WEAK MONADIC SECOND ORDER THEORY OF SUCCESOR IS NOT ELEMENTARY-RECURSIVE[†]

Albert R. Meyer

Let L_{SIS} be the set of formulas expressible in a weak monadic second order logic using only the predicates $[x = y+1]$ and $[x \in X]$. Büchi and Elgot[3,4] have shown that the truth of sentences in L_{SIS} (under the standard interpretation $< N$, successor $>$ with second order variables interpreted as ranging over finite sets) is decidable. We refer to the true sentences in L_{SIS} as WSIS. We shall prove that WSIS is not elementary-recursive in the sense of Kalmar. In fact, we claim a stronger result:

Theorem 1: There is a constant $\epsilon > 0$ such that if $\mathfrak{M}$ is a Turing machine which, started with any sentence in L_{SIS} on its tape, eventually halts in a designated accepting state if and only if the sentence is true, then for all sufficiently large n, there is a sentence of length n[†•] for which $\mathfrak{M}$'s computation requires

$$\left.{2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}}\right\} \lfloor \epsilon \cdot \log_2 n \rfloor$$

steps and tape squares.

[†•] By the length of a sentence we mean the number of characters in it including parentheses, digits in subscripts, etc. Any of the standard conventions for punctuating well-formed formulas may be used, except that in some cases conventions for matching parentheses may imply that for infinitely many n, there cannot be any wff's of length n. In this case, we assume that wff's may be lengthened by concatenating a finite sequence of "blank" symbols which leave the meaning of the wff unchanged, so that sentences of length n can be constructed for all sufficiently large n.

[†] Work reported here was supported in part 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-0362-0006 and the National Science Foundation under contract number GJ34671 Reproduction in whole or in part is permitted for any purpose of the United States Government.

Let $t_0(n) = n$, $t_{k+1}(n) = 2^{t_k(n)}$. A well-known characterization of the elementary-recursive functions by R.W. Ritchie[14] shows that a set of sentences is elementary-recursive iff it is recognizable in space bounded by

$$t_k(n) = \left.2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}\right\}k$$

for some fixed k and all inputs of length $n \geq 0$. Hence, <u>WSIS</u> is not elementary-recursive.

In these notes we prove a somewhat less powerful version of Theorem 1, which by Ritchie's result is still sufficient to establish the truth of our title.

<u>Theorem 2</u>: Let $\mathfrak{M}$ be a Turing machine which, started with any sentence in L_{SIS} on its tape, eventually halts in a designated accepting state iff the sentence is true. Then for any $k \geq 0$, there are infinitely many n for which $\mathfrak{M}$'s computation requires

$$\left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}}\right\}k$$

steps and tape squares for some sentence of length n.

The idea behind our proof will be to show that there are sentences in L_{SIS} of length n which describe the computation of Turing machines, provided the space required by the computation is not greater than $t_k(n)$. Since a Turing machine using a given amount of space can simulate and differ from all machines using less space, we will deduce that small sentences in L_{SIS} can describe inherently long computations, and hence L_{SIS} must itself be difficult to decide.

Actually it will be more convenient to develop an intermediate notation called γ-expressions for sets of finite sequences. We will show that γ-expressions can, in an appropriate sense, describe Turing machine computations, and that L_{SIS} can describe properties of γ-expressions.

Definition: Let Σ be a finite set whose elements are called symbols. Σ^* is the set of all finite sequences of symbols from Σ. For x, $y \in \Sigma^*$, the concatenation of x and y, written $x \cdot y$ or xy, is the sequence consisting of the symbols of x followed by those of y. An element $x \in \Sigma^*$ is called a word, and the length of x is written $\ell(x)$. We use λ to designate the vacuous sequence of length zero in Σ^* which by convention has the property that $x \cdot \lambda = \lambda \cdot x = x$ for any $x \in \Sigma^*$. (Σ^* is the free monoid with identity λ generated by Σ.) Concatenation is extended to subsets $A, B \subset \Sigma^*$ by the rule

$$A \cdot B = AB = \{xy \mid x \in A, y \in B\}.$$

For any $A \subset \Sigma^*$, we define

$$A^0 = \{\lambda\}, \; A^{n+1} = A^n \cdot A, \; A^* = \bigcup_{n=0}^{\infty} A^n.$$

These operations are familiar in automata theory. We introduce one further mapping.

Definition:[†] For any Σ, the function $\gamma_\Sigma \colon P(\Sigma^*) \to P(\Sigma^*)$ is defined by the rules

[†] $P(S) = \{A \mid A \subset S\}$ = the power set of S.

$$\gamma_{\Sigma}(\{x\}) = \{y \in \Sigma^* \mid \ell(x) = \ell(x)\} = \Sigma^{\ell(x)} \qquad \text{for } x \in \Sigma^*,$$

$$\gamma_{\Sigma}(A) = \bigcup_{x \in A} \gamma(\{x\}) \qquad \text{for } A \subset \Sigma^*.$$

We omit the subscript on γ_{Σ} when Σ is clear from context.

γ-expressions over Σ are certain words in $(\Sigma \cup \{\underset{\sim}{\gamma}, \underset{\sim}{\cdot}, \underset{\sim}{\neg}, \underset{\sim}{\cup}, \underset{\sim}{(}, \underset{\sim}{)}\})^*$ where $\underset{\sim}{\gamma}, \underset{\sim}{\cdot}, \underset{\sim}{\neg}, \underset{\sim}{\cup}, \underset{\sim}{(}, \underset{\sim}{)}$ are symbols <u>not</u> in Σ. Any γ-expression α defines a set $L(\alpha) \subset \Sigma^*$.

