THE EQUIVALENCE PROBLEM FOR REGULAR EXPRESSIONS WITH SQUARING REQUIRES EXPONENTIAL SPACE[†]

A.R. Meyer and L.J. Stockmeyer

Massachusetts Institute of Technology

Cambridge, Massachusetts

I. Introduction

There has been considerable interest recently in finding "natural" problems whose solutions require more than polynomial time. From the work of Hartmanis and Stearns [1] we know that given any arbitrarily large time bound T(n), there is a language recognition problem which requires time T(n) on some input of length n for all sufficiently large n. However, the diagonalization argument used to construct the language gives no insight into properties that the language possesses. In the first part of this paper we shall show that the problem of determining if two regular expressions describe the same set of strings requires exponential space (and hence exponential time) provided that the regular expressions can use a squaring abbreviation ($S^2 = S \cdot S$).

Closely related to the problem of finding non polynomial time languages is the question of whether or not nondeterministic polynomial time Turing machines can recognize a larger class of languages than deterministic polynomial time machines. The nondeterministic machine can be thought of as doing polynomial bounded quantification over deterministic polynomial time computable relations. In the second part of the paper we describe and give some simple properties of a "hierarchy" of languages. Each succeeding class of the hierarchy is obtained by allowing polynomial bounded quantification over relations in preceeding classes.

II. A Language Which Requires Exponential Time

The main goal of this section is to establish an exponential lower time bound for a certain language recognition problem. We will assume that the language recognition is being done by a one-tape one-head deterministic Turing machine (henceforth in this section called a machine) which, when started with an input x on its tape, eventually enters a designated accepting state if and only if x is in the language. Time and space bounds will be given as a function of n, the length of the input string. In the realm of exponential times, the one-tape one-head model is not restrictive since it can simulate multi-tape or random access models in time which is at most the square of the time required by the more powerful model.

The first language to be considered is the following.

Definition: Let Σ be a finite set. Define $RSQ(\Sigma) = \{\text{regular expressions with squaring } E \mid L(E) \neq \Sigma^*\}$ where L(E) denotes the language defined by the regular expression.

A regular expression with squaring may use the usual operations $\cup$, $\cdot$, *, as well as the squaring operation $S^2 = S \cdot S$.

Theorem 2.1: There is a finite set Σ such that if $\mathfrak{M}$ is any machine which recognizes $RSQ(\Sigma)$, then there is a constant $c > 1$ such that $\mathfrak{M}$ requires space (and hence time) c^n on some input of length n for infinitely many n.

Any procedure for determining equivalence of regular expressions with squaring can be used to decide RSQ, so that this lower bound also applies to the equivalence problem.

The method of proof will be to efficiently reduce the recognition of any language accepted by some $S(n) = 2^n$ space bounded machine to the recognition of $RSQ(\Sigma)$ for some Σ. The notion of efficient reduction is the following.

Definition: Let $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$ be two languages. We say that $L_1 \leq_{p\ell} L_2$ (L_1 is polynomial time linear space reducible to L_2) if there is a polynomial p(n), a constant $c > 0$, and a machine $\mathfrak{M}$ which, when started with any $x \in \Sigma_1^n$ on its tape, halts with some $y \in \Sigma_2^*$ on its tape such that

1) $x \in L_1 \Leftrightarrow y \in L_2$

2) $\mathfrak{M}$ carries out its computations within time p(n) and space cn (and hence $|y| \leq cn$).

The following simple fact relates the complexity of L_1 to the complexity of L_2.

Fact 1: If $L_1 \leq_{p\ell} L_2$ and if L_2 is accepted by some time T(n) and space S(n) machine, then L_1 is accepted by some time p(n) + T(cn) and space cn + S(cn) machine for some polynomial p and constant c.

For what follows, it will be useful to have a means of describing the computation of a space bounded Turing machine. Suppose Turing machine $\mathfrak{M}$ (possibly nondeterministic) accepts some language $L \subseteq \{0,1\}^*$ within space $S(n) \geq n$, has states Q and tape symbols T. Let $b \in T$ denote the blank tape symbol.

