LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-195

ON TIME-SPACE CLASSES AND THEIR RELATION TO THE THEORY OF REAL ADDITION

Anna R. Bruss

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

MIT/LCS/TR-195

ON TIME-SPACE CLASSES AND THEIR RELATION TO THE THEORY OF REAL ADDITION

by

Anna R. Bruss

March, 1978

This report was prepared in part with the support of National Science Foundation Grant no. MCS77-19764.

Massachusetts Institute of Technology
Laboratory for Computer Science

Cambridge Massachusetts 02139

ON TIME-SPACE CLASSES AND THEIR RELATION TO THE THEORY OF REAL ADDITION

by

Anna R. Bruss

Submitted to the Department of Electrical Engineering and Computer Science on February 28, 1978, in partial fulfillment of the requirements for the Degree of Master of Science.

ABSTRACT

A new lower bound on the computational complexity of the theory of real addition and several related theories is established: any decision procedure for these theories requires either space $2^{\epsilon n}$ or nondeterministic time $2^{\epsilon n^2}$ for some constant $\epsilon > 0$ and infinitely many n.

The proof is based on the families of languages TISP(T(n),S(n)) which can be recognized simultaneously in time T(n) and S(n) and the conditions under which they form a hierarchy.

Thesis Supervisor: Albert R. Meyer
Title: Professor of Electrical Engineering and Computer Science

Keywords: computational complexity, theory of real addition, time and space, log-space reducibility

To the memory of my mother, Gisela Bruss.

ACKNOWLEDGMENTS

I would like to thank my thesis supervisor, Professor Albert Meyer, for his guidance and inspiration throughout the preparation of this thesis. He proposed the main theorem and I am grateful for the many suggestions he made concerning the proof of it.

Furthermore I would like to thank Jeanne Ferrante and Karl Winklmann for their careful reading and helpful comments concerning this thesis, Bob Woodham for his patience in helping me edit it and Marilyn Matz for proofreading it.

I also want to thank my father, Johann Bruss, my sister Ellie and her husband Ron Saunders, and my friend Carol Schrager for their love and encouragement of my work.

TABLE OF CONTENTS

I. Introduction

We consider the computational complexity of the theory of real addition (Th<R,+>) and several related theories. Previous results provide the following bounds on the complexity of Th<R,+>:

1) Lower bound [FiR74]. Any decision procedure for Th<R,+> requires nondeterministic *time* $2^{O(n)}$ for infinitely many n.

2) Upper bound [FeR73]. Th<R,+> is decidable within *space* $2^{O(n)}$.

Because the precise relation between computation time and space remains unknown, there is an exponential discrepancy when upper and lower bounds are both expressed in terms of time or space alone. That is, the exponential lower bound (1) for time is only known to imply a linear space lower bound; the exponential upper bound (2) for space is only known to imply a double exponential upper bound for time.

In this thesis we improve the lower bound, showing in particular

Main Theorem: There is an $\epsilon > 0$ such that any decision procedure for Th<R,+> requires *either* more than space $2^{\epsilon n}$ *or* more than nondeterministic time $2^{\epsilon n^2}$ for infinitely many n.

Let (N)TISP(T(n),S(n)) be the family of languages recognizable by a (non)deterministic Turing machine which runs in time T(n) and space S(n) simultaneously for almost all n. The Main Theorem is equivalent to the assertion that Th<R,+> is not a member of $NTISP(2^{\epsilon n^2},2^{\epsilon n})$ for some $\epsilon > 0$.

We do not interpret the Main Theorem as suggesting the likelihood of an inherent time-space tradeoff among decision algorithms for Th<R,+>. The Theorem merely leaves open the possibility of such a tradeoff.

The Main Theorem applies to other theories such as monadic predicate calculus and exponentially bounded concatenation theory, all of which can be shown to be log-linear equivalent [SM73, STO74]. Recently Berman has observed that Th<R,+> is an example of a language complete under polynomial time reduction in what is essentially the class $Alt(2^n,n)$ of languages recognizable by alternating Turing machines using time 2^n and n alternations [BER77, CSTO76, KO76]. Our results imply that $NTISP(2^{n^2},2^n) \subset Alt(2^{O(n)},O(n))$, an observation which we interpret as supporting the conjecture that Berman's alternating machine complexity classes properly contain the languages recognizable in nondeterministic exponential time.

II. TIME-SPACE CLASSES

The basic computational model used is a deterministic or nondeterministic multitape Turing machine (DTM or NTM). It has a finite number of worktapes, each with a single read-write head which can move in both directions and a single input tape with a two-way read-only head. An accepting computation of a Turing machine M on input x is a computation of M which starts with the word x written on the input tape and the rest of the tapes blank, and terminates in an accepting state. The time of a computation is the number of steps in it; its space is the number of worktape squares visited during the computation (input tape squares not counted). By the linear speed-up theorem [HU69], it suffices to specify time and space bounds only to within a constant factor (e.g., it is unnecessary to specify the base of a logarithm). All time and space bounds are assumed to be positive valued functions on the positive integers.

Definition 1: Let T and S be functions from the positive integers to the positive integers. Then a (n)tisp(T,S)-machine is a (non)deterministic multitape Turing machine which on every input of length n computes for time at most T(n) and space at most S(n).

Remark: Both time and space bounds have to be observed by a single computation.

