The Complexity of Monadic Recursion Schemes: Exponential Time Bounds*

H. B. Hunt III and D. J. Rosenkrantz

Computer Science Department, State University of New York at Albany,
Albany, New York 12222

Received August 23, 1982; revised April 3, 1983

We study the computational complexity of decision problems for the class $\mathcal{R}$ of monadic recursion schemes. By the "executability problem" for a class $\mathcal{R}$ of monadic recursion schemes, we mean the problem of determining whether a given defined function symbol of a given scheme in $\mathcal{R}$ can be called during at least one computation. The executability problem for a class $\mathcal{C}$ of very simple monadic recursion schemes is shown to require deterministic exponential time. Using arguments about executability problems and about the class $\mathcal{K}$, a number of decision problems for $\mathcal{R}$ and for several of $\mathcal{K}$'s subclasses are shown to require deterministic exponential time. Deterministic exponential time upper bounds are also presented for several of these decision problems.

1. INTRODUCTION

Monadic recursion schemes, also called monadic functional schemes, are an extension of the single-variable program schemes [15, 16] that allow recursive function calls. They have been studied by a number of authors [2, 4–9], etc. Much of this work has dealt with the decidability, rather than the computational complexity, of their decision problems. However, the computational complexity of decision problems for the single-variable program schemes and for the linear monadic recursion schemes was studied in [11] and [12]. Here and in [13], we study the computational complexity of decision problems for the class $\mathcal{R}$ of monadic recursion schemes. Using the concept of "executability problems" for monadic recursion schemes developed in [13], we present the outline of a complexity theory for decision problems for $\mathcal{R}$.

We show that the executability problem for a class $\mathcal{C}$ of very simple monadic recursion schemes requires deterministic exponential time. Using extensions of the arguments about executability problems in [13], we show that deterministic

* A preliminary version of some of these results was presented at the 21st Annual Symposium on Foundations of Computer Science (October 1980).

† Research of the first author was supported in part by NSF Grants MCS 77-27197 and MCS 80-03353. Research of the second author was supported in part by NSF Grant MCS 78-03157.
exponential time lower bounds also hold for a number of decision problems for \( \mathcal{M} \) and several of its subclasses. These problems include the following:

1. the strong and weak computational identity problems,
2. the isomorphism problem,
3. the strong equivalence problem,
4. the divergence problem, and
5. the problems of testing if a monadic recursion scheme is strongly equivalent to a monadic single-variable program scheme, a linear monadic recursion scheme, or a free monadic recursion scheme.

(The totality problem for \( \mathcal{M} \) is shown to be \( \text{co-AV} \)-complete in \([13]\).) Deterministic exponential time upper bounds are presented for the executability problem, the strong and weak computational identity problems, the isomorphism problem, and the divergence problems for \( \mathcal{M} \). These latter results show that our lower bounds are fairly "tight" for a number of decision problems for \( \mathcal{M} \).

The rest of this section consists of definitions and notation about strings, relations, computational complexity, auxiliary pushdown machines \([3]\), and context-free grammars. Section 2 consists of definitions and properties of monadic recursion schemes.

We denote the length of a string \( S \) or the cardinality of a set \( S \) by \( |S| \). We denote the empty string by \( \lambda \).

**Definition 1.1.** Let \( D \) be a nonempty set. Let \( \rho, \sigma, \) and \( \tau \) be binary relations on \( D \) such that

(i) if \( x\rho y \) then \( x\sigma y \), and

(ii) if \( x\sigma y \) then \( x\tau y \).

Then we say that the relation \( \sigma \) is between \( \rho \) and \( \tau \).

**Definition 1.2.** Let \( \Sigma \) and \( A \) be finite nonempty alphabets. Let \( L \subset \Sigma^* \), and let \( M \subset A^* \). We say that \( L \) is polynomially reducible to \( M \) if and only if there exists a function \( f \) from \( \Sigma^* \) to \( A^* \) computable by a deterministic polynomially time-bounded Turing machine such that, for all \( x \in \Sigma^* \), \( x \in L \) if and only if \( f(x) \in M \).