Definition: An instantaneous description (i.d.) of $\mathfrak{M}$ is a string in $(T \cup (Q \times T))^*$ which contains exactly one symbol in $Q \times T$. Given any i.d. $r = y \cdot (q,t) \cdot z$ for $y, z \in T^*$, the set of possible next i.d.'s $\text{Next}_{\mathfrak{M}}(r)$ is defined in the obvious way from the transition of rules of $\mathfrak{M}$. For example, if when $\mathfrak{M}$ is in state q scanning symbol t, $\mathfrak{M}$ can enter state q', write symbol t', and shift its head right, then

$$y \cdot t' \cdot (q',u) \cdot w \in \text{Next}_{\mathfrak{M}}(r)$$

[†]Work reported herein was supported by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number N00014-70-A-0362-0001. Reproduction in whole or in part is permitted for any purpose of the United States Government.

where $z = uw$, $u \in T$. $\text{Next}_{\mathfrak{M}}(r)$ is empty if (q,t) is a halting condition. Let $\# \notin T \cup (Q \times T)$. The set of accepting computations $C_{\mathfrak{M}}(x)$ of $\mathfrak{M}$ on input $x \in \{0,1\}^n$ is all sequences of i.d.'s

$$\# \text{ i.d.}_1 \# \text{ i.d.}_2 \# \ldots \# \text{ i.d.}_k \#$$

such that

1) Each i.d. is of length $S(n)$

2) $\text{i.d.}_1 = (q_0, x_1)\, x_2 \ldots x_n\, b^{S(n)-n}$

where $x = x_1 \ldots x_n$, $x_j \in \{0,1\}$, $j = 1, \ldots, n$ and q_0 is a starting state of $\mathfrak{M}$.

3) $\text{i.d.}_{j+1} \in \text{Next}_{\mathfrak{M}}(\text{i.d.}_j) \quad \forall j \geq 1$.

4) i.d._k contains the accepting state q_a.

We now show that a regular expression can, in a sense, simulate a space bounded computation.

Lemma 2.1: Let $L \subseteq \{0,1\}^*$ be any language accepted by a space $S(n) = 2^n$ bounded (in general nondeterministic) Turing machine $\mathfrak{M}$. Then there is a finite set Σ such that $L \leq_{p\ell} \text{RSQ}(\Sigma)$.

Proof: Let $\Sigma = \{\#\} \cup T \cup Q \times T$ where Q and T are as above. Given $x = x_1 \ldots x_n$, we will construct a regular expression with squaring E of length cn for some constant c.

If $x \notin L$ then $L(E) = \Sigma^*$.

If $x \in L$ then $L(E) = \Sigma^* - C_{\mathfrak{M}}(x)$.

L(E) will have the above property if L(E) contains the following sets of strings.

1) Strings which don't begin with

$$\#(q_0, x_1)\, x_2 \ldots x_n\, b^{2^n - n} \#$$

2) Strings which don't contain the accepting state q_a.

3) Strings which aren't of the form

$$\# \text{ i.d.}_1 \# \text{ i.d.}_2 \# \ldots \# \text{ i.d.}_k \#$$

with $\text{i.d.}_{j+1} \in \text{Next}_{\mathfrak{M}}(\text{i.d.}_j) \quad \forall j \geq 1$.

These sets of strings can be described by the expressions

1) $((\Sigma - \#) \cup \# \cdot ((\Sigma - (q_0, x_1)) \cup x_1 \cdot ((\Sigma - x_2) \cup x_2 \cdot ((\Sigma - x_3) \cup \ldots \cdot (\Sigma - x_n))) \ldots) \cdot \Sigma^*$
$\cup\ \Sigma^{n+1} \cdot b^* \cdot (\Sigma - b - \#) \cdot \Sigma^*$
$\cup\ \# \cdot (\Sigma \cup \lambda)^{2^n - 1} \cdot \# \cdot \Sigma^*$
$\cup\ \# \cdot \Sigma^{2^n} \cdot (\Sigma - \#) \cdot \Sigma^*$

where λ denotes the empty string.

