that operates as follows: M_1 rejects its input $x \in (S \cup \{¢\})^*$ unless x has the form $x_1¢x_2¢\ldots x_m¢$ for some $m \geq 1$, $x_1, \ldots, x_m \in S^*$. If the input is of the correct form, M_1 acts like M on each segment x_i of x, using its work tapes (and oracle tape) just as M would. Then M_1 can be constructed to operate in time $(2c + d)n$, and $M_1(A) = (M(A)¢)^+ = (L¢)^+$, so $(L¢)^+ \in NL(\mathcal{C})$.

(b) Suppose $h_1: S^* \to T^*$ is a homomorphism with the property that, for some $k \geq 1$, for any $x \in L$, $|x| \leq k \cdot \max\{h_1(x), 1\}$, i.e., h_1

is a homomorphism that performs linear erasing on L. Let M_2 be a nondeterministic oracle machine which, given $y \in T^*$, first guesses a string $x \in S^*$ such that $h_1(x) = y$ and $|x| \le k \cdot \max\{|y|, 1\}$ and then accepts x if and only if M accepts y (relative to the same oracle set). Then M_2 operates in time $c'n + c'$ (where $c' = k(c+2) + d$) and $M_2(A) = h_1(L)$.

(c) Suppose h_2: $T^* \to S^*$ is a homomorphism. Let $k = \max\{|h(a)| : a \in T\}$; then for any x, $|h(x)| \le k|x|$. Let M_3 be an oracle machine which, given $y \in T^*$, first writes $h_2(y)$ on an extra tape and then accepts y if and only if M accepts $h_2(y)$. Then M_3 operates in time $k(c+2)n + d$ and $M_3(A) = h_2^{-1}(L)$.

(d) Suppose $L' \subseteq S^*$ is a language in DTIME(lin). Then L' is accepted by a deterministic Turing machine M' which operates in time kn for some $k \ge 1$. Let M_4 and M_5 be oracle machines which, given an input x, test first whether M accepts x and then test whether M' accepts x. M_4 accepts the input if and only if either of the tests succeeds and M_5 accepts if and only if both succeed. Then M_4 and M_5 both operate in time $(k+c+1)n+d$; and $M_4(A) = L \cup L'$ and $M_5(A) = L \cap L'$.

Note that if M is deterministic then the machines constructed in (a), (c) and (d) will also be deterministic. Closure under Kleene * follows from (a), (b) and (d): $L^* = h(((L \cap S^+)¢)^+) \cup \{e\}$, where h: $(S \cup \{¢\})^* \to S^*$ is the simple homomorphism determined by defining $h(s) = s$ for $s \in S$ and $h(¢) = e$. The homomorphism h is e-limited

on $((L \cap S^+)¢)^+$.

(2) Suppose $L_1, L_2 \subseteq S^*$ are elements of $NL(\mathcal{C})$. First, there exist a language $A_0 \in \mathcal{C}$ and nondeterministic linear-time oracle machines N_1 and N_2 such that $L_i = N_i(A_0)$ for $i = 1,2$. This is clearly the case if $\mathcal{C}$ consists of a single language, $\mathcal{C} = \{A_0\}$. If $\mathcal{C}$ contains more than one language, then since $L_1, L_2 \in NL(\mathcal{C})$ there are languages $A_1, A_2 \in \mathcal{C}$ and nondeterministic linear-time oracle machines N_1' and N_2' such that $L_i = N_i'(A_i)$, $i = 1,2$. Let T be an alphabet such that $A_1, A_2 \subseteq T^*$ and let $\#_1$, $\#_2$ be two distinct symbols not in T. If $\mathcal{C}$ is closed under marked union then $A_0 = \#_1 A_1 \cup \#_2 A_2 \in \mathcal{C}$. For $i = 1,2$, the machine N_i' can easily be altered to mark each string on its oracle tape with $\#_i$ before making an oracle call; the resulting oracle machine N_i will also operate in linear time and $N_i(A_0) = N_i'(A_i) = L_i$.

Now, since $NL(\mathcal{C})$ is closed under linear-erasing homomorphism, it suffices to show closure under marked product, marked union and intersection.

(a) Let $¢ \notin S$ be a new symbol. Let M_6 be an oracle machine which rejects its input x unless $x = x_1¢x_2$ for some $x_1, x_2 \in S^*$. On an input of the correct form, M_6 first tests whether N_1 accepts x_1 and then tests whether N_2 accepts x_2; M_6 accepts x if and only if both tests succeed. Since N_1 and N_2 operate in linear time, so will M_6, and $M_6(A_0) = L_1¢L_2$.

(b) Suppose $¢, \$ \notin S$. Let M_7 be an oracle machine which if given input $¢x$, $x \in S^*$, acts like N_1 on x; if given input $\$x$, $x \in S^*$, acts like N_2 on x; and rejects strings of any other form. Again since N_1 and N_2 operate in linear time, so will M_7, and $M_7(A_0) = ¢L_1 \cup \$L_2$.

(c) Let M_8 be an oracle machine which given input $x \in S^*$, first tests whether N_1 accepts x and then whether N_2 accepts x; M_8 accepts its input if and only if both tests succeed. Then M_8 will operate in linear time, and $M_8(A_0) = L_1 \cap L_2$.

Again, note that the three machines described above will be deterministic if N_1 and N_2 are deterministic. □

2.3. REPRESENTATION OF LANGUAGES ACCEPTED BY ORACLE MACHINES

In the following theorem, language-theoretic operations are used to obtain the language accepted by a linear-time oracle machine from the oracle set and a simpler language. This representation will be used extensively in Chapters 3 and 4, to aid in characterizing $NL(\mathcal{C})$ for certain classes of languages $\mathcal{C}$.

<u>Theorem 2.3.1</u>. (1) Let M be a nondeterministic linear-time oracle machine. Then there exist a length-preserving homomorphism h and a deterministic linear-time oracle machine D such that for any oracle set A, $M(A) = h(D(A))$.

(2) Let D be a deterministic linear-time oracle machine with tape alphabet S. Then there exist a language $L \in DTIME(lin)$, a length-preserving homomorphism h_1 and a homomorphism h_2 such that for any

$A \subseteq S^*$, $D(A) - \{e\} = h_1(L \cap h_2^{-1}(L'))$ where

$L' = (A \oplus (S^* - A))^* = (\#_1 A \cup \#_2(S^* - A))^*$ with $\#_1, \#_2 \notin S$.

<u>Proof</u>. First, some notation is necessary. If Γ is an alphabet and k is an integer, $k \geq 1$, let $\Gamma_k = \{[w]: w \in \Gamma^*, 1 \leq |w| \leq k\}$. That is, for each nonempty string $w \in \Gamma^*$ of length at most k, $[w]$ is a new symbol, and Γ_k is the set of these symbols. For $x \in \Gamma^*$, $x/k \in (\Gamma_k)^*$ is defined as follows. Suppose $|x| = mk + j$, $m \geq 0$, $0 \leq j \leq k-1$. If $j = 0$ then $x/k = [w_1][w_2]\ldots[w_m]$ where $x = w_1\ldots w_m$ and $|w_i| = k$ for $1 \leq i \leq m$; if $j \geq 1$, then $x/k = [w_1]\ldots[w_m][y]$ where $x = w_1\ldots w_m y$, $|w_i| = k$ for $1 \leq i \leq m$ and $|y| = j$. If $L \subseteq \Gamma^*$, let $L/k = \{x/k: x \in L\}$. Note that Γ^*/k is a regular set.

(1) Part (1) is proved by applying to oracle machines a technique used in [10,16,36]. Suppose a nondeterministic oracle machine M operates in time $cn + d$ and has input alphabet T. Let $k = c+d$. Suppose M has at most m choices of transition at any step and let $V = \{v_0, v_1, \ldots, v_m\}$ be an alphabet of $m+1$ distinct symbols. ("v_0" represents a call on the oracle, the only move possible from the query state.) Let $\Sigma = T \times (V_k \cup \{\#\}) = \{[b,\#]: b \in T\} \cup \{[b,[w]]: b \in T, w \in V^*, 1 \leq |w| \leq k\}$ where $\# \notin V_k$.

Let D_1 be a deterministic oracle machine which has input alphabet Σ and on input e simulates all the possible computations of M on e, using its oracle tape just as M would. D_1 rejects any nonempty input strings. Since there are only finitely many computations of M on the empty word, D_1 can be constructed to operate in linear time, and for any oracle set A, $D_1(A) = M(A) \cap \{e\}$. We now consider a deterministic oracle machine D_2 to follow computations of M on non-empty inputs.

D_2 will accept only strings of the form $\Theta(x,u/k) \in \Sigma^*$ with $x \in T^+$, $u \in V^+$, $|u/k| \le |x|$. Given such an input, D_2 accepts if and only if the choice of transitions given by u leads M to accept x (so that D_2 uses its oracle tape just as M would). That is, for any oracle set A, $D(A) = \{\Theta(x,u/k)\colon x \in T^+,\ u \in V^+,\ |u/k| \le |x|$ and u describes choices of transitions by which M accepts x relative to $A\}$. D_2 can be constructed to operate in linear time; since M operates in time $cn + d$ and $cn + d \le kn$ for $n \ge 1$, for any $x \ne e$ and any A, $x \in M(A)$ if and only if there is an accepting computation of M on x relative to A with at most $k|x|$ steps if and only if there is some $u \in V^+$ such that $\Theta(x,u/k) \in D_2(A)$. Using a construction similar to those in Theorem 2.2.4, there is a deterministic linear time oracle machine D such that for any A, $D(A) = D_1(A) \cup D_2(A)$. Let $h\colon \Sigma^* \to T^*$ be the length-preserving homomorphism defined by $h([b_1,b_2]) = b_1$ for $b_1 \in T$, $b_2 \in V_k \cup \{\#\}$; then $M(A) = h(D(A))$.

(2) Suppose D is a deterministic oracle machine that operates in time $cn + d$ and has input alphabet T and tape alphabet S. Let $\#_1, \#_2 \notin S$ be two new symbols and $U = S \cup \{\#_1,\#_2\}$. Let $\Sigma = T \times (U_k \cup \{\#\})$, where $k = c+d$.

Let $R \subseteq \Sigma^*$ be the regular set $R = \{\Theta(x,y)\colon x \in T^+,\ y \in U^*/k,\ |x| \ge |y|\}$. Let $h_2\colon \Sigma^* \to U^*$ be the homomorphism defined by $h_2([b,\#]) = e$ and for $w \in U^*$, $1 \le |w| \le k$, $b \in T$, $h_2([b,[w]]) = w$. Then if $A \subseteq S^*$ and $L' = (A \oplus (S^* - A))^*$, $h_2^{-1}(L') \cap R = \{\Theta(x,y)\colon x \in T^+,\ y \in L'/k,\ |x| \ge |y|\}$.

Let D' be the following deterministic Turing acceptor, with input alphabet Σ. D' rejects its input unless it is of the form $\Theta(x,u/k)$ with $x \in T^+$, $u \in U^*$ and $|u/k| \le |x|$. On an input of this form, D' acts like D would on input x, using the information in $u = \#_{i_1}u_1 \cdots \#_{i_m}u_m$ instead of oracle calls; that is, D' checks that D would query its oracle about $u_1, \ldots, u_m$ (in that order) and D' continues from the "yes" state if $i_j = 1$ and from the "no" state if $i_j = 2$, $1 \le j \le m$. D' accepts $\Theta(x,u/k)$ if and only if the answers in u lead D to accept x. Now since D operates in time $cn + d$ and the oracle tape is erased after an oracle call, if during a computation on x, D queries its oracle about strings $u_1, \ldots, u_m$, $m \ge 0$, and receives "answers" $i_1, \ldots, i_m$ then $m + |u_1| + \ldots + |u_m| \le c|x| + d \le k|x|$ for $x \ne e$; hence if $u = \#_{i_1}u_1 \cdots \#_{i_m}u_m$ then $|u/k| \le |x|$. Further, for any oracle set A, the answers in u are correct relative to A if and only if $u \in (\#_1A \oplus \#_2(S^* - A))^* = L'$. Therefore for any $x \in T^+$, $x \in D(A)$ if there exists $u \in U^*$ such that $\Theta(x,u/k) \in L(D')$ and $u \in L'$; or, $x \in D(A)$ if and only if there exists $y \in U_k^*$ with $|y| \le |x|$ such that $\Theta(x,y) \in L(D') \cap (h_2^{-1}(L') \cap R)$.

Let h_1: $\Sigma^* \to T^*$ be the length-preserving homomorphism determined by defining $h_1([b_1,b_2]) = b_1$. Let $L = L(D') \cap R$; since R is a regular set and D' can be constructed to operate in linear time, $L \in$ DTIME(lin). Then $D(A)-\{e\} = h_1(L \cap h_2^{-1}(L'))$. □

The two parts of Theorem 2.3.1 are combined with the closure properties given in Theorem 2.2.4 to yield the following corollary.

Corollary 2.3.2. For any class of languages $\mathcal{C}$,

$NL(\mathcal{C}) = \{h_1(L \cap h_2^{-1}((A \oplus (S^* - A))^*))$: $A \subseteq S^*$ in $\mathcal{C}$, h_1 a length-preserving homomorphism, h_2 a homomorphism, $L \in$ DTIME(lin)$\}$.

Proof. Suppose $A \subseteq S^*$ is in $\mathcal{C}$. Then from Theorem 2.2.4 A and $S^* - A$ are in NL({A}) and NL({A}) is closed under marked union, Kleene *, inverse homomorphism, intersection with languages in DTIME(lin) and linear-erasing (hence length-preserving) homomorphism. Therefore if $L \in$ DTIME(lin), h_1 is a length-preserving homomorphism and h_2 is an arbitrary homomorphism, then

$h_1(L \cap h_2^{-1}((A \oplus (S^* - A))^* \in NL(\{A\}) \subseteq NL(\mathcal{C})$.

On the other hand, if $L \in NL(\mathcal{C})$ then $L = M(A)$ for some $A \subseteq S^*$ in $\mathcal{C}$ and M a nondeterministic linear-time oracle machine. From Theorem 2.3.1 (1), there is a length-preserving homomorphism h and a deterministic linear-time oracle machine D such that $L = M(A) = h(D(A))$. From Theorem 2.3.1 (2), there is a length-preserving homomorphism g_1, a homomorphism h_2 and a language $L' \in$ DTIME(lin) such that

$D(A) - \{e\} = g_1(L' \cap h_2^{-1}((A \oplus (S^* - A))^*))$. Let h_1 be the homomorphism that is the composition of h with g_1; that is, h_1 is determined by defining, for a symbol a, $h_1(a) = h(g_1(a))$. Then h_1 is also a length-preserving homomorphism. Let $\hat{L} = L' \cup (D(A) \cap \{e\})$; since $D(A) \cap \{e\}$ is either empty or the singleton set $\{e\}$, it is a regular set, so $\hat{L} \in$ DTIME(lin). Then $L = h_1(\hat{L} \cap h_2^{-1}((A \oplus (S^* - A))^*))$. □

In Chapters 3 and 4, Theorem 2.3.1 will be used to give simpler representations for $NL(\mathcal{C})$ than Corollary 2.3.2 when $\mathcal{C}$ satisfies certain conditions; e.g., for some classes of languages $\mathcal{C}$,

$NL(\mathcal{C}) = \{h(L_1 \cap (S^* - L_2))$: $L_1, L_2 \subseteq S^*$ in $\mathcal{C}$, h a length-preserving homomorphism$\}$.

The constructions used in the proof of Theorem 2.3.1 can be applied to oracle machines which do not necessarily operate in linear time, yielding the following generalization of the representation to arbitrary time bounds.

Proposition 2.3.3. Suppose M is a nondeterministic oracle machine which operates in time $t(n)$ and has tape alphabet S. Then there exist homomorphisms h_1 and h_2 and a language $L_M \in$ DTIME(lin) such that for any oracle set $A \subseteq S^*$, $M(A) = h_1(L_M \cap h_2^{-1}((A \oplus (S^* - A))^*))$. Further, the homomorphism h_1 has the property that for any $z \in L_M$, $|z| \le t(|h_1(z)|)$.

Proof. (sketch) The language L_M is given by :

$L_M = \{\Theta(x,y,z)$: the transitions described in y and the information given in z about the oracle set cause M to accept $x\}$. The homomorphisms h_1 and h_2, then, satisfy $h_1(\Theta(x,y,z)) = x$ and $h_2(\Theta(x,y,z)) = z$. Since M operates in time $t(n)$, if $\Theta(x,y,z) \in L_M$ then $|\Theta(x,y,z)| = |y| \le t(|x|) = t(|h_1(\Theta(x,y,z))|)$. Since the homomorphism h_1 does some erasing, the language L_M does not depend on A. Note that the strings y and z are now written "one symbol per square," rather than k symbols as in Theorem 2.3.1; thus for a symbol a, $|h_1(a)|, |h_2(a)| \le 1$. □

If we drop the convention that the oracle tape is erased after an oracle call, then a representation similar to Proposition 2.3.3 still holds. However, it may no longer be the case that for $\Theta(x,y,z) \in L_M$, $|z| \le |y| \le t(|x|)$, although $|z| \le (t(|x|))^2$.

The conclusion of Proposition 2.3.3 may be restated as follows: for $B = M(A)$,

$x \in B$ iff there exists a string $\Theta(x,y,z)$ such that $\Theta(x,y,z) \in L_M$ and $z \in (A \oplus (S^* - A))^*$.

In this form, it is similar to the definition given in [47] of the relation "B is r.e. in A." Both allow separate consideration of two conditions that must be satisfied by an alleged computation of M with oracle set A: (1) moves from the query state must be consistent with the answers from the oracle and other moves must follow from the transition function of M; and (2) the answers from the oracle must be correct relative to A. This separation of the oracle calls from the other moves

can be used to simplify proofs about oracle machines, in particular, the proof of the following fact, which states that the multiple work tapes of a nondeterministic oracle machine may be replaced with a fixed number of tapes at the cost of only a linear increase in time.

<u>Corollary 2.3.4</u>. Suppose M is an oracle machine, which operates in time $t(n)$. Then there exists a nondeterministic oracle machine M' with 3 work tapes such that for any oracle set A

(i) $M'(A) = M(A)$; and

(ii) every accepting computation of M' relative to A on an input of length n has to most $3t(n)$ steps (i.e., M' "accepts" in time $3t(n)$).

<u>Proof</u>. Let M be an oracle machine and let S be the tape alphabet of M. From the previous proposition, there exist homomorphisms h_1, h_2 and a language $L_M \in$ DTIME(lin) such that for any oracle set $A \subseteq S^*$ $M(A) = h_1(L_M \cap h_2^{-1}((A \oplus (S^* - A))^*))$. Also, if M operates in time $t(n)$, then $|u| \le t(|h_1(u)|)$ for $u \in L_M$.

Since $L_M \in$ DTIME(lin) $\subseteq$ NTIME(n), from [9] there is a nondeterministic real-time Turing machine M_1 with 2 work tapes that accepts L_M. The nondeterministic oracle machine M' operates as follows: given an input string x (over the input alphabet of M), M' guesses, one symbol per step, a string $u = \Theta(x,y,z)$ for some y and z, reading x while guessing the first $|x|$ symbols of u. As each symbol u_i of u is guessed, M' writes the corresponding symbol $h_2(u_i)$ of z on tape 1 and uses u_i as the next input symbol to M_1. M' uses tapes

2 and 3 as the work tapes of M_1; the third tape is the oracle tape of M' but the oracle is not queried during this phase of the computation. If M_1 enters an accepting state then the string $\theta(x,y,z)$ guessed up to that point is in L_M, so its length is at most $t(|x|)$. If M_1 accepts, then M' proceeds to copy each segment $z_j \in S^*$ of $z = \#_{i_1} z_1 \ldots \#_{i_m} z_m$ $(m \geq 0)$ onto the oracle tape, querying its oracle about z_j and comparing the response to the marker $\#_{i_j}$. This phase of the computation takes $|z| + m \leq 2|z|$ steps. For any z and A, M' accepts x relative to A if and only if there exist strings y,z such that $\theta(x,y,z) \in L_M$ and $z \in (A \oplus (S^* - A))^*$, so $M'(A) = M(A)$. If M' does accept x (relative to A), by guessing strings y,z, then $|y|,|z| \leq t(|x|)$; therefore that computation of M' on x has length at most $|\theta(x,y,z)| + 2|z| = |y| + 2|z| \leq 3t(|x|)$. □

Recall that a function $t(n)$ is a "running time" if there is a deterministic Turing machine which on any input of length n takes exactly $t(n)$ steps. If in Corollary 2.3.4 $t(n)$ is a running time, then with the addition of some finite number of extra tapes, the oracle machine M' can be constructed to operate (rather than accept) in time $3t(n)$. In the context of linear time bounds, this fact will be used in Chapter 4, so it is stated as a separate corollary.

Corollary 2.3.5. For any class of languages $\mathcal{C}$,

$NL(\mathcal{C}) = \{M(A)\colon A \in \mathcal{C},\ M$ a nondeterministic linear-time oracle machine with 4 tapes$\}$. □

2.4. A RELATIVIZED TIME-HIERARCHY THEOREM

As is the case with the analogous result for Turing acceptors, Corollary 2.3.4 allows construction of a nondeterministic oracle machine which can simulate computations of any oracle machine that operates in time $t(n)$ and which itself operates in time not much larger than $t(n)$. The known methods of deterministic simulation with a fixed number of tapes of deterministic oracle machines are less efficient. These simulation techniques are used in the following theorem to give conditions on functions $t_1(n), t_2(n)$ which ensure that relative computation in time $t_2(n)$ is more powerful than relative computation in time $t_1(n)$.

Theorem 2.4.1. Suppose A is a recursive language and $t_2(n)$ is a running time.

(1) If $\lim_{n\to\infty} \dfrac{t_1(n)\lg(t_1(n))}{t_2(n)} = 0$ then

$DTIME(t_2(n), A) \not\subseteq DTIME(t_1(n), A)$.

(2) If $\lim_{n\to\infty} \dfrac{t_1(n+1)}{t_2(n)} = 0$ then

$NTIME(t_2(n), A) \not\subseteq NTIME(t_1(n), A)$. □

See Appendix A for the proof of Theorem 2.4.1. The proofs of parts (1) and (2) are essentially the same as the proofs for the analogous results for Turing acceptors [28,49,50]. In part (1), it is not necessary that A be recursive, since the proof is by diagonalization; the recursiveness of A is used to derive a contradiction in the proof of (2). In both cases the proof is uniform, in the sense that the oracle machine constructed to demonstrate the non-containment does not depend on A but only on an alphabet Σ such that $A \subseteq \Sigma^*$.

Theorem 2.4.1 will only be applied in the context of polynomial time bounds. It implies in particular that for a recursive language A, NP({A}) (and P({A})) can be decomposed into an infinite hierarchy of classes based on the degree of the polynomial used as a time-bounding function.

<u>Corollary 2.4.2</u>. For any recursive language A

(1) $NL(\{A\}) \subsetneq NP(\{A\})$; and

(2) for any polynomial p(n),

$$DTIME(p(n), A) \subsetneq P(\{A\})$$

$$NTIME(p(n), A) \subsetneq NP(\{A\}).$$

<u>Proof</u>. Part (2) follows easily from the theorem. For part (1), suppose A is a language and M is a nondeterministic oracle machine which operates in time $cn + d$. The language $L = \{x \in M(A): |x| < c + d\}$ is finite, so let M' be a nondeterministic oracle machine which acts like M on input strings of length at least $c + d$ and accepts strings in L by simply reading its input. Then M' operates in time n^2 and $M'(A) = M(A)$; so $NL(\{A\}) \subseteq NTIME(n^2, A)$. If A is a recursive language then from part (2), $NL(\{A\}) \subseteq NTIME(n^2, A) \subsetneq NP(\{A\})$. □

One might consider extending Theorem 2.4.1 to classes of languages, to find, for example, conditions on a class of languages $\mathcal{C}$ such that $NL(\mathcal{C}) \subsetneq NTIME(n^3, \mathcal{C}) = \cup \{NTIME(n^3, A): A \in \mathcal{C}\}$. If the class NTIME(poly) satisfied those conditions, then it will follow from results in Chapter 5 that NTIME(poly) could not be closed under complementation. However, the proofs of Theorem 2.4.1 (1) and (2) seem to apply only when the class $\mathcal{C}$ has a simple structure; and there exist classes $\mathcal{C}$ for which $\mathcal{C} = NL(\mathcal{C}) = NP(\mathcal{C})$ (e.g., $\mathcal{C}$ = PSPACE).

In the remainder of this dissertation, frequent application will be made of the "mathematical machinery," representation theorems and closure properties established in this chapter.

Chapter 3: THE RUDIMENTARY RELATIONS AND RELATIVE COMPUTATION

In this chapter the rudimentary relations will be defined and investigated as a family of formal languages. The class of rudimentary relations is the smallest class of string relations containing the concatenation relations "xy = z" and closed under some natural operations, called the rudimentary operations (basically the Boolean operations and a form of bounded quantification). Extending the work of Quine [44] on definability from concatenation, Smullyan [51] introduced the rudimentary relations (or "attributes") and used them in a development of recursive function theory based on string manipulation. By taking this approach, Smullyan identified small bases for the recursively enumerable sets and proved normal form theorems without relying on number theory.