By a deterministic linearly space-bounded auxiliary pushdown machine, we mean a deterministic linearly bounded automaton augmented with an auxiliary pushdown store. Such a machine \( M \) is specified in terms of:

1. a finite set \( Q \) of states,
2. a finite input tape alphabet \( \Sigma \),
3. a finite pushdown store alphabet \( \Gamma \),
4. a start state \( q_0 \in Q \).
5. two distinct endmarkers $\vdash$ and $\dashv$ not in $\Sigma$,
6. a bottom of stack marker $Z_0 \in \Gamma$,
7. a finite set $F \subset Q$ of accepting states, and
8. a transition function $\delta$

from $Q \times (\Sigma \cup \{\vdash, \dashv\}) \times \Gamma$ to $Q \times (\Sigma \cup \{\vdash, \dashv\}) \times \Gamma^* \times \{0, 1, -1\}$.

We interpret the transition $\delta(s, a, z) = (t, b, \eta, p)$ as follows—when in state $s$, scanning input tape symbol $a$, and having top stack symbol $z$, the machine can in one move

i. change state to $t$,
ii. replace its scanned input symbol by $b$,
iii. replace its top stack symbol by the string $\eta$, and
iv. move its input tape head one square to the left, move its input tape head one square to the right, or keep its input tape head stationary if $p = -1$, $p = 1$, or $p = 0$, respectively.

The transition function $\delta$ preserves the integrity of endmarkers as follows:

a. if $a = \vdash$ then $b = \vdash$ and $p \in \{0, 1\}$,

b. if $a = \dashv$ then $b = \dashv$ and $p \in \{0, -1\}$, and

c. if $a \in \Sigma$ then $b \in \Sigma$.

A configuration of a deterministic linearly space-bounded auxiliary pushdown machine $M$ is a four-tuple $(s, z, y, j)$, where $s \in Q$, $z = \vdash x \dashv$ for some $x \in \Sigma^*$, $y \in \Gamma^*$, and $0 \leq j \leq |x| + 1$. A transition between configurations denoted by

$(s, \vdash x \dashv, y, j) \xrightarrow{M} (t, \vdash y \dashv, \xi, k)$

holds if and only if

i. the $j$th symbol of the string $\vdash x \dashv$ is $a$,

ii. the rightmost symbol of the string $y$ is $z$,

iii. $\delta(s, a, z) = (t, b, \eta, p)$,

iv. the string $\vdash y \dashv$ is the string $\vdash x \dashv$ with its $j$th symbol replaced by $b$,

v. the string $\xi$ is the string $y$ with its rightmost symbol replaced by $\eta$, and

vi. $k = j + p$.

For configurations $\alpha$ and $\beta$ of $M$, if $\beta$ is obtained from $\alpha$ by means of a sequence of $i$ transitions, we denote this by

$\alpha \xrightarrow{M}^i \beta$.

The language that $M$ accepts by final state and empty stack is the set of all string $x \in \Sigma^*$ such that

$$(q_0, \vdash x \dashv, Z_0, 1) \xrightarrow{M}^i (q_f, \vdash y \dashv, \lambda, k)$$

for some nonnegative integer $i$, state $q_f \in F$, string $y \in \Sigma^*$, and nonnegative integer $k$. 
The deterministic exponential time lower bounds presented here are based upon well-known time hierarchy results for deterministic Turing machines [10] and the following property of deterministic linearly space-bounded auxiliary pushdown machines due to Cook [3].

**PROPOSITION 1.3.** The class of all languages accepted by deterministic linearly space-bounded auxiliary pushdown machines equals the class of all languages accepted by deterministic Turing machines that operate within time $2^{cn}$ for some $c > 0$.