Definition 2: Let Σ be a finite alphabet. Then (N)TISP(T(n),S(n)) is the set of languages $A \subseteq \Sigma^*$ for which there exists a (n)tisp(T,S)-machine M such that for all $x \in \Sigma^*$ (where n denotes the length of x)

i) if $x \in A$ then there is an accepting computation of M on x,

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

We will show that under some familiar "honesty" conditions [GLI71, SFM73] upon T and S, TISP(NTISP) defines a hierarchy in the following sense: for small increases in the growth rate of T and S new languages can be accepted which could not be accepted before.

Definition 3: [SFM73] A function is *fully constructible* if there is a DTM M such that for each input of length n M halts in precisely space S(n) with the string $\#\not{b}^{S(n)-2}\#$ on one of its work tapes.

Definition 4: [SFM73] A function T(n) is a *running time* if there is a DTM M such that for each input of length n, the computation of M has precisely T(n) steps.

Definition 5: Two functions T(n) and S(n) are *compatible* if each of them is computable by a tisp(T,S)-machine.

Remark: If two functions T and S are compatible then T is a running time and S is fully constructible. It is a major open problem for which pairs of functions the converse holds.

Theorem 1: Let T_1 and S_1 and T_2 and S_2 be compatible functions respectively. If

$$\text{(I)}\quad T_1(n)\log(T_1(n)) = o(T_2(n)) \qquad \text{and}$$

$$\text{(II)}\quad S_1(n) = o(S_2(n)) \quad ,$$

then

$$\mathrm{TISP}(T_1(n),S_1(n)) \subsetneq \mathrm{TISP}(T_2(n),S_2(n)) \ .$$

Proof:

It is a well-known result that condition (I) suffices to show that DTIME($T_1(n)$) (i.e., the class of languages recognized by a DTM within time $T_1(n)$) is properly contained in DTIME($T_2(n)$). Likewise condition (II) suffices to obtain a similar result for deterministic space [HU69]. It is straightforward to combine these proofs to obtain the separation result for TISP. We omit the details. □

Theorem 2: Let $S_2(n) \geq \log(n)$ and let T_1 and S_1 and T_2 and S_2 be compatible respectively.

If

(I) $T_1(n+1) = o(T_2(n))$ and

(II) $S_1(n+1) = o(S_2(n))$,

then

$$NTISP(T_1(n),S_1(n)) \subsetneq NTISP(T_2(n),S_2(n)) .$$

Proof:

Condition (I) suffices to obtain a separation result for nondeterministic time classes whereas condition (II) is adequate to get a similar result for nondeterministic space classes [SFM73]. We will sketch how to combine the proofs of these results - assuming familiarity with the notation of [SFM73] - to obtain a proof of Theorem 2. The conditions for the program code (Appendix I in [SFM73]) are the same as for the time and space theorem (Theorem 1 and Theorem 2 in [SFM73]) . The universal simulator first lays off $S_2(n)$ squares and then behaves like the clocked version. Only in the case when $k \geq T(|x|)$ and $\log(k) \geq S(|x|)$ does the machine M' behave like the machine M. In all other cases it behaves like U_1. □

Basic for proving the Main Theorem is the notion of log-linear reducibility defined in [STO77].

<u>Lemma 1</u>:

Let $A \leq_{\text{log-lin}} B$, $T(n)$ and $S(n)$ be monotone nondecreasing functions. Then there is some polynomial p and some constant $c > 0$ such that

(i)

$$A \notin \left\{ \begin{array}{l} \mathrm{DTIME}(T(n)+p(n)) \\ \mathrm{DSPACE}(S(n)+\log(n)) \\ \\ \mathrm{NTIME}(T(n)+p(n)) \\ \mathrm{NSPACE}(S(n)+\log(n)) \end{array} \right. \quad \Longrightarrow B \notin \left\{ \begin{array}{l} \mathrm{DTIME}(T(cn)) \\ \mathrm{DSPACE}(S(cn)) \\ \\ \mathrm{NTIME}(T(cn)) \\ \mathrm{NSPACE}(S(cn)) \end{array} \right.$$

(ii)

$$A \notin \left\{ \begin{array}{l} \mathrm{TISP}(T(n)p(n),S(n)+\log(n)) \\ \\ \mathrm{NTISP}(T(n)p(n),S(n)+\log(n)) \end{array} \right. \quad \Longrightarrow B \notin \left\{ \begin{array}{l} \mathrm{TISP}(T(cn),S(cn)) \\ \\ \mathrm{NTISP}(T(cn),S(cn)) \end{array} \right.$$

For a proof of part (i) of this Lemma see [STO74]. Part (ii) can be shown in an analogous way.

III. THE THEORY OF REAL ADDITION

Let $\mathfrak{R}$ = <R,+> be the structure consisting of the set of all real numbers with the operation of addition. Let Th($\mathfrak{R}$) be the first order theory of $\mathfrak{R}$, i.e., the set of all first order sentences true in $\mathfrak{R}$.

As a technical tool for the proof of the Main Theorem as stated in the introduction we will use the first order theory of string concatenation and what we call t-bounded concatenation theory. Meyer [FMS76B] has shown that 2^n-bounded concatenation theory is log-lin reducible to Th($\mathfrak{R}$). We will show that NTISP(2^{n^2},2^n) is log-lin reducible to 2^n-bounded concatenation theory. The Main Theorem then follows immediately from Lemma 1, Theorem 2 and the transitivity of log-lin reducibility.