2) $(\Sigma - (\bigcup_{t \in T} (q_a, t)))^*$

3) $\bigcup_{\sigma_1, \sigma_2, \sigma_3 \in \Sigma} (\Sigma^* \cdot \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdot \Sigma^{2^n - 1} \cdot (\Sigma - N(\sigma_1, \sigma_2, \sigma_3)) \cdot \Sigma^*)$

where $N(\sigma_1, \sigma_2, \sigma_3) \subseteq \Sigma$ is the set of symbols that could legally occupy the j^{th} symbol of $\text{Next}_{\mathfrak{M}}(r)$ given that the j-1, j, and $j+1^{\text{th}}$ symbols of r are σ_1, σ_2, and σ_3 respectively. Also $N(\sigma_1, \#, \sigma_3) = \{\#\} \quad \forall \sigma_1, \sigma_3 \in \Sigma$.

Set difference has been used above only to simplify the descriptions of certain sets whose size is independent of n.

These expressions are seen to be of length cn for some c once it is noted that for any $k \geq 1$ there is a regular expression with squaring of length d log k which describes Σ^k, for some constant d.

E is the union of the expression 1), 2), 3). The reader may verify that a Turing machine can carry out the construction in polynomial time and linear space. □

Now we use the following fact to find a language which requires space 2^n for all sufficiently large n.

Fact 2: Let $S_1(n)$, $S_2(n)$ be tape constructable functions such that

$$\inf_{n \to \infty} \frac{S_1(n)}{S_2(n)} = 0 \quad \text{and} \quad S_2(n) \geq \log n.$$

Then there is a language $L \subseteq \{0,1\}^*$ accepted by some space $S_2(n)$ machine but by no space $S_1(n)$ machine.

Proof of Theorem 2.1: Let L be as in Fact 2 with $S_2(n) = 2^n$. Let Σ be such that $L \leq_{p\ell} \text{RSQ}(\Sigma)$.

If $\text{RSQ}(\Sigma)$ is accepted by some space S(n) machine, then L is accepted by some space S(cn) + cn machine for some constant c.

$$\inf_{n \to \infty} \frac{S(cn) + cn}{2^n} > 0 \quad \text{implies that}$$

$S(n) \geq d^n$ for some constant $d > 1$ and all n which are multiples of c. □

Note that Theorem 2.1 can easily be strengthened by filling in the gaps between multiples of c, thus replacing the "infinitely many n" condition by an "all sufficiently large n" condition.

A simple coding can be used to show that Theorem 2.1 applies to smaller alphabets.

Lemma 2.2: For any finite Σ,

$$\text{RSQ}(\Sigma) \leq_{p\ell} \text{RSQ}(\{0,1\}).$$

Proof: Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ and let $h: \Sigma \to \{0,1\}^*$ be the uniquely decodable code $h(\sigma_i) = 1^{i-1}\, 0$, $i = 1, \ldots, k$. Let $C = h(\Sigma)$ be the set of code words. An expression for $\{0,1\}^* - C^*$ is

$$(0 \cup 1)^* \cdot 1^k \cdot (0 \cup 1)^*.$$

Given any regular expression with squaring E over Σ, form the expression

$$E' = h(E) \cup (0 \cup 1)^* \cdot 1^k \cdot (0 \cup 1)^*$$

over $\{0,1\}$, where h(E) is the expression obtained from

E by replacing each occurrence in E of a symbol in Σ by its code word. Now $L(E) = \Sigma^* \Leftrightarrow L(E') = \{0,1\}^*$ □

Corollary 2.1: The statement of Theorem 2.1 is true with $\Sigma = \{0,1\}$.

The next objective will be to get some rough upper bounds on the complexity of $RSQ(\Sigma)$.