Inspection of the proof of Proposition 1.3 in [3] (in particular the proof that (c) ⇒ (a) of Theorem 1 in [3]) and known time hierarchy results for deterministic Turing machines [10] yield the following corollary of Proposition 1.3 and its proof.

**PROPOSITION 1.4.** The exists $c > 0$ and a deterministic linearly space-bounded auxiliary machine $N$ such that

1. the recognition of the language $L$ accepted by $N$ requires more than $2^{cn}$ steps infinitely often on any deterministic Turing machine,
2. the machine $N$ halts for all inputs, and
3. $N$'s input tape alphabet is $\{0, 1\}$.

Finally, we assume that the reader is familiar with the definitions and basic properties of context-free grammars, otherwise see [10]. We specify a context-free grammar $G$ in terms of

1. a finite set $N$ of nonterminals,
2. a finite set $\Sigma$ of terminals,
3. a symbol $S \in N$ called the start symbol of $G$, and
4. a finite set of productions of the form $A \rightarrow \omega$, where $A \in N$ and $\omega \in (N \cup \Sigma)^*$.

The size of a context-free grammar $G$, denoted by $\|G\|$, equals the sum of the number of symbols occurring in its productions. We denote the language generated by a context-free grammar $G$ by $L(G)$.

2. **MONADIC RECURSION SCHEMES—DEFINITIONS, PROPERTIES, AND CONSTRUCTIONS**

In this section we present the basic definitions, notation, and properties of monadic recursion schemes, interpretations, and computations used in this paper.

We assume that DFS, BFS, and PS are pairwise disjoint countably infinite sets called the sets of *defined function symbols*, *basis function symbols*, and *predicate symbols*, respectively. A defining statement is a string of the form

$$ Fx := \text{if } px \text{ then } ax \text{ else } bx $$  

(*)
where :=, if, then, and else are thought of as single symbols, \( F \in \text{DFS} \), \( p \in \text{PS} \), and \( \alpha \) and \( \beta \) are elements of \((\text{DFS} \cup \text{BFS})^*\). A defining statement of the form (*) is called a defining statement for (or of) \( F \). The strings \( \alpha \) and \( \beta \) are called the embedded strings of (*).

**Definition 2.1.** A monadic recursion scheme \( S = (\text{DFS}(S), \text{BFS}(S), \text{PS}(S), F_0, \text{DS}(S)) \), where

1. \( \text{DFS}(S) \) is a finite subset of \( \text{DFS} \),
2. \( \text{BFS}(S) \) is a finite subset of \( \text{BFS} \),
3. \( \text{PS}(S) \) is a finite subset of \( \text{PS} \),
4. \( F_0 \in \text{DFS}(S) \) is called the initial defined function symbol of \( S \), and
5. \( \text{DS}(S) \) is a finite set of defining statements of the form

\[
F_x := \text{if } px \text{ then } ax \text{ else } \beta x
\]

where \( F \in \text{DFS}(S) \), \( p \in \text{PS}(S) \), \( \alpha \in (\text{DFS}(S) \cup \text{BFS}(S))^* \), \( \beta \in (\text{DFS}(S) \cup \text{BFS}(S))^* \), and there is exactly one defining statement in \( \text{DS}(S) \) for each \( F \in \text{DFS}(S) \).

We represent monadic recursion schemes, henceforth also called schemes, by finite lists of defining statements with the defining statement for the initial defined function symbol first. Since there is a defining statement for each defined function symbol of a scheme \( S \) in any representation of \( S \), it is easy to infer from a representation of \( S \) which symbols in an embedded string are defined function symbols and which are basis function symbols. We sometimes represent a defining statement of the form

\[
F_x := \text{if } px \text{ then } ax \text{ else } \beta x
\]

by

\[
F_x := ax.
\]

We also sometimes taken the notational liberty of using nested “if-then-else” statements.

**Definition 2.2.** The size of a monadic recursion scheme \( S \), denoted by \( \| S \| \), is the number of symbols appearing in the defining statements of \( S \).