<u>Definition</u>: For any Σ, <u>γ-expressions over Σ</u> and the function L:{γ-expressions over Σ} $\to P(\Sigma^*)$ are defined inductively as follows:

1) σ is a γ-expression over Σ for any $\sigma \in \Sigma$, and $L(\sigma) = \{\sigma\}$;

2) if α, β are γ-expressions over Σ, then $\underset{\sim}{(}\ \alpha \underset{\sim}{\cdot} \beta\ \underset{\sim}{)}$, $\underset{\sim}{(}\ \alpha \underset{\sim}{\cup} \beta\ \underset{\sim}{)}$, $\underset{\sim}{\neg}\underset{\sim}{(}\ \alpha\ \underset{\sim}{)}$, and $\underset{\sim}{\gamma}\ \underset{\sim}{(}\ \alpha\ \underset{\sim}{)}$, are γ-expressions over Σ, and
$L(\underset{\sim}{(}\ \alpha \underset{\sim}{\cdot} \beta\ \underset{\sim}{)}) = L(\alpha)\cdot L(\beta)$, $L(\underset{\sim}{(}\ \alpha \underset{\sim}{\cup} \beta \underset{\sim}{)}) = L(\alpha) \cup L(\beta)$,
$L(\underset{\sim}{\neg}\underset{\sim}{(}\ \alpha\ \underset{\sim}{)}) = \Sigma^* - L(\alpha)$, and $L(\underset{\sim}{\gamma}\ \underset{\sim}{(}\ \alpha\ \underset{\sim}{)}) = \gamma(L(\alpha))$,

3) That's all.

Having thus made clear the distinction between a γ-expression α and the set $L(\alpha)$ it defines, we will frequently ignore the distinction when there can be no confusion. Thus we write $\Sigma^* = \sigma \cup \neg(\sigma)$ instead of $\Sigma^* = L(\ \underset{\sim}{(}\ \sigma \underset{\sim}{\cup} \underset{\sim}{\neg} \underset{\sim}{(}\ \sigma\ \underset{\sim}{)}\ \underset{\sim}{)}\ )$. Similarly, for any set of letters $V \subset \Sigma$,

$$V^* = \neg(\Sigma^* \cdot (\Sigma - V)\cdot\Sigma^*)$$

since V^* consists precisely of those words in Σ^* which do not contain a symbol not in V. Thus there is a γ-expression α over Σ such that $L(\alpha) =$

V^*. DeMorgan's law gives us intersection, and then the identities

$$V^n = \Sigma^n \cap V^*, \text{ and}$$
$$\Sigma^n = \gamma(V^n)$$

imply that from a γ-expression of length s for Σ^n we can obtain a γ-expression of length $s + c$ for V^n, and conversely from V^n to Σ^n, for some constant c and all $s, n \in N$. We shall show below that in general s may be much smaller than n.

Definition: Empty $(\Sigma) = \{\alpha \mid \alpha$ is a γ-expression over Σ and $L(\alpha) = \emptyset\}$.

Since the regular (finite automaton recognizable) subsets of Σ^* are closed under $\cdot$, $\cup$, $\neg$, and γ, it follows that Empty(Σ) is recursive and in fact primitive recursive. One simply constructs a finite automaton for $L(\alpha)$ and tests whether the automaton accepts some word; there are well-known procedures to do this. *A priori* analysis of this procedure however indicates that from *deterministic* automata for γ-expressions α, β one would obtain a *non-deterministic* automaton for $\alpha \cdot \beta$ or $\gamma(\alpha)$, and then would have to apply the "subset construction" of Rabin-Scott [13] to obtain an automaton for $\neg(\alpha \cdot \beta)$ or $\neg\gamma(\alpha)$. Since the subset construction can exponentially increase the number of states in the automaton, γ-expressions in which k complementations alternated with γ's and concatenations can lead to an automaton with $t_k(2)$ states. The time and space required by a Turing machine which recognizes Empty (Σ) by the procedure outlined above *can* be bounded above by

$$t_n(c) = \left.2^{2^{\cdot^{\cdot^{\cdot^{2^c}}}}}\right\} n$$

for some constant c and all γ-expressions of length $n \geq 0$. It will follow from results below that such absurd inefficiency is inevitable.

Definition: A Turing machine $\mathfrak{M}$ recognizes a set $A \subset \Sigma^*$ if, when started with any word $x \in \Sigma^*$ on its tape, $\mathfrak{M}$ halts in a designated accepting state iff $x \in A$.

Let $f: \mathbb{N} \to \mathbb{N}$. The space complexity of a set $A \subset \Sigma^*$ is at most f almost everywhere, written

$$\text{SPACE } (A) \leq f \text{ (a.e.)}$$

iff there is a Turing machine which recognizes A and which, for all but finitely many $x \in A$, uses at most $f(\ell(x))$ tape squares in its computation on input x. The space complexity of A exceeds f infinitely often written

$$\text{SPACE } (A) > f \text{ (i.o.)}$$

iff it is not true that SPACE (A) $\leq$ f (a.e.).

We shall use Turing's original one tape, one read-write head model of Turing machine, and define the number of tape squares used during the computation on input x to be the larger of $\ell(x)$ and the number of tape squares visited by the read-write head. Then by convention at least $\max\{\ell(x), 1\}$ tape squares are used in a computation on any input word x.

We briefly review some well-known facts, first established by Stearns, Hartmanis, and Lewis [15], about space-bounded Turing machine computations.

Definition: A function $f: N \to N$ is tape constructible iff there is a Turing machine which, started with any input word of length $n \geq 0$, halts having used exactly $f(n)$ tape squares.

Fact 1: $t_0 + 1 = \lambda n[n+1]$ is tape constructible. For any $k > 0$, t_k is tape constructible.

Fact 2: If $f: N \to N$ is tape constructible, and SPACE $(A) \leq f$ (a.e.) for some $A \subset \Sigma^*$, then there is a Turing machine which recognizes A which halts on every input $x \in \Sigma^*$ using at most $f(\ell(x))$ tape squares.