Definition 6: Let Σ be a finite set and let L(Σ) be the first order language with equality, with constants $\underline{\sigma}$ for each $\sigma \in \Sigma$, and whose only atomic formulae (other than equalities) are of the form $\underline{\text{cat}}$(x,y,z). The *elementary theory of concatenation*, CT(Σ), is the set of true sentenes in L(Σ) under the following interpretation: Σ^* is the underlying domain, the constant symbols denote the elements $\sigma \in \Sigma$, and for a,b,c $\in \Sigma^*$, $\underline{\text{cat}}$(a,b,c) is true iff a is the concatenation of b and c.

We assume that one of the standard formats is used for writing well formed formulae in CT(Σ) which are built up with propositional connectives and quantifiers as usual. The *length* of a formula is the number of symbols in the formula where subscripts are written in binary.

By bounding the length of strings in CT(Σ) (in a sense made precise in the following definition), we obtain bounded concatentation theory.

Definition 7: Let Σ be a finite set and let L(Σ) be the first order language with equality, with constants $\underline{\sigma}$ for each $\sigma \in \Sigma$, and whose only atomic formulae (other than equalities) are of the form $\underline{\mathrm{bcat}}(a,b,c,\underline{n})$. Then for any function $t : N \rightarrow N$, we define *t-bounded concatenation theory* (t-BCT(Σ)) as the set of true sentences in L(Σ) under the following interpretation: Σ^* is the underlying domain, the constant symbols denote the elements $\sigma \in \Sigma$, and for a,b,c $\in \Sigma^*$, $\underline{\mathrm{bcat}}(a,b,c,\underline{n})$ is true iff a is the concatenation of b and c and the length of a is smaller than or equal to t(n), where $\underline{n}$ is the unary numeral for the nonnegative integer n.

Remark: As $\underline{n}$ is written in unary the length of the atomic formula $\underline{\mathrm{bcat}}(a,b,c,\underline{n})$ is proportional to n plus the size of the variables a,b and c.

In reducing NTISP to bounded concatenation theory it is convenient to restrict the underlying computational model to be a "simple" one-tape Turing machine (STM) [STO77]. This can be done without loss of generality because an STM can simulate a multitape Turing machine with only a quadratic time loss and no space loss [HU69]. Furthermore, we assume that any move which shifts the head off the left end of the tape causes the STM to halt and reject the input.

In the reduction we will describe the computation of an STM with short formulae in 2^{3n}-BCT(Σ). Let M be an STM, let Q denote the set of its states and S its tape alphabet. An instantaneous description (i.d.) of M is any word in S^*QS^*. As in [STO77] we define the function $\mathrm{Next}_M : S^*QS^* \to 2^{S^*QS^*}$, where $\mathrm{Next}_M(d)$ is the set of i.d.'s that can occur one step after i.d. d. We remark here that Next_M is length preserving. It suffices to make "local checks" within i.d. d_1 and i.d. d_2 to decide if $d_2 \in \mathrm{Next}_M(d_1)$. The reason for this is that in one step only a few symbols around the state symbol can change.

<u>Lemma 2</u>: [STO77] Let M be an STM, $\$ \notin S \cup Q$, and $\Sigma = S \cup Q \cup \{\$\}$. There is a function $N_M : \Sigma^3 \to 2^{\Sigma^3}$ with the following properties:

Let d_1 be any i.d. of M, let k be the length of d_1 and suppose

$\$d_1\$ = d_{10}d_{11}d_{12}......d_{1k}d_{1,k+1}$ where $d_{1j} \in \Sigma$ for $0 \le j \le k+1$ and

$\$d_2\$ = d_{20}d_{21}d_{22}......d_{2k}d_{2,k+1}$ where $d_{2j} \in \Sigma$ for $0 \le j \le k+1$

then

$d_2 \in \mathrm{Next}_M(d_1)$ iff $d_{2,j-1}d_{2,j}d_{2,j+1} \in N_M(d_{1,j-1}d_{1,j}d_{1,j+1})$ for all $1 \le j \le k$

For a proof of Lemma 2 see [STO74]. Informally N_M specifies all possibilities of how the symbols of one i.d. can change in one step.

The classes 1-TISP and 1-NTISP are defined for STM's in the same way that TISP and NTISP were given in Definition 2 above for (n)tisp(T,S)-machines. Then the main lemma can be stated as following:

<u>Lemma 3</u>: $(\forall A \in 1\text{-NTISP}(2^{n^2}, 2^n))(\exists \Sigma)(A \le_{log\text{-}lin} 2^{3n}\text{-BCT}(\Sigma))$.

Proof:

Let M be a nondeterministic STM recognizing $A \subseteq \Theta^*$ simultaneously within time 2^{n^2} and space 2^n. Let Σ be the alphabet for M given in Lemma 2. For each $x \in \Theta^*$ we will describe a sentence S_x in 2^{3n}-BCT(Σ) which asserts that there is an accepting computation of M on input x. Thus $x \in A$ iff S_x is true in 2^{3n}-BCT(Σ). We will then observe that the function mapping x to S_x is computable in deterministic logspace and is linear bounded, viz., the length of S_x is at most proportional to the length of x. This will then complete the proof.

Let $n = |x|$. The computation to be described is 2^{n^2} steps long, thus a word consisting of a representation of the whole computation would be of length $2^{O(n^2)}$ and therefore too long to be expressed in the language of 2^{3n}-BCT(Σ). Instead we shall define the formula S_x based on the construction of the formula $P_{k,n}(z)$ which, for all integers k,n and for all $z \in \Sigma^*$, is true iff

1) z is a string of the form $\$z_1\$z_2\$...\$z_{2^n}\$$,

2) z_j represents an i.d. , $\quad 1 \le j \le 2^n$,

3) $|z_j| = 2^n+1$, $\quad 1 \le j \le 2^n$,

4) in some computation of M which is started in i.d. z_j the i.d. z_{j+1} can be reached in at most 2^{kn} steps using space at most 2^n , $1 \le j \le 2^n$.