The reader should recall that we think of :=, if, then, and else as single symbols. The meaning of monadic recursion schemes is defined in terms of interpretations, configurations, and computations in the standard manner.

**Definition 2.3.** An interpretation \( I \) consists of

1. a nonempty set \( D \) called the domain of \( I \),
2. an assignment of an element of \( D \) to the symbol “\( x \),”
3. an assignment of a function \( f_i \) from \( D \) to \( D \) to each basis function symbol \( f \), and

4. an assignment of a predicate \( p_i \) from \( D \) to \{True, False\} to each predicate symbol \( p \).

**Definition 2.4.** A free (or Herbrand) interpretation \( I \) is an interpretation such that

1. the domain of \( I \) is \( \text{BFS}^* \cdot \{x\} \),
2. the assignment of the letter "\( x \)" to the symbol "\( x \),"
3. the assignment of a function \( f_i \) from \( \text{BFS}^* \cdot \{x\} \) to \( \text{BFS}^* \cdot \{x\} \) to each basis function symbol \( f \), where \( f_i \) is defined by \( f_i(\alpha \cdot x) \) equals the string "\( f \cdot \alpha \cdot x \)," and
4. the assignment of a predicate \( p_i \) from \( \text{BFS}^* \cdot \{x\} \) to \{True, False\} to each predicate symbol \( p \).

**Definition 2.5.** Let \( S \) be a monadic recursion scheme. We denote the set \( \text{BFS}(S)^* \cdot \{x\} \) by \( \mathcal{A}[S] \). Let \( I \) be a free interpretation. A configuration of \( S \) under \( I \) is a triple \( \langle w, y, Q \rangle \), where \( w \in [\text{DFS}(S) \cup \text{BFS}(S)]^* \), \( y \in \mathcal{A}[S] \), and \( Q = \{ p \in \text{PS}(S) | p_i(y) = \text{True} \} \).

The binary relation \( \vdash \), on the set of configurations of \( S \) under \( I \) is defined by

\[
\langle w, y, Q \rangle \vdash \langle w', y', Q' \rangle \quad \text{if and only if}
\]

1. \( w = v \cdot f \), where \( f \in \text{BFS}(S) \), \( w' = v \), \( y' = f \cdot y \), and \( Q' = \{ p \in \text{PS}(S) | p_i(y') = \text{True} \} \) or
2. \( w = v \cdot F \), where \( F \in \text{DFS}(S) \), the defining equation for \( F \) in \( S \) is

\[
Fx := \text{if } px \text{ then } ax \text{ else } bx
\]

and either

(i) \( p \in Q, w' = v \cdot \alpha, y' = y \), and \( Q' = Q \), or
(ii) \( p \not\in Q, w' = v \cdot \beta, y' = y \), and \( Q' = Q \).

**Definition 2.6.** The computation of a monadic recursion scheme \( S \) under a free interpretation \( I \) is the sequence of configurations of \( S \) under \( I \)

\[
c_0, c_1, \ldots, c_k, \ldots
\]

having the following properties:

1. \( c_0 = \langle F_0, x, \{ p \in \text{PS}(S) | p_i(x) = \text{True} \} \rangle \), where \( F_0 \) is the initial defined function symbol of \( S \).
2. For all \( i \geq 0 \) such that the sequence has a term \( c_{i+1}, c_i \vdash c_{i+1} \).
3. If the sequence is finite, then its last term is \( \langle \lambda, y, R \rangle \) for some \( y \in \mathcal{A}[S] \) and \( R \subset \text{PS}(S) \).
If \( c_i = \langle w \cdot F, y, Q \rangle \), where \( F \in \text{DFS}(S) \) and the defining statement for \( F \) in \( S \) is

\[
F_x := \text{if } px \text{ then } ax \text{ else } bx,
\]

we say that \( F \) is called at the \( i \)th step of the computation of \( S \) under \( I \). If in addition \( p \in Q \), we say that \( \alpha \) is the selected embedded string of the call. Otherwise, we say that \( \beta \) is the selected embedded string of the call.