Hence, SPACE $(A) \leq f \Leftrightarrow$ SPACE $(\Sigma^* - A) \leq f$.

Fact 3: If $f: N \to N$ is tape constructible, then there is an $A \subset \{0,1\}^*$ such that

$$\text{SPACE }(A) \leq f \quad \text{and}$$

$$\text{SPACE }(A) > g \quad \text{(i.o.)}.$$

for any $g: N \to N$ such that

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0.$$

Our proof consists of a sequence of reductions of one decision or recognition problem to another. In contrast to the usual reductions of recursive function theory, our reductions must be computationally efficient. We introduce a particular notion of efficient reduction which is sufficient for our purposes.

Definition: Let Σ_1, Σ_2 be finite sets of symbols, and $A_1 \subset \Sigma_1^*$, $A_2 \subset \Sigma_2^*$.

A_1 is efficiently reducible to A_2, written

$$A_1 \ \underline{\text{eff}}\ A_2$$

providing there is a polynomial p and a Turing machine which, started with any word $x \in \Sigma_1^*$ on its tape, eventually halts with a word $y \in \Sigma_2^*$ on its tape such that

1) $x \in A_1 \Leftrightarrow y \in A_2$, and

2) the number of tape squares used in the computation on input x is at most $p(\ell(x))$ (and <u>a fortiori</u> $\ell(y) \leq p(\ell(x))$).

We remark that all the reductions which are described below require only a linear polynomial number of tape squares and a polynomial number of steps, but to minimize the demands on the readers intuition (since we never actually give a flow-chart or table of quadruples for the Turing machines we describe) we allow polynomials of any degree. Even so, <u>eff</u> is much more restricted than is necessary to prove Theorem 2.

<u>Fact 4</u>. <u>eff</u> is a transitive relation on sets of words.

<u>Fact 5</u>. If A_1 <u>eff</u> A_2 and SPACE $(A_2) \leq f$ (a.e.), then there is a polynomial p such that

$$\text{SPACE } (A_1) \leq \lambda n[\ \max\{f(m) \mid m \leq p(n)\} + p(n)]\ \text{(a.e.)}$$

<u>Fact 6</u>. If A_1 eff A_2 and SPACE $(A_1) > t_{k+1}$ (i.o.), then SPACE $(A_2) > t_k$ (i.o.).

Proof. Immediate from Fact 5 and that observation that for any polynomial p, $t_k(p(n)) + p(n) \leq t_{k+1}(n)$ for all sufficiently large n.

The proof of Theorem 2 can now be summarized.

<u>Proof of Theorem 2</u>: We will establish below that

<u>Empty</u> $(\{0,1\})$ <u>eff</u> <u>WSIS</u>

<u>Empty</u> (Σ) <u>eff</u> <u>Empty</u> $(\{0,1\})$ for any finite Σ,

and finally that for any k and for any set $A \subset \{0,1\}^*$ such that SPACE (A) $\le t_k$ (a.e.) there is a finite Σ such that

A eff Empty (Σ)

From fact 4, we have A eff WSIS for any A and k such that SPACE (A) $\le t_k$ (a.e.).

Then from facts 1, 3 and 6 we conclude that SPACE (WSIS) $> t_{k-1}$(i.o.) for any k. Q.E.D.

It remains only to establish the required reductions.

Lemma 1: Empty($\{0,1\}$) eff WSIS

Proof: For any γ-expression α over $\{0,1\}$ we shall show how to construct a formula $F_\alpha \in L_{SIS}$ with two free integer variables and one free set variable. For any set $M \subset N$, let $C_M: N \to \{0,1\}$ be the characteristic function of M, that is, $C_M(n) = 1 \Leftrightarrow n \in M$. The formula F_α will be constructed so that for n, m $\in$ N, M $\subset$ N, M finite:

$$F_\alpha(n,m,M) \text{ is true} \Leftrightarrow [[n < m \text{ and } C_M(n)\cdot C_M(n+1) \ \ldots\ C_M(m-1) \in L(\alpha)] \text{ or } [n = m \text{ and } \lambda \in L(\alpha)]].$$

F_α is constructed by induction on the definition of γ-expressions. If α is 0 or 1, then

$F_0(x,y,X)$ is $[y = x+1$ and $\neg (x \in X)]$,

$F_1(x,y,X)$ is $[y = x+1$ and $x \in X]$.

If α is $\underset{\sim}{(}\ \beta \underset{\sim}{\cdot}\ \delta \underset{\sim}{)}$, then

$F_\alpha(x,y,X)$ is $(\exists z)[x \le z$ and $z \le y$ and $F_\beta(x,z,X)$ and $F_\delta(z,y,X)]$.

If α is $\underset{\sim}{\gamma}\,\underset{\sim}{(}\,\beta\,\underset{\sim}{)}$, then

$$F_\alpha(x,y,X) \text{ is } (\exists X_0)[F_\beta(x,y,X_0)].$$

If α is $\underset{\sim}{(}\,\beta\,\underset{\sim}{\cup}\,\delta\underset{\sim}{)}$ or $\underset{\sim}{\neg}\,\underset{\sim}{(}\,\beta\,\underset{\sim}{)}$, then F_α is $[F_\beta$ <u>or</u> $F_\delta]$ or $[x \le y$ <u>and</u> $\neg F_\beta(x,y,X)]$, respectively.