Theorem 2.2: For any finite Σ, $RSQ(\Sigma)$ can be recognized in nondeterministic space c^n and hence deterministic space $(c^2)^n$ and time d^{d^n} for some $c > 1$, $d > 1$.

Proof: Given a regular expression with squaring of length n, first eliminate the squaring abbreviations obtaining a standard regular expression E of length at most 2^n. From this expression, construct a description of a nondeterministic finite state machine $\mathfrak{M}$ which accepts L(E). The length of this description is at most c^n for some $c > 1$. The "state" of $\mathfrak{M}$ after some input has been read is the subset of states $\mathfrak{M}$ could be in depending on what nondeterministic choices it made while reading the input. Any such subset can be stored in space c^n. Starting with the subset consisting of all starting states, symbols from Σ are nondeterministically chosen and the subset "state" updated accordingly after each choice. The original regular expression is accepted if $\mathfrak{M}$ ever enters a subset "state" which contains no final state. □

Therefore, the space bound of Theorem 2.1 is good to within the base of the exponential but the time bound isn't. What is needed is a stronger relation between the space and time complexities of a computation. Such a relation would be provided by a proposed result of Shamir [3] which states that a time T(n) machine can be simulated by a space $(\log T(n))^2$ machine. Such a relation would imply that the language L used in the proof of Theorem 2.1 requires time d^{d^n} for some $d > 1$ and all sufficiently large n.

Proposition 2.1: The time bound of Theorem 2.1 can be raised to d^{d^n} for some $d > 1$ provided that there is a constant $k > 0$ such that any time T(n) machine can be simulated by a space $(\log T(n))^k$ machine.

Such a time-space relation would also allow one to establish a greater than polynomial time bound for the following simpler language.

Definition: Let $REG(\Sigma)$ denote the set of all standard regular expressions which do not describe Σ^*.

Lemma 2.3: Let $L \subseteq \{0,1\}^*$ be any context sensitive language. Then $L \leq_{p\ell} REG(\{0,1\})$.

Proof: The proof is very similar to that of Lemma 2.1. The only difference is that an expression for Σ^k now is of length dk. But since the machine being simulated uses only space n, the length of the simulating regular expression is still only cn. □

Proposition 2.2: A $c^{n^{1/k}}$ lower time bound holds for $REG(\{0,1\})$ provided that a time-space relation as in Proposition 2.1 holds.

Conclusion: The general method of obtaining the lower bounds described in this section is to efficiently reduce the computation of any member of a general class of computation to a particular computation. A lower bound on the particular computation can then be obtained if there is a diagonalization (or other) argument which states that some members of the general class possess a certain complexity. It is expected that this method will find other uses than the ones described here.

One application so far is a proof that the weak monadic second order theory of successor and several other decidable theories to which it can be efficiently reduced are not elementary-recursive [4].

III. A Hierarchy of Languages

Related to the question of existence of natural non-polynomial time languages is the question of whether or not NP = P where the classes NP, P are defined as

NP(P) = {languages accepted by some nondeterministic (deterministic) polynomial time Turing machine}

Definition: Given a set of languages S and a transitive relation $\leq$ on S, a particular language L is complete in S with respect to $\leq$ if

1) $L \in S$

2) $L' \in S \Rightarrow L' \leq L$.

Cook [5] and Karp [6] have studied the class NP and have exhibited a variety of combinatorial problems which are complete in NP with respect to the reducibility $\leq_p$ defined as follows.

Definition: Given languages L_1, L_2, we say $L_1 \leq_p L_2$ if L_1 is accepted by some deterministic polynomial time machine $\mathfrak{M}$ with an L_2 oracle. The L_2 oracle can, in a single step of $\mathfrak{M}$, determine whether or not $y \in L_2$ where y is some string written on $\mathfrak{M}$'s tape.

This definition is due to Cook. Karp uses a stronger form of reducibility in which the oracle is called only once as the last step of the computation of $\mathfrak{M}$. The result of their work is that either all or none of the complete problems are recognizable in deterministic polynomial time.