Let \( S \) be a scheme. Let \( I \) be a free interpretation. Let \( c_0, c_1, \ldots, c_k \ldots \) be the computation of \( S \) under \( I \). Then it is easy to verify that \( c_j = \langle w, y, Q \rangle \) if and only if the following hold.

After \( j \) steps of the computation of \( S \) under \( I \),

1. \( w = w_1 \cdot w_2 \cdot \ldots \cdot w_m \), where \( w_1, \ldots, w_m \) are the basis and defined function symbols not yet expanded with \( w_m \) the next function symbol to be expanded;
2. the value of \( x \) is \( y \); and
3. \( Q = \{ p \in \text{PS}(S) \mid p(I)(y) = \text{True} \} \).

Examples of a scheme and a corresponding computation under a free interpretation appear in Figure 1. The definitions of a configuration and a computation for arbitrary interpretations can be obtained by extending Definitions 2.5 and 2.6 in the obvious manner.

**Definition 2.7.** We say that two strings \( \alpha \) and \( \beta \) in \( (\text{DFS} \cup \text{BFS})^* \) are compatible if and only if \( |\alpha| = |\beta| \) and, for each \( i \) with \( 1 \leq i \leq |\alpha| \), either the \( i \)th symbol of \( \alpha \) and the \( i \)th symbol of \( \beta \) are the same basis function symbol, or they both are defined function symbols.

**Definition 2.8.** Let \( S \) and \( T \) be monadic recursion schemes.

1. The scheme \( S \) is divergent if, for all interpretations \( I \), the computation of \( S \) under \( I \) diverges. The scheme \( S \) is nondivergent if it is not divergent.

2. The schemes \( S \) and \( T \) are strongly computationally identical if, for all interpretations \( I \), the sequences of defining statements called during the computations of \( S \) and of \( T \) under \( I \) are identical (even having identical names of defined function symbols).

\[
F_x := \text{if } px \text{ then } x \text{ else } fFgx
\]

\[
G_x := \text{if } qx \text{ then } Ggx \text{ else } fx
\]

\[
\langle F, x, \{q\} \rangle, \langle fFg, x, \{q\} \rangle, \langle fFgg, x, \{q\} \rangle, \langle fFg, hx, \{p\} \rangle,
\]

\[
\langle fFg, hx, \{p\} \rangle, \langle fFg, fhx, \{\} \rangle, \langle fFg, fhx, \{\} \rangle, \langle fFg, fhx, \{p, q\} \rangle,
\]

\[
\langle f, ffhx, \{p, q\} \rangle, \langle f, ffhx, \{p, q\} \rangle
\]

**Figure 1**
3. The schemes \( S \) and \( T \) are *weakly computationally identical* if, for all interpretations \( I \), the sequences of defining statements called during the computations of \( S \) and \( T \) under \( I \) have identical names, identical predicate symbols, and identical selected embedded strings (but not necessarily identical unselected embedded strings).

4. The schemes \( S \) and \( T \) are *isomorphic* if, for all interpretations \( I \), the sequences of defining statements called during the computations of \( S \) and of \( T \) under \( I \) have identical predicate symbols and compatible selected embedded strings (although names of defined function symbols can differ).

5. The schemes \( S \) and \( T \) are *strongly equivalent* if, for all interpretations \( I \), either both the computations of \( S \) and of \( T \) under \( I \) diverge or both halt with the same values of \( x \).

6. The scheme \( S \) is *contained by* the scheme \( T \) if, for all interpretations \( I \), whenever the computation of \( S \) under \( I \) halts, the computation of \( T \) under \( I \) halts with the same value of \( x \).

7. The schemes \( S \) and \( T \) are *weakly equivalent* if, for all interpretations \( I \) for which both the computations of \( S \) and of \( T \) under \( I \) halt, \( S \) and \( T \) halt with the same values of \( x \).