The formulae $P_{k,n}(z)$ will be defined inductively. First we will write them in $CT(\Sigma)$ to clarify the idea underlying the construction of the appropriate formulae in 2^{3n}-$BCT(\Sigma)$.

As a notational convenience we will introduce some abbreviations for formulae in concatentation theory. Let $\Delta = \{\sigma_1,\sigma_2,\ldots,\sigma_k\}$, where $\sigma_i \in \Sigma$ for $1 \leq i \leq k$.

Abbreviation	Formula
$p = qr$	$cat(p,q,r)$
$p = qrs$	$(\exists x)(p = qx \wedge x = rs)$
$p \in \Delta$	$(p = \sigma_1) \vee \ldots. \vee (p = \sigma_k)$
$p \in \Delta^*$	$(\forall x,y,z)(p = xyz \wedge y \in \Sigma \rightarrow y \in \Delta)$
$p \subset q$	$(\exists x,y)(q = xpy)$

We also define for each $k \in N$ a formula $\ell_k(x)$ of concatenation theory which is true iff the length of x is equal to k. We define $\ell_k(x)$ inductively in such a way that the length of the formula itself is proportional to $\log(k)$ plus the length of the variable x.

$$\ell_1(x) \quad := \quad x \in \Sigma$$

$$\ell_{2k}(x) \quad := \quad (\exists y,z)(x = yz \wedge (\forall w)((w = y \vee w = z) \rightarrow \ell_k(w)))$$

$$\ell_{2k+1}(x) \quad := \quad (\exists y,z)(x = yz \wedge \ell_{2k}(y) \wedge \ell_1(z))$$

Furthermore we note that the new variables introduced in constructing ℓ_{2k}

from ℓ_k need only be distinct from each other and from the free variables of ℓ_k. Thus only a constant number of different variables is needed to construct ℓ_k.

The formula Form(z) will assert conditions 1) - 3) above. We use the convention that in every i.d. the state symbol q is positioned immediately to the left of the symbol being scanned.

$$\mathrm{Form}(z) := (\exists w)(z=\$w\$) \wedge \ell_{2^n(2^n+2)+1}(z) \wedge (\forall z_1)\{(\$z_1\$ \subset z \wedge \$ \not\subset z_1) \rightarrow$$
$$[\ell_{2^n+1}(z_1) \wedge (\exists w_1,w_2,q)(w_1,w_2 \in S^* \wedge q \in Q \wedge w_2 \neq \epsilon \wedge z_1 = w_1 q w_2\ )]\} \qquad (1)$$

As the induction base we will construct the formula $P_{0,n}(z)$ which satisfies the conditions 1) - 3) above and the conditions that each of the successive i.d.'s are either identical or follow in one step.

$$P_{0,n}(z) := \mathrm{Form}(z) \wedge (\forall z_1,z_2)[(\$z_1\$z_2\$ \subset z \wedge \$ \not\subset z_1 \wedge \$ \not\subset z_2) \rightarrow (\exists w_1,s_1,q,s_2,w_2,u)$$
$$(s_1,s_2 \in S \cup \{\$\} \wedge q \in Q \wedge \$z_1\$ = w_1 s_1 q s_2 w_2 \wedge z_2 = w_1 u w_2 \wedge u \in N_M(s_1 q s_2))] \qquad (2)$$

For the induction step we will write a formula $P_{k+1,n}(z)$ using $P_{k,n}(z)$ as a subformula. The basic idea is that i.d. z_{j+1} can be reached in $2^{(k+1)n}$ steps from i.d. z_j iff there is a string w which has z_j as a prefix, z_{j+1} as a suffix and for which $P_{k,n}(w)$ holds. Thus $P_{k+1,n}(z)$ can be written as :

$$P_{k+1,n}(z) := \mathrm{Form}(z) \wedge (\forall z_1,z_2)[(\$z_1\$z_2\$ \subset z \wedge \$ \not\subset z_1 \wedge \$ \not\subset z_2) \rightarrow$$
$$(\exists w_1)(P_{k,n}(\$z_1\$w_1\$z_2\$))] \qquad (3)$$

This completes the inductive construction of $P_{k,n}(z)$.

We remark here that the length of $P_{k+1,n}$ is equal to a constant plus the length of $P_{k,n}$ and the length of the formula Form. The formula Form is of length $O(n)$. Hence $P_{n,n}$ is of length $O(n^2)$ primarily because of the n occurrences of Form. However there is a standard "abbreviation trick" [RA75, FeR78] which allows n occurrences of subformulae which are the same - except for the name of the variables - to be replaced by single occurrences of n distinct variables and one occurrence of the subformula. Applying this abbreviation trick to $P_{n,n}$ would yield an equivalent formula $P'_{n,n}$ of length $O(n\log(n))$.

We wish now to construct a short formula b-$P_{n,n}(z)$ in the language of 2^{3n}-BCT(Σ) which is true iff conditions 1) - 4) as above hold. The straightforward way to obtain such a b-$P_{n,n}$ is to first rewrite $P'_{n,n}$ so that the formula <u>bcat</u> replaces each occurrence of <u>cat</u>. Since there are only proportional to n occurrences of <u>cat</u> in $P'_{n,n}$ and the length of <u>bcat</u> is $O(n)$, one could next apply the standard abbreviation trick on the multiple occurrences of <u>bcat</u> to obtain a formula b-$P_{n,n}$ which is also of length $O(n\log(n))$. This would be enough to prove a version of our Main Theorem with $2^{O(n^2/\log^2(n))}$ in place of $2^{O(n^2)}$ and $2^{O(n/\log(n))}$ in place of $2^{O(n)}$. In the paragraphs below we will give a slightly more complicated construction yielding a formula b-$P_{n,n}$ which is actually of length $O(n)$.