The rudimentary relations are "constructively" definable from concatenation; hence they may be viewed as a string-theoretic analog of the constructive arithmetic relations, also defined by Smullyan [51]. The class of constructive arithmetic relations is the smallest class of relations on natural numbers containing addition and multiplication and closed under number-theoretic versions of the rudimentary operations (the Boolean operations, finite quantification and explicit transformation). Since most formal models for computation and, in particular, for restricted computation are based on string manipulation, it is appropriate for the study of computational complexity to consider the rudimentary relations rather than the constructive arithmetic relations. Results about

either class apply to both, however, since Bennett [4] has shown that the two classes are the same when strings are viewed as numerals. Interest in the rudimentary relations is strengthened by the observation that although it is a class of "low" complexity (contained in Grzegorczyk's class $\mathcal{E}^0_*$ [25]) yet it contains exponentiation (as the relation "$n^m = p$" [4]) and forms a basis for the recursively enumerable sets. We will see in this chapter and the next how questions which remain open about the rudimentary relations are tied to some important open questions in automata-based computational complexity.

The first section of this chapter contains the definition used here for the class of rudimentary relations; it is equivalent to the definitions used by Smullyan and others [4, 33, 57]. To allow comparison of the rudimentary relations to classes of languages, the mapping θ (given in Chapter 1) is used. No information is lost in passing from the rudimentary relations to the family of languages associated with it under this encoding, since a relation is rudimentary if and only if the language encoding it is a (1-ary) rudimentary relation. In Section 2, some properties of the rudimentary relations are established and the machinery developed in Chapter 2 is applied to the class of (encodings of) rudimentary relations. As a result, the family of rudimentary relations is characterized as the smallest nonempty class of languages "closed under" the operator NL( ). Finally, in Section 3, parts of the proof of this characterization are examined more closely, to extract the information they contain about classes of languages in general and about language-theoretic closure properties. In particular, the class of rudimentary relations is shown to be the class of languages generated by the language $\{a^n b^n\colon n \geq 0\}$ under the operations of inverse homomorphism, length-preserving homomorphism and the Boolean operations.

3.1. DEFINITION OF THE RUDIMENTARY RELATIONS

We begin with a definition of the rudimentary operations and the rudimentary relations. The definition is in a more general form than that given by Smullyan [51] and is based on the definition of Jones [33].

Definition 3.1.1. (1) We shall call the following operations the "rudimentary operations."

(i) Explicit transformation: An explicit transformation of a relation R is obtained by adding redundant variables, identifying or permuting variables, or substituting a string for a variable. That is, $Q \subseteq [S^*]^n$ is defined by explicit transformation from $R \subseteq [S^*]^m$ if and only if $Q = \{(x_1, \ldots, x_n)\colon (t_1, \ldots, t_m) \in R\}$ where for $1 \leq i \leq m$, t_i is a string containing symbols from S or variables $x_1, \ldots, x_n$ (or both). For example, t_1 might be x_2 or a string $w \in S^*$ or $x_1 w x_2$.

(ii) Boolean operations: The Boolean operations are union, intersection and difference of relations over the same alphabet.

(iii) Bounded existential quantification: Suppose $R \subseteq [S^*]^{n+1}$, $n \geq 0$. A relation $Q \subseteq [S^*]^{n+1}$ is defined by bounded existential quantification from R if and only if $Q = \{(x_1, \ldots, x_n, y)\colon$ for some $z \in S^*$ such that $|z| \leq |y|$, $(x_1, \ldots, x_n, z) \in R\}$. This will also be written

$Q = \exists^{\leq} R$.

(2) For an alphabet S, the concatenation relation C_S on S^* is defined to be $C_S = \{(x, y, z)\colon x,y,z \in S^*, xy = z\}$. Define $RUD(S)$ to be the class of relations on S^* definable from C_S by a finite number of applications of the rudimentary operations; that is, $RUD(S)$ is the smallest class of relations containing C_S and closed under the Boolean operations, bounded existential quantification and explicit transformation. Finally, define $RUD = \cup \{RUD(S)\colon S$ any finite alphabet$\}$.

The definition of the rudimentary relations in [51] restricts them to the alphabet $\{1,2\}$, and in [4] separate classes of m-rudimentary relations are defined for each $m \geq 1$; in both these definitions the operation of explicit transformation is restricted to a simpler form, in which each term t_i is either a constant string or one of the variables. Clearly if $\#(S) = \#(T)$ then $RUD(S)$ is equal to $RUD(T)$ (i.e., isomorphic under a renaming of the symbols); it is shown in [33] that for $\#(S) = m$, $RUD(S)$ is equal to the m-rudimentary relations. Moreover, in a certain sense (discussed below) the alphabet may be restricted to two letters without changing the class of relations defined.

A string relation may be viewed as a relation on natural numbers, in the following way. If $S = \{s_1, \ldots, s_m\}$ is an alphabet with m symbols, let $e_S\colon \mathbb{N} \to S^*$ be the bijection which assigns to a number its m-adic notation. That is, $e_S(0) = e$ and for $n \geq 1$, $e_S(n) = s_{i_1} s_{i_2} \cdots s_{i_k}$ if and only if $n = \sum_{j=1}^{k} i_j m^{k-j}$. The function e_S

is extended to tuples of numbers and to relations on numbers in the obvious way. The results in [4,33] show that for $\#(S),\#(T) \geq 2$, if W is a relation on $\mathbb{N}$ then $e_S(W) \in RUD(S)$ if and only if $e_T(W) \in RUD(T)$; further, for $\Sigma = \{1,2\}$ a relation W on $\mathbb{N}$ is constructive arithmetic if and only if $e_\Sigma(W) \in RUD(\Sigma)$. Hence $RUD = RUD(\Sigma)$ when they are viewed as relations on natural numbers, and, also in this sense, *RUD* is the class of constructive arithmetic relations.

The class of rudimentary relations is closed under two other forms of bounded existential quantification. If $n \geq 1$, $R \subseteq [S^*]^{n+1}$, define

(i) $\varepsilon(R) = \{(x_1, \ldots, x_n)$: there exists $z \in S^*$ such that $|z| \leq \max\{|x_i|: 1 \leq i \leq n\}$ and $(x_1, \ldots, x_n, z) \in R\}$; and

(ii) for $1 \leq j \leq n$,
$\varepsilon_j(R) = \{(x_1, \ldots, x_n)$: there exists $z \in S^*$ such that $|z| \leq |x_j|$ and $(x_1, \ldots, x_n, z) \in R\}$.

Let $Q = \exists{\leq}\, R$ be as in Definition 3.1.1. Then $\varepsilon_j(R)$ is an explicit transformation of Q: for $1 \leq j \leq n$,

$$\varepsilon_j(R) = \{(x_1, \ldots, x_n): (x_1, \ldots, x_n, x_j) \in Q\}.$$

Also, $\varepsilon(R)$ can be defined from Q using union and explicit transformation: $\varepsilon(R) = \varepsilon_1(R) \cup \varepsilon_2(R) \cup \ldots \cup \varepsilon_n(R)$.

The same class *RUD* of relations results if the operations ε_i, $i \geq 1$, are used in place of $\exists{\leq}$ in the definition of the rudimentary

operations. To see this, let $R' \subseteq [S^*]^{n+2}$ be the explicit transformation of R defined by

$$R' = \{(x_1, \ldots, x_{n+2}): (x_1, \ldots, x_n, x_{n+2}) \in R\};$$

then $Q = \varepsilon_{n+1}(R')$. When ε_i is used to define a relation R_1 from R, this will also be written

$$(x_1, \ldots, x_n) \in R_1 \iff (\exists z)_{x_i} [(x_1, \ldots, x_n, z) \in R].$$

Notice that this is a quantification in which the length of z is bounded by the length of x_i (rather than one in which the number represented by z is bounded by the number x_i represents).

The class of rudimentary relations is also preserved if constant multiples of the lengths of the variables are used to bound the quantification. For example, suppose k is an integer and

$Q' = \{(x_1, \ldots, x_n, y)$: there exists $z \in S^*$ such that $|z| \leq k|y|$ and $(x_1, \ldots, x_n, z) \in R\}$.

Then Q' is an explicit transformation of Q:

$$Q' = \{(x_1, \ldots, x_n, y): (x_1, \ldots, x_n, y^k) \in Q\}$$

where $y^k = \underbrace{yy\ldots y}_{k}$ is the string y concatenated with itself k times.

In Theorem 3.2.7 the rudimentary relations will be compared to a certain class of languages; to make this comparison we associate the language $\theta(R)$ with a string relation R. Myhill used the mapping θ in a proof that any rudimentary relation could be accepted by a deterministic linear-bounded automaton, i.e., $RUD \subseteq$ DSPACE(n). Other encodings of tuples of strings are possible, for instance the "sequential" one, taking $(x_1, \ldots, x_n)$ to $x_1\# \ldots \#x_n$. The "parallel" encoding θ is used here because it gives rise to simple relationships between the rudimentary operations and language-theoretic operations.

Proposition 3.1.2. If a relation Q is defined using rudimentary operations from a relation R, then $\theta(Q)$ can be defined from $\theta(R)$ and languages in DTIME(lin) by application of Boolean operations, homomorphism and inverse homomorphism. Specifically:

(1) If $R, R' \subseteq [S^*]^n$ and $\# \notin S$ then $\theta(R \cup R') = \theta(R) \cup \theta(R')$, $\theta(R \cap R') = \theta(R) \cap \theta(R')$ and $\theta(R - R') = \theta(R) - \theta(R')$;

(2) If Q is defined from R by bounded existential quantification, then there exist a regular set L_0, a homomorphism h and a length-preserving homomorphism h' such that $\theta(Q) = h'(L_0 \cap h^{-1}(\theta(R)))$; and

(3) If Q is an explicit transformation of R then $\theta(Q)$ can be formed from $\theta(R)$ by applying inverse homomorphism, linear-erasing homomorphism and intersection with languages in DTIME(lin).

Proof. Verification of (1) is straightforward.

For (2), suppose $Q, R \subseteq [S^*]^{n+1}$ and $Q = \exists{\le} R$. Let $L_0 = \{\theta(x_1, \ldots, x_n, y, z): |z| \le |y|\} \subseteq ([S_\#]^{n+2})^*$; then L_0 is a regular set. Let $h: ([S_\#]^{n+2})^* \to ([S_\#]^{n+1})^*$ be the homomorphism determined by defining

$$h([b_1, \ldots, b_{n+2}]) = \begin{cases} e & \text{if } b_1 = \ldots = b_n = b_{n+2} = \# \\ [b_1, \ldots, b_n, b_{n+2}] & \text{else.} \end{cases}$$

Let $h': ([S_\#]^{n+2})^* \to ([S_\#]^{n+1})^*$ be the length-preserving homomorphism determined by defining $h'([b_1, \ldots, b_{n+2}]) = [b_1, \ldots, b_{n+1}]$. Then $h(\theta(x_1, \ldots, x_n, y, z)) = \theta(x_1, \ldots, x_n, z)$ and $h'(\theta(x_1, \ldots, x_n, y, z)) = \theta(x_1, \ldots, x_n, y)$ if $|z| \le |y|$; hence $\theta(Q) = h'(L_0 \cap h^{-1}(\theta(R)))$.

For (3) it is sufficient to show that the statement holds in the following three cases, since any explicit transformation can be built up from transformations of these forms. Suppose $R \subseteq [S^*]^n$ and let

(i) $Q_1 = \{(x_1, \ldots, x_n): (x_{\pi(1)}, \ldots, x_{\pi(n)}) \in R\}$ where π is some permutation on $\{1, 2, \ldots, n\}$;

(ii) $Q_2 = \{(x_1, \ldots, x_{n+1}): (x_1, \ldots, x_n) \in R\}$; and

(iii) $Q_3 = \{(x_1, \ldots, x_{n-1}): (x_1, \ldots, x_{n-1}, t(x_1, \ldots, x_{n-1})) \in R\}$ where $t(x_1, \ldots, x_{n-1})$ is a string containing variables and symbols from S. Let $h_1: ([S_\#]^n)^* \to ([S_\#]^n)^*$ be the length-preserving homomorphism determined by defining $h_1([b_1, \ldots, b_n]) = [b_{\pi(1)}, \ldots, b_{\pi(n)}]$; then $\theta(Q_1) = h_1(\theta(R))$. Define a homomorphism $h_2: ([S_\#]^{n+1})^* \to ([S_\#]^n)^*$ by

$$h_2([b_1, \ldots, b_{n+1}]) = \begin{cases} e & \text{if } b_1 = \ldots = b_n = \# \\ [b_1, \ldots, b_n] & \text{else.} \end{cases}$$

Then $h_2(\theta(x_1, \ldots, x_{n+1})) = \theta(x_1, \ldots, x_n)$ and $\theta(Q_2) = h_2^{-1}(\theta(R)) \cap \theta([S^*]^n)$. (Note that $\theta([S^*]^n)$ is a regular set.) Let $h_3: ([S_\#]^n)^* \to ([S_\#]^{n-1})^*$ be the homomorphism (similar to h_2)

determined by defining

$$h_3([b_1, \ldots, b_n]) = \begin{cases} e & \text{if } b_1 = \ldots = b_{n-1} = \# \\ [b_1, \ldots, b_{n-1}] & \text{else.} \end{cases}$$

Let $L_1 = \{\theta(x_1, \ldots, x_n)\colon\ x_n = t(x_1, \ldots, x_{n-1})\}$. Then $\theta(Q_3) = h_3(\theta(R) \cap L_1)$. Since $t(x_1, \ldots, x_{n-1})$ is formed by concatenating some of the variables and some strings in S^*, it is easy to see that L_1 can be accepted in linear time by a deterministic Turing machine with two-way input and one work tape. (Notice that if $t(x_1, \ldots, x_{n-1})$ is of a simple form, either a constant string or one of the variables, then L_1 is a regular set.) Also because of the form of $t(x_1, \ldots, x_{n-1})$, there are constants $c_0, c_1, \ldots, c_{n-1}$ such that for any $x_1, \ldots, x_{n-1} \in S^*$,

$|t(x_1, \ldots, x_{n-1})| \le c_0 + c_1|x_1| + \ldots + c_{n-1}|x_{n-1}|$. Therefore if $c = n\cdot\max\{c_i\colon\ 0 \le i \le n-1\}$ then for any $z \in L_1$, $|z| \le c|h_3(z)|$; that is, h_3 is a linear-erasing homomorphism on L_1, hence on $L_1 \cap \theta(R)$. □

If explicit transformations are restricted to forms (i) and (ii) above and form (iii) with $t(x_1, \ldots, x_{n-1})$ either a constant string or one of $x_1, \ldots, x_{n-1}$, then part (3) of this proposition can be strengthened as follows: for any explicit transformation Q of R, there exist a regular set L and homomorphisms g_1, g_2 such that $\theta(Q) = g_1(g_2^{-1}(\theta(R)) \cap L)$ and g_1 is e-limited on $g_2^{-1}(\theta(R)) \cap L$.

3.2. A CHARACTERIZATION OF THE RUDIMENTARY RELATIONS

The distinction between the rudimentary relations and the class of languages $\{\theta(R)\colon\ R \in \mathit{RUD}\}$ will now be ignored. We will see that the rudimentary relations are closed under some useful language-theoretic operations and contain the languages accepted by certain types of resource-bounded automata. In particular, any language accepted in linear time by a nondeterministic multitape Turing acceptor is rudimentary. This fact combined with the representation given in Theorem 2.3.1 yields the principal result of this section, that *RUD* is the smallest nonempty class $\mathcal{L}$ satisfying $\mathrm{NL}(\mathcal{L}) \subseteq \mathcal{L}$.

The comparison of *RUD* to a class of languages defined by automata is easier if the latter class can be shown to be generated by some rudimentary language under operations which preserve *RUD*. The Dyck sets, defined below, are the basis for two such algebraic characterizations.

Definition 3.2.1. Let Σ_2 be the alphabet $\{a_1, a_2, \bar{a}_1, \bar{a}_2\}$ and let $\Sigma_1 \subseteq \Sigma_2$ be the alphabet $\{a_1, \bar{a}_1\}$. Let $\sim$ be the binary relation on Σ_2^* defined by $x \sim y$ if and only if $x = ua_i\bar{a}_iv$ for some $u,v \in \Sigma_2^*$, $i = 1$ or 2, and $y = uv$. That is, $x \sim y$ if y results when some pair $a_1\bar{a}_1$ or $a_2\bar{a}_2$ is cancelled from x. Let $\overset{*}{\sim}$ denote the reflexive and transitive closure of $\sim$. Then D_2, the Dyck set on two letters, is defined to be $D_2 = \{x \in \Sigma_2^*\colon\ x \overset{*}{\sim} e\}$ and D_1, the Dyck set on one letter, is defined to be $D_1 = \{x \in \Sigma_1^*\colon\ x \overset{*}{\sim} e\}$.

Consider a_1 and $\overline{a}_1$, and a_2 and $\overline{a}_2$ to be two types of matching parentheses. Then D_1 consists of strings of balanced parentheses of one type, and D_2 consists of strings that are properly nested and balanced parentheses of both types. The Dyck sets are context-free languages; the importance of the Dyck set on two letters lies in the fact that it generates the context-free languages under the operations of intersection with regular sets, inverse homomorphism and homomorphism.

Proposition 3.2.2 (Chomsky-Schutzenberger). If L is a context-free language then there exist a regular set L' and homomorphisms h_1, h_2, with h_2 length-preserving, such that $L = h_2(L' \cap h_1^{-1}(D_2))$. □

The first version of the Chomsky-Schutzenberger Theorem appeared in [14]. In that proof, as well as in subsequent refinements [18], the homomorphism h_2 is not necessarily even e-limited on $L' \cap h_1^{-1}(D_2)$; a proof of the version of the theorem given above can be found in [8].

The following result when combined with Proposition 3.2.2 shows that the class NTIME(n) is generated by the Dyck set on two letters and the regular sets under the operations of intersection, inverse homomorphism and length-preserving homomorphism.

Proposition 3.2.3 ([9]). If $L \in$ NTIME(n) then there exist a length-preserving homomorphism h and context-free languages L_1, L_2, L_3 such that $L = h(L_1 \cap L_2 \cap L_3)$. □

The next proposition brings together some facts about rudimentary relations that will be useful for the proof of Theorem 3.2.7.

Proposition 3.2.4. (1) (Jones [33]) For any relation R, $R \in$ *RUD* if and only if $\theta(R) \in$ *RUD*.

(2) For any relation R, if $\theta(R) \in$ NSPACE (lg(n)) $\in$ then R *RUD* (i.e., NSPACE(lg(n)) $\subseteq$ *RUD*).

(3) Suppose $L \subseteq S^*$, h: $T^* \to S^*$ is a homomorphism and h': $S^* \to T^*$ is a homomorphism that performs linear erasing on L. Then $L \in$ *RUD* implies $h^{-1}(L), h'(L) \in$ *RUD*.

(4) For any relation R, if $\theta(R) \in$ NTIME(n) then $R \in$ *RUD*.

(5) If L_1, L_2 are rudimentary languages then $(L_1 \oplus L_2)^* \in$ *RUD*.

Note that part (1) justifies our equating the family of rudimentary relations and the family of languages $\{\theta(R)\colon R \in RUD\}$.

Proof. (1) For $n \geq 1$, let $T_n = \{(x_1,\ldots,x_n,z)\colon z = \theta(x_1,\ldots,x_n)\}$. Jones [33] has shown that T_n is rudimentary for each n (and any alphabet); the proof relies on the fact that relations such as "$|x| = |y|$" and "a is the $|x|$-th symbol in y" are rudimentary. Recall that $|\theta(x_1, \ldots, x_n)| = \max\{|x_i|\colon 1 \leq i \leq n\}$. Now for any relation R,

$$\theta(R) = \{z\colon \text{there exist } x_1, \ldots, x_n \text{ such that } |x_i| \leq |z| \text{ for } 1 \leq i \leq n,\ (x_1, \ldots, x_n, z) \in T_n \text{ and } (x_1, \ldots, x_n) \in R\};$$

and

$$R = \{(x_1, \ldots, x_n)\colon \text{there exists } z \text{ such that } |z| \leq \max_i |x_i|,\ (x_1, \ldots, x_n, z) \in T_n \text{ and } z \in \theta(R)\}.$$

As remarked previously, the types of quantification used in the two expressions above can be replaced by bounded existential quantification (and explicit transformation and union); therefore R and $\theta(R)$ can be defined from each other and T_n by use of rudimentary operations.

(2) If $R \subseteq [S^*]^m$ and $\# \notin S$, let
$\sigma(R) = \{x_1\#x_2\#\ldots\#x_m: (x_1, \ldots, x_m) \in R\}$.
It is shown in [42] that R is rudimentary if there are constants $k \geq 1$ and e, $0 < e < 1$, such that $\sigma(R)$ is accepted in time n^k and space n^e by a Turing acceptor with a two-way read-only input tape and one work tape (to which the space bound applies). The proof is similar in form to that in [45], where Turing acceptors are arithmetized to show that a class of relations contains a basis for the recursively enumerable sets; in this case the arithmetization must be done to allow use of only bounded quantification, so that the resulting relations are rudimentary. A similar theorem appears in [57].

It is easy to see that $\sigma(R) \in$ NSPACE(lg(n)) if and only if $\theta(R) \in$ NSPACE(lg(n)) and that any language in NSPACE(lg(n)) can be accepted by a device which satisfies the conditions of the theorem cited above; therefore, if $\theta(R) \in$ NSPACE(lg(n)) then R is rudimentary.

(3) Given $L \subseteq S^*$ and $h: T^* \to S^*$ a homomorphism, let $m = \max\{|h(a)|: a \in T\}$. Let R_1 be the binary relation on $(S \cup T)^*$ defined by $R_1 = \{(x,y): x \in T^*, y \in S^*, h(x) = y\}$. Then $\theta(R_1) \in$ DSPACE(lg(n)), so R_1 is rudimentary. Let R_2 be the explicit transformation of L given by $R_2 = \{(x,y): x \in T^*, y \in L\}$. Let R_3 be defined from $R_1 \cap R_2$ by bounded existential quantification:

$(x,y) \in R_3 \iff (\exists z)_y[(x,z) \in R_1 \cap R_2]$. Then $h^{-1}(L) = \{x: (x,x^m) \in R_3\}$ is an explicit transformation of R_3; hence $h^{-1}(L)$ is rudimentary if L is rudimentary.

Suppose $h': S^* \to T^*$ is a homomorphism with the property that $|x| \leq k|h'(x)|$ for any $x \in L$. Let $Q_1 = \{(x,y): h'(y) = x\}$ and $Q_2 = \{(x,y): x \in T^*, y \in L\}$. Then $\theta(Q_1) \in$ DSPACE(lg(n)) and Q_2 is an explicit transformation of L, so both are rudimentary if L is. If Q_3 is defined from $Q_1 \cap Q_2$ by bounded existential quantification, then $h'(L) = \{x: (x,x^k) \in Q_3\}$ and therefore $h'(L)$ is rudimentary.

(4) Since $\theta(R) \in$ *RUD* implies $R \in$ *RUD*, it is sufficient to show that any language $L \in$ NTIME(n) is rudimentary. In [46] a deterministic automaton is described which accepts the Dyck set on two letters and uses lg(n) space on an input of length n; hence $D_2 \in$ DSPACE(lg(n)) and, using part (2), $D_2 \in$ *RUD*. Also, any regular set is rudimentary, since any regular set is in DSPACE(lg(n)). Therefore, from Proposition 3.2.2 and part (3), any context-free language is rudimentary. Using the characterization of NTIME(n) given in Proposition 3.2.3 and the closure properties of *RUD*, any language in NTIME(n) is therefore rudimentary. (In [34,57] the Chomsky-Schutzenberger Theorem is also used to show that any context-free language is rudimentary, with different methods for showing that the Dyck sets and regular sets are rudimentary.)

(5) Suppose $L_1, L_2 \subseteq S^*$ and $\#_1, \#_2 \notin S$, and let $L_1 \oplus L_2 = \#_1L_1 \cup \#_2L_2$. Let $L = (\{\#_1,\#_2\}S^*)^* - (L_1 \oplus L_2)^*$; since

$(\{\#_1,\#_2\}S^*)^*$ is a regular set, $(L_1 \oplus L_2)^*$ is rudimentary if and only if L is rudimentary.