We were first led to extend the class NP by considering the language MINIMAL which denotes the set of well-formed Boolean expressions for which there is no shorter equivalent expression. We do not know if either MINIMAL or its complement are in NP. However it is clear that $\neg$MINIMAL can be recognized by a polynomial time nondeterministic machine with an oracle for determining equivalence (or non-equivalence) of Boolean expressions. This latter non-equivalence problem is in NP. The method of obtaining the second class containing $\neg$ MINIMAL can be repeated to obtain a "hierarchy" of languages. The following relation is useful.

Definition: For languages L_1, L_2, we say $L_1 \, R_n \, L_2$ if L_1 is accepted by some nondeterministic polynomial time machine with oracle language L_2.

We have not used a reducibility notation for this relation because it is not clear that it is transitive. The classes of the polynomial hierarchy are defined as follows.

Definition: $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \emptyset$.

$$\Sigma_{i+1}^p = \{L \mid L\ R_n\ L' \text{ for some } L' \in \Sigma_i^p\}$$

$$\Pi_{i+1}^p = \{L \mid \neg\, L\ R_n\ L' \text{ for some } L' \in \Sigma_i^p\}$$

$$\Delta_{i+1}^p = \{L \mid L \leq_p L' \text{ for some } L' \in \Sigma_i^p\}$$

for all i = 0, 1, 2, ...

In particular Σ_1^p = NP and Δ_1^p = P.

The obvious inclusion properties are

1) $\Delta_i^p \subseteq \Sigma_i^p \cap \Pi_i^p$, $\quad i \geq 0$

2) $\Sigma_i^p \cup \Pi_i^p \subseteq \Delta_{i+1}^p$, $\quad i \geq 0$.

This hierarchy has the same inclusion structure as the Kleene arithmetical hierarchy[7]. In fact, the arithmetical hierarchy is given by the above definition if the relations R_n and $\leq_p$ are replaced by "is r.e. in" and "is recursive in" respectively. Unfortunately, the diagonalization arguments by which one can prove proper inclusion between successive classes in the arithmetical hierarchy do not appear to apply to the polynomial hierarchy. We do not know if the inclusions 1) or 2) are proper. Proving that the inclusion 2) is proper for all i is probably difficult because it implies in particular that NP ≠ P as the following illustrates.

Lemma 3.1: For any $k \geq 1$, if $\Sigma_k^p \cup \Pi_k^p = \Delta_k^p$, then $\Sigma_i^p = \Pi_i^p = \Delta_k^p$ for all $i \geq k$. In particular, if NP = P, then $\Sigma_i^p = \Pi_i^p = P$ for all $i > 0$.

Proof: The result is true for i = k. Assume true for some $i > k$. Now $L \in \Sigma_{i+1}^p \Rightarrow L\ R_n\ L'$ for some $L' \in \Sigma_i^p$ or $L' \in \Delta_i^p$ by induction. Therefore $L' \leq_p L''$ for some $L'' \in \Sigma_{k-1}^p$.

But $L\ R_n\ L' \leq_p L'' \Rightarrow L\ R_n\ L'' \Rightarrow L \in \Sigma_k^p \Rightarrow L \in \Delta_k^p$. □

Other analogies can be drawn between the arithmetical and polynomial hierarchies. However, it seems that almost anything non-trivial that can be said about the polynomial hierarchy implies in particular that NP ≠ P.

Lemma 3.1 shows that the problem of proving NP ≠ P is reduced to the problem of exhibiting some language $L \in \Sigma_k^p \cup \Pi_k^p$ for any $k \geq 1$ such that $L \notin P$. For example, if we could show that $RSQ(\{0,1\}) \in \Sigma_k^p \cup \Pi_k^p$ for some $k \geq 1$, we could conclude NP ≠ P. However, the following two results seem to indicate that REG({0,1}) (and hence RSQ({0,1})) is not in the hierarchy.