The idea of the construction is as follows: the formula $S_{k,n}(a,b,c,d,e,z)$ will mean the same as

$$\mathrm{bcat}(a,b,c,\underline{n}) \wedge \ell_{2^n+1}(d) \wedge \ell_{2^n(2^n+2)+1}(e) \wedge P_{k,n}(z).$$

Thus the formula $S_{k+1,n}(a,b,c,d,e,z)$ will be equivalent to

$$(\exists f,g,u)S_{k,n}(a,b,c,f,g,u) \wedge (\exists u)S_{k,n}(\epsilon,\epsilon,\epsilon,d,e,u) \wedge P_{k+1,n}(z)$$

(where ϵ denotes the empty string).

Now note that as in (3) $P_{k+1,n}(z)$ is equivalent to

$$\mathrm{Form}(z) \wedge (\forall z_1,z_2)[(\$z_1\$z_2\$ c z \wedge \$\notin z_1 \wedge \$\notin z_2) \rightarrow$$
$$(\exists w_1,f,g)(S_{k,n}(\epsilon,\epsilon,\epsilon,f,g,\$z_1\$w_1\$z_2\$))]. \qquad (4)$$

Similarly the formula Form(z) as in (1) is equivalent to

$$(\exists w)(z=\$w\$) \wedge (\exists f,u)S_{k,n}(\epsilon,\epsilon,\epsilon,f,z,u) \wedge (\forall z_1)\{(\$z_1\$ c z \wedge \$\notin z_1) \rightarrow$$
$$[(\exists g,u)S_{k,n}(\epsilon,\epsilon,\epsilon,z_1,g,u) \wedge (\forall w_1,w_2,q)(w_1,w_2\in \$^* \wedge q\in Q \wedge w_2\neq\epsilon \wedge z_1=w_1qw_2)]\} \qquad (5)$$

We observe now that the meaning of formulae equivalent to (4) and (5) does not change if each occurrence of the formula cat is replaced by the formula bcat as the length of all strings in (4) and (5) is bounded by 2^{3n}. Thus (4) and (5) can equivalently be written by replacing each occurrence of cat(p,q,r) by $(\exists f,g,u)S_{k,n}(p,q,r,f,g,u)$. So we can conclude that a formula $S'_{k+1,n}$ equivalent to $S_{k+1,n}$ can be written using a fixed number (independent of k and n) of copies of $S_{k,n}$ plus a fixed number of additional quantifiers, variables and logical connectives. We assume now that the reader is familiar with the technical details of the abbreviation trick and we merely summarize its application to $S'_{k+1,n}$. Applying the abbreviation trick to $S'_{k+1,n}$ yields the formula $S_{k+1,n}$ which has only one copy of $S_{k,n}$ as a subformula and a fixed number of quantifiers, variables and logical connectives. Again we note that no difficulty arises if the new variables introduced in constructing $S_{k+1,n}$ coincide with variables bound inside $S_{k,n}$ Thus only a constant number of additional variables are needed to construct $S_{k+1,n}$ from $S_{0,n}$. Therefore the length of $S_{k+1,n}$ is O(k+1) plus the length of $S_{0,n}$.

We will proceed now in constructing a formula $S_{0,n}(a,b,c,d,e,z)$ whose meaning is the same as bcat(a,b,c,$\underline{n}$) and $\ell_{2^n+1}(d)$ and $\ell_{2^n(2^n+2)+1}(e)$ and $P_{0,n}(z)$. As we want the length of $S_{0,n}$ to be proportional to n, we shall require a formula b-$\ell_{m,n}(a,b,c,d)$ written in the language of 2^{3n}-BCT(Σ) whose length is O(n) plus O(log(m)) plus the length of a,b,c and d and which means that bcat(a,b,c,$\underline{n}$) holds and that the length of d is m, where m is any integer $\leq 2^{3n}$. The construction of b-$\ell_{m,n}(a,b,c,d)$ is similar to the one for $\ell_m(d)$.

We henceforth use the same notational abbreviations as were introduced for formulae in CT(Σ) except that p = qr is an abbreviation for the formula bcat(p,q,r,$\underline{n}$).

$$\text{b-}\ell_{1,n}(a,b,c,d) \quad := \text{bcat}(a,b,c,\underline{n}) \wedge d \in \Sigma$$

$$\text{b-}\ell_{2m,n}(a,b,c,d) \quad := (\exists e,f)(\forall p,q,r,s)[(\langle p,q,r,s\rangle=\langle d,e,f,e\rangle \vee \langle p,q,r,s\rangle=\langle a,b,c,f\rangle) \rightarrow \text{b-}\ell_{m,n}(p,q,r,s)]$$

$$\text{b-}\ell_{2m+1,n}(a,b,c,d) := (\exists e,f)(\text{b-}\ell_{2m,n}(d,e,f,e) \wedge \text{b-}\ell_{1,n}(a,b,c,f))$$

By carefully reusing bound variables in the construction above, only a fixed number of distinct variables is needed. Thus the length of b-$\ell_{m,n}(a,b,c,d)$ is $O(n)$ plus $O(\log(m))$ plus the length of a,b,c and d. Note that therefore the lengths of the formulae b-$\ell_{2^n+1,n}(a,b,c,d)$ and b-$\ell_{2^n(2^n+2)+1,n}(a,b,c,d)$ are both proportional to n plus the length of a,b,c and d.