Let $T = S \cup \{\#_1,\#_2\}$. Rather than giving an explicit definition of L from C_T using the rudimentary operations, the previously established facts about *RUD* will be used, specifically the facts that *RUD* is closed under nonerasing homomorphism, inverse homomorphism and intersection with regular sets. Note that

$$L = (\{\#_1,\#_2\}S^*)^*(\{\#_1\}(S^* - L_1) \cup \{\#_2\}(S^* - L_2))(\{\#_1,\#_2\}S^*)^*.$$

Let $U = \{\bar{a}: a \in S\}$ be an alphabet isomorphic to S, with $U \cap T = \emptyset$. For $i = 1,2$ let $R_i \subseteq (U \cup T)^*$ be the regular set

$R_i = (\{\#_1,\#_2\}U^*)^*\{\#_i\}S^*(\{\#_1,\#_2\}U^*)^*$. Let $h_1: (U \cup T)^* \to T^*$ be the length-preserving homomorphism determined by defining $h_1(\#_1) = \#_1$, $h_1(\#_2) = \#_2$, and for $a \in S$, $h_1(a) = h_1(\bar{a}) = a$. Let $h_2: (U \cup T)^* \to S^*$ be the homomorphism determined by defining $h_2(\#_1) = h_2(\#_2) = e$ and for $a \in S$, $h_2(a) = a$ and $h_2(\bar{a}) = e$. Thus, applying h_2^{-1} to a string $x \in S^*$ inserts some symbols from $U \cup \{\#_1,\#_2\}$ into x, and applying h_1 to a string $y \in (U \cup T)^*$ changes symbols $\bar{a} \in U$ occurring in y to the corresponding symbols $a \in S$. Then

$$L = h_1((h_2^{-1}(S^* - L_1) \cap R_1) \cup (h_2^{-1}(S^* - L_2) \cap R_2)),$$

so L is rudimentary. □

Recall that $NL(C) = \{M(L): L \in C,$ M a nondeterministic linear-time oracle machine$\}$. We now define iterations and the closure of the operator NL( ).

Definition 3.2.5. Let C be a class of languages. Define $NL^0(C) = C$, and for $k \geq 0$, $NL^{k+1}(C) = NL(NL^k(C))$. Define $NL^*(C) = \cup \{NL^k(C): k \geq 0\}$. Thus, $NL^*(C)$ is the closure of C under the operator NL( ), the smallest class of languages $\mathcal{L}$ satisfying $C \subseteq \mathcal{L}$ and $NL(\mathcal{L}) \subseteq \mathcal{L}$.

The following closure properties of classes defined by $NL^*(\ )$ are easily established using Theorem 2.2.4.

Proposition 3.2.6. Suppose C is a class of languages which either consists of a single language or is closed under marked union. Then $NL^*(C)$ is closed under the Boolean operations, linear-erasing homomorphism and inverse homomorphism, and contains the class DTIME(lin). □

Note that if C is a nonempty class then $\emptyset \in NL(C)$; hence if C is nonempty and $NL(C) \subseteq C$, then $NL^*(\{\emptyset\}) \subseteq C$. Therefore, $NL^*(\{\emptyset\})$ is (by definition) the smallest nonempty class that is closed under NL( ).

We now have the necessary preliminaries for the proof of the characterization.

Theorem 3.2.7. A relation R is rudimentary if and only if $\theta(R) \in NL^*(\{\emptyset\})$, i.e., $RUD = NL^*(\{\emptyset\})$. Thus the class of rudimentary relations is the smallest nonempty class C satisfying $NL(C) \subseteq C$.

Proof. First note that for any finite alphabet S, $\theta(C_S) \in$ DTIME(lin), where $C_S = \{(x,y,z): xy = z,\ x,y,z \in S^*\}$. Using Proposition 3.2.6, $\theta(C_S) \in NL^*(\{\emptyset\})$. Also from Proposition 3.2.6, $NL^*(\{\emptyset\})$ is closed under inverse homomorphism, linear-erasing homomorphism and the Boolean operations. Combining this with Proposition 3.1.2, we see that the class of relations $\{R: \theta(R) \in NL^*(\{\emptyset\})\}$ is closed under the Boolean operations, explicit transformation and bounded existential quantification, so from the definition of *RUD*, if R is a rudimentary relation then $\theta(R) \in NL^*(\{\emptyset\})$.

Clearly $\emptyset \in$ *RUD*. Since $\theta(R) \in$ *RUD* implies that R is rudimentary, to show that $NL^*(\{\emptyset\}) \subseteq$ *RUD* it suffices to show that NL(*RUD*) $\subseteq$ *RUD*. Let L be a language in *RUD* and let M be a nondeterministic linear-time oracle machine. Using the representation given in Corollary 2.3.2, there exist a length-preserving homomorphism h_1, a homomorphism h_2 and a language $L' \in$ DTIME(lin) such that $M(L) = h_1(L' \cap h_2^{-1}((L \oplus (S^* - L))^*))$, where $L \subseteq S^*$. Now *RUD* is closed under intersection and difference and, from Proposition 3.2.4 (3-5), under the other operations used in this expression; therefore $M(L) \in$ *RUD*. □

The family DSPACE(n) contains DTIME(lin) and is closed under the Boolean operations, inverse homomorphism and nonerasing homomorphism; therefore from Corollary 2.3.2 NL(DSPACE(n)) $\subseteq$ DSPACE(n). Since the empty set is clearly in DSPACE(n), we can conclude the following known inclusion.

<u>Corollary 3.2.8</u> (Myhill [41]). Every rudimentary relation can be accepted in linear space by a deterministic Turing machine:

RUD $\subseteq$ DSPACE(n). □

This corollary can be proved by another method. It is known [45] that the family DSPACE(n) is Grzegorczyk's class $\mathcal{E}_*^2$ [25], again viewing strings as numerals in m-adic notation. From the definitions of the classes it is not hard to show that every rudimentary relation is in $\mathcal{E}_*^0$ [51], hence *RUD* $\subseteq \mathcal{E}_*^0 \subseteq \mathcal{E}_*^2$ = DSPACE(n). (A direct proof that *RUD* $\subseteq \mathcal{E}_*^2$ is given in [45].) It is not known whether $\mathcal{E}_*^0 \subsetneq \mathcal{E}_*^2$, or whether *RUD* $\subsetneq \mathcal{E}_*^2$; note that if *RUD* = DSPACE(n) then $\mathcal{E}_*^0 = \mathcal{E}_*^2$. This problem will be discussed further in the next chapter.

3.3. EXTENSIONS

In the proofs of Proposition 3.2.4 and Theorem 3.2.7 only some of the properties of the rudimentary relations were used in each part. In this section the ideas of those proofs will be applied to classes of languages in general.

We first restate the closure properties of NL($\mathcal{C}$) and NL*($\mathcal{C}$) in the following form.

<u>Proposition 3.3.1</u>. If $\mathcal{C}$ is a class of languages closed under marked union or which consists of a single language then:

(1) the closure of $\mathcal{C}$ under union, intersection, product, Kleene *, inverse homomorphism and linear-erasing homomorphism is contained in NL($\mathcal{C}$); and

(2) the closure of C under the Boolean operations, product, Kleene *, inverse homomorphism and linear-erasing homomorphism is contained in $NL^*(C)$. □

An "abstract family of languages" (AFL) [20] is a class of languages containing at least one nonempty language and closed under union, intersection with regular sets, product, Kleene *, inverse homomorphism and nonerasing homomorphism. Thus Proposition 3.3.1 states that, for a nonempty class C satisfying the condition, $NL(C)$ contains the AFL-closure of C (i.e., the smallest AFL containing C); and $NL^*(C)$ contains the Boolean and AFL closure of C. The containment may be proper: for instance, the class $\mathcal{R}$ of regular sets is closed under the Boolean and AFL operations, but $NL(\mathcal{R}) = NTIME(n)$ and $NL^*(\mathcal{R}) = RUD$, both of which properly contain the regular sets.

We now consider conditions under which equality does hold in Proposition 3.3.1.

Proposition 3.3.2. Suppose C is a class of languages containing D_2, the Dyck set on two letters. Then $NL^*(C)$ is contained in the closure of C under intersection, difference with regular sets, inverse homomorphism and length-preserving homomorphism.

Proof. The proof is similar to the proof of Theorem 3.2.7. Let C be a class of languages that contains D_2 and let C_0 denote the closure of C under intersection, difference with regular sets, inverse homomorphism and length-preserving homomorphism; it must be proven that

$NL^*(C) \subseteq C_0$.

Since $C \subseteq C_0$, for $NL^*(C) \subseteq C_0$ it is sufficient that $NL(C_0) \subseteq C_0$. It is easy to see that C_0 is closed under union and contains every regular set. Since D_2 is in C_0 and C_0 is closed under inverse homomorphism, length-preserving homomorphism and intersection (with regular sets), from Propositions 3.2.2 and 3.2.3, $NTIME(n) \subseteq C_0$. Referring again to Corollary 2.3.2, if M is a nondeterministic linear-time oracle machine and $L \subseteq S^*$ is an oracle set, then $M(L) \in C_0$ if $(L \oplus (S^* - L))^* \in C_0$, since $M(L)$ can be formed from $(L \oplus (S^* - L))^*$ using length-preserving homomorphism, inverse homomorphism and intersection with a language in $DTIME(lin)$. Recall from the proof of Proposition 3.2.4 (5), that for $L_1, L_2 \subseteq S^*$, regular sets R_1, R_2 and homomorphisms h_1, h_2 were constructed (with h_1 length-preserving) such that

$$(\{\#_1,\#_2\}S^*)^* - (L_1 \oplus L_2)^* = h_1((h_2^{-1}(S^* - L_1) \cap R_1) \cup (h_2^{-1}(S^* - L_2) \cap R_2)).$$

Therefore if $L \subseteq S^*$ is in C_0, then $(L \oplus (S^* - L))^* \in C_0$, and so $NL(C_0) \subseteq C_0$. □

In the proof above, to allow the conclusion that $NTIME(n) \subseteq C_0$, it is sufficient (and also necessary) that the Dyck set on two letters be in C_0. Thus the condition on C in Proposition 3.3.2 can be weakened if D_2 can be generated from some other language in C.

Proposition 3.3.3. If C is a class of languages containing the language $\{0^n1^n : n \geq 0\}$ then the Dyck set on two letters is in the closure of C under intersection, difference with regular sets, inverse homomorphism and length-preserving homomorphism. □

The proof of Proposition 3.3.3 can be found in Appendix B. It is first shown that D_2 can be formed from D_1 by application of Boolean operations with regular sets, union, inverse homomorphism and length-preserving homomorphism; and then that D_1 can be formed by applying those operations to $\{0^n1^n: n \geq 0\}$.

Theorem 3.3.4. Suppose $\mathcal{C}$ is a class of languages which contains $\{0^n1^n: n \geq 0\}$ and which is either a singleton class or is closed under marked union. Then $NL^*(\mathcal{C})$ is equal to the closure of $\mathcal{C}$ under intersection, difference with regular sets, inverse homomorphism and length-preserving homomorphism. If further $\mathcal{C}$ is closed under inverse homomorphism, then $NL^*(\mathcal{C})$ is equal to the closure of $\mathcal{C}$ under intersection, difference with regular sets and length-preserving homomorphism.

Proof. The first part follows easily from Propositions 3.3.1-3.3.3. For the second part, let $\mathcal{C}_1$ denote the closure of $\mathcal{C}$ under intersection, difference with regular sets and length-preserving homomorphism; it must be shown that $\mathcal{C}_1$ is closed under inverse homomorphism.

Let $\mathcal{D}_0 = \mathcal{C}$ and for $k \geq 0$, let $\mathcal{D}_{k+1} = \{L_1 \cap L_2,\ R - L_1,\ h(L_1): L_1, L_2 \in \mathcal{D}_k,\ R$ a regular set, h a length-preserving homomorphism$\}$. Then $\mathcal{C}_1 = \bigcup_k \mathcal{D}_k$. Since by assumption $\mathcal{C}$ is closed under inverse homomorphism, if $L \in \mathcal{D}_0$ and h is a homomorphism then $h^{-1}(L) \in \mathcal{C}_1$. For any languages L_1, L_2 and any homomorphism h, $h^{-1}(L_1 \cap L_2) = h^{-1}(L_1) \cap h^{-1}(L_2)$ and $h^{-1}(L_1 - L_2) = h^{-1}(L_1) - h^{-1}(L_2)$.

Furthermore, inverse homomorphism and length-preserving homomorphism "commute," in the following way:

Claim. If $h_1: S^* \to T^*$ is a nonerasing homomorphism, $h_2: U^* \to T^*$ is a homomorphism and $L \subseteq S^*$, then there exist a regular set L', a length-preserving homomorphism h_3 and a homomorphism h_4 such that $h_2^{-1}(h_1(L)) = h_3(L' \cap h_4^{-1}(L))$.

This construction is given in [21, pp. 43-44].

Thus an induction argument can be given to show that for all $k \geq 0$, if $L \in \mathcal{D}_k$ and h is a homomorphism then $h^{-1}(L) \in \mathcal{C}_1$, and therefore $\mathcal{C}_1$ is closed under inverse homomorphism. □

Theorem 3.3.4 yields other, algebraic, characterizations of the rudimentary relations.

Corollary 3.3.5. (1) (Yu [57]) *RUD* is equal to the closure of the context-free languages under the Boolean operations and length-preserving homomorphism.
(2) *RUD* is the smallest class of languages containing $\{0^n1^n: n \geq 0\}$ and closed under the Boolean operations, inverse homomorphism and length-preserving homomorphism. □

Note that part (1) of the corollary above holds if the context-free languages are replaced by any class of languages contained in *RUD*, closed under inverse homomorphism and containing $\{0^n1^n: n \geq 0\}$, e.g., the deterministic context-free languages, the linear context-free languages, the one-counter languages, the class DSPACE(lg(n)). Recall that the closure of the regular sets under the Boolean operations and length-preserving homomorphism is just the regular sets, which is properly contained in *RUD*. Since $\{0^n 1^n: n \geq 0\}$ is a simple nonregular set, this raises the question of whether there is a class of languages for which the closure in part (1) properly contains the regular sets but is itself properly contained in *RUD*. Alternately, is there a nonregular language L such that the smallest Boolean-closed AFL containing L is properly contained in *RUD*?

Part (2) of this corollary has the following interpretation. It can be shown that for any alphabet S the language $\{\theta(x,y,z): x,y,z \in S^*, \ xy \neq z\}$ can be accepted by a nondeterministic one-counter automaton with the property that during any computation the counter makes at most one turn (i.e., one change from increasing to decreasing). Furthermore, the class of nondeterministic one-turn one-counter languages is generated by the language $\{0^n 1^n: n \geq 0\}$ under the operations of intersection with regular sets, inverse homomorphism and homomorphism [22].

Nonerasing or linear-erasing homomorphism can be used in Corollary 3.3.5, with the same results. On the other hand, the closure of the (linear) context-free languages under intersection and arbitrary homomorphism is the family of recursively enumerable sets [26, 2]; and any r.e. set can be generated from $\{a^n b^n: n \geq 0\}$ by application of the AFL operations, intersection and arbitrary homomorphism [27].

Recall that NL*(C) is closed under linear-erasing homomorphism for

any class of languages C. There exist classes of languages closed under nonerasing homomorphism (and the other AFL operations) but not under linear erasing [23]; however, from Theorem 3.3.4 it is clear that any sufficiently large Boolean-closed AFL is closed under linear-erasing homomorphism.

<u>Corollary 3.3.6</u>. If C is a class of languages containing $\{0^n 1^n: n \geq 0\}$ and closed under difference with regular sets, union, intersection, inverse homomorphism and length-preserving homomorphism, then C is an AFL closed under linear-erasing homomorphism. □

In [12] conditions on C are given for which the closure of C under intersection and nonerasing homomorphism is itself closed under linear erasing homomorphism; the proof, like that for Corollary 3.3.6, uses constructions involving automata. Once these results have been stated it is possible to give proofs which make no reference to automata, but which, however, still rely on such algebraic characterizations as those given in Propositions 3.2.2 and 3.2.3.

It is possible to strengthen Cor. 3.3.6 as follows: if C contains the (deterministic) linear context-free languages and is closed under intersection, inverse homomorphism, length-preserving homomorphism and union with $\{e\}$, then C is also closed under linear-erasing homomorphism.

Now we consider the relationship of the operator NL( ) to language-theoretic closure properties.

<u>Theorem 3.3.7</u>. Suppose C is a class of languages closed under marked product, marked +, inverse homomorphism, union with languages in DTIME(lin) and intersection with and product with regular sets. Then

$$NL(C) = \{h(L_1 \cap (\Sigma^* - L_2)): L_1, L_2 \subseteq \Sigma^* \text{ in } C,\ h \text{ a nonerasing homomorphism}\}.$$

<u>Proof</u>. For the containment from right to left, first recall from Theorem 2.2.4 that $NL(C)$ is closed under nonerasing homomorphism and union with regular sets and contains L and $\Sigma^* - L$ whenever $L \subseteq \Sigma^*$ is in C. If M_1 and M_2 are nondeterministic linear-time oracle machines (with the same input and tape alphabets) then for any oracle sets L_1, L_2, it is easy to construct a nondeterministic linear-time oracle machine M_0 such that $M_1(L_1) \cap M_2(L_2) = M_0(L_0)$ where $L_0 = (L_1 \cup \{e\})¢(L_2 \cup \{e\})$. Thus if C is closed under marked product and union with regular sets, $NL(C)$ is closed under intersection, and the containment follows.

Suppose $L \in NL(C)$, so that $L = M(L_1)$ with $L_1 \in C$, M a nondeterministic linear-time oracle machine. From Theorem 2.3.1 (1), there is a length-preserving homomorphism h and a deterministic linear-time oracle machine D such that $M(L_1) = h(D(L_1))$; since the composition of nonerasing homomorphisms is a nonerasing homomorphism, we may assume that M is deterministic.

Now suppose M operates in time $cn + d$ and $L \subseteq T^*$, $L_1 \subseteq S^*$. Let $\#$, $\#_1$, $\#_2$ be three new symbols and

$$U = T \times (\{\#\} \cup (S \cup \{\#_1,\#_2\})_k) = \{[a,\#]: a \in T\} \cup$$
$$\{[a,w]: a \in T,\ w \in (S \cup \{\#_1,\#_2\})^*,\ 1 \leq |w| \leq k\},$$

where $k = c + d$. Let $h_1: U^* \to T^*$ be the homomorphism determined by defining $h_1([a,x]) = a$, and $h_2: U^* \to (S \cup \{\#_1,\#_2\})^*$, by defining $h_2([a,\#]) = e$ and $h_2([a,w]) = w$. The proof of Theorem 2.3.1 (2) can be modified slightly to yield the following: there exists a language $L_2 \subseteq U^*$ in DTIME(lin) such that $L = M(L_1) = h_1(L_2 \cap h_2^{-1}(L_3))$ where $L_3 = (L_1\#_1 \cup (S^* - L_1)\#_2)^*$.

Let $\bar{S} = \{\bar{a}: a \in S\}$ be an alphabet disjoint from S. Let $h_3: (S \cup \bar{S} \cup \{\#_1,\#_2\})^* \to (S \cup \{\#_1,\#_2\})^*$ be the homomorphism determined by defining $h_3(a) = h_3(\bar{a}) = a$, $a \in S$, $h_3(\#_1) = \#_1$ and $h_3(\#_2) = \#_2$. Let U_1 be the alphabet $T \times (\{\#\} \cup (S \cup \bar{S} \cup \{\#_1,\#_2\})_k)$. Define languages $L_1' \subseteq \bar{S}^*$, $L_4, L_5 \subseteq U_1^*$ by:

$L_1' = \{x \in \bar{S}^*: h_3(x) \in S^* - L_1\}$;

$L_4 = \{\theta(x,v/k): x \in T^*,\ |x| \geq |v/k|,\ v \in (L_1\#_1 \cup \bar{S}^*\#_2)^*\}$; and

$L_5 = \{\theta(x,v/k): x \in T^*,\ |x| \geq |v/k|,\ v \in (S^*\#_1 \cup L_1'\#_2)^*,$
$\qquad \theta(x,h_3(v)/k) \in L_2\}$.

Now if $h_4: U_1^* \to T^*$ is the length-preserving homomorphism determined by defining $h_4([a,x]) = a$, then $L = h_1(L_2 \cap h_2^{-1}(L_3)) = h_4(L_4 \cap L_5)$. The containment from left to right will therefore follow if it is shown that L_4 and $U_1^* - L_5$ are in C.

Let $h_5: (S \cup \bar{S} \cup \{\#_1,\#_2\})^* \to (S \cup \{\#_1\})^*$ be the homomorphism that erases symbols in $\bar{S} \cup \{\#_2\}$: $h_5(a) = a$, $h_5(\bar{a}) = e$ for $a \in S$, $h_5(\#_1) = \#_1$ and $h_5(\#_2) = e$. Let $h_6: U_1^* \to (S \cup \{\#_1\})^*$ be the homomorphism determined by defining $h_6([a,\#]) = e$, $h_6([a,w]) = h_5(w)$, so that $h_6(\theta(x,v/k)) = h_5(v)$. Let $L_6 \subseteq U_1^*$ be the regular set

$L_6 = \{\theta(x,v/k): x \in T^*,\ v \in (S^*\#_1 \cup \bar{S}^*\#_2)^*,\ |x| \ge |v/k|\}$. Then

$L_4 = h_6^{-1}((L_1\#_1)^+ \cup \{e\}) \cap L_6$. Hence since $\mathcal{C}$ is closed under marked +, inverse homomorphism and intersection and union with regular sets, $L_4 \in \mathcal{C}$.

Let $h_7: (S \cup \bar{S} \cup \{\#_1,\#_2\})^* \to (S \cup \{\#_2\})^*$ be the homomorphism (similar to h_5) determined by defining $h_7(\bar{a}) = a$, $h_7(a) = e$, $h_7(\#_1) = e$, $h_7(\#_2) = \#_2$; and let $h_8: U_1^* \to (S \cup \{\#_2\})^*$ be determined by defining $h_8([a,\#]) = e$, $h_8([a,w]) = h_7(w)$. Note that

$h_7^{-1}((S^*\#_2)^*L_1\#_2(S^*\#_2)^*) \cap (S^*\#_1 \cup \bar{S}^*\#_2)^* = (S^*\#_1 \cup \bar{S}^*\#_2)^* - (S^*\#_1 \cup L_1'\#_2)^*$.

If $L_7 \subseteq U_1^*$ is defined to be

$L_7 = h_8^{-1}((S^*\#_2)^*L_1\#_2(S^*\#_2)^*) \cap L_6$ then

$L_7 = \{\theta(x,v/k): x \in T^*,\ |x| \ge |v/k|,\ v \in (S^*\#_1 \cup \bar{S}^*\#_2)^* - (S^*\#_1 \cup L_1'\#_2)^*\}$.

$L_7 \in \mathcal{C}$ since $\mathcal{C}$ is closed under product by regular sets, inverse homomorphism, and intersection with regular sets. Let

$L_8 = \{\theta(x,v/k): \theta(x,v/k) \in L_6$ and $\theta(x,h_3(v)/k) \in U^* - L_2\} \cup (U_1^* - L_6)$.

Since L_2 is in DTIME(lin) and L_6 is a regular set, also L_8 is in

DTIME(lin). Since $\mathcal{C}$ is closed under union with languages in DTIME(lin), $L_7 \cup L_8 \in \mathcal{C}$; but $L_7 \cup L_8 = U_1^* - L_5$, so $U_1^* - L_5 \in \mathcal{C}$. □

<u>Notation</u> [19]. If $\mathcal{L}_1$, $\mathcal{L}_2$ are classes of languages, let

$\mathcal{L}_1 \wedge \mathcal{L}_2 = \{L_1 \cap L_2: L_i \in \mathcal{L}_i,\ i = 1,2\}$; and

$H[\mathcal{L}_1] = \{h(L): L \in \mathcal{L}_1,\ h$ a nonerasing homomorphism$\}$.

In this notation, Theorem 3.3.7 becomes: for a class $\mathcal{C}$ satisfying the conditions, $NL(\mathcal{C}) = H[\mathcal{C} \wedge \text{co-}\mathcal{C}]$. In particular, we have the following corollary.

<u>Corollary 3.3.8</u>. If $\mathcal{C}$ is an AFL containing DTIME(lin) then $H[\mathcal{C} \wedge \text{co-}\mathcal{C}]$ is an AFL closed under intersection and linear erasing. □

Part of Corollary 3.3.8 also follows from results in [21], namely that if $\mathcal{C}$ is an AFL then so is $H[\mathcal{C} \wedge \text{co-}\mathcal{C}]$. However, closure under linear erasing and under intersection do not seem to follow, unless $\mathcal{C}$ is closed under intersection.

In the preceding comparisons of $NL(\mathcal{C})$ and $NL^*(\mathcal{C})$ with closures of $\mathcal{C}$, only sufficient conditions were given, because the necessary conditions that can be derived are not informative (e.g., if the conclusion of Theorem 3.3.7 holds then DTIME(lin) $\subseteq H[\mathcal{C} \wedge \text{co-}\mathcal{C}]$). There is a class of languages that fails to satisfy the conditions of Theorem 3.3.7 and for which the containment $NL(\mathcal{C}) \subseteq H[\mathcal{C} \wedge \text{co-}\mathcal{C}]$ is open: the family *DCF* of deterministic context-free languages. It is known that co-*DCF* = *DCF* (see [18]) and *DCF* is closed under the operations used in the proof of Theorem 3.3.7 except union with DTIME(lin). (Recall that only a simple form of product with regular sets was used.) Since $DCF \subseteq$ DTIME(lin), NL(DCF) = NTIME(n); the results in [9] can be used to show that NTIME(n) = H[$DCF \wedge DCF \wedge DCF$]. However, it is unknown whether H[$DCF \wedge DCF$] = H[$DCF \wedge DCF \wedge DCF$].