It is clear that there is a Turing machine which, given an input $\alpha \in \{0,\ 1,\ \underset{\sim}{(}\ ,\ \underset{\sim}{)},\ \underset{\sim}{\cup},\ \underset{\sim}{\gamma},\ \underset{\sim}{\cdot},\ \underset{\sim}{\neg}\}^*$, can test whether α is a well-formed γ-expression and, if so, print out the sentence

$$\neg\,(\exists x)(\exists y)(\exists X)[F_\alpha(x,y,X)],$$

never using more space than some fixed polynomial in $\ell(\alpha)$. (If α is not well-formed, the machine prints out $(\exists x)[x = x+1]$.) Hence, <u>Empty</u>$(\{0,1\})$ <u>eff</u> <u>WSIS</u>. Q.E.D.

It will be convenient to work with larger symbol sets than $\{0,1\}$, but a trivial coding will demonstrate that this involves no loss of generality.

Let Σ be any finite set of symbols with $|\Sigma| \ge 2$. Say $0,\ 1 \in \Sigma$, $0 \ne 1$. Then for any $n \ge 1$, there is a γ-expression over Σ for $(\Sigma^n)^*$. To see this, consider a word in Σ^* <u>not</u> in $(0^{n-1}\ 1)^*$. Such a word either fails to begin with $0^{n-1}\ 1$, fails to end with 1, or contains a subword in $0\ \Sigma^{n-1}(\Sigma\text{-}0)$ or $1\ \Sigma^{n-1}(\Sigma\text{-}1)$. Hence

$$\lambda \cup \neg((0^{n-1}1)^*) = \neg(0^{n-1}1\ \Sigma^*) \cup \neg(\Sigma^*\ 1) \cup (\Sigma^*\ 0\ \Sigma^{n-1}(\Sigma\text{-}0)\ \Sigma^*)$$
$$\cup(\Sigma^*\ 1\ \Sigma^{n-1}(\Sigma\text{-}1)\ \Sigma^*),$$

and so, noting that $\lambda = \neg(\Sigma\cdot\Sigma^*)$, we have

$$(0^{n-1}\ 1)^* = \neg(\lambda \cup \neg((0^{n-1}\ 1)^*)) \cup \lambda,$$

and

$$(\Sigma^n)^* = \gamma((0^{n-1}\ 1)^*).$$

Now given any finite set Σ_1 choose n sufficiently large that $|\Sigma^n| \geq |\Sigma_1|$ and let h: $\Sigma_1 \to \Sigma^n$ be any one-one function. Extend h to a one-one map from $P(\Sigma_1^*)$ into $P((\Sigma^n)^*)$ by the obvious rules $h(\lambda) = \lambda$, $h(x\sigma_1) = h(x) \cdot h(\sigma_1)$ for $x \in \Sigma_1^*$, $\sigma_1 \in \Sigma_1$, and $h(A) = \bigcup\limits_{x \in A} \{h(x)\}$ for $A \subset \Sigma_1^*$. There is then a γ-expression over Σ for $h(\Sigma_1^*)$, because a word <u>fails</u> to be in $h(\Sigma_1^*)$ either because its length is not a multiple of n, or else because it contains a subword of length n not in $h(\Sigma_1)$ which begins at a position congruent to one modulo n:

$$\Sigma^* - h(\Sigma_1^*) = \neg((\Sigma^n)^*) \cup (\Sigma^n)^* \cdot (\Sigma^n - h(\Sigma_1)) \cdot (\Sigma^n)^*.$$

<u>Lemma 2</u>: (Coding) Let Σ_1, Σ be finite sets of symbols with $|\Sigma| \geq 2$. Let h: $P(\Sigma_1^*) \to P((\Sigma^n)^*)$ be the extension of a one-one function from Σ_1 to Σ^n for some $n \geq 1$. There is a Turing machine which, started with a γ-expression α over Σ_1, halts with a γ-expression β over Σ on its tape such that

$$h(L(\alpha)) = L(\beta).$$

Moreover the space used during the computation with input α is bounded by a polynomial in $\ell(\alpha)$.

<u>Proof</u>. The transformation of α to β operates by applying the following rules recursively.

If $\alpha \in \Sigma_1$, β is set equal to an expression for $h(L(\alpha))$.

If α is $\underset{\sim}{(}\ \alpha_1 \underset{\sim}{\cdot} \alpha_2 \underset{\sim}{)}$ or $\underset{\sim}{(}\ \alpha_1 \underset{\sim}{\cup} \alpha_2 \underset{\sim}{)}$, then β is $\underset{\sim}{(}\ \beta_1 \underset{\sim}{\cdot} \beta_2 \underset{\sim}{)}$ or $\underset{\sim}{(}\ \beta_1 \underset{\sim}{\cup} \beta_2 \underset{\sim}{)}$, respectively, where β_1, β_2 are the transforms of α_1, α_2.

If α is $\underset{\sim}{\gamma} \underset{\sim}{(}\ \alpha_1 \underset{\sim}{)}$, then β is

$$\utilde{\neg}\,\utilde{(}\,\utilde{(}\,\utilde{\neg}\,\utilde{(}\,\utilde{\gamma}\,\utilde{(}\,\beta_1\,\utilde{)}\,\utilde{)}\,\utilde{\cup}\,\beta_{\Sigma_1}\,\utilde{)}\,\utilde{)}$$

where β_1 is the transform of α_1 and β_{Σ_1} is a γ-expression over Σ for $\Sigma^*-h(\Sigma_1^*)$. (Note that $h(\gamma_{\Sigma_1}(A)) = \gamma_\Sigma(h(A)) \cap h(\Sigma_1^*)$ for $A \subset \Sigma_1^*$, which justifies this rule.)

Finally, if α is $\utilde{\neg}\,\utilde{(}\,\alpha_1\,\utilde{)}$, then β is $\utilde{\neg}\,\utilde{(}\utilde{(}\beta_1\,\utilde{\cup}\,\beta_{\Sigma_1}\utilde{)}\,\utilde{)}$, since $h(\Sigma_1^*-A) = h(\Sigma_1^*) -h(A) = \Sigma^* -(h(A) \cup (\Sigma^*-h(\Sigma_1^*))$ for $A \subset \Sigma_1^*$.