Now let b-$P_{0,n}(z)$ be the formula obtained from $P_{0,n}(z)$ as given in (2) by first replacing each of occurrence of $\ell_m(d)$ by b-$\ell_{m,n}(\epsilon,\epsilon,\epsilon,d)$ and then by substituting the formula bcat(p,q,r,$\underline{n}$) for each occurrence of the formula cat(p,q,r). As only a fixed number (independent of n) of copies of the formulae $\underline{\text{b-}\ell}_{2^n+1,n}$, $\underline{\text{b-}\ell}_{2^n(2^n+2)+1,n}$ and $\underline{\text{bcat}}$ (not including the $\underline{\text{bcat}}$'s inside $\underline{\text{b-}\ell_{m,n}}$) are needed to write the formula b-$P_{0,n}$, the length of b-$P_{0,n}(z)$ is proportional to n. Finally, we take $S_{0,n}(a,b,c,d,e,z)$ to be

$$\text{b-}\ell_{2^n+1,n}(a,b,c,d) \wedge \text{b-}\ell_{2^n(2^n+2)+1,n}(\epsilon,\epsilon,\epsilon,e) \wedge \text{b-}P_{0,n}(z) \text{ .}$$

Therefore the length of $S_{0,n}(a,b,c,d,e,z)$ is $O(n)$.

Thus we have shown how to construct a formula $S_{n,n}(a,b,c,d,e,z)$ in the language of 2^{3n}-BCT(Σ) whose meaning is the same as the conjunction of $b\text{-}\ell_{2^n+1,n}(a,b',c,d)$, $b\text{-}\ell_{2^n(2^n+2)+1,n}(\epsilon,\epsilon,\epsilon,e)$ and $b\text{-}P_{n,n}(z)$ and whose length is proportional to n.

Now to complete the construction of S_x we will need, in addition to $S_{n,n}$, a formula $IN_{x,n}(w)$ which is true iff w is the string x, Let $x_1,x_2,\ldots,x_n$ be the successive symbols in x. Then the straightforward way to write the formula $IN_{x,n}(w)$ uses n different variables and therefore its length would be nlog(n). Instead we will define a formula $I_{x,n}(a,b,c,w)$ whose meaning is the same as the conjunction of $IN_{x,n}(w)$ and $bcat(a,b,c,\underline{n})$, such that the length of $I_{x,n}$ is O(n).

$I_{\epsilon,n}(a,b,c,w) := bcat(a,b,c,\underline{n}) \wedge w = \epsilon$.

For $u\epsilon\Sigma^*$, $\sigma\epsilon\Sigma$, we define

$$I_{u\sigma,n}(a,b,c,w) := (\exists w_1)(\forall p,q,r,s)[(\langle p,q,r,s\rangle=\langle w,w_1,\sigma,w_1\rangle \vee \langle p,q,r,s\rangle=\langle a,b,c,w_1\rangle) \rightarrow I_{u,n}(p,q,r,s)]$$

Again we note that $I_{x,n}$ can be constructed using a fixed number of distinct variables.

Finally let S_x be the following formula, where q_0 denotes the initial state, q_a the accepting state and ƀ the blank tape symbol.

$$S_x := (\exists w,b,z,u)[I_{x,n}(\epsilon,\epsilon,\epsilon,w) \wedge b\epsilon\{ƀ\}^* \wedge \$q_0wb\$q_au\$cz \wedge S_{n,n}(\epsilon,\epsilon,\epsilon,q_au,z,z)] \quad .$$

Clearly $x \in A$ iff S_x is (true) in 2^{3n}-BCT(Σ). We have already shown that the function mapping x to S_x is linear bounded. The results of [SM73, LIN74] may be used to show that the computation of S_x can be carried out within deterministic logspace; we leave the verification of this final claim to the reader. Hence the transformation of x to S_x implies that $A \leq_{\text{log-lin}} 2^{3n}$-BCT($\Sigma$).

Remark: For any $c > 1$ and any alphabet Σ there exists an alphabet Θ such that 2^{cn}-BCT(Σ) is log-lin reducible to 2^n-BCT(Θ).

Lemma 3 and the preceding remark, together with the reduction of 2^n-BCT(Σ) to Th<R,+> completes the proof of the Main Theorem.

IV. OPEN PROBLEMS

In this thesis we classified logical theories with respect to both computation time and space. The basic open question remaining is to characterize the complexity of Th<R,+> (or equivalently Alt(2^n,n)) more precisely in terms of time and space. Note that the claims that Alt(2^n,n) is equivalent to NTIME(2^n) or equivalent to SPACE(2^n), or both for that matter, remain consistent with our Main Theorem.

A second related open problem is to improve the known lower bounds on the complexity of Presburger Arithmetic. Such improvements do not follow directly by the same method used to bound Th<R,+>, as can easily be seen by parameterizing our main result. We have shown that for $f(n)=2^n$, the class NTISP($f(n)^n$,$f(n)$) reduces to Th<R,+>. The same proof shows only that NTISP($g(n)^n$,$g(n)$) reduces to Presburger Arithmetic where $g(n)=2^{2^n}$, a result which degenerates to the known results [FIR74] that NTIME(2^{2^n}) reduces to Presburger Arithmetic.