Theorem 3.3.7 will be used in the next chapter to show that for certain classes C, NL(C) = H[co-C].

Chapter 4: THE LINEAR HIERARCHY

In this chapter, the structure of the class of rudimentary relations is examined more closely. Using the characterization given in Chapter 3, the class of rudimentary relations is decomposed into the "linear hierarchy," a structure of classes of languages analogous to the arithmetic hierarchy defined using linear-time oracle machines. The k+1-st class σ_{k+1} in the linear hierarchy is defined from the k-th class σ_k by σ_{k+1} = NL(σ_k), so that a language is in σ_{k+1} if and only if it can be accepted nondeterministically in linear time given an oracle for some language in σ_k. It is not known whether the linear hierarchy is in fact an infinite hierarchy of classes; a positive answer to this question would close several open questions in automata theory and logic.

Section 1 gives the definition of the linear hierarchy and establishes some of its basic properties, which follow easily from the more general results proved in Chapters 2 and 3. In particular, a simpler, alternate definition of the linear hierarchy is given (Theorem 4.1.4): for $k \geq 1$, a language belongs to the k+1-st class if and only if it is the image under a nonerasing homomorphism of the complement of a language in the k-th class. In Section 2 this alternate definition is used to prove a result that strengthens the analogy between the linear hierarchy and the arithmetic hierarchy: the k-th class in the linear hierarchy consists of exactly those languages that can be obtained from languages in a basis class by application of k alternations of bounded quantification (Theorem 4.2.2). Thus the class in the linear hierarchy to which a rudimentary relation belongs is closely related to the syntactic form of its definition from concatenation.

We turn in Section 3 to investigation of the internal structure of the classes in the linear hierarchy, employing the notions of "efficient reducibility" and "complete sets." Each class is shown to possess a complete set with respect to reductions of a simple form (Theorem 4.3.5); this property of the classes allows some conclusions to be drawn about the relationship of the linear hierarchy and of the rudimentary relations to other families of languages.

4.1. DEFINITION

The linear hierarchy consists of the classes of languages σ_k, π_k and δ_k $(k \geq 0)$, defined immediately below. The names of the classes were selected to suggest the analogy to the arithmetic hierarchy.

<u>Definition 4.1.1</u>. Define $\sigma_0 = \pi_0 = \delta_0 = \{\emptyset\}$. For $k \geq 0$, define

$$\sigma_{k+1} = \mathrm{NL}(\sigma_k);$$

$$\pi_{k+1} = \text{co-}\sigma_{k+1}; \text{ and}$$

$$\delta_{k+1} = \{M(L)\colon L \in \sigma_k,\ M \text{ a deterministic linear-time oracle machine}\}.$$

Note that $\sigma_k = \mathrm{NL}^k(\{\emptyset\})$ for all $k \geq 0$; hence $RUD = \bigcup\{\sigma_k\colon k \geq 0\}$. Primary attention will be given to the classes σ_k. For $k \geq 1$, the class π_k consists of languages whose complements are in σ_k, and δ_{k+1} consists of languages that can be recognized deterministically in linear time relative to some language in σ_k. Except for the trivial case $k = 0$, it is not known whether $\sigma_k \subsetneq \sigma_{k+1}$. A proof of either proper containment or equality must rely on properties specific to the classes rather than only on general properties of the operator NL( ), since, by employing techniques used in [3] in the context of polynomial time, classes of recursive languages $\mathcal{C}_1$ and $\mathcal{C}_2$ can be found such that $\mathcal{C}_1 = \mathrm{NL}(\mathcal{C}_1)$ but $\mathcal{C}_2 \subsetneq \mathrm{NL}(\mathcal{C}_2)$.

The following proposition applies some results from Chapter 2 and related constructions to the linear hierarchy. As well as providing information about the classes, these containment and positive closure properties will be useful in the investigation to follow.

<u>Proposition 4.1.2</u>.

(1) For each $k \geq 0$, $\sigma_k \cup \pi_k \subseteq \delta_{k+1} \subseteq \sigma_{k+1} \cap \pi_{k+1}$.

(2) $\sigma_1 = \mathrm{NTIME}(n)$; $\delta_1 = \mathrm{DTIME}(\mathrm{lin})$.

(3) For each $k \geq 1$, σ_k is closed under union, intersection, product, Kleene *, inverse homomorphism and linear erasing homomorphism.

(4) For each $k \geq 1$, π_k and δ_k are closed under union, intersection, marked product, marked +, and inverse homomorphism; δ_k is closed under complementation.

Note that closure under complementation for σ_k and closure under nonerasing homomorphism for δ_k are not asserted; in Proposition 4.1.3 it will be seen that proof of either of these closure properties is equivalent to proving the finiteness of the linear hierarchy.

Proof.

(1) For any alphabet S, it is easy to construct deterministic oracle machines D_1 and D_2, both operating in time $n+1$, such that for any $L \subseteq S^*$, $D_1(L) = D_2(S^*-L) = L$. Hence, if $L \in \sigma_k$ then $L = D_1(L) \in \delta_{k+1}$, and if $L \in \pi_k$ then $S^*-L \in \sigma_k$ and so $L = D_2(S^*-L) \in \delta_{k+1}$. The second containment follows from the facts that $\delta_{k+1} \subseteq \sigma_{k+1}$ (by definition) and that δ_{k+1} is closed under complementation. It is possible that $\delta_{k+1} \subsetneq \sigma_{k+1} \cap \pi_{k+1}$, which is not the case for the corresponding classes of the arithmetic hierarchy (e.g., a set is recursive if and only if both it and its complement are r.e.).

(2) Since $\sigma_0 = \{\emptyset\}$, these equalities are consequences of Proposition 2.2.2.

(3) The class σ_0 consists of a single language; since $\sigma_k = NL(\sigma_{k-1})$, by using induction on k and Theorem 2.2.4 we see that for each $k \geq 1$, σ_k is closed under marked union as well as the other operations listed.

(4) The proof that δ_k is closed under these operations for each $k \geq 1$ uses that fact that σ_{k-1} is closed under marked union and simple machine constructions (omitted here), similar to those given in Theorem 2.2.4. Closure of the classes π_k under the operations listed can be seen from part (3) and the following identities: if $L_1, L_2 \subseteq S^*$,

$\not{c} \notin S$, $T = S \cup \{\not{c}\}$ and $h: U^* \to S^*$ is a homomorphism, then

$(S^*-L_1) \cup (S^*-L_2) = S^* - (L_1 \cap L_2)$;

$(S^*-L_1) \cap (S^*-L_2) = S^* - (L_1 \cup L_2)$;

$(S^*-L_1)\not{c}(S^*-L_2) = T^* - [(T^* - (S^*\not{c}S^*)) \cup L_1\not{c}S^* \cup S^*\not{c}L_2]$;

$((S^*-L_1)\not{c})^+ = T^* - [(T^* - (S^*\not{c})^+) \cup (S^*\not{c})^* L\not{c}(S^*\not{c})^*]$; and

$h^{-1}(S^* - L_1) = U^* - h^{-1}(L_1)$. □

It is not known whether any of the classes δ_k and π_k is closed under nonerasing homomorphism; that one of them should be so closed is a necessary and sufficient condition for the linear hierarchy to "collapse" at that point. Similarly, a class σ_k is closed under complementation if and only if the linear hierarchy is finite (and $RUD = \sigma_k$).

Proposition 4.1.3.

(1) For all $k \geq 1$, δ_k is closed under nonerasing homomorphism if and only if $\sigma_k = \delta_k$ if and only if for all $j \geq 1$, $\sigma_j \subseteq \delta_k$.

(2) For all $k \geq 1$, π_k is closed under nonerasing homomorphism if and only if σ_k is closed under complementation if and only if for all $j \geq 1$, $\sigma_j \subseteq \sigma_k$.

Proof. First note that if for some $k \geq 1$, $\sigma_{k+1} = NL(\sigma_k)$ is contained in σ_k, then for all $j \geq 0$, $NL^j(\sigma_k) \subseteq \sigma_k$ and hence for all $j \geq 1$, $\sigma_j \subseteq \sigma_k$. Also, if σ_k is closed under complementation then, using the other closure properties given in Proposition 4.1.2(3), if $L \in \sigma_k$, $L \subseteq S^*$, then for any length-preserving homomorphism h_1, any homomormorphism h_2 and any language $L' \in \sigma_k$,

$$h_1(L' \cap h_2^{-1}((L \oplus (S^*-L))^*)) \in \sigma_k.$$

Therefore, from Corollary 2.3.2 if M is any nondeterministic linear-time oracle machine and $L \in \sigma_k$, then $M(L) \in \sigma_k$; that is, $\sigma_{k+1} = NL(\sigma_k) \subseteq \sigma_k$.

To see the first part of the proposition, recall from Theorem 2.3.1(1) that if M is a nondeterministic linear-time oracle machine, then there exist a deterministic linear-time oracle machine D and a length-preserving homomorphism h such that for any language L, $M(L) = h(D(L))$. In the context of the linear hierarchy, this implies that for all $k \geq 1$, $\sigma_k \subseteq \{h(L)\colon L \in \delta_k,\ h$ a nonerasing homomorphism$\}$. In fact, since $\delta_k \subseteq \sigma_k$ and σ_k is closed under nonerasing homomorphism, we see that for all $k \geq 1$, σ_k is equal to the closure of δ_k under nonerasing homomorphism. Thus if for some k, δ_k is closed under nonerasing homomorphism then $\delta_k = \sigma_k$. Now since δ_k is closed under complementation, if $\delta_k = \sigma_k$ then σ_k is also closed under complementation, so by the remark above, for all $j \geq 1$, $\sigma_j \subseteq \sigma_k = \delta_k$. On the other hand, if for all $j \geq 1$, $\sigma_j \subseteq \delta_k$ then in particular $\sigma_k \subseteq \delta_k$, so that $\sigma_k = \delta_k$ and since σ_k is closed under nonerasing homomorphism, also δ_k is closed under nonerasing homomorphism.

For the second part, suppose for some $k \geq 1$, π_k is closed under nonerasing homomorphism. Then, since $\delta_k \subseteq \pi_k$, the closure of δ_k under nonerasing homomorphism is contained in π_k; hence, from the proof of part (1), $\sigma_k \subseteq \pi_k$. Since $\pi_k = \text{co-}\sigma_k$, if $\sigma_k \subseteq \pi_k$, then

$\sigma_k = \pi_k$ and so σ_k is closed under complementation. Again this implies that for all $j \geq 1$, $\sigma_j \subseteq \sigma_k$. For the reverse implication, if for all $j \geq 1$, $\sigma_j \subseteq \sigma_k$, then in particular $\sigma_{k+1} \subseteq \sigma_k$. Since $\pi_k \subseteq \sigma_{k+1}$, this implies that $\pi_k \subseteq \sigma_k$; hence $\pi_k = \sigma_k$ and π_k is closed under nonerasing homomorphism. □

In the proof of the proposition above, it was shown (using Theorem 2.3.1) that for all $k \geq 1$, $\sigma_k = \{h(L)\colon L \in \delta_k,\ h$ a nonerasing homomorphism$\} = H[\delta_k]$. Use of Theorem 3.3.7 allows the following result, which is stronger in case $\pi_{k-1} \not\subseteq \delta_k$.

<u>Theorem 4.1.4</u>. For all $k \geq 2$, $\sigma_k = H[\pi_{k-1}]$.

<u>Proof</u>. Since each class σ_k is closed under nonerasing homomorphism and since $\pi_{k-1} \subseteq \sigma_k$, the containment from right to left is clear.

Consider σ_{k+1} for $k \geq 1$. From the closure properties of σ_k given in Proposition 4.1.2(3) and the fact that $\text{DTIME(lin)} = \delta_1 \subseteq \sigma_k$, we see that σ_k satisfies the conditions of Theorem 3.3.7: i.e., σ_k is closed under marked product, marked +, inverse homomorphism, union with DTIME(lin), and intersection with and product with regular sets. Therefore $\sigma_{k+1} = NL(\sigma_k) = H[\sigma_k \wedge \text{co-}\sigma_k] = H[\sigma_k \wedge \pi_k]$. Clearly $\pi_k \subseteq H[\pi_k]$. Also, $\sigma_k = H[\delta_k] \subseteq H[\pi_k]$ so both π_k and σ_k are contained in $H[\pi_k]$. From [21, p. 45] we see that the closure properties of π_k given in Proposition 4.1.2(4) ensure that $H[\pi_k]$ is closed under intersection and nonerasing homomorphism (i.e., π_k is a "pre-AFL" closed under intersection). Therefore $H[\sigma_k \wedge \pi_k] \subseteq H[\pi_k]$ and so $\sigma_{k+1} = H[\pi_k]$. □

Theorem 4.1.4 gives an alternate definition for the "σ classes" of the linear hierarchy: $\sigma_1 = H[\mathrm{DTIME}(\mathrm{lin})]$ and for $k \geq 1$, $\sigma_{k+1} = H[\text{co-}\sigma_k]$. The original definition allowed simple proofs of closure of the classes under language-theoretic operations, for which well-known constructions involving automata could be applied. On the other hand, because of the close relationship between the operations of nonerasing homomorphism and bounded existential quantification (partially expressed in Proposition 3.1.2), this definition makes virtually immediate the "syntactic" characterization of the classes σ_k to be given in the next section. As a further preliminary for that characterization, we consider another operation on string relations, bounded universal quantification.

Definition 4.1.5. The operation of bounded universal quantification is defined as follows: Suppose $n \geq 0$ and $R \subseteq [S^*]^{n+1}$. Then $Q \subseteq [S^*]^{n+1}$ is defined by bounded universal quantification from R (written $Q = \forall \leq R$) if and only if $Q = \{(x_1,\ldots,x_n,z)$: for every $y \in S^*$ such that $|y| \leq |z|$, $(x_1,\ldots,x_n,y) \in R\}$.

Note that if $Q = \forall \leq R$, then $[S^*]^{n+1} - Q = \exists \leq ([S^*]^{n+1} - R)$, so the rudimenatry relations are closed under bounded universal quantification.

If two relations Q and R satisfy $Q = \forall \leq [\exists \leq R]$ then Q is just an explicit transformation of R; therefore rather than classifying rudimentary relations by the number of applications of bounded quantification used in their definition, we consider quantifications bounded by (the length of) one of the variables. Recall from Chapter 3

that any rudimenatry relation can be defined using the Boolean operations and explicit transformations, and quantification of the form

$\varepsilon_i(R) = \{(x_1,\ldots,x_n)$: there is some $y \in S^*$ such that $|y| \leq |x_i|$ and $(x_1,\ldots,x_n,y) \in R\}$

where $1 \leq i \leq n$, $R \subseteq [S^*]^{n+1}$. The analogous form of universal quantification is given by:

$\alpha_i(R) = \{(x_1,\ldots,x_n)$: for every $y \in S^*$, if $|y| \leq |x_i|$ then $(x_1,\ldots,x_n,y) \in R\}$.

To increase readability, the definition of relations $Q_1 = \varepsilon_i(R)$ and $Q_2 = \alpha_i(R)$ by use of these operations is written:

$(x_1,\ldots,x_n) \in Q_1 \iff (\exists y)_{x_i}[(x_1,\ldots,x_n,y) \in R]$; and

$(x_1,\ldots,x_n) \in Q_2 \iff (\forall y)_{x_i}[(x_1,\ldots,x_n,y) \in R]$.

4.2. A SYNTACTIC CHARACTERIZATION OF THE LINEAR HIERARCHY

In this section we consider a certain classification of the rudimentary relations, by the number of alternations of applications of bounded existential and universal quantification used in their definition. If R_0 denotes the class of relations definable from concatenation relations by a finite number of applications of the Boolean operations and explicit transformation, then any rudimentary relation can be obtained by applying some number of quantifications and explicit transformations to a relation in R_0. It is easy to see that if a relation R is in R_0 then $\Theta(R) \in \mathrm{DTIME}(\mathrm{lin})$; we will see that, for example, if a relation R is defined from a relation in R_0 by applying bounded universal quantification, explicit transformation and bounded existential quantification (in that order) then $\theta(R) \in \sigma_2$.

The following definition, of classes of languages E_k, $k \geq 1$, is made only for notational convenience; Theorem 4.2.2 states that for all $k \geq 1$, E_k and σ_k contain exactly the same languages.

Definition 4.2.1. For each $k \geq 1$, define the class E_k as:

(1) If k is odd, then a language L is in E_k if and only if for some language $L_0 \in$ DTIME(lin), for all x,

$$x \in L \iff (\exists y_1)_x(\forall y_2)_x \cdots (\exists y_k)_x[\theta(x,y_1,\ldots,y_k) \in L_0].$$

(2) If k is even, then L is in E_k if and only if there exists $L_0 \in$ DTIME(lin) such that for all x,

$$x \in L \iff (\exists y_1)_x(\forall y_2)_x \cdots (\forall y_k)_x[\theta(x,y_1,\ldots,y_k) \in L_0].$$

The quantifiers in these expressions alternate between existential and universal, so that the j-th quantification (from the left) is $(\exists y_j)_x$ if j is odd, and $(\forall y_j)_x$ if j is even.

Thus the class E_k consists of languages that can be defined from languages in DTIME(lin) by use of k alternations of quantification bounded by x and with an existential quantification applied last (i.e., $\exists$ is the leftmost quantifier in the prefix). If a langauge $L \in E_k$ is defined from $L_0 = \theta(R_0) \in$ DTIME(lin), then L is an explicit transformation of a relation defined by successive application of bounded universal or existential quantification and explicit transformation to R_0.

The definition of E_k could have been made to allow multiple

occurrences of quantifiers between alternations; the same classes of languages would have resulted. Suppose, for example. that $L_0 \in$ DTIME(lin) and a language L satisfies: $x \in L$ if and only if

$$(\exists y)(\exists z)[|y|,|z| \leq |x| \text{ and } \theta(x,y,z) \in L_0].$$

Let L_1 be the language $L_1 = \{\theta(x,w): w = \theta(y,z)$ for some y and z, and $\theta(x,y,z) \in L_0\}$; then also $L_1 \in$ DTIME(lin). Since L satisfies $x \in L$ if and only if $(\exists w)_x[\theta(x,w) \in L_1]$, $L \in E_1$.

It is not hard to see that a relation R is rudimentary if and only if for some $k \geq 1$, $\theta(R) \in E_k$; in fact the classification of *RUD* given by $\bigcup_k E_k$ is the same as that given by the linear hierarchy.

Theorem 4.2.2. For all $k \geq 1$, $E_k = \sigma_k$.

The proof is by induction on k. In the induction step for the containment from left to right, we use Theorem 4.1.4 $(\sigma_k = H[\pi_{k-1}])$ and the fact that application of bounded existential quantification followed by complementation is equivalent to application of complement followed by bounded universal quantification. For the reverse containment we essentially show that $E_{k+1} = H[\text{co-}E_k]$.

Proof. For the basis of the induction, recall that $\sigma_1 = \text{NTIME}(n) = H[\text{DTIME(lin)}]$. Therefore if $L \in \sigma_1$, $L \subseteq S^*$, then there is a language $L_1 \in$ DTIME(lin), $L_1 \subseteq T^*$, and a nonerasing homomorphism $h: T^* \to S^*$ such that $L = h(L_1)$. Let $L_0 = \{\theta(x,y):$ $h(y) = x$ and $y \in L_1\}$; then also $L_0 \in$ DTIME(lin). Since h is nonerasing, if $h(y) = x$ then $|y| \leq |x|$, so for all x, $x \in L$ if and only if

$$(\exists y)[|y| \le |x| \text{ and } \theta(x,y) \in L_0];$$

therefore $L \in E_1$. The proof that $E_1 \subseteq \sigma_1$ is essentially the same as the proof to be given for this containment in the induction step, and is omitted.

Suppose $E_k = \sigma_k$ for some $k \ge 1$, and further suppose that k is even. (The proof for the case k odd is a simple notational variant.) If $L \subseteq S^*$ is a language in E_{k+1}, let $L_0 \in \mathrm{DTIME(lin)}$ be the language such that

$$x \in L \iff (\exists y_1)_x(\forall y_2)_x \cdots (\exists y_{k+1})_x[\theta(x,y_1,\ldots,y_{k+1}) \in L_0].$$

Let L_0' be the language $L_0' = \{\theta(z,y_2,\ldots,y_{k+1})\colon z = \theta(x,y_1)$ for some x and y_1 such that $\theta(x,y_1,\ldots,y_{k+1}) \in L_0\}$; it is easy to see that $L_0' \in \mathrm{DTIME(lin)}$ if $L_0 \in \mathrm{DTIME(lin)}$. Let L' be the language defined by

$$z \in L' \iff (\exists y_1)_z(\forall y_2)_z \cdots (\forall y_k)_z[\theta(z,y_1,\ldots,y_k) \in L_0'].$$

Since $L_0' \in \mathrm{DTIME(lin)}$, $L' \in E_k$. Recall that if $|y_1| \le |x|$, then $|\theta(x,y_1)| = |x|$; hence for any x, $x \in L$ if and only if

$$(\exists y_1)[|y_1| \le |x| \text{ and } \theta(x,y_1) \in L'].$$

Now suppose T and $\# \notin T$ are such that $L' \subseteq ([T_\#]^2)^*$. Let $L'' \subseteq ([T_\#]^2)^*$ be the regular set $L'' = \{\theta(x,y)\colon x \in S^*,\ y \in T^*,\ |y| \le |x|\}$. Since $L' \in E_k = \sigma_k$ and L'' is regular, $L'' - L' \in \pi_k$. If $h\colon ([T_\#]^2)^* \to (T \cup \{\#\})^*$ is the length-preserving homomorphism determined by defining $h([a,b]) = a$, then $L = h(L'' - L') \in H[\pi_k]$. Therefore, from Theorem 4.1.4, $L \in \sigma_{k+1}$.

To see that $\sigma_{k+1} \subseteq E_{k+1}$, suppose $L \in \sigma_{k+1}$. From Theorem 4.1.4 there is a nonerasing homomorphism h and a language $L' \in \pi_k$ such that $L = h(L')$. Since $\pi_k = \text{co-}\sigma_k$ and $\sigma_k = E_k$, there is a language $L_0' \in \mathrm{DTIME(lin)}$ such that for all x (over the appropriate alphabet)

$$x \notin L' \iff (\exists y_1)_x(\forall y_2)_x \cdots (\forall y_k)_x[\theta(x,y_1,\ldots,y_k) \in L_0'].$$

Define a language L_0 by:

$\theta(x,y_1,\ldots,y_{k+1}) \in L_0$ if and only if

(i) $h(y_1) = x$;

(ii) for $1 \le j \le k/2$, $|y_{2j+1}| \le |y_1|$; and

(iii) if for $1 \le j \le k/2$ $|y_{2j}| \le |y_1|$ then

$$\theta(y_1,y_2,\ldots,y_{k+1}) \notin L_0'.$$

Now if h is a nonerasing homomorphism, $h(y_1) = x$ implies $|y_1| \le |x|$; therefore

$$x \in L \iff (\exists y_1)_x(\forall y_2)_x \cdots (\exists y_{k+1})_x[\theta(x,y_1,\ldots,y_{k+1}) \in L_0],$$

so $L \in E_{k+1}$. □

A "direct" proof of the containment $\sigma_{k+1} \subseteq E_{k+1}$ (that is, one not relying on Theorem 4.1.4) is much longer, since it requires showing that if $L \subseteq S^*$ is in E_k, then $(L \oplus (S^*-L))^* \in E_{k+1}$.

If the linear hierarchy is finite, so that for some $k \ge 1$, $\bigcup\{\sigma_j\colon j \ge 1\} = \sigma_k$, then also $RUD = \sigma_k = E_k$. Hence for any rudimentary relation R, $\theta(R)$ could be obtained from a language in $\mathrm{DTIME(lin)}$ by use of a finite number (k) of alternations of bounded existential and universal quantification. Equivalently, if it could be shown that no finite number of applications of bounded quantification will suffice to define all rudimentary relations, then $\sigma_1 \subsetneq RUD$, so σ_1 = NTIME(n) could not be closed under complementation. In the next section, it will be shown that a stronger result about definability of rudimentary relations can be derived from the assumption that the linear hierarchy is finite: if it is finite, then there is an integer m such that for any rudimentary relation R, $\Theta(R)$ can be obtained by applying m operations to the language $\{0^n 1^n : n \geq 0\}$ (or to a Dyck set).

4.3. COMPLETE SETS IN THE LINEAR HIERARCHY

The concepts of efficient reduction of one (recognition) problem to another and of completeness have been found useful in studying the computational complexity of languages. The reducibilities considered are restrictions of the many-one reducibility of recursive function theory [43, 47]; their general form, and the corresponding definition of complete set, may be stated as follows.

Definition 4.3.1. Suppose $\mathcal{F}$ is a class of string-to-string functions.