Lemma 3.2: If $L \in \Sigma_k^p \cup \Pi_k^p$ for any $k \geq 0$, then there is a polynomial p(n) such that L is recognized by some deterministic space p(n) machine.

Proof: The proof follows easily by induction on k.

Theorem 3.1: If $L \in \Sigma_k^p \cup \Pi_k^p$ for any $k \geq 0$, then $L \leq_p REG(\{0,1\})$.

Proof: Let p(n) be as in the previous Lemma. As in Lemma 2.3, given x of length n, construct a regular expression E of length $c \cdot p(n)$, for some constant c, such that

$$L(E) \neq \{0,1\}^* \Leftrightarrow x \in L.$$

The construction requires at most polynomial time. □

We conclude this section by exhibiting a complete language at each stage of the hierarchy.

Definition: For k = 1, 2, ... define

$B_k = \{A(\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_k) \mid A$ is a well-formed Boolean expression of the 0-1 valued variables

$$\bar{X}_j = \{X_{j1}, \ldots, X_{jm_j}\},\ j = 1, \ldots, k$$

and $(\exists \bar{X}_1)(\forall \bar{X}_2)(\exists \bar{X}_3) \ldots (\forall \bar{X}_k)[A(\bar{X}_1, \ldots, \bar{X}_k) = 1]\}$

In particular, $B_1 = \{$satisfiable formulas$\}$.

Theorem 3.2: For any $k \geq 1$,

1) $B_k \in \Sigma_k^p$.

2) $L \in \Sigma_k^p \cup \Pi_k^p \Rightarrow L \leq_p B_k$.

The case k = 1 was proven by Cook[5]. We do not claim that the languages B_k are natural problems. However they may provide a useful intermediate step in exhibiting natural problems which are complete in some class above NP, just as Cook's proof of the case k =1 provided a handle on the class NP.

We will give here only the proof of the k = 2 case. The general case follows easily by induction using essentially no new ideas. The proof is very similar to Cook's proof of the k = 1 case and it is assumed that the reader has some familiarity with that proof.

Proof (k = 2):

1) $B_2 \in \Sigma_2^p$ is clear.

2) Assume $L \in \Sigma_2^p$ is accepted by some nondeterministic time p(n) machine $\mathfrak{M}$ with oracle language $L' \in NP$ for some polynomial p.

We will assume that $\mathfrak{M}$ has a second tape, called the oracle tape, and three special states q_C, q_Y, q_N. Whenever $\mathfrak{M}$ enters state q_C, the next state is q_Y if $y \in L'$ or q_N if $y \notin L'$ where y is the string currently written on the oracle tape. For simplicity, assume L' has been coded so that $L' \subseteq \{0,1\}^*$ and that the head scanning the oracle tape can write only 0, 1, and b (blank).

Given some x of length n, we will construct a formula $A_x(\bar{X}_1, \bar{X}_2)$ such that

$$x \in L \Leftrightarrow (\exists \bar{X}_1)(\forall \bar{X}_2)[A_x(\bar{X}_1, \bar{X}_2) = 1].$$

The construction will take time polynomial in n.

As in [5], the formula A_x will be (in part) a function of the variables:

$P^i_{s,t}$ = 1 iff tape square s contains symbol σ_i at time t for $1 \le s,t \le p(n)$,

$R^i_{s,t}$ = 1 iff oracle tape square s contains symbol i at time t for $1 \le s,t \le p(n)$, $i \in \{0,1,b\}$,

and similar variables Q^i_t, $S_{s,t}$, $S'_{s,t}$ whose values determine the state and the two head positions. These variables will be called the main variables.

A_x will be a conjunction of terms. As in [5], terms will be constructed so that $A_x = 1$ iff the main variables describe a legal accepting computation of $\mathfrak{M}$ on input x. The only difference is that now terms must be constructed to insure that the state variables are forced to assume the correct values following oracle calls. This is done as follows.