We hope that the framework we have set up leads to a better understanding of the relation between the computational resources time and space.

BIBLIOGRAPHY

[BER77] Berman, L., "Precise Bounds for Presburger Arithmetic and the Reals with Addition," *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, Providence, Rhode Island, 1977.

[CSTO76] Chandra, A.K., and L.,J. Stockmeyer, "Alternation," *Proceedings of the 17th Annual Symposiom on Foundations of Computer Science*, Houston, Texas, 1976.

[FeR73] Ferrante, J., and C. Rackoff, "A Decision Procedure for the First Order Theory of Real Addition with Order," *MIT Project MAC TM 33*, 1973.

[FeR78] Ferrante, J., and C. Rackoff, *"The Computational Complexity of Logical Theories,"* Springer Verlag, 1978 (to appear).

[FIR74] Fischer, M.J., and M.O. Rabin "Super-Exponential Complexity of Presburger Arithmetic", *SIAM-AMS Proceedings Vol.VII*, *American Mathematical Society*, Providence, Rhode Island, 1974.

[FMS76A] Fleischmann, K., B. Mahr, and D., Siefkes, "Bounded Concatentation Theory as a Uniform Method for Proving Lower Complexity Bounds," Purdue University, CSD-TR202.

[FMS76B] Fleischmann, K., B. Mahr, and D. Siefkes, "Complexity of Decision Problems," Technische Universität Berlin, Bericht Nr.76-09, 1976.

[GLI71] Glinert, E.P., "On Restricting Turing Computability," Mathematical Systems Theory, Vol.5, No.4, Springer Verlag, 1971.

[HU69] Hopcroft, J.E., and J.,D. Ullman, *Formal Languages and their Relation to Automata*, Addison-Wesley, Reading, Massachusetts, 1969.

[KO76] Kozen, D., "On Parallelism in Turing Machines," *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*,Houston, Texas, 1976.

[LIN74] Lind, J.C., "Computing in Logarithmic Space," *MIT Project MAC TM 52*, 1974.

[RA75] Rackoff, C., "The Computational Complexity of Some Logical Theories," *MIT Project MAC TR 144*, 1975.

[SEI74] Seiferas, J.I., "Nondeterministic Time and Space Complexity Classes," *MIT Project MAC TR 137*, 1974.

[SFM73] Seiferas, J.I., M.J. Fisher, and A.R. Meyer, "Refinements of the Nondeterministic Time and Space Hierarchies," *Proceedings of the 14th Annual Symposium on Programming and Automata Theory*, Iowa City, Iowa, 1973.

[SM73] Stockmeyer, L.J., and A.R. Meyer, "Word Problems Requiring Exponential Time: Preliminary Report," *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, 1973.

[STO74] Stockmeyer, L.J., "The Complexity of Decision Problems in Automata Theory and Logic," *MIT Project MAC TR 133*, 1974.

[STO77] Stockmeyer, L.J., "The Polynomial Time Hierarchy," *Theoretical Computer Science*, Vol.3, North Holland Publishing Company, 1977.

APPENDIX: RELATION BETWEEN THE COMPLEXITY CLASSES $S_1(f(n),g(n))$ [BER77] AND $Alt(t(n),a(n))$

In his paper Berman [BER77] introduced a new complexity measure based on the specification of sets by bounded quantification of linear-time predicates.

<u>Definition 1</u>: [BER77] A set is in the complexity class $S_1(f(n),g(n))$ if there is a linear-time predicate $R(-)$ on strings such that

$$A=\{x \mid \exists y_1 \forall y_2 \ldots Qy_{g(|x|)}[R(x\#^{f(|x|)}y_1\#\ldots.\#y_{g(|x|)}) \wedge |y_1|<f(|x|) \wedge \ldots \wedge |y_{g(|x|)}|<f(|x|)]\}$$

Furthermore he observes that $Th\langle R,+\rangle$ is complete in $\bigsqcup_{k\geq 1} S_1(2^{kn},n)$ under a polynomial time reduction. We will show that the complexity measure S_1 is essentially the same as the measure Alt defined by:

<u>Definition 2</u>: A set is in the complexity class $Alt(t(n),a(n))$ if there is an alternating Turing machine [CSTO76] which accepts A within time $t(n)$ using at most $a(n)$ alternations.

Lemma 1:

Let $f(n)$ and $g(n)$ be computable in time $f(n)g(n)$ and $f(n) \geq n$. Then

$$S_1(f(n),g(n)) \subset Alt(f(n)g(n),g(n))$$

Proof:

Let $A \in S_1(f(n),g(n))$. To show that A is also a member of $Alt(f(n),g(n))$ we will describe a computation of an alternating Turing machine M which accepts A within time $f(n)g(n)$ using at most $g(n)$ alternations.