(1) If $L \subseteq S^*$, $L' \subseteq T^*$ are languages, then L is $\mathcal{F}$-reducible to L' if there is a function $f: S^* \to T^*$ in $\mathcal{F}$ such that for all $x \in S^*$, $x \in L$ iff $f(x) \in L'$ (i.e., $L = f^{-1}(L')$). A family of languages $\mathcal{C}$ is said to be $\mathcal{F}$-reducible to a language L' if every language $L \in \mathcal{C}$ is $\mathcal{F}$-reducible to L'.

(2) A language L_0 is $\mathcal{F}$-complete in a family of languages $\mathcal{C}$ (or, complete in $\mathcal{C}$ with respect to $\mathcal{F}$-reductions) if $L_0 \in \mathcal{C}$ and $\mathcal{C}$ is $\mathcal{F}$-reducible to L_0.

Note that if a language L_0 is $\mathcal{F}$-complete for $\mathcal{C}$ then $\mathcal{C} \subseteq \{f^{-1}(L_0): f \in \mathcal{F}\}$. If the class $\mathcal{C}$ is closed under application of inverses of functions in $\mathcal{F}$ then since $L_0 \in \mathcal{C}$, equality holds; that is, L_0 generates $\mathcal{C}$ under the operations $\{f^{-1}: f \in \mathcal{F}\}$.

In applying these ideas to the linear hierarchy, we will not consider specific languages but rather view existence of complete sets and of reductions of languages in one class of languages to languages in another as properties of the classes of languages.

The appropriate class of functions to consider with the linear hierarchy is that consisting of functions which can be computed in linear time by deterministic Turing machines (say, with a two-way input tape and a one-way output tape). It is easy to see that this class of functions contains all identity functions and is closed under composition; hence the reducibility relation it defines is reflexive and transitive. We therefore abbreviate "L is reducible to L' by a function that can be computed in linear time" by: $L \leq^{lin} L'$. Two other reducibilities which have been extensively used and which will arise in the next chapter are: $\leq^{lg}$, corresponding to the functions that can be computed in lg(n) space [35, 53], and $\leq^{p}_{m}$, corresponding to the functions that can be computed in polynomial time [36]. Note that if either $L \leq^{lin} L'$ or $L \leq^{lg} L'$ then $L \leq^{p}_{m} L'$.

The following proposition states that each class in the linear hierarchy is "closed under linear-time reductions." The concept of closure under a reducibility is useful in comparing classes of languages and in deriving conditions for one class to be contained in another [7].

Proposition 4.3.2. For all $k \geq 1$, if $A \leq^{lin} B$ and $B \in \sigma_k$ (respectively, π_k, δ_k) then $A \in \sigma_k$ (respectively, π_k, δ_k).

Proof. The proof is a simple construction. Suppose $A \subseteq S^*$, $B \subseteq T^*$ and $f: S^* \to T^*$ is a function which can be computed in linear time and which reduces A to B: for all $x \in S^*$, $x \in A$ iff $f(x) \in B$. Note that if f can be computed in time $cn+d$, then for all x, $|f(x)| \leq c|x| + d$. First suppose $B \in \sigma_k$, so there is a language $C \in \sigma_{k-1}$ and a linear-time oracle machine M such that $B = M(C)$. The machine M and the machine that computes f can be combined to construct a linear-time oracle machine M' such that $M'(C) = A$. Given an input $x \in S^*$, M' first computes $f(x)$ as the input to M, accepting x if and only if M accepts $f(x)$ relative to the same oracle set. Then $M'(C) = \{x \in S^*: f(x) \in M(C) = B\} = f^{-1}(B) = A$. Since the length of $f(x)$ is bounded by a linear function of $|x|$, M' can be constructed to operate in linear time. Thus $A \in \sigma_k$. In the preceding construction, if M is deterministic then M' will also be deterministic; hence if $B \in \delta_k$ then $A \in \delta_k$. Moreover, since f also reduces S^*-A to T^*-B, if $B \in \pi_k$ then $A \in \pi_k$. □

From Proposition 4.3.2 it is clear that for any $k \geq 1$ and any $L \in \sigma_k$, $\{f^{-1}(L): f$ a linear-time computable function$\} \subseteq \sigma_k$. It will now be shown that σ_k is generated by a single language A_k under application of inverses of a subset of this class of functions. An

inductive scheme is used to define the generators; a generator is defined from the previous one by means of the "universal" linear-time oracle machine described in the following theorem.

Theorem 4.3.3. Let $\Sigma = \{0,1\}$. One can construct a nondeterministic linear-time oracle machine M_0 with input and tape alphabet Σ which has the following property: If M is any nondeterministic linear-time oracle machine (with alphabet Σ), then there is a homomorphism $h_M: \Sigma^* \to \Sigma^*$ such that for any oracle set $L \subseteq \Sigma^*$, $M(L) - \{e\} = h_M^{-1}(M_0(L))$.

Proof. The theorem essentially states that $M_0(L)$ is a "hardest" language for $NL(\{L\})$ in the sense of [24]. The construction of M_0 uses a technique in [55, 11].

First, suppose N is a (nondeterministic) oracle machine with four tapes, the last one the oracle tape, and input and tape alphabet Σ. Then there is a string $\overline{N} \in \{0,1,¢\}^*$ that describes N in such a way that information about the action of N can be extracted easily from $\overline{N}$ (e.g., using an encoding similar to that in Appendix A). Also, we can assume that no such encoding is a substring of any other, so that for $x = a_1(\overline{N})^d \ldots a_m(\overline{N})^d$ with $d \geq 1$, $a_1,\ldots,a_m \in \Sigma$, $m \geq 1$, the strings $a_1 \ldots a_m$ and $\overline{N}$ and the integer d can be determined uniquely from x.

Let $h_0: \{0,1,¢\}^* \to \Sigma^*$ be the homomorphism determined by defining $h_0(0) = 00$, $h_0(1) = 11$ and $h_0(¢) = 01$. Note that h_0 is a one-to-one function. The oracle machine M_0 will reject its input $x \in \Sigma^*$ unless $x = h_0(a_1(\overline{N})^d \ldots a_m(\overline{N})^d)$ for some $m \geq 1$, $a_1,\ldots,a_m \in \Sigma$,

$d \geq 1$ and $\overline{N}$ a description of a four-tape oracle machine. M_0 can be constructed so that checking that the input is in the correct form takes only linear time. On an input of the correct form, M_0 simulates some computation of N on $y = a_1 \ldots a_m$, using 4 of its tapes, including its oracle tape, just as N would. M_0 clocks the simulation, however: if within $|x|$ of its steps M_0 simulates an accepting computation of N on y, then M_0 accepts x, and otherwise it rejects x. M_0 can be constructed so that it needs at most $2|\overline{N}|$ steps to simulate one step of N. Then M_0 operates in linear time, and accepts x if and only if it simulates an accepting computation of N on y of length at most $|x|/2|\overline{N}|$.

Now suppose M is any nondeterministic linear-time oracle machine with alphabet Σ. From Corollary 2.3.5 there is an equivalent nondeterministic linear-time oracle machine with four tapes, so we may assume that M itself has 4 tapes, named in such a way that the last one is the oracle tape. Let M operate in time $cn+d$ and let $k = c+d$. Define the homomorphism $h_M: \Sigma^* \to \Sigma^*$ by $h_M(a) = h_0(a(\overline{M})^k)$ for $a \in \Sigma$. Note that for any $y \in \Sigma^*$, $|h_M(y)| = 2k|y| \cdot |\overline{M}|$. Then for any $y \in \Sigma^*$ and $L \subseteq \Sigma^*$, if M_0 accepts $h_M(y)$ relative to L, then by construction of M_0, $y \neq e$ and $y \in M(L)$. On the other hand, if y is a nonempty string in $M(L)$ then there is an accepting computation of M on y relative to L, which has at most $c|y| + d \leq k|y| = |h_M(y)|/2|\overline{M}|$ steps; therefore $h_M(y) \in M_0(L)$. Thus for any $L \subseteq \Sigma^*$ and $y \in \Sigma^* - \{e\}$, $y \in M(L)$ if and only if $h_M(y) \in M_0(L)$, or $M(L) - \{e\} = h_M^{-1}(M_0(L))$. □

The construction in Theorem 4.3.3 can be extended to any honest, superadditive time-bounding function $t(n)$, by allowing M_0 to take $t(|x|)$, rather than $|x|$, steps during the simulation phase. The resulting machine M_0 will be universal for the class of oracle machines which operate in time linear in $t(n)$.

The oracle machine described in the proof above gives a uniform method for producing the desired generators for the classes σ_k.

<u>Definition 4.3.4</u>. Let M_0 be the linear-time oracle machine of Theorem 4.3.3. Define $A_0 = \emptyset$ and for $k \geq 0$, $A_{k+1} = M_0(A_k)$.

It is apparent from the definition that $A_k \in \sigma_k$ for each $k \geq 0$. Therefore (from Proposition 4.1.2) for $k \geq 1$, σ_k contains the family of languages generated by A_k under application of inverse homomorphisms and union with $\{e\}$; the following theorem implies that in fact σ_k is equal to this family of languages.

<u>Theorem 4.3.5</u>.

(1) For all $k \geq 1$, $\sigma_k = NL(\{A_{k-1}\})$.

(2) For all $k \geq 0$, if $L \in \sigma_k$ then there is a homomorphism h such that $L - \{e\} = h^{-1}(A_k)$.

<u>Proof</u>. Both parts of the theorem will be proved together, using induction on k. By definition, part (2) holds for $k = 0$; it will be shown that if part (2) holds for σ_{k-1}, then both (1) and (2) hold for σ_k.

Suppose for some $k \geq 1$, for every language $L \in \sigma_{k-1}$, there is a homomorphism h such that $L - \{e\} = h^{-1}(A_{k-1})$. Let $L \subseteq S^*$ be any

language in σ_k and suppose $S = \{s_1, \ldots, s_p\}$. Let $h_1\colon S^* \to \{0,1\}^*$ be the homomorphism determined by defining for $1 \le j \le p$, $h_1(s_j) = 01^j0$. Then $L_1 = h_1(L)$ is also in σ_k so there exist a language $L_2 \in \sigma_{k-1}$ and a nondeterministic linear-time oracle machine M_1 such that $L_1 = M_1(L_2)$. Without loss of generality we may assume that $e \notin L_2$, so since $L_2 \in \sigma_{k-1}$ there is a homomorphism h_2 such that $L_2 = h_2^{-1}(A_{k-1})$. Let M_2 be the nondeterministic linear-time oracle machine that acts like M_1 except that it uses tape symbols of M_1 encoded as strings in $\{0,1\}^*$, and if M_1 would query its oracle about a string z, M_2 instead queries its oracle about $h_2(z)$. Then $M_2(A_{k-1}) = M_1(h_2^{-1}(A_{k-1})) = M_1(L_2)$. Therefore $L_1 = M_1(L_2) \in NL(\{A_{k-1}\})$; since $L = h_1^{-1}(L_1)$, also $L \in NL(\{A_{k-1}\})$ and so $\sigma_k \subseteq NL(\{A_{k-1}\})$. Further, M_2 satisfies the conditions of Theorem 4.3.3, so there is a homomorphism h_3 such that $M_2(A_{k-1}) - \{e\} = h_3^{-1}(M_0(A_{k-1})) = h_3^{-1}(A_k)$. Let $h\colon S^* \to \{0,1\}^*$ be the homomorphism that is the composition of h_3 with h_1: $h(s) = h_3(h_1(s))$ for $s \in S$. Then
$L - \{e\} = h_1^{-1}(L_1 - \{e\}) = h_1^{-1}(h_3^{-1}(A_k)) = h^{-1}(A_k)$. □

The following corollary is easily proved from part (2) of Theorem 4.3.5, using a construction similar to that used to prove Theorem 4.3.5(1).

<u>Corollary 4.3.6</u>. For all $k \ge 1$, $\delta_k = \{M(A_{k-1})\colon M$ a deterministic linear-time oracle machine$\}$. □

We now consider the consequences for the linear hierarchy and for

the class of rudimentary relations of the representation given in Theorem 4.3.5. The corollaries which follow will be used in the discussion of the polynomial hierarchy in Chapter 5 and have significance for basic questions in automata-based computational complexity.

Suppose the linear hierarchy is finite, so that $RUD = \sigma_k$ for some $k \ge 1$. Since $A_k \in RUD$, from Corollary 3.3.5(2) A_k can be defined from the language $\{0^n1^n\colon n \ge 0\}$ by use of some finite number, say m, of applications of the Boolean operations, inverse homomorphism and length-preserving homomorphism. But from Theorem 4.3.5(2), $\sigma_k = \{h^{-1}(A_k),\ h^{-1}(A_k) \cup \{e\}\colon h$ a homomorphism$\}$. Therefore if the linear hierarchy is finite, then for any rudimentary relation R, $\Theta(R)$ can be defined from $\{0^n1^n\colon n \ge 0\}$ by at most $m+2$ applications of the Boolean operations, inverse homomorphism and length-preserving homomorphism. Conversely, if a language L is obtained from $\{0^n1^n\colon n \ge 0\}$ by use of k of these operations then $L \in \sigma_k$. Hence the linear hierarchy is infinite if and only if $\{\Theta(R)\colon R \in RUD\}$ cannot be generated from one language by use of a bounded number of applications of language-theoretic operations. The representation of the classes σ_k in terms of operations applied to the language A_k also shows that σ_k possesses a $\le^{\text{lin}}$-complete language.

<u>Corollary 4.3.7</u>. For all $k \ge 1$, the language $\{\$\} \cup \{\$\}A_k$ is complete in σ_k with respect to linear-time reductions.

<u>Proof</u>. Suppose $k \ge 1$. Recall that by definition, $e \notin A_k$ and $A_k \subseteq \{0,1\}^*$; let $A_k' = \{\$\} \cup \{\$\}A_k$. As noted previously, $A_k \in \sigma_k$, so

$A_k' \in \sigma_k$. If $L \subseteq S^*$ is any language in σ_k, let $h\colon S^* \to \{0,1\}^*$ be the homomorphism such that $L - \{e\} = h^{-1}(A_k)$. If $e \in L$, let $f\colon S^* \to \{0,1,\$\}^*$ be the function defined by $f(x) = \$h(x)$ for all $x \in S^*$; if $e \notin L$, define f by $f(e) = e$ and for $x \neq e$, $f(x) = \$h(x)$. In either case, for any $x \in S^*$, $x \in L$ iff $f(x) \in A_k'$, so f reduces L to A_k'. Moreover, the function f can be computed in time $mn+1$ (where $m = \max\{|h(a)|: a \in S\}$) by a deterministic Turing machine with only a one-way input tape and a one-way output tape, so $L \leq^{\text{lin}} A_k'$. □

The arguments of Theorem 4.3.5 and Corollary 4.3.7 can be used to show that, in general, if C is a class of languages and L_0 is complete in C with respect to linear-time reductions, then $NL(C) = NL(\{L_0\})$ and $NL(C)$ possesses a linear-time complete language.

The existence of a complete set for each class σ_k yields the following information on the question of the finiteness of the linear hierarchy.

Corollary 4.3.8.

(1) The linear hierarchy is finite if and only if the class of rudimentary relations contains a language that is complete with respect to linear-time reductions.

(2) If the linear hierarchy is infinite then the rudimentary relations are properly contained in $\mathcal{E}_*^2$.

Proof. If the linear hierarchy is finite then there is some k such that $RUD = \sigma_k$. Then since $A_k' = \{\$\} \cup \{\$\}A_k$ is $\leq^{\text{lin}}$-complete in σ_k, for every rudimentary relation R, $\Theta(R)$ would be reducible to $A_k' \in RUD$ in linear time; that is, A_k' would be complete in RUD with respect to linear-time reductions. On the other hand, suppose $L_0 \in RUD$ is $\leq^{\text{lin}}$-complete for RUD. Since $RUD = \bigcup\{\sigma_j: j \geq 1\}$ there is some k such that $L_0 \in \sigma_k$. If R is any rudimentary relation then $\Theta(R) \leq^{\text{lin}} L_0 \in \sigma_k$, so from Proposition 4.3.2, $\Theta(R) \in \sigma_k$. Hence $RUD \subseteq \sigma_k$ and the linear hierarchy is finite.

To see part (2), recall from Corollary 3.2.8 and the discussion there that $RUD \subseteq \text{DSPACE}(n)$ and $\mathcal{E}_*^2 = \text{DSPACE}(n)$. There is a language $L_0 \in \text{DSPACE}(n)$ such that $\text{DSPACE}(n) \leq^{\text{lin}} L_0$ [55]; the language L_0 is similar in form to the language accepted by the machine M_0 of Theorem 4.3.3:

$L_0 = \{\bar{a}_1 \bar{M} \bar{a}_2 \bar{M} \ldots \bar{a}_n \bar{M}$: M is a deterministic linear-bounded automaton and $a_1 a_2 \ldots a_n \in L(M)\}$. Therefore if $\mathcal{E}_*^2 \subseteq RUD$, then $L_0 \in RUD$, so RUD contains a complete set with respect to linear-time reductions and from part (1), the linear hierarchy must be finite. The reverse implication of part (2) seems implausible; it states that if for some k, $RUD = \sigma_k$ then $\text{DSPACE}(n) = \sigma_k$, and hence

$\text{DSPACE}(n) = \{h^{-1}(A_k), h^{-1}(A_k) \cup \{e\}$: h a homomorphism$\}$. □

The classes $\text{DSPACE}(\lg(n))$ and $\text{NSPACE}(\lg(n))$ are contained in the rudimentary relations (Proposition 3.2.4), hence contained in $\bigcup\{\sigma_j: j \geq 1\}$. Whether either of these families of languages is comparable to any σ_k is not known; however, the structure of the classes σ_k revealed in Theorem 4.3.5 gives partial information on this question.

Corollary 4.3.9. For all $k \geq 1$, σ_k is not equal to either DSPACE(lg(n)) or NSPACE(lg(n)).

Proof. As remarked previously, Theorem 4.3.5(2) implies that for all $k \geq 1$, $\sigma_k = \{h^{-1}(A_k), h^{-1}(A_k) \cup \{e\}: h$ a homomorphism$\}$. However, such a representation (as a class generated by a single language under those operations) cannot hold for the classes DSPACE(lg(n)) and NSPACE(lg(n)): they are each the union of an infinite hierarchy of classes that are closed under inverse homomorphism and union with $\{e\}$ [32, 49,50].

There is an alternate proof of Corollary 4.3.9 using a "translational" argument [7]; this approach will be taken in Chapter 5 to generalize the statement of the corollary to DSPACE($(\lg(n))^j$) and NSPACE($(\lg(n))^j$) for all $j \geq 1$.

Neither DSPACE(lg(n)) nor NSPACE(lg(n)) is known to be closed under nonerasing homomorphism; their closure under this operation has the following consequences for the linear hierarchy.

Corollary 4.3.10.

(1) If NSPACE(lg(n)) is closed under nonerasing homomorphism then NTIME(n) is not closed under complementation.

(2) If DSPACE(lg(n)) is closed under nonerasing homomorphism then the linear hierarchy is infinite (and, in particular, NTIME(n) is not closed under complementation).

Proof.

(1) The class NSPACE(lg(n)) is closed under inverse homomorphism and intersection and contains the Dyck sets and regular sets. Using Propositions 3.2.2 and 3.2.3, if NSPACE(lg(n)) is closed under nonerasing homomorphism then σ_1 = NTIME(n) $\subseteq$ NSPACE(lg(n)). From Proposition 4.1.3, if σ_1 is closed under complementation then for all $j \geq 1$, $\sigma_j \subseteq \sigma_1$ so NSPACE(lg(n)) $\subseteq$ *RUD* $\subseteq \sigma_1$ = NTIME(n). Since NTIME(n) $\neq$ NSPACE(lg(n)), a contradiction results if both closure properties are assumed.

(2) Suppose DSPACE(lg(n)) is closed under nonerasing homomorphism. It is also closed under the Boolean operations and inverse homomorphism and contains $\{0^n 1^n: n \geq 0\}$; hence from Corollary 3.3.5(2), *RUD* $\subseteq$ DSPACE(lg(n)), so *RUD* = $\bigcup\{\sigma_j: j \geq 1\}$ = DSPACE(lg(n)). If also the linear hierarchy is finite, then there is some k such that *RUD* = σ_k = DSPACE(lg(n)), contradicting Corollary 4.3.9; therefore the linear hierarchy must be infinite. From Proposition 4.1.3, if the linear hierarchy is infinite then for all $k \geq 1$, σ_k is not closed under complementation. □

It was possible to draw a more general conclusion in part (2) of this corollary than in part (1) because NSPACE(lg(n)) is not known to be closed under complementation. This difference between the two lg(n)-space classes is reflected in the facts that (i) if DSPACE(lg(n)) is closed under nonerasing homomorphism then DSPACE(lg(n)) is the rudimentary relations; while (ii) if NSPACE(lg(n)) is closed under nonerasing homomorphism then it is the class of <u>positive rudimentary</u>

relations, defined by Bennett [4]. In terms of the definitions used here, the class of positive rudimentary relations is the smallest class of string relations containing the concatenation relations and closed under union, intersection, explicit transformation, bounded existential quantification and universal subpart quantification (that is, quantification of the form "for every z that is a substring of y ..."). This is a "positive" class in that the operations of complementation and bounded universal quantification are excluded. The proof in [42] in fact shows that any language in NSPACE(lg(n)) is a positive rudimentary relation; hence σ_1 = NTIME(n) is contained in the class of positive rudimentary relations. On the other hand, it is not known whether even δ_2 is contained in the positive rudimentary relations. In the next chapter we will see that a positive answer to this question would imply that the class NTIME(poly) is closed under complementation.

Chapter 5: THE POLYNOMIAL HIERARCHY

This chapter explores the polynomial hierarchy of Meyer and Stockmeyer [40,52,53], employing primarily the connections between it and the linear hierarchy. The polynomial hierarchy can be used for the classification of problems whose solution is not known to require more than polynomial time, but for which no polynomial-time algorithm (even nondeterministic) is known to exist. This hierarchy, like the linear hierarchy, is a structure analogous to the arithmetic hierarchy; its definition, using polynomial-time oracle machines, extends the analogy between the recursive sets (those sets for which it is possible to decide membership) and the class DTIME(poly) of sets recognizable deterministically in polynomial time (those sets for which it is "practical" to decide membership [15,36]).

Section 1 presents the definition of the classes in the polynomial hierarchy and establishes that proper containment holds between corresponding classes in the linear and polynomial hierarchies. A further relationship between the two structures is derived in the second section: each language in a class in the polynomial hierarchy is represented in the corresponding class of the linear hierarchy by application of "polynomial padding." The third section contains facts about the linear and polynomial hierarchies that follow from this relationship.

5.1. DEFINITION

The following definition differs only in notation from that given in [40]. In particular, the superscript "p" has been dropped from the names of the classes, since the classes of the arithmetic hierarchy are not referred to here by name.

Definition 5.1.1. Define $\Sigma_0 = \Pi_0 = \Delta_0 = \{\phi\}$. For $k \geq 0$, define

$$\Sigma_{k+1} = NP(\Sigma_k)$$
$$\Pi_{k+1} = co\text{-}NP(\Sigma_k)$$
$$\Delta_{k+1} = P(\Sigma_k) .$$

Finally, define $PH = \cup\{\Sigma_k : k \geq 0\}$.

Note that Δ_1 = DTIME(poly) and Σ_1 = NTIME(poly).

It is apparent from this definition that each class in the polynomial hierarchy contains the corresponding class in the linear hierarchy (e.g., $\sigma_1 \subseteq \Sigma_1$). In fact, using the time-hierarchy theorem given in Chapter 2 (Theorem 2.4.1), the containments can be shown to be proper.

Proposition 5.1.2. For all $k \geq 1$, $\sigma_k \subsetneq \Sigma_k$, $\pi_k \subsetneq \Pi_k$ and $\delta_k \subsetneq \Delta_k$.

Proof. Suppose $k \geq 1$. Recall from Theorem 4.3.5 that $\sigma_k = NL(\{A_{k-1}\})$ with $A_{k-1} \in \sigma_{k-1}$. From Corollary 2.4.2, $NL(\{A_{k-1}\}) \subsetneq NP(\{A_{k-1}\})$ so $\sigma_k \subsetneq NP(\sigma_{k-1}) \subseteq \Sigma_k$. Similarly, using Corollaries 4.3.6 and 2.4.2, $\delta_k \subseteq DTIME(n^2, A_{k-1}) \subsetneq P(\{A_{k-1}\}) \subseteq P(\Sigma_{k-1})$ so $\delta_k \subsetneq \Delta_k$. Now if $\pi_k = \Pi_k$ then (from the definition) $\sigma_k = co\text{-}\pi_k = co\text{-}\Pi_k = \Sigma_k$; hence $\pi_k \subsetneq \Pi_k$. □

By using constructions similar to those given for Theorem 2.2.4, it can be seen that, for all $k \geq 1$, Σ_k possesses the same positive closure properties as σ_k: Σ_k is closed under union, intersection, product, Kleene *, inverse homomorphism and linear-erasing homomorphism. Moreover, Σ_k is closed under application of polynomial-erasing homomorphisms; that is, if $L \subseteq S^*$ is a language in Σ_k and $h: S^* \to T^*$ is a homomorphism with the property that for some polynomial $p(n)$, for every $x \in L$, $|x| \leq p(|h(x)|)$, then $h(L) \in \Sigma_k$. We will make occasional (implicit) use of these facts.