It is clear that a Turing machine can carry out this recursive transformation within the required space bound. Q.E.D.

<u>Corollary</u>: <u>Empty</u> (Σ) <u>eff</u> <u>Empty</u> $\{0,1\}$ for any finite Σ.

<u>Proof</u>: Code Σ into $\{0,1\}$ via h as in Lemma 2. Then $\alpha \in$ <u>Empty</u> $(\Sigma) \Leftrightarrow$ $L(\alpha) = \emptyset \Leftrightarrow h(L(\alpha)) = \emptyset \Leftrightarrow L(\beta) = \emptyset \Leftrightarrow \beta \in$ <u>Empty</u> $\{0,1\}$. Q.E.D.

We now show how, given a γ-expression for Σ^n, one can construct a γ-expression of about the same size describing any desired computation of a Turing machine, providing the states and symbols of the Turing machine can be represented in Σ and the computation only requires n tape squares. This construction will be applied recursively to obtain γ-expressions of size n for $\Sigma^{t_k(n)}$, and will then finally be used to conclude that A <u>eff</u> <u>Empty</u> (Σ) for any $A \subset \{0,1\}^*$ such that SPACE $(A) \leq t_k$ (a.e.).

Definition: Let $\mathfrak{M}$ be any Turing machine with tape symbols T and states S. Assume $b \in T$ where "b" designates a blank tape square. An instantaneous description (i.d.) of $\mathfrak{M}$ is a word in $(T \cup (S \times T))^*$[†] which contains exactly one symbol in $S \times T$. Given any i.d. $x = y\cdot(s,t)\cdot z$ for $y, z \in T^*$, $s \in S$, $t \in T$, the next i.d., $\text{Next}_{\mathfrak{M}}(x)$ is defined as follows: if when $\mathfrak{M}$ is in state s with its read-write head scanning symbol t, $\mathfrak{M}$ enters state s' and writes symbol $t' \in T$, then $\underline{\text{Next}}_{\mathfrak{M}}(x)$ is

$y\cdot(s', t')\cdot z$	if $\mathfrak{M}$ does not shift its head,
$y\cdot t'(s', u)\cdot w$	if $\mathfrak{M}$ shifts its head right and $z = uw$ for $u \in T$, $w \in T^*$,
$w\cdot(s', u)\cdot t'\cdot z$	if $\mathfrak{M}$ shifts its head left and $y = wu$ for $u \in T$, $w \in T^*$.

$\text{Next}_{\mathfrak{M}}(x)$ is undefined if (s,t) is a halting condition, or if (s,t) is the rightmost (leftmost) symbol of x and $\mathfrak{M}$ shifts right (left). Let $\underline{\text{Next}}_{\mathfrak{M}}(x,0) = x$ if x is an i.d., undefined otherwise; $\underline{\text{Next}}_{\mathfrak{M}}(x, n+1) = \text{Next}_{\mathfrak{M}}(\text{Next}_{\mathfrak{M}}(x,n))$.

Finally, let # be a symbol not in $T \cup (S \times T)$. The computation $\text{Comp}(\mathfrak{M},x)$ of $\mathfrak{M}$ from x is singleton set consisting of the following word in $(\{\#\} \cup T \cup (S \times T))^*$:

$$\text{Comp}(\mathfrak{M},x) = \{\#\cdot\text{Next}_{\mathfrak{M}}(x,0)\cdot\#\cdot\text{Next}_{\mathfrak{M}}(x,1)\cdot\#\cdot\cdots\cdot\#\cdot\text{Next}_{\mathfrak{M}}(x,n)\cdot\#\}$$

[†] $S \times T = \{(s,t) \mid s \in s \text{ and } t \in T\}$. We assume $T \cap (S \times T) = \emptyset$

where n is the least integer such that (q_a, t) occurs in $\text{Next}_{\mathfrak{M}}(x,n)$ for some $t \in T$ and designated halting state q_a. $\text{Comp}(\mathfrak{M},x) = \phi$ if there is no such n.

<u>Remark</u>: Note that our definition of computation differs from the one commonly in the literature. The computation $\text{Comp}(\mathfrak{M},x)$ is defined for <u>i.d.'s</u> x, <u>not</u> input words x. Moreover, all i.d.'s in $\text{Comp}(\mathfrak{M},x)$ have exactly the same length. A key property of $\text{Comp}(\mathfrak{M},x)$ is given next.

<u>Fact 7</u>: Given $\mathfrak{M}$ as in the preceding definition, let $\Sigma = \{\#\} \cup T \cup (S \times T)$. Then for any i.d. $y \in \Sigma^*$, the n-1st, nth and n+1st symbols of y uniquely determine the nth symbol of $\text{Next}_{\mathfrak{M}}(y)$ for $1 < n < \ell(y)$ providing $\text{Next}_{\mathfrak{M}}(y)$ is defined.

Hence, there is a partial function $f_{\mathfrak{M}}: \Sigma^3 \to \Sigma$ such that if $\sigma_1, \sigma_2, \sigma_3$ are the n-1st, nth, n+1st symbols of $\text{Comp}(\mathfrak{M},x)$, then $f_{\mathfrak{M}}(\sigma_1,\sigma_2,\sigma_3)$ is the $n+\ell(x)+1$st symbol of $\text{Comp}(\mathfrak{M},x)$ for $1 < n < \ell(\text{Comp}(\mathfrak{M},x)) - \ell(x)$ and any i.d. x such that $\text{Comp}(\mathfrak{M},x) \neq \phi$. Also, $f_{\mathfrak{M}}(\sigma_1,\sigma_2,\sigma_3) = \phi$, if $\sigma_2 \in (S \times T)$ and σ_2 is a halting condition of $\mathfrak{M}$.