The proof is very similar to the one of Theorem 5 in [KO76] and we assume familiarity with the notions used there. Let $x \in A$, $n = |x|$. On input x M first writes $x\#^{f(n)}$ on its tape. Now it enters an existential state to write down $x\#^{f(n)}y_1$ (where $|y_1| \leq f(n)$). Then by changing into an universal state it writes down $x\#^{f(n)}y_1\#y_2$ for all y_2 with $|y_2| \leq f(n)$. It proceeds now alternating existential and universal states until $x\#^{f(n)}y_1\#\ldots\#y_{g(n)}$ is written on the tape. This can be done with at most $f(n)g(n)$ steps and $g(n)$ alternations. Now M checks the predicate $R(x\#^{f(n)}y_1\#\ldots\#y_{g(n)})$ and accepts iff $R(-)$ is true. As $R(-)$ is a linear-time predicate, M uses at most $f(n)g(n)$ steps to check it. As we can speed up the whole computation by a constant factor, M accepts A within time $f(n)g(n)$ using at most $g(n)$ alternations. □

Lemma 2:

Let $t(n) \geq n$. Then

$$Alt(t(n),a(n)) \subset S_1(t(n),a(n))$$

Proof:

Let A be a set accepted by an alternating Turing machine M within time $t(n)$ using at most $a(n)$ alternations. To simplify the following proof we assume (w.l.o.g.) that M has only one tape on which initially the input is written. Furthermore we adopt the convention that once M enters the accepting state q_a it keeps running in q_a. We also assume that the initial state is an existential state. We want to show that A is also in the class $S_1(t(n),a(n))$. To clarify the construction of the predicate R(-) as required in Definition 1 we will first show that A is in the class $S_1(t(n)^2,a(n))$.

Let $x \in A$ and $n = |x|$. A computation of M on x can be described by a sequence of strings $y_1,y_2,\ldots,y_{a(n)}$ where each y_i is a sequence of i.d.s all of which only contain states of one kind, universal if i is even, existential if i is odd. We define the predicate $R_1(x\#^{t(n)^2}y_1\#\ldots\#y_{a(n)})$ to be true iff $y_1\#y_2\#\ldots\#y_{a(n)}$ describes an accepting computation of M on input x (i.e., $y_1\#y_2\#\ldots\#y_{a(n)} = d_1\#d_2\#\ldots\#d_{t(n)}$ with d_1 being the initial i.d., $d_{t(n)}$ the accepting i.d. and for $1 \leq i \leq t(n)-1$ the i.d. d_{i+1} follows from i.d. d_i in one computational step) *or* there is an i.d. d_i which is a substring of some y_{2j} $1 \leq j \leq \lfloor t(n)/2 \rfloor$ and is not a successor i.d. of d_{i-1}.

It is now straightforward to verify that

$\{x \mid x \text{ accepted by } M\} =$

$\{x \mid \exists y_1 \forall y_2 \ldots Q y_{a(n)} [R_1(x\#^{t(n)^2} y_1\#\ldots\#y_{a(n)}) \wedge |y_1|<f(n) \wedge \ldots \wedge |y_{a(n)}|<f(n)]\}$

We remark only that without the second clause in the definition of R_1, the predicate $\exists y_1 \forall y_2 \ldots Q y_{a(n)} R_1(\text{-})$ is never true as the quantifiers range over *all* strings.

Now note that at most 2 symbols change between two consecutive i.d.s. These changes are determined by the next move function of M. Now let $u_1,\ldots,u_{t(n)}$ be a sequence of strings which describes a sequence of moves in a computation of M. That means that for each i, $1 \le i \le t(n)$, u_i is a string of the form pdq, where p denotes the symbol to be printed, d the direction of the move of the head and q the state to be entered. A computation of M contains at most a(n) alternations. Therefore up to t(n) consecutive moves correspond to situations where M does not change between universal and existential state and we will replace each such sequence by single variables w_j, $1 \le j \le a(n)$. As the length of the strings u_i is constant the length of the strings w_j is at most of order t(n). We will construct now a linear-time predicate $R(x\#^{t(n)} w_1\#\ldots\#w_{a(n)})$ which is true iff there is an accepting computation of M on x determined by u_1 through $u_{t(n)}$ *or* for some u_i which is part of some w_{2j}, $1 \le j \le \lfloor t(n)/2 \rfloor$ the following is true: u_i does not describe a legal move for the configuration obtained by applying the moves u_1 through u_{i-1} on input x.

The predicate $R(x\#^{t(n)} w_1\#\ldots\#w_{a(n)})$ can be computed in time linear in its input by the following straightforward procedure:

```
1) construct the initial i.d. from $x$
2) for each $i$ check if $u_i$ describes a legal move (this can be done by
   comparing $u_i$ against the 3-tuples determined by the transition function)
   if $u_i$ describes a legal move: update the current i.d.
   otherwise: halt and output true if $u_i$ is in part of some $w_j$ and $j$ is even
   otherwise halt and output false
3) check if the string $u_{t(n)}$ contains the symbol $q_a$.
```

Clearly the above procedure does not take more than $O(t(n))$ steps. □

CS-TR Scanning Project
Document Control Form

Date : 10/26/95

Report # LCS-TR-195

Each of the following should be identified by a checkmark:
Originating Department:

□ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) □ Technical Memo (TM)
□ Other:

Document Information

Number of pages: 31 (37-images)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
□ Double-sided

Intended to be printed as :
□ Single-sided or
☒ Double-sided

Print type:
□ Typewriter □ Offset Press □ Laser Print
□ InkJet Printer ☒ Unknown □ Other:

Check each if included with document:

□ DOD Form □ Funding Agent Form ☒ Cover Page
☒ Spine □ Printers Notes □ Photo negatives
□ Other:

Page Data:

Blank Pages(by page number):

Photographs/Tonal Material (by page number):

Other (note description/page number):

Description : Page Number:

IMAGE MAP! (1-31) UN#'ED TITLE PAGE, 2-31
(32-37) SCANCONTROL, COVER, SPINE, TRGT'S (3)

Scanning Agent Signoff:

Date Received: 10/26/95 Date Scanned: 11/15/95 Date Returned: 11/16/95

Scanning Agent Signature: Michael W. Cook

Rev 9/94 DS/LCS Document Control Form cstrform.vsd