The same relationships hold between classes in the polynomial hierarchy as hold between the corresponding classes in the linear hierarchy. Thus, for each $k \geq 1$, $\Sigma_{k-1} \cup \Pi_{k-1} \subseteq \Delta_k \subseteq \Sigma_k \cap \Pi_k$. As is the case with the linear hierarchy, it is not known whether any of the inclusions $\Sigma_k \subseteq \Sigma_{k+1}$ for $k \geq 1$ is proper, i.e., whether the polynomial hierarchy is finite or infinite. (If DTIME(poly) = NTIME(poly) then none of these inclusions can be proper [40].) By restating the proof of Proposition 4.1.3 in the context of the polynomial hierarchy, it can be seen that if for some $k \geq 1$, $\Sigma_k = \Sigma_{k+1}$ (or, equivalently, $\Sigma_k = \Pi_k$) then the polynomial hierarchy collapses at that point, that is, $\Sigma_j \subseteq \Sigma_k$ for all $j \geq 1$.

5.2. REPRESENTATION IN THE LINEAR HIERARCHY

Many properties of the classes in the polynomial hierarchy can be established (as indicated above) by using arguments similar to those in Chapter 4. However, some properties can be arrived at more simply by making use of the fact that a language in the polynomial hierarchy has a certain representation in the corresponding class of the linear hierarchy.

Definition 5.2.1. Suppose $L \subseteq S^*$ is a language, $¢ \notin S$ is a new symbol and $p(n)$ is a polynomial. The language $\{x¢^m : x \in L, m = p(|x|)\}$ is termed a polynomial representative of L.

It will be shown that the polynomial hierarchy is "polynomially represented" in the linear hierarchy; we first establish some preliminary results.

Recall that for languages A and B, $A \leq^{lin} B$ if there is a string function f that can be computed in linear time such that $A = f^{-1}(B)$ (i.e., for all x, $x \in A$ iff $f(x) \in B$). In connection with the polynomial hierarchy, we extend the class of functions allowed for reductions to include functions computable in polynomial time.

Definition 5.2.2. Let $A \subseteq S^*$ and $B \subseteq T^*$ be languages.

(1) $A \leq^P_m B$ if there is a function $f : S^* \to T^*$ and a polynomial $p(n)$ such that $A = f^{-1}(B)$ and for any $x \in S^*$, $f(x)$ can be computed in time $p(|x|)$.

(2) $A \leq^{lg} B$ if there is a function $f : S^* \to T^*$ such that $A = f^{-1}(B)$ and for any $x \in S^*$, $f(x)$ can be computed in space $lg(|x|)$.

The model for the computation of these reduction functions is a Turing machine with two-way (read-only) input, a one-way output tape and multiple work tapes. The space bound applies only to the number of squares used on the work tapes. The class of functions computable in

this way in lg(n) space is known to be closed under composition [35,53] and clearly contains the identity function, so $\leq^{lg}$ is a transitive and reflexive relation. Similarly, $\leq^P_m$ is transitive and reflexive. Since a Turing machine that operates in lg(n) space (and halts) also operates in polynomial time, $\leq^{lg}$ is a restriction of $\leq^P_m$. The relation $\leq^P_m$ is itself a restriction of the relation on languages defined by the operator P(): i.e., if $A \leq^P_m B$ then $A \in P(\{B\})$.

Using essentially the same argument as was given for Proposition 4.3.2, the following proposition can be established; it states that the classes in the polynomial hierarchy are closed under polynomial-time reductions (hence closed under lg(n)-space reductions).

Proposition 5.2.3. For any $k \geq 1$ and languages L_1, L_2, if $L_1 \leq^P_m L_2$ and $L_2 \in \Sigma_k$ (resp., Π_k, Δ_k) then $L_1 \in \Sigma_k$ (resp., Π_k, Δ_k). □

The statement of Proposition 5.2.3 can be easily seen to hold in general; that is, if $A \leq^P_m B$ then, for example, $B \in NP(\{C\})$ implies $A \in NP(\{C\})$ for any language C. The next proposition might be termed the "inverse" of this fact: if two languages are polynomially equivalent then they give rise to the same class of languages when used as oracle sets with polynomial-time oracle machines.

Proposition 5.2.4. For any language A, $P(P(\{A\})) \subseteq P(\{A\})$ and $NP(P(\{A\})) \subseteq NP(\{A\})$. Hence if $L_1 \leq^P_m L_2$ then $P(\{L_1\}) \subseteq P(\{L_2\})$ and $NP(\{L_1\}) \subseteq NP(\{L_2\})$.

Proof. The proof uses a construction similar to that of Proposition 2.1.5. Suppose M_1 is an oracle machine that operates in time $p_1(n)$ and

M_2 is a deterministic oracle machine that operates in time $p_2(n)$, where $p_1(n)$ and $p_2(n)$ are polynomials. Let $q(n) = p_1(n) + p_1(p_2(n))$. The action of M_1 on input strings can be composed with the action of M_2 on the oracle strings to construct an oracle machine M such that: (i) M is deterministic if M_1 is deterministic; (ii) M operates in time q(n); and (iii) if A is an oracle set for M_2 and $B = M_2(A)$ then $M(A) = M_1(B)$ (i.e., $M(A) = M_1(M_2(A))$). The details of the construction are straightforward. Note that if M_2 is not assumed to be deterministic, then this construction fails, since there may be nonaccepting computations of M_2 relative to A on a string $z \in M_2(A)$. □

Returning to "polynomial representation," we see that it is a restricted form of lg(n)-space equivalence.

Proposition 5.2.5. If L' is a polynomial representative of L then $L' \equiv^{lg} L$.

Proof. Suppose $L \subseteq S^*$ and $L' = \{x¢^m : x \in L, m = p(|x|)\}$ where $¢ \notin S$ and p(n) is some polynomial. Let $f : S^* \to (S \cup \{¢\})^*$ be the function $f(x) = x¢^{p(|x|)}$. Let $g : (S \cup \{¢\})^* \to (S \cup \{¢\})^*$ be the function defined by: for $y \in (S \cup \{¢\})^*$, if $y = x¢^{p(|x|)}$ for some $x \in S^*$ then $g(y) = x$ and for y not of this form, $g(y) = ¢$. Clearly f reduces L to L'; since $L \subseteq S^*$, $¢ \notin L$ so g reduces L' to L. Moreover for a string z of length n, both f(z) and g(z) can be computed in space linear in lg(n). □

A polynomial representative L' of a language L contains strings from L padded to increase their lengths. Since computation time is measured in terms of the length of the input, by a suitable choice of

the length of the padding the apparent complexity of L' can be made smaller than that of L. Applying this principle to polynomial-time oracle machines yields the following result.

Proposition 5.2.6. Suppose M is a polynomial-time oracle machine. Then there is a linear-time oracle machine M' such that (i) M' is deterministic if M is deterministic; and (ii) for any oracle set A, M'(A) is a polynomial representative of M(A).

Proof. Let M be an oracle machine that operates in time p(n), where p(n) is some polynomial. Let S be the input alphabet of M and $¢ \notin S$. The oracle machine M' operates as follows: given an input $y \in (S \cup \{¢\})^*$, M' first tests whether y is of the form $x¢^{p(|x|)}$ for some $x \in S^*$. If the test is successful, then M' proceeds to accept y if and only if M accepts x (relative to the same oracle set). Clearly M' will be deterministic if M is deterministic. Since M operates in time p(n), the second phase of the computation of M' takes at most $p(|x|) \le |y|$ steps. Thus M' operates in linear time, and for any oracle set A, $M'(A) = \{x¢^{p(|x|)} : x \in M(A)\}$ is a polynomial representative of M(A). □

Based on the preceding propositions, the proof of the desired "representation theorem" is straightforward.

Theorem 5.2.7. For any $k \ge 0$ and any language L, $L \in \Sigma_k$ (resp., Π_k, Δ_k) if and only if L has a polynomial representative in σ_k (resp., π_k, δ_k).

Proof. The statement is clearly true for k = 0, since the only polynomial representative of the empty set is the empty set.

Suppose $k \ge 1$, L is a language and L' is a polynomial representative of L. If $L' \in \sigma_k$ (π_k, δ_k) then also $L' \in \Sigma_k$ (Π_k, Δ_k); from Proposition 5.2.5, $L \leq^{lg} L'$, so from Proposition 5.2.3, $L \in \Sigma_k$ (Π_k, Δ_k).

As noted above, the implication in the other direction holds for $k = 0$; suppose it holds for some $k \geq 0$ and let L be any language in Σ_{k+1}. By definition there is a language $A \in \Sigma_k$ such that $L \in NP(\{A\})$. Since $A \in \Sigma_k$, A has a polynomial representative A' in σ_k. Since $A \leq^{lg} A'$, from Proposition 5.2.4, $NP(\{A\}) \subseteq NP(\{A'\})$ so $L \in NP(\{A'\})$. Hence from Proposition 5.2.6, L has a polynomial representative in $NL(\{A'\}) \subseteq NL(\sigma_k) = \sigma_{k+1}$. A similar argument, again using Propositions 5.2.4 and 5.2.6, shows that for $L \in \Delta_{k+1} = P(\Sigma_k)$, L has a polynomial representative in δ_{k+1}. If $L \subseteq S^*$ is a language in Π_{k+1} then for $L_1 = S^* - L$, we have $L_1 \in \Sigma_{k+1}$. Since the statement was shown to hold in this case, there is a polynomial p(n) and a symbol $¢ \notin S$ such that $L_1' = \{x¢^m : x \in L_1, m = p(|x|)\}$ is in σ_{k+1}. Let $L_p = \{x¢^m : x \in S^*, m = p(|x|)\}$; then $L_p \in$ DTIME(lin), so $L_1' \cup [(S\cup\{¢\})^* - L_p] \in \sigma_{k+1}$. Therefore $(S\cup\{¢\})^* - [L_1' \cup [(S\cup\{¢\})^* - L_p]] = L_p - L_1' \in \pi_{k+1}$; and $L_p - L_1' = \{x¢^m : m = p(|x|), x \in L\}$ is a polynomial representative of L. □

The remainder of this chapter is devoted to consequences of Theorem 5.2.7 for the linear and polynomial hierarchies.

5.3. PROPERTIES OF THE HIERARCHIES

We first use the representation theorem to show that if the linear hierarchy is finite then the polynomial hierarchy must be finite as well.

The proof technique does not seem to allow proof of the reverse implication; however, necessary and sufficient conditions for the finiteness of the polynomial hierarchy which involve the classes of the linear hierarchy can be established. The relationships between the questions of finiteness of the two hierarchies follow from a more general statement (Proposition 5.3.2) about families of languages that contain a class in the linear hierarchy.

Definition 5.3.1. A family of languages $\mathcal{C}$ is said to be closed under removal of polynomial padding if whenever L' is a polynomial representative of L and $L' \in \mathcal{C}$, also $L \in \mathcal{C}$.

Proposition 5.3.2. Suppose $\mathcal{C}$ is a family of languages which is closed under removal of polynomial padding. If for some $k \geq 1$, σ_k (resp., π_k, δ_k) is contained in $\mathcal{C}$, then Σ_k (resp., Π_k, Δ_k) is contained in $\mathcal{C}$.

Proof. Suppose $k \geq 1$ and $\sigma_k \subseteq \mathcal{C}$, where $\mathcal{C}$ is closed under removal of polynomial padding. If L is any language in Σ_k then from Theorem 5.2.7 there is a language $L' \in \sigma_k$ such that L' is a polynomial representative of L. Then $L' \in \mathcal{C}$ so since $\mathcal{C}$ is closed under removal of polynomial padding, $L \in \mathcal{C}$; hence $\Sigma_k \subseteq \mathcal{C}$. The other two cases ($\pi_k$ and Π_k, and δ_k and Δ_k) follow from a similar argument. □

It is easy to see (using Propositions 5.2.3, 5.2.5) that every class in the polynomial hierarchy is closed under removal of polynomial padding. Thus, for example, if $\sigma_2 \subseteq \Sigma_1$ then $\Sigma_2 \subseteq \Sigma_1$ and therefore $PH = \bigcup_j \Sigma_j \subseteq \Sigma_1$; since $\sigma_2 \subseteq \Sigma_2$ the converse also holds. The general statements that follow from this reasoning are contained in the next corollary.

Corollary 5.3.3. For every $k \geq 1$:

(1) $\pi_k \subseteq \Sigma_k$ if and only if $\delta_{k+1} \subseteq \Sigma_k$ if and only if for all $j \geq 1$, $\Sigma_j \subseteq \Sigma_k$.

(2) If $\cup\{\sigma_j : j \geq 1\} = \sigma_k$ then $PH = \Sigma_k$. □

For the case $k = 1$, the first part of this corollary implies the following fact: co-NTIME(n) is contained in NTIME(poly) if and only if NTIME(poly) is closed under complementation. The next corollary also generalizes a result known for NTIME(poly) = Σ_1 to the other classes in the polynomial hierarchy; in this case, the fact that NTIME(poly) = DTIME(poly) if and only if DTIME(poly) is closed under nonerasing homomorphism.

Corollary 5.3.4. For all $k \geq 1$ the following are equivalent:

(1) $\sigma_k \subseteq \Delta_k$;

(2) $\Sigma_k = \Delta_k$;

(3) Δ_k is closed under nonerasing homomorphism. □

From Propositions 5.3.2 and 5.1.2 it is clear that if C is a family of languages closed under removal of polynomial padding and, e.g., $\sigma_k \subseteq C$ then $\sigma_k \subsetneq C$. Therefore, no class in the linear hierarchy can be equal to any class closed under removal of polynomial padding. Many families of languages defined using resource-bounded automata can be shown to be closed under this operation; for some of these families (e.g., PSPACE) the fact that they properly contain σ_k for each k can be derived from the containment $\sigma_k \subseteq$ DSPACE(n). Of greater interest are families

defined using space bounds of the form $(\lg(n))^j$, i.e., polynomials in $\lg(n)$. Except for the case $j = 1$, these families are not known to be comparable to any class in the linear hierarchy or to the class of rudimentary relations.

Corollary 5.3.5.

(1) For all $j,k \geq 1$, $\sigma_k \neq$ DSPACE($(\lg(n))^j$) and $\sigma_k \neq$ NSPACE($(\lg(n))^j$).

(2) For all $k \geq 1$, $\sigma_k \neq \cup\{$DSPACE $((\lg(n))^j) : j \geq 1\}$.

(Recall that $\bigcup_j$ NSPACE($(\lg(n))^j$) = $\bigcup_j$ DSPACE($(\lg(n))^j$)[48]).

The polynomial hierarchy was originally presented as a structure for classifying the complexity of problems (encoded as languages). Reducibilities such as $\leq_m^P$ have been found to be useful in such classification; moreover, it can be more easily done when complete sets are known. Use of Theorem 5.2.7 (in conjunction with Theorem 4.3.5) establishes one sequence of complete sets for the classes in the polynomial hierarchy, the languages $A_0, A_1, \ldots$ from Chapter 4. These complete sets are not "natural", taking "natural" to mean a language that arises from a problem the solution of which is of independent interest. However, the relationship between the linear and polynomial hierarchies can be used to simplify proofs of completeness for other languages and (as in Chapter 4) the existence of complete sets gives some information about the classes in the polynomial hierarchy.

Proposition 5.3.6. For all $k \geq 1$,

(1) $\Sigma_k \leq^{\lg} A_k$; and

(2) $\Sigma_k = NP(\{A_{k-1}\})$. □

In general, if L_0 is a language such that $\sigma_k \leq_m^P L_0$ then also $\Sigma_k \leq_m^P L_0$ and $\Sigma_{k+1} = NP(\{L_0\})$.

The representation of the classes Σ_k given in part (2) of this proposition shows that the first part cannot be strengthened; that is, for any reducibility more "efficient" than $\leq^{lg}$, such complete sets cannot exist. Any lg(n)-space computable function can be computed in polynomial time; on the other hand, any polynomial can be "clocked" in lg(n) space (i.e., for any j there is a machine that on an input of length n will use lg(n) space and run for exactly n^j steps). The latter property is essential for the reduction of a language in Σ_k to the language A_k (or any other language).

<u>Corollary 5.3.7</u>. Suppose F is a class of functions with the property that for some polynomial p(n), any function in F can be computed in time p(n) except for finitely many inputs. Then for all $k \geq 1$ Σ_k cannot have an F-complete set.

<u>Proof</u>. The proof is by contradiction, employing a technique used in [6,7,19]. Suppose L_0 is F-complete in Σ_k for some $k \geq 1$. Since then $L_0 \in \Sigma_k = NP(\{A_{k-1}\}) = \cup\{NTIME(q(n),A_{k-1}) : q(n) \text{ a polynomial}\}$, there is some polynomial $q_0(n)$ such that $L_0 \in NTIME(q_0(n),A_{k-1})$. Now if L is any language in Σ_k then L is F-reducible to L_0: for some $f \in F$, $L = f^{-1}(L_0)$. The obvious construction, combining the time $q_0(n)$ oracle machine for L_0 and the machine that computes f, yields $L \in NTIME(2q_0(p(n)), A_{k-1})$. Hence $\Sigma_k \subseteq NTIME(2q_0(p(n)),A_{k-1})$ so using Corollary 2.4.2, $\Sigma_k \subsetneq NP(\{A_{k-1}\}) = \Sigma_k$, the desired contradiction. □

Suppose a space bounding function S satisfies

$$\lim_{n\to\infty} \frac{S(n)}{lg(n)} = 0$$

and f is a function that can be computed in space S(n). Then, by counting configurations of a machine that computes f, it can be seen that f can also be computed in time n^3 (except for finitely many inputs, the number of which depends on f); hence a result similar to Proposition 5.3.6 (1) cannot hold if the space allowed is further restricted.

Corollary 5.3.7 reveals, for example, that none of the classes Σ_k can possess a set that is complete with respect to linear-time reductions. This knowledge allows generalization of some facts concerning NTIME(poly) to other classes in the hierarchy: each of the classes on the right in the corollary below contains such a complete set (see [6, Lemma 3.4]).

<u>Corollary 5.3.8</u>. For all $k \geq 1$,

(1) $\Sigma_k \neq DSPACE(n^r)$ or $NSPACE(n^r)$ for any $r > 0$;

(2) $\Sigma_k \neq DTIME(2^{lin})$;

(3) $\Sigma_k \neq NTIME(2^{lin})$. □

The statement of part (3) can be strengthened in the case k = 1[5]: since $NTIME(poly) \subseteq NTIME(2^{lin})$, we see that $NTIME(poly) \subsetneq NTIME(2^{lin})$. For $k > 1$ it is unknown whether $\Sigma_k \subseteq NTIME(2^{lin})$. Corollaries 5.3.7 and 5.3.8 can be shown to hold for the classes PH, Π_k and Δ_k as well.

Corollary 2.4.2 implies that for any index $k \geq 1$, any language $L \in \Sigma_{k-1}$ and any polynomial p(n), Σ_k properly contains NTIME(p(n),L). However, it does not exclude the possibility that $\Sigma_k \subseteq NTIME(p(n),L)$ for some language L, even L in Σ_k, and some polynomial p(n). If $L \in \Sigma_k$ has this property for Σ_k, then Corollary 5.3.7 requires that the power of relative computation must be exploited to overcome the restriction to time p(n). The existence of such an L would imply that the polynomial hierarchy contains at least two distinct classes, so that DTIME(poly) $\neq$ NTIME(poly).

Corollary 5.3.9. For all $j \geq 1$, if for some polynomial p(n) and language $L \in PH$, $\Sigma_j \subseteq$ NTIME(p(n),L) then $\Sigma_j \subsetneq \Sigma_{j+1}$ and therefore NTIME(poly) is not closed under complementation. □

Now we consider the extension to the polynomial hierarchy of the representation of languages in the linear hierarchy in terms of bounded quantifiers. Quantification bounded by a polynomial function of the length of a string is used, instead of simply the length of the string. This polynomial-bounded quantification takes the following form.

Notation. Suppose L is a language and q a polynomial. Let languages L_1 and L_2 be defined by:

$x \in L_1 \iff \exists y[\ |y| \leq q(|x|)$ and $\theta(x,y) \in L]$ and

$x \in L_2 \iff \forall y[$ if $|y| \leq q(|x|)$ then $\theta(x,y) \in L]$.

Then L_1 (L_2) will be said to be defined from L by polynomial-bounded existential (universal) quantification. The expression defining L_1 will also be written $(\exists y)_q[\theta(x,y) \in L]$, and that for L_2, $(\forall y)_q[\theta(x,y) \in L]$. In the case of multiple quantifiers, the subscripting polynomials will all refer to bounds in terms of x; e.g., $(\exists y)_{q_1}(\forall z)_{q_2}[\theta(x,y,z) \in L]$ denotes $(\exists y)(\forall z)[|y| \leq q_1(|x|)$ and if $|z| \leq q_2(|x|)$ then $\theta(x,y,z) \in L]$.

The following characterization of the classes Σ_k and Π_k follows easily from Theorems 4.2.2 and 5.2.7. It was in this form--definition by the number of alternations of bounded quantifiers--that the polynomial hierarchy was suggested by Karp [36].

Proposition 5.3.10. Let $L \subseteq S^*$ be a language. For any $k \geq 1$: $L \in \Sigma_k$ if and only if there is a language $L' \in$ DTIME(poly) and a sequence of k polynomials $p_1,\ldots,p_k$ such that for all $x \in S^*$

$x \in L \iff (\exists y_1)_{p_1}(\forall y_2)_{p_2}\cdots(Q\ y_k)_{p_k}[\theta(x,y_1,\ldots,y_k) \in L']$.

Dually, $L \in \Pi_k$ if and only if

$x \in L \iff (\forall y_1)_{p_1}(\exists y_2)_{p_2}\cdots(Q'y_k)_{p_k}[\theta(x,y_1,\ldots,y_k) \in L]$

for some $L' \in$ DTIME(poly) and polynomials $p_1,\ldots,p_k$. □

The quantifiers in these expressions alternate, so that if k is odd then Q is $\exists$ and Q' is $\forall$, and if k is even then Q is $\forall$ and Q' is $\exists$. This result was announced, without proof, in [53]. Note that in the implication from left to right it is sufficient to take $L' \in$ DTIME(lin), and the polynomials $p_2,\ldots,p_k$ can all be the function $p_i(n) = n$.

The form of the expressions in Proposition 5.3.10 recalls another sequence of complete sets for the polynomial hierarchy. Suppose propositional formulas containing variables from the set $\{x\langle i,j\rangle : i,j \geq 1\}$ are encoded as strings over some finite alphabet. For each $k \geq 1$ let B_k be a certain set of such strings, defined as follows. The encoding of a formula F is in B_k is and only if

$(\exists \underline{X}^1)(\forall \underline{X}^2)\ \ldots\ (Q\ \underline{X}^k)\ [F(\underline{X}^1,\ldots,\underline{X}^k)$ is true] where the variables in F are exactly $\underline{X}^1 = \{x\langle 1,1\rangle,\ldots,x\langle 1,n_1\rangle\}$, ... , $\underline{X}^k = \{x\langle k,1\rangle,\ \ldots\ ,x\langle k,n_k\rangle\}$

for some $n_1, n_2, \ldots, n_k \geq 1$, and where (as above) Q is $\exists$ if k is odd and $\forall$ if k is even.

Thus B_1 encodes the set of satisfiable formulas; as remarked in [35], the proof of Cook [16] (see also [1]) can be used to show that B_1 is $\leq^{lg}$-complete in NTIME(poly) = Σ_1. In general, B_k is $\leq^{lg}$-complete in Σ_k, $k \geq 1$ [53]. Using Cook's result for the basis, the general case follows by induction on k. The induction step can be done by "relativizing" the proof for k = 1 to oracle machines (as in [53]) or, more easily, by making use of the following fact, which can be derived from Theorems 5.2.7 and 4.1.4: for each $k \geq 1$, $\Sigma_{k+1} = \{h(L) : L \in \pi_k$, h a polynomial-erasing homomorphism$\}$ [56].

The family of "extended rudimentary relations" was suggested by Bennett [4, p. 67]: in terms of the definitions used here, it consists of those string relations that are definable from rudimentary relations by one application of polynomial-bounded existential quantification. From Theorem 5.2.7 and the fact that the union of the linear hierarchy is the rudimentary relations, we see that $PH = \{\theta(R) : R$ an extended rudimentary relation$\}$. The characterization given in Proposition 5.3.10 therefore shows that the extended rudimentary relations can also be defined as the smallest class containing the concatenation relations and closed under the Boolean operations, explicit transformation and polynomial-bounded existential quantification.