Since $L' \in NP$, there is a Boolean expression $G(\bar{I},\bar{Y})$ where $\bar{I} = \{I^i_s \mid 1 \le s \le p(n), i \in \{0,1,b\}\}$ such that if the variables $\bar{I}$ are given values corresponding to some string $y = y_1y_2 \cdots y_m$, $y_j \in \{0,1\}$, $j = 1, \ldots, m$, in the sense that

$$I^i_s = \begin{cases} 1 \text{ if} & i = y_s \\ \text{or} & i = b \text{ and } s > m \\ 0 & \text{otherwise} \end{cases}$$

then $y \in L' \Leftrightarrow (\exists \bar{Y})[G(\bar{I},\bar{Y}) = 1]$.

Moreover, G can be constructed in time polynomial in p(n). G is constructed as in [5] except that some of the tape symbol variables are left free for t = 1, $1 \le s \le p(n)$, instead of being bound to some fixed input. These free variables become the $\bar{I}$ input variables above.

A_x will contain the term

$$J = \bigwedge_{t=1}^{p(n)} [Q^C_t \rightarrow [(\neg G_{1t}(\bar{R}_t, \bar{Y}_{1t}) \rightarrow Q^N_{t+1}) \wedge (G_{2t}(\bar{R}_t, \bar{Y}_{2t}) \rightarrow Q^Y_{t+1})]]$$

where G_{jt} are different copies of the formula G above, each with distinct variables $\bar{Y}_{jt}$. For each time t,

$$\bar{R}_t = \{R^i_{s,t}\}$$

binds the input of G_{1t} and G_{2t} to the oracle tape at this time.

The variables are distributed in the two blocks as

$$\bar{X}_1 = \text{main variables} \cup \bigcup_{t=1}^{p(n)} \bar{Y}_{1t}.$$

$$\bar{X}_2 = \bigcup_{t=1}^{p(n)} \bar{Y}_{2t}.$$

A_x is the conjunction of the terms in [5] together with the new term J.

The reader may verify that the main variables can describe a legal accepting computation of $\mathfrak{M}$ on x if and only if $(\exists \bar{X}_1)(\forall \bar{X}_2)[A_x(\bar{X}_1, \bar{X}_2) = 1]$. □

It would be interesting to find some other problems which are complete in $\Sigma^p_2 \cup \Pi^p_2$. One approach is to add another level of quantification to problems known to be complete in NP. One possibility is the set of all graph pair descriptions (G_1, G_2) such that any subgraph of G_1 is isomorphic to some subgraph of G_2. It would be less artificial to find problems for which the alternation of quantifier is not so obvious. The language MINIMAL defined at the beginning of this section is one possibility.

Open Question: Is MINIMAL complete in $\Sigma^p_2 \cup \Pi^p_2$?

Acknowledgment

Michael J. Fischer contributed several ideas to this work. His help and interest are greatly appreciated.

References

1. Hartmanis, J. and R.E. Stearns, "On the Computational Complexity of Algorithms", Trans. Amer. Math. Soc. 117, 1965, 285-306.

2. Hartmanis, J., P.M. Lewis, and R.E. Stearns, "Hierarchies of Memory Limited Computations", IEEE Conf. Rec. on Switching Th. and Logical Design, Sixth Ann. Symp., 1965, 179-190.

3. Shamir, E., "Relations Between Tape and Time Complexity Via Pushdown Machines", Abstract 72T-C23, Notices of Amer. Math. Soc. 19, 3, April 1972.

4. Meyer, A.R., "Weak Monadic Second Order Theory of Successor is Not Elementary-Recursive", manuscript, M.I.T., Project MAC, Cambridge, Mass., 1972.

5. Cook, S., "The Complexity of Theorem - Proving Procedures", Conf. Rec. of Third ACM Symp. on Th. of Computing, 1970, 151-158.

6. Karp, R.M., "Reducibility Among Combinatorial Problems", Tech. Report 3, Dept. of Computer Sci., University of California, Berkeley, 1972.

7. Rogers, H. Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967.