<u>Lemma 3</u>: (Simulation) Let $\mathfrak{M}$ be a Turing machine with states S, symbols T, and designated halting state $q_a \in S$. Let $\Sigma = \{\#\} \cup T \cup (S \times T)$. There is a Turing machine $\mathcal{F}(\mathfrak{M})$ which, started with any word $y \cdot \# \cdot \alpha$ on its tape where y is an i.d. of $\mathfrak{M}$ and α is a γ-expression over Σ such that $L(\alpha) = \Sigma^n$ for some $n > 0$, halts with a γ-expression β over Σ such that

$$L(\beta) = \text{Comp}(\mathfrak{M}, b^n \cdot y \cdot b^n).$$

Moreover, there is a polynomial p such that $\mathfrak{F}(\mathfrak{M})$ never uses more than $p(\ell(y\cdot\#\cdot\alpha))$ tape squares in its computation.

Proof: We shall describe how to construct the γ-expression β for $\mathrm{Comp}(\mathfrak{M}, b^n y b^n)$ from $y\cdot\#\cdot\alpha$ where $L(\alpha) = \Sigma^n$. We begin by noting that the words in Σ^* not equal to $\mathrm{Comp}(\mathfrak{M}, b^n y b^n)$, i.e., $\neg(\mathrm{Comp}(\mathfrak{M}, b^n y b^n))$, can be characterized as follows:

1) words that do not begin with $\# b^n y b^n \#$, or
2) words that do not contain q_a, or
3) words that do not end with #, or
4) words that violate the functional condition determined by $f_{\mathfrak{M}}$ in Fact 7.

These four sets of words can also be described by the formulas

1') $\neg(\#\cdot(L(\alpha) \cap b^*)\cdot y\cdot(L(\alpha) \cap b^*)\cdot\#\cdot\Sigma^*)$,

2') $\neg(\Sigma^*\cdot(\{q_a\} X T)\cdot\Sigma^*)$,

3') $\neg(\Sigma^*\cdot\#)$,

4') $\bigcup_{\sigma_1,\sigma_2,\sigma_3\in\Sigma} [\Sigma^*\cdot\sigma_1\sigma_2\sigma_3\cdot L(\alpha)\cdot\Sigma^{\ell(y)-1}\cdot L(\alpha)\cdot(\Sigma - f_{\mathfrak{M}}(\sigma_1,\sigma_2,\sigma_3))\cdot\Sigma^*]$

But it is easy to see how to construct γ-expressions directly from (1')-(4'), and therefore β is simply the complement of the union of these four expressions. Note that $\ell(\beta) \leq c\cdot\ell(y\#\alpha)$ for some constant c which depends only on $\mathfrak{M}$, and not on y or α. Moreover a Turing machine $\mathfrak{F}(\mathfrak{M})$ which constructs β from $y\#\alpha$ need never use more tape squares than $\ell(\beta)$, and so certainly runs within a polynomial space bound. Q.E.D.

Definition: A $\underline{\Sigma\text{-}t_k\text{-TM}}$ is a Turing machine such that for some polynomial p, some function $f_k \geq t_k$, and all $n > 0$, when the Turing machine is started with 0^n on its tape, it halts with a word α on its tape such that

1) α is a γ-expression over Σ and $L(\alpha) = \Sigma^{f_k(n)}$,

2) the number of tape squares used in the computation is at most $p(n)$.

Lemma 4: If there is a $\Sigma'\text{-}t_k$-TM for any finite Σ', then there is a $\Sigma\text{-}t_k$-TM for any Σ such that $|\Sigma| \geq 2$.

Proof: Code Σ' into Σ as in Lemma 2. Details are left to the reader.

Q.E.D.

Lemma 5: For any $k \geq 0$ and any Σ with $|\Sigma| \geq 2$, there is a $\Sigma\text{-}t_k$-TM.

Proof: A $\Sigma\text{-}t_0$-TM simply prints an expression for $\gamma(\sigma^n)$ from input 0^n, where $\sigma \in \Sigma$ is any symbol. Proceeding by induction, assume there is a $\Sigma\text{-}t_k$-TM. Let $\mathfrak{M}_k$ be a Turing machine which, started with 0^n on its tape for any $n > 0$, lays out $t_k(n)$ tape squares on its tape and then uses these tape squares to cycle through some number $f_{k+1}(n) \geq 2^{t_k(n)} = t_{k+1}(n)$ steps before finally halting. Since t_k is tape-constructible, it is easy to obtain $\mathfrak{M}_k$ as described. Choose Σ as in the simulation lemma applied to $\mathfrak{M}_k$.

The $\Sigma\text{-}t_{k+1}$-TM operates as follows: Given 0^n, use the $\Sigma\text{-}t_k$-TM to obtain α such that $L(\alpha) = \Sigma^{f_k(n)}$. Apply $\mathcal{F}(\mathfrak{M}_k)$ of the simulation lemma to $(q_0,0)0^{n-1}\cdot\#\cdot\alpha$ where q_0 is the start state of $\mathfrak{M}_k$. This yields a

γ-expression β such that $L(\beta) = \mathrm{Comp}(\mathfrak{M}_k, x)$ where $x = b^{f_k(n)} \cdot (q_0, 0)0^{n-1} \cdot b^{f_k(n)}$.

But $\mathrm{Comp}(\mathfrak{M}_k, x)$ is defined since $\mathfrak{M}_k$ halts on input 0^n within $t_k(n) \leq f_k(n)$ tape squares. Moreover, $\ell(\mathrm{Comp}(\mathfrak{M}_k, x)) \geq t_{k+1}(n)$ since $\mathfrak{M}_k$ runs for at least $t_{k+1}(n)$ steps. Hence, the output of the Σ-t_{k+1}-TM is simply $\gamma(\beta)$.