By using the proof technique of Corollary 5.3.8 it can be seen

that the class of extended rudimentary relations is not equal to any class DSPACE(n^r), $r > 0$. (In particular, it is not equal to $\mathcal{E}_*^2$ = DSPACE(n).) However, PSPACE = $\bigcup_r$ DSPACE(n^r) can be easily shown to contain *PH* (which is the extended rudimentary relations); the question of proper containment remains open. The class PSPACE possesses a complete set with respect to lg(n)-space reductions [53] so we obtain a result lifting Corollary 4.3.8 to the polynomial hierarchy.

Proposition 5.3.11. If the polynomial hierarchy is infinite then the extended rudimentary relations are properly contained in PSPACE (and the rudimentary relations are properly contained in DSPACE(n)). □

Recall that no class in the linear hierarchy can be equal to any family of languages that is closed under removal of polynomial padding. Hence, in particular, for all $j,k \geq 1$, $\sigma_j \neq \Sigma_k$; that is, for $k < j$, $\sigma_j \neq \Sigma_k$ and for $k \geq j$, $\sigma_j \subsetneq \Sigma_k$. Now, if the linear hierarcy is finite, then for some k, $RUD = \sigma_k$, and the polynomial hierarchy is also finite. Thus we can conclude the following.

Corollary 5.3.12. If the linear hierarchy is finite then the rudimentary relations are properly contained in the extended rudimentary relations. □

Two other classes of string relations defined by Bennett [4] are of interest in connection with the linear and polynomial hierarchies. Recall from Chapter 4 that the class of positive rudimentary relations is the smallest class of string relations containing the concatenation relations and closed under union, intersection, explicit transformation, bounded existential quantification and universal subpart quantification. A relation is an extended positive rudimentary relation if it can be obtained from a positive rudimentary relation by one application of polynomial-bounded existential quantification; as noticed by Cobham [15], the class of extended positive rudimentary relations is in fact the class NTIME(poly).

It is not hard to see that the positive rudimentary relations are contained both in the rudimentary relations and in the extended positive rudimentary relations [4]. Whether either containment is proper is not known, nor is any inclusion relation known to hold between the rudimentary relations and the extended positive rudimentary relations (i.e., between *RUD* and NTIME(poly)). If *RUD* is in fact equal to the positive rudimentary relations then the linear hierarchy is all contained in NTIME(poly) and so (from Corollary 5.3.3) *PH* = NTIME(poly). The same conclusion can be drawn if *RUD* is contained in the extended positive rudimentary relations.

As remarked in Chapter 4, σ_1 = NTIME(n) is contained in the positive rudimentary relations but this inclusion is not known to hold for classes in the linear hierarchy with larger indices. If even δ_2 is contained in the positive rudimentary relations then also $\delta_2 \subseteq$ NTIME(poly) so (again from Corollary 5.3.3) every class in the polynomial hierarchy is contained in NTIME(poly).

We have seen in this chapter that a strong and useful relationship

exists between the linear and polynomial hierarchies. Questions that remain open about the polynomial hierarchy can be solved if the corresponding questions about the linear hierarchy have certain solutions (e.g., if the linear hierarchy collapses at a class then the polynomial hierarchy collapses at the corresponding class.) Properties desired for the classes in the polynomial hierarchy need only be proven in the context of the linear hierarchy (that is, only linear time bounds, rather than polynomials, need be considered); they can then be lifted to the polynomial hierarchy. Thus, for example, to show that a language $L_0 \in \Sigma_k$ is complete in Σ_k with respect to polynomial-time reductions, it is sufficient to show that any language in σ_k is polynomial-time reducible to L_0.

Appendix A: DISCUSSION OF THE RELATIVIZED TIME-HIERARCHY THEOREM

This appendix contains further discussion of the time hierarchy theorems for oracle machines that were stated in Chapter 2.

Theorem 2.4.1. Suppose A is a recursive language and $t_2(n)$ is a running time.

(1) If $\lim_{n\to\infty} \dfrac{t_1(n)\, \lg(t_1(n))}{t_2(n)} = 0$ then $\mathrm{DTIME}(t_2(n),A) \not\subseteq \mathrm{DTIME}(t_1(n),A)$.

(2) If $\lim_{n\to\infty} \dfrac{t_1(n+1)}{t_2(n)} = 0$ then $\mathrm{NTIME}(t_2(n),A) \not\subseteq \mathrm{NTIME}(t_1(n),A)$.

We concentrate on the proof for the nondeterministic case. For the deterministic case, first note that by applying the construction of [29] to all the tapes of an oracle machine except the oracle tape, it can be established that $\mathrm{DTIME}(t_1(n),A) \subseteq \{M(A)$: M is a deterministic oracle machine with 3 tapes that operates in time linear in $t_1(n)\lg(t_1(n))\}$ for any language A. Using changes similar to those described below, the diagonalization proof of [28] can then be modified to apply to oracle machines. This yields the result stated in (1) for any language A (not only recursive A).

The nondeterministic case of Theorem 2.4.1 follows from a more general result (stated below), a relativized version of the theorem of Seiferas for Turing acceptors [49, Theorem 13]. The differences between the proof for Turing machines and that for

oracle machines will be emphasized in the description of the proof.

To simplify the proof we consider <u>off-line</u> oracle machines. These machines differ from (on-line) oracle machines only in that the input tape serves as a Turing tape; that is, an off-line oracle machine can read its input tape in both directions and can write on it. As with off-line Turing machines, the reading head of the input tape is initially positioned at the left end of the input. The mechanism for an off-line oracle machine to query its oracle is the same as for an on-line oracle machine; in particular, the oracle tape is reset to blanks after an oracle call. Also as in the on-line case, an off-line oracle machine M is said to operate in a time bound $t(n)$ if for any oracle set A for M and any input x, every computation of M on x relative to A has at most $t(|x|)$ steps.

Theorem A.1. Suppose $t_2(n)$ is a running time and A is a recursive language. Let $\mathcal{B} = \{t_1 : \mathbb{N} \to \mathbb{N} : t_1(n) \geq n$ for all n and for some recursively bounded, strictly increasing function f, $\lim_{n\to\infty} \dfrac{t_1(f(n+1))}{t_2(f(n))} = 0\}$. Then $\{M(A)$: M is a nondeterminstic off-line oracle machine that operates in time $t_2(n)\}$ − $\{M(A)$: M is a nondeterministic off-line oracle machine that operates in time $t_1(n)$ for some $t_1 \in \mathcal{B}\}$ is nonempty.

Theorem 2.4.1 (2) is easily derived from this theorem. First note that for any language A and time bound $t(n)$, $\mathrm{NTIME}(t(n),A) =$

$\{M(A)$: M is an off-line oracle machine that operates in time $t(n)\}$. The containment from left to right follows from the definitions. On the other hand, an off-line oracle machine M can be converted to an (on-line) oracle machine without loss of time, by the addition of two pushdown stores to act jointly as the input tape for M. Now if $\lim_{n\to\infty} \frac{t_1(n+1)}{t_2(n)} = 0$ then $t_1 \in \mathcal{B}$ for the recursive, strictly increasing function $f(n) = n$; hence Theorem A.1 may be applied to yield a contradiction if Theorem 2.4.1 (2) is assumed to be false.

The proof of Theorem A.1 has the same structure as the proof of the corresponding theorem for Turing acceptors [49, p. 23]. For any alphabet Σ, a particular nondeterminstic oracle machine U_2 is constructed satisfying: (i) U_2 operates in time $t_2(n)$; and (ii) for any recursive language $A \subseteq \Sigma^*$, $U_2(A)$ is not equal to $U_1(A)$ for any off-line oracle machine U_1 that operates in a time bound in $\mathcal{B}$. In describing the proof of the relativized version, it will be assumed that the reader is familiar with Seiferas's proof [49, pp. 21-28]. The changes necessary are due to the larger alphabet involved ($A \subseteq \Sigma^*$) and to additional linear factors in the timing of the machines.

Suppose M is an off-line oracle machine and C is an oracle set for M. Extending the notation of [49], for $x \in M(A)$ let $\mathrm{Time}_{M,C}(x)$ be the length of a shortest accepting computation of M on x relative to C. For $x \notin M(A)$, let $\mathrm{Time}_{M,C}(x) = \infty$;

i.e., $m \leq \mathrm{Time}_{M,A}(x)$ is true for every integer m.

The proof uses a "universal simulator" with certain properties, so we establish an encoding to describe off-line oracle machines as strings for input to that machine. Suppose $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ is an alphabet with $\{0,1,¢\} \subseteq \Sigma$. Let a four-tape off-line oracle machine M with input alphabet $\{0,1,¢\}$ and tape alphabet Σ be given as a 9-tuple $M = (K, \{0,1,¢\}, \Sigma, \delta, q_0, q_?, q_{yes}, q_{no}, F)$ where we assume that the first tape is the input tape and the last tape is the oracle tape. As usual, K is the set of states of M, $F \subseteq K$ is the set of accepting states, $q_0 \in K$ is the initial state, $q_? \in K$ is the query state and q_{yes}, $q_{no} \in K$ are the response states. The four distinguished states $(q_0, q_?, q_{yes}, q_{no})$ are required to be distinct. The set δ of transitions of M is a subset of

$$[(K-\{q_?\}) \times \{0,1,¢,B\} \times (\Sigma \cup \{B\})^3] \times [K \times (\{0,1,¢,B\} \times \{L,R,C\}) \times ((\Sigma \cup \{B\}) \times \{L,R,C\})^3]$$

with the usual interpretation. ($B \notin \Sigma$ is the blank tape symbol; "L", "R", and "C" instruct the machine to move that head to the left, to the right and not at all, respectively.) The string $\bar{M} \in \{0,1,¢\}^*$ describing M is constructed as follows. Let K be a (finite) subset of $\{p_1, p_2, \ldots\}$ and define $\bar{p}_i$ to be i written in binary. For $1 \leq j \leq m$, let $\bar{\sigma}_j = 01^{j+1}0$ and let $\bar{B} = 010$. Let $\bar{L} = 10$, $\bar{R} = 01$ and $\bar{C} = 00$. Suppose $t = (q,a_1,a_2,a_3,a_4,p,b_1,d_1,\ldots,b_4,d_4)$ is a transition in δ with $q \in K-\{q_?\}$, $p \in K$, $a_1, b_1 \in \{0,1,¢,B\}$, $a_2,\ldots,a_4$, $b_2,\ldots,b_4 \in \Sigma \cup \{B\}$, $d_1,\ldots,d_4 \in \{L,R,C\}$. Then $\bar{t} = ¢\bar{q}¢\bar{a}_1¢\ldots¢\bar{a}_4¢\bar{p}¢\bar{b}_1¢\bar{d}_1¢\ldots\bar{b}_4¢\bar{d}_4¢$. If $t_1,\ldots,t_r$ are the transitions of M and $f_1,\ldots,f_p$ are its accepting states, then

$\bar{M} = ¢¢¢\bar{q}_0¢\bar{q}_?¢\bar{q}_{yes}¢\bar{q}_{no}¢\bar{t}_1 \ldots \bar{t}_r¢\bar{f}_1¢\bar{f}_2¢ \ldots \bar{f}_p¢¢¢$.

Let $L^4_{p.c.} = \{\bar{M}$: M is a four-tape off-line oracle machine with input alphabet $\{0,1,¢\}$ and tape alphabet $\Sigma\}$. If $e \in L^4_{p.c.}$ then M_e will denote the off-line oracle machine M such that $\bar{M} = e$. To simplify the notation, for $e \in L^4_{p.c.}$ let $Time_{e,C}(x)$ deonte $Time_{M_e,C}(x)$, i.e., the length of a shortest accepting computation of M_e on x relative to c (if one exists).

The set $L^4_{p.c.}$ of program codes is easily seen to satisfy the following conditions, analogs of the conditions in [49].

<u>Condition 1</u>. $L^4_{p.c.}$ is prefix-free and can be recognized in linear time by a determinstic on-line Turing machine.

<u>Condition 2</u>. There is a nondeterministic off-line oracle machine U_0 such that for any $C \subseteq \Sigma^*$

$$U_0(C) = \{ex : e \in L^4_{p.c.},\ x \in \{0,1,¢\}^*,\ x \in M_e(C)\}$$

and for any $e \in L^4_{p.c.}$ there is a constant c_e such that for any C and x

$$Time_{U_0,C}(x) \le c_e \cdot Time_{e,C}(x).$$

<u>Condition 3</u>. There is a recursive function $f_4 : L^4_{p.c.} \to L^4_{p.c.}$ such that for any $e \in L^4_{p.c.}$, $M_{f_4(e)}$ spends $|e|$ steps writing e (backwards) on its second tape and then acts according to the rules of M_e.

Except for references to oracle machines rather than Turing acceptors, only Condition 1 differs from the statement in [49] : linear time rather than real-time seems necessary to check that no transitions begin with the query state. Since the alphabet Σ is

fixed, while U_0 is simulating a computation of M_e on x, it can use its oracle tape (and three others) just as M_e would during that computation.

<u>Lemma A.2</u>. Suppose M is a four-tape off-line oracle machine with input alphabet $\{0,1,¢\}$ and tape alphabet Σ. Then there is an index $e_0 \in L^4_{p.c.}$ such that for any $C \subseteq \Sigma^*$

$$M_{e_0}(C) = \{x \in \{0,1,¢\}^* : e_0 x \in M(C)\}$$

and there is a constant c such that for any C and x

$$Time_{e_0,C}(x) \le c + Time_{M,C}(x). \quad \square$$

The following lemma, essential to the proof of the theorem, has a slightly weaker statement for oracle machines than for Turing acceptors (see [49, Lemma 4]).

<u>Lemma A.3</u>. Let M_1, M_2 be off-line oracle machines with the same input alphabet. One can construct an off-line oracle machine M such that (i) for any oracle set C, $M(C) = M_1(C) \cup M_2(C)$; and (ii) there is a constant d_0 such that for any C and x,

$$Time_{M,C}(x) \le d_0 \cdot \min\{Time_{M_1,C}(x),\ Time_{M_2,C}(x)\}. \quad \square$$

The linear factor in the timing of M arises from oracle calls made by M_1 and M_2. The steps of M_1 and M_2 in which neither queries its oracle can be run by M in "parallel", as with Turing acceptors. However, if (say) M_1 wishes to query its oracle, the contents of the tape serving as the oracle tape for M_1 must be copied by M onto its oracle tape before the call can be made. Since the oracle tape of a machine is reset to blanks after a query, the total length of all strings that must be copied by M onto its oracle tape will be less than twice the number of non-copying (or "parallel") steps M takes; therefore only a linear (rather than quadratic) time loss results.

The argument of Corollary 2.3.4 can be modified to yield a final lemma.

Lemma A.4. If M_1 is an off-line oracle machine then one can construct a nondeterministic four-tape off-line oracle machine M_2 with the same alphabet such that (i) for any oracle set C for M_1, $M_2(C) = M_1(C)$; and (ii) there is a constant c such that for any C and x,

$$\text{Time}_{M_2,C}(x) \le c\cdot\text{Time}_{M_1,C}(x). \qquad \square$$

Both of the following contribute to the linear factor c: the action of M_2 in quessing and writing down an entire computation of M_1 (and then following it); and recoding the extra symbols used by M_2 into the original tape alphabet of M_1.

We can now proceed with the proof of Theorem A.1. Suppose $t_2(n)$ is a running time. Let U_0 be the universal oracle machine of Condition 2 above. Let an oracle machine U_2 be constructed from U_0 by adding a timer for $t_2(n)/2$; then U_2 operates in time $t_2(n)$. (The timer cannot run during steps of U_2 that are oracle calls, but during any initial segment of a computation of U_0 the number of steps that are not oracle calls must exceed the number of oracle calls.) It will be shown that for any recursive $A\subseteq\Sigma^*$, $U_2(A)$ cannot be accepted relative to A by any off-line oracle machine that operates in time $t_1(n)$ for any $t_1\in\mathcal{B}$.

Assume to the contrary that $t_1\in\mathcal{B}$ and $U_2(A) = U_1(A)$ for U_1 an off-line oracle machine that operates in time $t_1(n)$. Let f be the function that demonstrates that t_1 is in $\mathcal{B}$: $\lim_{n\to\infty} \frac{t_1(f(n+1))}{t_2(f(n))} = 0$. Let U be the oracle machine constructed from U_0 and U_1 as in Lemma A.3. Then $U(A) = U_0(A)\cup U_1(A) = U_0(A)$ and there is a constant d_0 such that for any $e\in L^4_{p.c.}$ and $x\in\{0,1,¢\}^*$,

$$\text{Time}_{U,A}(ex) \le d_0\cdot\min\{\text{Time}_{U_0,A}(ex),\ \text{Time}_{U_1,A}(ex)\}$$
$$\le d_0\cdot\min\{c_e\cdot\text{Time}_{e,A}(x),\ \text{Time}_{U_1,A}(ex)\}.$$

Suppose $e\in L^4_{p.c.}$ and $x\in M_e(A)$. If $c_e\cdot\text{Time}_{e,A}(x)\le t_2(|ex|)/2$ then U_2 accepts ex relative to A, so $ex\in U_1(A)$ and therefore $\text{Time}_{U_1,A}(ex) \le t_1(|ex|)$. Hence if $c_e\cdot\text{Time}_{e,A}(x)\le t_2(|ex|)/2$ then $\text{Time}_{U,A}(ex)\le d_0\cdot t_1(|ex|)$.

The machine U is now used to show that any recursive language over {0,1} can be accepted relative to A within a fixed recursive time bound. Combining this with Proposition 2.1.5 (since A is recursive) we can conclude that for some recursive function h, NTIME(h(n)) contains all the recursive sets, a contradiction. Hence $U_2(A)$ cannot be accepted relative to A in time $t_1(n)$ and the theorem is proved.

So suppose $L\subseteq\{0,1\}^*$ is any recursive language and let M be a Turing acceptor for L that operates in time t(n) for some running time t(n). An oracle machine M' is constructed from M and U just as in [49, p. 24]. M' rejects any input not in $L^4_{p.c.}\cdot\{0,1\}^*\cdot\{¢\}^*$.

On an input $e x ¢^k$ ($e \in L^4_{p.c.}$, $x \in \{0,1\}^*$, $k \geq 0$) M' operates as follows:

1) if $k \geq t(|x|)$ then M' acts like M would act on input x; and

2) if $k < t(|x|)$ then M' quesses some value $k' > k$ and then acts like U on $e x ¢^{k'}$.

Thus $e x ¢^k \in M'(A)$ if and only if either (1) $k \geq t(|x|)$ and $x \in L$; or (2) $k < t(|x|)$ and there exists $k' > k$ such that $x ¢^{k'} \in M_e(A)$. Using Condition 1 and the fact that t(n) is a running time, there is some constant d_1 such that for $e x ¢^k \in M'(A)$

$$\mathrm{Time}_{M',A}(e x ¢^k) \leq \begin{cases} d_1 \cdot |e x ¢^k| & \text{if } k \geq t(|x|) \\ \min\limits_{k'>k} \{d_1 \cdot |e x ¢^{k'}| + \mathrm{Time}_{U,A}(e x ¢^{k'})\} & \text{if } k < t(|x|). \end{cases}$$

Applying Lemma A.4 and then Lemma A.2 to M', there is a program code e_0 and a constant d_2 such that

$$M_{e_0}(A) = \{x ¢^k : x \in \{0.1\}^*,\ k \geq 0,\ e_0 x ¢^k \in M'(A)\}$$

and for any $x \in \{0,1\}^*$ and $k \geq 0$

$$\mathrm{Time}_{e_0,A}(x ¢^k) \leq d_2 \cdot \mathrm{Time}_{M',A}(e_0 x ¢^k).$$

The following three claims can be established as in [49, pp.25-28].

Claim 1. For all $x \in \{0,1\}^*$ and $k \geq 0$, $x ¢^k \in M_{e_0}(A)$ if and only if $x \in L$.

Claim 2. For every sufficiently long string $x \in L$, for every $n \geq |e_0 x|$

$$\mathrm{Time}_{e_0,A}(x ¢^{f(n) - |e_0 x|}) \leq d_3 \cdot t_1(f(n+1))$$

where $d_3 = d_2 \cdot (d_1 + d_0)$.

(In this proof, x is chosen long enough that $c_{e_0} \cdot d_3 \cdot t_1(f(n+1)) \leq t_2(f(n))/2$ for every $n \geq |e_0 x|$.)

Claim 3. For every sufficiently long string $x \in L$,

$$\mathrm{Time}_{e_0,A}(x) \leq d_3 \cdot t_1(f(|e_0 x| + 1)).$$

An oracle machine M" can be easily constructed from M_{e_0} to accept L relative to A; using Claim 3, $\mathrm{Time}_{M'',A}(x) \leq d_3 \cdot t_1(f(|e_0 x| + 1))$ for any $x \in L$. Since f is bounded above by a recursive function, this implies (as in [49, p. 28]) that there is a recursive function h_1 such that $L \in \mathrm{NTIME}(h_1(n),A)$, leading to the desired contradiction.

Appendix B: <u>CHARACTERIZATIONS OF THE DYCK SETS</u>

Two representations of Dyck sets in terms of simpler languages are given here. First, the Dyck set on two letters is shown to be definable from the Dyck set on one letter by used of language-theoretic operations; the method generalizes easily to the Dyck set on k letters for any $k \geq 2$. Second, the Dyck set on one letter is expressed in terms of the language $L_0 = \{0^n 1^n\colon\ n \geq 0\}$. In both cases complementation (i.e., difference with regular sets) is used as well as some of the AFL operations; use of complementation is necessary.

For completeness we restate the definition of the Dyck sets. Let $\Sigma_1 = \{a_1, \bar{a}_1\}$ and $\Sigma_2 = \{a_1, a_2, \bar{a}_1, \bar{a}_2\}$. Define the binary relation $\sim$ on Σ_2^* as follows: for any $u, v \in \Sigma_2^*$, $u a_1 \bar{a}_1 v \sim uv$ and $u a_2 \bar{a}_2 v \sim uv$. Let $\overset{*}{\sim}$ denote the reflexive and transitive closure of $\sim$; that is, $x \overset{*}{\sim} y$ if and only if $x = y$ or for some $n \geq 1$ and $z_1, \ldots, z_n \in \Sigma_2^*$, $x \sim z_1 \sim z_2 \sim \ldots \sim z_n \sim y$. Then $D_2 = \{x \in \Sigma_2^*\colon\ x \overset{*}{\sim} e\}$ and $D_1 = \{x \in \Sigma_1^*\colon\ x \overset{*}{\sim} e\}$. Two properties of the Dyck sets are apparent:

(1) no string in D_1 begins with $\bar{a}_1$ or ends with a_1; and

(2) for $i = 1,2$, for any x and y, if $x \sim y$ then $x \in D_i$ if and only if $y \in D_i$.

Let $h\colon\ \Sigma_2^* \to \Sigma_1^*$ be the homomorphism determined by defining

$h(a_1) = h(a_2) = a_1$ and $h(\bar{a}_1) = h(\bar{a}_2) = \bar{a}_1$. It is easy to see that $h(D_2) = D_1$, so that $D_2 \subseteq h^{-1}(D_1)$; moreover, we will see that the difference between D_2 and $h^{-1}(D_1)$ can be expressed in terms of $h^{-1}(D_1)$.

<u>Notation</u>. Let $A = (\Sigma_2^* a_1 (\Sigma_2^* - h^{-1}(D_1)\bar{a}_1 \Sigma_2^*) \cup$

$$(\Sigma_2^* a_2 (\Sigma_2^* - h^{-1}(D_1)\bar{a}_2 \Sigma_2^*).$$

Note that for $x \in \Sigma_2^*$, $x \notin A$ if and only if whenever $x = u a_i v$ for $i = 1$ or 2 then $v \in h^{-1}(D_1)\bar{a}_i \Sigma_2^*$; that is, $x \notin A$ if and only if for every occurrence of a_i in x there is a "matching" occurrence of $\bar{a}_i$.

The language A contains the strings which are in $h^{-1}(D_1)$ but not in D_2. To see this, we first prove two lemmas about the language $h^{-1}(D_1) - A$.

<u>Lemma B.1</u>. For any $x, y \in \Sigma_2^*$, if $x \sim y$, then $x \in h^{-1}(D_1) - A$ if and only if $y \in h^{-1}(D_1) - A$.

<u>Proof</u>. Suppose $x \sim y$. Then by definition, $x = u a_j \bar{a}_j v$ for some u and v, and $y = uv$. Then $h(x) = h(u) a_1 \bar{a}_1 h(v) \sim h(u) h(v) = h(y)$, so $h(x) \in D_1$ if and only if $h(y) \in D_1$. It remains to show that $x \in A$ if and only if $y \in A$.

(1) If $x \in A$ then for some $i = 1$ or 2, and some u_1, v_1, $x = u_1 a_i v_1$ and $v_1 \notin h^{-1}(D_1)\bar{a}_i \Sigma_2^*$. Now if $u_1 = u$ then $i = j$ and

$v_1 = \bar{a}_i v \in h^{-1}(D_1)\bar{a}_i \Sigma_2^*$; therefore $u_1 \neq u$. Two cases remain:

(i) $|u_1| < |u|$: Then $u = u_1 a_i u_2$ for some u_2 and $v_1 = u_2 a_j \bar{a}_j v$. Now $h(u_2 a_j) = h(u_2)a_1 \notin D_1$, so it is not hard to see that if $u_2 v \in h^{-1}(D_1)\bar{a}_i \Sigma_2^*$ then also $v_1 \in h^{-1}(D_1)\bar{a}_i \Sigma_2^*$, a contradiction. Since $y = u_1 a_i (u_2 v)$, $y \in A$.