Since by hypothesis α is obtainable in space $p_1(n)$ for some polynomial p_1, and similarly β is obtainable in space $p_2(n+1 + p_1(n))$ for some polynomial p_2, the entire process requires only polynomial space. Q.E.D.

Lemma 6: For any set $A \subset \{0,1\}^*$, if $\mathrm{Comp}(A) \leq t_k$ (a.e.) for some $k \geq 0$, then there is a finite Σ such that A eff Empty(Σ).

Proof: Let $\mathfrak{M}$ be a Turing machine which recognizes $\{0,1\}^*$-A and for every $x \in \{0,1\}^*$, $\mathfrak{M}$ halts using at most $t_k(\ell(x))$ tape squares. By Fact 2, there is such an $\mathfrak{M}$.

Choose Σ as in the simulation lemma applied to $\mathfrak{M}$.

The Turing machine which efficiently reduces A to Empty (Σ) operates as follows: given $x \in \{0,1\}^*$, use a Σ-t_k-TM to obtain a γ-expression α such that $L(\alpha) = \Sigma^{f_k(n)}$ for $n = \ell(x)$. Apply $\mathcal{F}(\mathfrak{M})$ of the simulation lemma to $(q_0, u) \cdot w \cdot \# \cdot \alpha$ where q_0 is the start state of $\mathfrak{M}$, and $x = uw$ for $u \in \{0,1\}$, $w \in \{0,1\}^*$. (We ignore the case $x = \lambda$.) This yields a γ-expression β which we claim is the desired output.

Since $\mathfrak{M}$ requires space at most $t_k(n)$, we conclude that $\mathrm{Comp}(\mathfrak{M}, y)$ where $y = b^{f_k(n)} \cdot (q_0, u) \cdot w \cdot b^{f_k(n)}$ is nonempty iff x is accepted by $\mathfrak{M}$. Hence $x \in A \Leftrightarrow x$ is not accepted by $\mathfrak{M} \Leftrightarrow \mathrm{Comp}(\mathfrak{M}, y) = \emptyset \Leftrightarrow L(\beta) = \emptyset \Leftrightarrow \beta \in$ Empty(Σ). This verifies our claim that β is a correct output.

As in the preceding lemma, the Turing machine transforming x to β requires space at most a polynomial in $\ell(x)$. Q.E.D.

This completes the lemmas required for Theorem 2.

It is not hard to extend this argument to obtain Theorem 1. We use a stronger form of Fact 3 due to Blum [1] to obtain from the proof of Theorem 2 more information about the frequency of the (i.o.) condition in the statement that Comp(WSIS) > t_k (i.o.).

Theorem 3: The following decidable full and weak second order theories are not elementary-recursive: two successors, countable linear order, countable well-order, unary function with countable domain, unit interval under $\leq$. Also, first order theory of two successors with length and prefix predicates, and the first order theory of $\langle N,+,P\rangle$, where $P(x,y) \equiv$ [x is a power of two and x divides y], are decidable but not elementary.†

These results follow by reasonably straightforward efficient reductions of WSIS to each of these theories.

γ-expressions are themselves of interest as a decidable but non-elementary word problem.

Corollary: Empty($\{0,1\}$) is not elementary-recursive.

Further remarks:

(1) The results and methods described here were developed in May, 1972. [9] This paper is a revised version of a preliminary report with the same title written at that time. Since then, in collaboration with

† The decidability of these theories is shown in [6,12].

M.J. Fischer, M.O. Rabin, and L. Stockmeyer, J. Ferrante and C. Rackoff, close upper and lower bounds on space or time have been obtained for most of the classical decidable theories in logic as well as for various notations related to γ-expressions.

Some of the more interesting results to appear in forthcoming papers are

(i) (Meyer) The satisfiability problem for sentences in the first order theory of linear order is not elementary; in fact space $t_{\epsilon \cdot n}(n)$ is required for some $\epsilon > 0$. <u>WSIS</u> also requires this much space. An upper bound $t_{c \cdot n}(n)$ follows from Rabin's proof that S 2 S is decidable [12].†

(ii) (Stockmeyer) The emptiness problem for expressions involving only the operation of $\cup$, $\cdot$, $\neg$ is not elementary, that is, the γ-operation is unnecessary. The simulation lemma and its proof become considerably more subtle.

(iii) (Fischer-Rabin) Any decision procedure for the first-order theory of $\langle N,+\rangle$, that is, Presburger's arithmetic, requires $t_2(\epsilon \cdot n)$ steps even on nondeterministic Turing machines. Ferrante and Rackoff[7], following Cooper[5] and Oppen[11], have established an upper bound of space $t_2(\epsilon \cdot n)$.

(iv) (Fischer-Rabin) Any decision procedure for the first order theory of $\langle N-\{0\},\cdot\rangle$ requires time $t_3(\epsilon \cdot n)$ even on nondeterministic Turing machines. Rackoff has shown that space $t_3(c \cdot n)$ is sufficient.

† In [12], Rabin inaccurately claims his decision procedure is elementary. In a personal communication, he has informed me that he was aware that his procedure required space $t_{c \cdot n}(n)$, but that he misunderstood the definition of elementary.

(v) (Fischer) Let $\mathbb{S}$ be any class of structures with a binary associative operator $*$ and the property that for arbitrarily large n there exists $s \in S \in \mathbb{S}$ such that

$$s^n \neq s^m \text{ for } 1 \leq m < n,$$

where $s^m = \underbrace{s * s * \cdots * s}_{m}$. Then any decision procedure for satisfiability over $\mathbb{S}$ of sentences in the first order language of $*$ requires $t_1(\epsilon \cdot n)$ steps. This general result applies to nearly all the familiar decidable theories in logic, except for the propositional calculus and pure equality.

(vi) (Meyer) The decision problem for satisfiability of sentences in monadic predicate calculus with only seven (approximately) quantifiers requires time $t_1(\epsilon \cdot n)$ even on nondeterministic Turing machines; time $t_1(c \cdot n)$ is achievable on nondeterministic Turing machines.

(vii) (Fischer-Meyer) The decision problem for satisfiability of sentences in the first order language of a single monadic function is not elementary.

(2) Abstract complexity theory has been open to the criticism of being unable to exhibit "natural" decision problems in which phenomena such as speed-up appeared. Applying Blum's results [2] on effective speed-up to our simulation of Turing machines via <u>WSIS</u>, we can show that given <u>any</u> decision procedure for <u>WSIS</u>, one can <u>effectively</u> construct a new decision procedure for <u>WSIS</u> which is much faster (faster by t_k for any k) than the given procedure on at least one

sentence of length n for all sufficiently large integers n. Similar results apply to the other decision procedures mentioned above.

(3) The relation eff can be characterized in a manner similar to the definition of the elementary functions or the primitive recursive functions. $\varepsilon^{2.5}$, so called because it lies properly between the Grzegorczyk classes ε^2 and ε^3, is defined inductively as follows:

1. $x \dot{-} y$, $x+y$, $x \cdot y$, $x^{\lfloor \log_2 y \rfloor} \in \varepsilon^{2.5}$,
2. $\varepsilon^{2.5}$ is closed under explicit transformation (substituting constants and renaming or identifying variables),
3. $\varepsilon^{2.5}$ is closed under composition of functions, and
4. $\varepsilon^{2.5}$ is closed under limited recursion, limited sum and limited minimization.[†]
5. That's all.

If we identify words in Σ^* with the integers they represent in $|\Sigma|$-adic notation, and for any set $A \subset \Sigma^*$ let $C_A: N \rightarrow \{0,1\}$ be the characteristic function of the set of integers identified with A, then B eff A if and only if $C_A(x) = C_B(f(x))$ for some $f \in \varepsilon^{2.5}$ and all $x \in N$.

Essentially $\varepsilon^{2.5}$ provides a high-level programming language in which one can formally express the procedures we informally claimed could be carried out by polynomial space-bounded Turing machines. In this manner our proof could be presented in a completely formal fashion without appeal to intuition about the space requirements of computations. We prefer the latter approach.

[†]See Grzegorczyk's paper for definitions. [8]. Closure under limited recursion actually implies closure under limited sum and limited minimization.

Acknowledgments: Larry Stockmeyer's proof, that any problem decidable in nondeterministic polynomial time is deterministic polynomial time reducible to the regular expressions not equal to Σ^*, provided the key idea of the simulation lemma. Jeanne Ferrante and Charles Rackoff worked out the efficient reductions of WSIS mentioned in Theorem 3. Patrick Fischer correctly suggested that the use of * in my original proof was inessential.

My colleague Michael J. Fischer's suggestions and attention were extremely helpful, as they invariably have been in the past.

October, 1973
Cambridge, Mass.

REFERENCES

1. Blum, M. A machine-independent theory of the complexity of recursive functions, Jour. Assoc. Comp. Mach., 14, 2 (April, 1967), 322-336.

2. Blum, M. On effective procedures for speeding up algorithms, Jour. Assoc. Comp. Mach., 18, 2 (April, 1971), 290-305.

3. Büchi, J.R. and C.C. Elgot, Decision problems of weak second order arithmetics and finite automata, Part I, (abstract), AMS Notices, 5 (1959), 834.

4. Büchi, J.R. Weak second order arithmetic and finite automata, Zeit. f. Math. Log. and Grund. der Math., 6 (1960), 66-92.

5. Cooper, D.C. Theorem-proving in arithmetic without multiplication, Computer and Logic Group Memo. No. 16, U.C. of Swansea, April, 1972, to appear in Machine Intelligence 7.

6. Elgot, C.C. and M.O. Rabin, Decidability and undecidability of extensions of second (first) order theory of (generalized) successor, Jour. Symb. Logic, 31, 2 (June, 1966), 169-181.

7. Ferrante, J. and C. Rackoff, A decision procedure for the first order theory of real addition with order, Project MAC Tech. Memo 33, Mass. Inst. of Technology (May, 1973), 16pp., to appear SIAM Jour. Comp.

8. Grzegorczyk, A. Some classes of recursive functions, Rozprawy Matematyczne, 4 (1953), Warsaw, 1-45.

9. Meyer, A.R. Weak SIS cannot be decided (abstract 72T-E67), AMS Notices, 19, 5 (August, 1972), p. A-598.

10. Meyer, A.R. and L.J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, 13th Switching and Automata Theory Symp. (Oct. 1972), IEEE, 125-129.

11. Oppen, D.C. Elementary bounds for Presburger arithmetic, 5th ACM Symp. Theory of Computing (April, 1973), 34-37.

12. Rabin, M.O. Decidability of second-order theories and automata on infinite trees, Trans. AMS, 141 (July, 1969), 1-35.

13. Rabin, M.O. and D. Scott, Finite automata and their decision problems, IBM Jour. Research and Development, 3 (1959), 115-125.

14. Ritchie, R.W. Classes of predictably computable functions, Trans. AMS, 106 (1963), 139-173.

15. Stearns, R.E., J. Hartmanis, and P.M. Lewis, III, Hierarchies of memory-limited computations, 6th Switching Theory and Logical Design Symp. (1965), IEEE, 179-190.

16. Stockmeyer, L.J. and A.R. Meyer, Word problems requiring exponential time, 5th ACM Symp. Theory of Computing (April, 1973), 1-9.