(ii) $|u_1| > |u|$: Then $u_1 = u a_j \bar{a}_j u_2$ for some u_2 so $y = u u_2 a_i v_1$. Since $v_1 \notin h^{-1}(D_1)\bar{a}_i \Sigma_2^*$, $y \in A$.

(2) If $y \in A$ then $y = u_1 a_i v_1$ and $v_1 \notin h(D_1)\bar{a}_i \Sigma_2^*$. Again there are two cases, $|u_1| < |u|$ and $|u_1| \geq |u|$; using arguments similar to those above, it can be seen that $x \in A$. □

Lemma B.2. For any $x \neq e$ in Σ_2^*, if $x \in h^{-1}(D_1) - A$ then $x = u a_i \bar{a}_i v$ for some $u, v \in \Sigma_2^*$ and $i = 1$ or 2.

Proof. Suppose $x = x_1 \ldots x_n$ is in Σ_2^* with $n \geq 1$, $x_i \in \Sigma_2$ for $1 \leq i \leq n$. Let $f\colon \Sigma_2^* \to \mathbb{Z}$ (where $\mathbb{Z}$ denotes the integers) be the homomorphism determined by defining $f(a_i) = 1$ and $f(\bar{a}_i) = -1$, $i = 1,2$. Let $m = \max\{f(x_1 \ldots x_i)\colon 1 \leq i \leq n\}$ and $k = \min\{j\colon 1 \leq j \leq n,\ f(x_1 \ldots x_j) = m\}$; that is, k is the leftmost position in x at which the maximum depth m is achieved. Since $h(x) \in D_1$ and every nonempty string in D_1 begins with a_1, $m > 0$. Let $u = x_1 \ldots x_{k-1}$ (that is, if $k = 1$, then $u = e$). By the choice of k, $f(x_1 \ldots x_k) \geq f(u)$, so x_k is either a_1 or a_2, say a_1. Since

every nonempty string in D_1 ends with $\bar{a}_1$, $k < n$, so let $v = x_{k+2} \ldots x_n$. Then $x = u a_1 x_{k+1} v$ and since $x \notin A$, $x_{k+1} v \in h^{-1}(D_1)\bar{a}_1 \Sigma_2^*$. Since $f(x_1 \ldots x_k) = m \geq f(x_1 \ldots x_{k+1})$, x_{k+1} is a "barred" symbol, either $\bar{a}_1$ or $\bar{a}_2$. But no string in $h^{-1}(D_1)$ can begin with a "barred" symbol so in fact $x_{k+1} = \bar{a}_1$ and $x = u a_1 \bar{a}_1 v$. □

Recall that A was defined from $h^{-1}(D_1)$ by use of Boolean operations and product with regular sets. Thus the following proposition gives a definition of D_2 from D_1.

Proposition B.3. $D_2 = h^{-1}(D_1) - A$.

Proof. The proof is by induction on $|x|$ for $x \in \Sigma_2^*$. For the basis step, $|x| = 0$, note that $e \in D_2$, $e \in h^{-1}(D_1)$ but $e \notin A$. For the induction step Lemmas B.1 and B.2 are used along with the fact that when $x \sim y$, $x \in D_2$ if and only if $y \in D_2$.

Proposition B.3 can be rephrased as follows: a string $x \in \Sigma_2^*$ is in D_2 if and only if

(1) $h(x) \in D_1$; and

(2) whenever $x = u a_i v$ $(i = 1,2)$, there is a string $w \in h^{-1}(D_1)$ such that $w\bar{a}_i$ begins v (i.e., is an initial substring of v).

Descriptions of D_2 similar to this one have been used to construct automata which accept D_2.

Corollary B.4. (i) ([46]) $D_2 \in$ DSPACE(lg(n)).

(ii) ([31]) D_2 can be accepted by a deterministic two-way one-counter automaton. □

The automaton given by Ritchie and Springsteel [46] is a device with two-way (read-only) input and has for storage counters which are bounded by the length of the input, so there is a lg(n) tape-bounded Turing machine that accepts the same set. The counter of the device given by Hotz and Messerschmidt [31] is also bounded by the length of the input. Both automata operate by checking condition (2) above for each symbol a_i $(i = 1,2)$ in the input $x = ua_iv$, using a counter to determine which initial substrings of v are elements of $h^{-1}(D_1)$. They also check that every symbol $\bar{a}_i$ in x has a "matching" symbol a_i to its left; it is not hard to see, however, that if conditions (1) and (2) are satisfied by x (i.e., if $x \in h^{-1}(D_1) - A$) then x also satisfies:

(3) whenever $x = u\bar{a}_iv$ $(i = 1,2)$ there exists $w \in h^{-1}(D_1)$ such that a_iw ends u.

Once it is established that the Dyck set on one letter is a rudimentary relation (see Proposition B.9), Proposition B.3 can be used to show that the Dyck set on two letters is rudimentary and, hence, every context-free language is a rudimentary relation. For the purpose of showing the context-free languages to be rudimentary, other definitions of D_2 have been given by Jones [34] and Yu [57]. The characterization given by Jones is similar in form to the restatement above of Proposition

B.3. For $b \in \Sigma_2$ and $w \in \Sigma_2^*$, let $\#_b(w)$ denote the number of occurrences of the symbol b in w. For $w \in \Sigma_2^*$, define w to be balanced if $\#_{a_1}(w) = \#_{\bar{a}_1}$ and $\#_{a_2}(w) = \#_{\bar{a}_2}(w)$. Then Jones's representation of D_2 may be stated as follows:

$x \in D_2$ iff (1') x is balanced;

(2') whenever $x = ua_iv$ $(i = 1$ or $2)$ there is a balanced string w such that $w\bar{a}_i$ begins v; and

(3') whenever $x = u\bar{a}_iv$ $(i = 1$ or $2)$ there is a balanced string w such that a_iw ends u.

Note that $h^{-1}(D_1)$ and the set of balanced strings are incomparable. It is not clear whether condition (3') can be omitted. The characterization of the Dyck sets suggested by Yu can be stated more easily using language-theoretic operations. Let $f_1\colon \Sigma_2^* \to \Sigma_1^*$ and $f_2\colon \Sigma_2^* \to \Sigma_1^*$ be the homomorphisms (similar to the homomorphism h of Proposition B.3) determined by defining for $i = 1,2$, $f_i(a_i) = a_1$, $f_i(\bar{a}_i) = \bar{a}_1$ and $f_i(a_j) = f_i(\bar{a}_j) = e$ for $j \neq 1$, $j = 1,2$. Define a language $K \subseteq \Sigma_2^*$ by

$$K = \Sigma_2^* a_1(f_1^{-1}(D_1) \cap f_2^{-1}(D_1))\bar{a}_2 \Sigma_2^* \cup \Sigma_2^* a_2(f_1^{-1}(D_1) \cap f_2^{-1}(D_1))\bar{a}_1 \Sigma_2^*.$$

Then $D_2 = [f_1^{-1}(D_1) \cap f_2^{-1}(D_1)] - K$. The language $f_1^{-1}(D_1) \cap f_2^{-1}(D_1)$ is properly contained both in $h^{-1}(D_1)$ and in the set of balanced strings. Recognition of D_2 using this representation seems to require two counters.

The operation of product with regular sets was used to define A from $h^{-1}(D_1)$. The following lemma shows that closure of a class of languages under product with regular sets is implied by closure under intersection with regular sets, inverse homomorphism and length-preserving homomorphism.

Lemma B.5. Suppose S is an alphabet, $L, L_1, L_2 \subseteq S^*$ and L_1 and L_2 are regular sets. Then there exist homomorphisms h_1, h_2, with h_1 length-preserving, and a regular set R such that $L_1LL_2 = h_1(h_2^{-1}(L) \cap R)$.

Proof. Suppose $L, L_1, L_2 \subseteq S^*$ are languages as in the statement of the lemma. Let $T = \{\bar{a}: a \in S\}$ be an alphabet isomorphic to S, with $S \cap T = \emptyset$. For $i = 1,2$, let $R_i = \{\bar{a}_1 \ldots \bar{a}_n: n \geq 0, a_1 \ldots a_n \in L_i\}$; since L_1 and L_2 are regular so are R_1 and R_2. Let $R \subseteq (S \cup T)^*$ be the regular set $R = R_1S^*R_2$. Let $h_1: (S \cup T)^* \to S^*$ and $h_2: (S \cup T)^* \to S^*$ be the homomorphisms determined by defining, for $a \in S$, $h_1(a) = h_1(\bar{a}) = a$, $h_2(a) = a$ and $h_2(\bar{a}) = e$. Note that h_1 is a length-preserving homomorphism. Then $L_1LL_2 = h_1(h_2^{-1}(L) \cap R)$. □

The following fact is easily proven using Proposition B.3 and Lemma B.5.

Proposition B.6. If $\mathcal{C}$ is a class of languages containing D_1 and closed under intersection, difference with regular sets, inverse homomorphism and length-preserving homomorphism, then $D_2 \in \mathcal{C}$. □

Recall that the closure of D_1 under the AFL operations is the class of nondeterministic one-counter languages [22], which does not contain D_2; hence the operation of complementation is necessary.

Now we turn to the representation of D_1 in terms of $L_0 = \{0^n1^n: n \geq 0\}$. For $w \in \{0,1\}^*$ let $\#_0(w)$ denote the number of occurrences of 0 in w, and $\#_1(w)$, the number of occurrences of 1 in w. Let $L_1 = \{w \in \{0,1\}^*: \#_0(w) = \#_1(w)\}$. We first define D_1 from L_1 using language operations, in a representation similar to that of Prop. B.3, and then define L_1 from L_0.

Let $h_1: \Sigma_1^* \to \{0,1\}^*$ be the homomorphism determined by defining $h_1(a_1) = 0$ and $h_1(\bar{a}_1) = 1$, so that $x \in h_1^{-1}(L_1)$ if and only if $\#_{a_1}(x) = \#_{\bar{a}_1}(x)$. Define $B = \{x \in \Sigma_1^*:$ for some prefix y of x, $\#_{a_1}(y) < \#_{\bar{a}_1}(y)\}$. Then the language B contains those strings which are in $h_1^{-1}(L_1)$ but not in D_1:

Proposition B.7. $D_1 = h_1^{-1}(L_1) - B$. □

The proof of this equality is essentially the same as the proof of Proposition B.3, using facts about $h_1^{-1}(L_1) - B$ similar to Lemmas B.1 and B.2. Proposition B.7 may be restated as: for $x \in \Sigma_1^*$, $x \in D_1$ if and only if

(1) $\#_{a_1}(x) = \#_{\bar{a}_1}(x)$; and

(2) for every prefix y of x, $\#_{a_1}(y) \geq \#_{\bar{a}_1}(y)$.

Corollary B.8. If $\mathcal{C}$ is a class of languages containing L_1 and closed under intersection, difference with regular sets, inverse homomorphism and length-preserving homomorphism, then D_1 (and hence every Dyck set) is in $\mathcal{C}$.

Proof. Recalling Lemma B.5, it is sufficient to show that B can be defined from L_1 by use of inverse homomorphism, length-preserving homomorphism and intersection and product with regular sets.

Let $h_2\colon \{0,1,¢\}^* \to \{0,1\}^*$ and $h_3\colon \{0,1,¢\}^* \to \Sigma_1^*$ be the homomorphisms determined by defining $h_2(0) = 0$, $h_2(1) = 1$ and $h_2(¢) = e$, and $h_3(0) = a_1$ and $h_3(1) = h_3(¢) = \bar{a}_1$. Note that h_3 is length-preserving. Let R be the regular set $R = \{0,1,¢\}^* \{¢\}\{0,1,¢\}^*$. Then $h_3(h_2^{-1}(L_1) \cap R) = \{x \in \Sigma_1^*\colon \#_{a_1}(x) < \#_{\bar{a}_1}(x)\}$; hence $B = (h_3(h_2^{-1}(L_1) \cap R))\ \Sigma_1^*$. □

We will now see that L_1 can be defined from L_0 and regular sets by use of inverse homomorphism, length-preserving homomorphism and union and intersection. The operation of complementation need not be used, because L_1 can be accepted in linear time by a deterministic automaton with two counters, each of which makes only one turn during any computation. Therefore there exist two one-turn one-counter languages L_1' and L_1'' such that L is the image under a linear-erasing homomorphism of $L_1' \cap L_1''$. Recall also that L_0 generates the one-turn one-counter languages under the AFL operations. The algebraic definition of L_1 from L_0 (which reduces the linear-erasing homomorphism to a length-preserving

homomorphism) is based on these ideas.

Let $C_1 = \{e\} \cup \{u¢v¢w\colon u,v,w \in \{0,1\}^*,\ \#_0(u) = \#_0(vw),\ \#_1(uv) = \#_1(w)$ and $\#_0(u) = \#_1(w) \geq 1\}$. Note that if $x = u¢v¢w$ is in C_1 then $\#_0(x) = \#_1(x)$, $\#_0(x)$ is even, and the two occurrences of $¢$ in x mark the positions in x where half the 0's and half the 1's in x have occurred. Similarly, let $C_2 = \{u¢v¢w\colon u,v,w \in \{0,1\}^*,\ \#_0(u)+1 = \#_0(vw),\ \#_1(uv)+1 = \#_1(w)$ and $\#_0(u)+1 = \#_1(w) \geq 1\}$. Let $g_1\colon \{0,1,¢\}^* \to \{0,1,¢\}^*$ be the homomorphism that interchanges 0's and 1's: $g_1(0) = 1$, $g_1(1) = 0$ and $g_1(¢) = ¢$. Let $C_3 = C_1 \cup g_1(C_1) \cup C_2 \cup g_1(C_2)$. Let $g_2\colon \{0,1,¢\}^* \to \{0,1\}^*$ be the homomorphism defined by $g_2(0) = 0$, $g_2(1) = 1$ and $g_2(¢) = e$. Then $L_1 = g_2(C_3)$. Since any word in C_3 has at most two occurrences of the symbol $¢$, g_2 is e-limited on C_3. The effect of an e-limited homomorphism can be achieved by use of length-preserving homomorphism, inverse homomorphism and intersection with a regular set [21, p.44], so it suffices to show that C_1 and C_2 can be formed from L_0.

C_1 is the intersection of three one-turn one-counter languages, each of which checks one of the conditions on the number of symbols in a word. Let $C_4 = \{u¢v¢w\colon \#_0(u) = \#_0(vw) \geq 1\}$, $C_5 = \{u¢v¢w\colon \#_1(uv) = \#_1(w) \geq 1\}$, and $C_6 = \{u¢v¢w\colon \#_0(u) = \#_1(w) \geq 1\}$; then $C_1 = \{e\} \cup (C_4 \cap C_5 \cap C_6)$. It is not hard to see that C_4, C_5 and C_6 are inverse a-transducer mappings [20] of L_0, hence can be defined from L_0 by use of length-preserving homomorphism, inverse homomorphism and intersection with regular sets. We give the definition only for C_4; that for C_5 is essentially the same, and that for C_6 is simpler. Define four homomorphisms as follows:

r_1: $\{0,1,\$\}^* \to \{0,1\}^*$ $\quad r_1(0) = 0,\ r_1(1) = 1,\ r_1(\$) = e$

r_2: $\{0,1,\$\}^* \to \{0,\$\}^*$ $\quad r_2(0) = r_2(1) = 0,\ r_2(\$) = \$$

r_3: $\{0,1,¢,\$\}^* \to \{0,1,\$\}^*$ $\quad r_3(0) = 0,\ r_3(1) = r_3(¢) = e,\ r_3(\$) = \$$

r_4: $\{0,1,¢,\$\}^* \to \{0,1,¢\}^*$ $\quad r_4(0) = 0,\ r_4(1) = 1,\ r_4(¢) = r_4(\$) = ¢.$

Let R_1 and R_2 be the regular sets:

$R_1 = \{0^m\$1^n:\ m,n \geq 1\}$

$R_2 = \{0,1\}^*\{\$\}\{0,1\}^*\{¢\}\{0,1\}^*.$

Then

$r_1^{-1}(L_0) \cap R_1 = \{0^n\$1^n:\ n \geq 1\};$

$r_2(r_1^{-1}(L_0) \cap R_1) = \{0^n\$0^n:\ n \geq 1\};$

$r_3^{-1}(r_2(r_1^{-1}(L_0) \cap R_1)) \cap R_2 = \{u\$v¢w:\ u,v,w \in \{0,1\}^*,\ \#_0(u) = \#_0(vw) \geq 1\};$

and

$r_4(r_3^{-1}(r_2(r_1^{-1}(L_0) \cap R_1)) \cap R_2) = C_4.$

The demonstration that C_2 can be formed from L_0 is similar.

The preceding discussion is summarized in the following proposition.

Proposition B.9. If C is a class of languages containing $\{0^n1^n:\ n \geq 0\}$ and closed under intersection, difference with regular sets, inverse homomorphism and length-preserving homomorphism then every Dyck set is in C. □

Again, the operation of complementation cannot be deleted, since the closure of L_0 under the AFL operations is properly contained in the family of context-free languages, and hence does not contain the Dyck sets.

Using the algebraic characterizations of the context-free languages and of the class NTIME(n), it can be seen that for any class C satisfying the conditions of Proposition B.9, the context-free languages are properly contained in C and NTIME(n) is contained in C.

REFERENCES

1. A. Aho, J. Hopcroft and J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

2. B. Baker and R. Book, Reversal-bounded multipushdown machines, J. Computer and Systems Science 8(1974), 315-332.

3. T. Baker, J. Gill and R. Solovay, Relativizations of the P =? NP question, SIAM J. Computing, to appear.

4. J. H. Bennett, On Spectra, Ph.D. dissertation, Princeton University, 1962.

5. R. Book, On languages accepted in polynomial time, SIAM J. Computing 1(1972), 281-287.

6. R. Book, Comparing complexity classes, J. Computer and Systems Science 9(1974), 213-229.

7. R. Book, Translational lemmas, polynomial time and $(\log n)^j$-space, Theoretical Computer Science 1(1975), to appear.

8. R. Book, On the Chomsky-Schutzenberger Theorem, Technical Report, Department of Computer Science, Yale University, 1975.

9. R. Book and S. Greibach, Quasirealtime languages, Math. Systems Theory 4(1970), 97-111.

10. R. Book, S. Greibach and B. Wegbreit, Time- and tape-bounded Turing acceptors and AFLs, J. Computer and Systems Science 4(1970), 606-621.

11. R. Book, S. Greibach, B. Wegbreit and O. Ibarra, Tape-bounded Turing acceptors and principal AFLs, J. Computer and Systems Science 4(1970), 622-625.

12. R. Book and M. Nivat, Linear languages and the intersection closure of classes of languages, in preparation.

13. R. Book, M. Nivat and M. Paterson, Reversal-bounded acceptors and intersections of linear languages, SIAM J. Computing 3(1974), 283-295.

14. N. Chomsky and M. P. Schutzenberger, The algebraic theory of context-free languages, in P. Braffort and D. Hirschberg, eds., Computer Programming and Formal Systems, North-Holland, Amsterdam, 1970, 118-161.

15. A. Cobham, The intrinsic computational difficulty of functions, Proc. 1964 Congress for Logic, Math., and Phil. of Science, North-Holland, Amsterdam, 1964, 24-30.

16. S. Cook, The complexity of theorem-proving procedures, Proc. Third ACM Symp. on Theory of Computing (1971), 151-158.

17. P. Fischer, A. Meyer and A. Rosenberg, Real-time simulation of multihead tape units, J. ACM 19(1972), 590-607.

18. S. Ginsburg, The Mathematical Theory of Context-free Languages, McGraw-Hill, New York, 1966.

19. S. Ginsburg and S. Greibach, Principal AFL, J. Computer and Systems Science 4(1970), 308-338.

20. S. Ginsburg and S. Greibach, Abstract families of languages, Memoir 87, American Mathematical Society, Providence, R. I., 1969, 1-32.

21. S. Ginsburg, S. Greibach and J. Hopcroft, Pre-AFL, Memoir 87, American Mathematical Society, Providence, R. I., 1969, 41-51.

22. S. Greibach, An infinite hierarchy of context-free languages, J. ACM 16(1969), 91-106.

23. S. Greibach, Erasing in context-free AFLs, Inf. and Control 31 (1972), 436-465.

24. S. Greibach, The hardest context-free language, SIAM J. Computing 2(1973), 304-310.

25. A. Grzegorczyk, Some classes of recursive functions, Rozprawy Matematyczne IV(1953), 1-46.

26. J. Hartmanis, Context-free languages and Turing machine computations, Proc. Symp. Applied Math. 19(1967), 42-51.

27. J. Hartmanis and J. Hopcroft, What makes some language theory problems undecidable, J. Computer and Systems Science 4(1970), 368-376.

28. J. Hartmanis and R. Stearns, On the computational complexity of algorithms, Trans. Amer. Math. Soc. 117(1965), 285-306.

29. F. Hennie and R. Stearns, Two-tape simulation of multitape machines, J. ACM 13(1966), 533-546.

30. J. Hopcroft and J. Ullman, Formal Languages and Their Relation to Automata, Addison-Wesley, Reading, Mass., 1969.

31. G. Hotz and J. Messerschmidt, Dyck-Sprachen sind in Bandkomplexitat log n analysierbar, Technical Report, Universitat des Saarlandes, 1975.

32. O. Ibarra, On two-way multihead automata, J. Computer and Systems Science 7(1973), 28-36.

33. N. Jones, Formal Languages and Rudimentary Attributes, Ph.D. dissertation, University of Western Ontario, London, Canada, 1967.

34. N. Jones, Context-free languages and rudimentary attributes, Math. Systems Theory 3(1969), 102-109.

35. N. Jones, Reducibility among combinatorial problems in log n space, Proc. Seventh Annual Princeton Conf. on Information Sciences and Systems (1973).

36. R. Karp, Reducibility among combinatorial problems, in R. Miller and J. Thatcher, eds., Complexity of Computer Computations, Plenum Press, 1972, 85-104.

37. R. Ladner, On the structure of polynomial time reducibility, J. ACM 22(1975), 155-171.

38. R. Ladner, N. Lynch and A. Selman, A comparison of polynomial time reducibilities, Theoretical Computer Science, to appear.

39. N. Lynch, Relativization of the Theory of Computational Complexity, Project MAC Technical Report 99, Massachusetts Institute of Technology, 1972.

40. A. Meyer and L. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, Conf. Record IEEE Thirteenth Annual Symp. on Switching and Automata Theory (1972), 125-129.

41. G. Myhill, Linear Bounded Automata, Wright Air Dev. Div. Technical Note 60-165, 1960.

42. V. A. Nepomnyashchii, Rudimentary interpretation of two-tape Turing computation, Kibernetika 6(1970), 29-35. Translated in Cybernetics, Dec. (1972), 43-50.

43. E. L. Post, Recursively enumerable sets of positive integers and their decision problems, in M. Davis, ed., The Undecidable, Raven Press, Hewlett, New York, 1965, 305-337.

44. W. V. Quine, Concatenation as a basis for arithmetic, Journal of Symbolic Logic 11(1946), 105-114.

45. R. Ritchie, Classes of predictably computable functions, Trans. Amer. Math. Soc. 106(1963), 139-173.

46. R. Ritchie and F. Springsteel, Language recognition by marking automata, Inf. and Control 20(1972), 313-330.

47. H. Rogers, Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967.

48. W. Savitch, Relationships between nondeterministic and deterministic tape complexities, J. Computer and Systems Science 4(1970), 177-192.

49. J. Seiferas, Nondeterministic Time and Space Complexity Classes, Project MAC Technical Report 137, Massachusetts Institute of Technology, 1974.

50. J. Seiferas, M. Fischer and A. Meyer, Refinements of the nondeterministic time and space hierarchies, Conf. Record IEEE Fourteenth Annual Symp. on Switching and Automata Theory (1973), 130-137.

51. R. Smullyan, Theory of Formal Systems, Annals of Mathematics Studies No. 47, Princeton University Press, 1961.

52. L. Stockmeyer, The Polynomial-Time Hierarchy, IBM Research Report RC-5379, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1975.

53. L. Stockmeyer and A. Meyer, Word problems requiring exponential time, Proc. Fifth ACM Symp. on Theory of Computing (1973), 1-9.

54. A. Turing, Systems of logic based on ordinals, in M. Davis, ed., The Undecidable, Raven Press, Hewlett, New York, 1965, 155-222.

55. B. Wegbreit, A generator of context-sensitive languages, J. Computer and Systems Science 3(1969), 456-461.

56. C. Wrathall, Complete sets and the polynomial hierarchy, in preparation.

57. Y. Yu, Rudimentary Relations and Formal Languages, Ph.D. dissertation, University of California, Berkeley, 1970.