The differences between the definitions of strong computational identity, weak computational identity, and isomorphism can be seen by comparing the schemes \( F \), \( G \), \( H \), and \( K \) of Figure 2. The schemes \( F \) and \( G \) are strongly computationally identical. The schemes \( F \) and \( H \) are weakly computationally identical but are not strongly computationally identical. The schemes \( F \) and \( K \) are isomorphic but are not weakly computationally identical.

**Definition 2.9.** Let \( \mathcal{E} \) be a class of monadic recursion schemes.

1. The *executability problem* for \( \mathcal{E} \) is the problem of determining, for \( S \in \mathcal{E} \) and defined function symbol \( B \) of \( S \), if \( B \) is called during some computation of \( S \).

1. The scheme \( F \)
   \[
   F_0 x := \text{if } p, x \text{ then } F_1 x \text{ else } x \\
   F_1 x := \text{if } p, x \text{ then } f x \text{ else } F_2 x \\
   F_2 x := \text{if } p, x \text{ then } f x \text{ else } g x
   \]

2. The scheme \( G \)
   \[
   F_0 x := \text{if } p, x \text{ then } F_1 x \text{ else } x \\
   F_1 x := \text{if } p, x \text{ then } f x \text{ else } F_2 x \\
   F_2 x := \text{if } p, x \text{ then } g x \text{ else } h x
   \]

3. The scheme \( H \)
   \[
   F_0 x := \text{if } p, x \text{ then } F_1 x \text{ else } x \\
   F_1 x := \text{if } p, x \text{ then } f x \text{ else } t x
   \]

4. The scheme \( K \)
   \[
   K_0 x := \text{if } p, x \text{ then } K_1 x \text{ else } x \\
   K_1 x := \text{if } p, x \text{ then } f x \text{ else } w x
   \]

**Figure 2**
2. The divergence problem for $\mathcal{C}$ is the problem of determining, for $S \in \mathcal{C}$, if $S$ is divergent.

3. The strong computational identity problem for $\mathcal{C}$ is the problem of determining, for $S, T \in \mathcal{C}$, if $S$ and $T$ are strongly computationally identical. The weak computational, isomorphism, strong equivalence, containment, and weak equivalence problems are defined similarly to the strong computational identity problem.

**Definition 2.10.** Let $S = (\text{DFS}(S), \text{BFS}(S), \text{PS}(S), F_0, \text{DS}(S))$ be a monadic recursion scheme. Then, the value language of $S$, denoted by $\text{VAL}(S)$, is the set
\[
\{w \in [\text{BFS}(S)]^* | \langle F_0, x, P \rangle \text{ and } \langle \lambda, w, Q \rangle \text{ are configurations of } S \text{ under a free interpretation } I; \text{ and } \langle F_0, x, P \rangle \vdash^* \langle \lambda, w \cdot x, Q \rangle \}.
\]

**Definition 2.11.**
1. A monadic recursion scheme $S$ is said to be free if, for every free interpretation $I$, the computation of $S$ under $I$ does not test a predicate with the same term $t \in \mathcal{M}|S|$ more than once.

2. A monadic recursion scheme $S$ is said to be executable if every $F \in \text{DFS}(S)$ is executable in $S$.

3. A monadic recursion scheme $S$ is said to be linear if every embedded string of $S$ has at most one occurrence of a defined function symbol.

**Definition 2.12.** Monadic recursion scheme $S = (\text{DFS}(S), \text{BFS}(S), \text{PS}(S), F_0, \text{DS}(S))$ is an executable subscheme of monadic recursion scheme $T = (\text{DFS}(T), \text{BFS}(T), \text{PS}(T), G_0, \text{DS}(T))$ if $\text{DFS}(S) \subseteq \text{DFS}(T)$, $\text{BFS}(S) \subseteq \text{BFS}(T)$, $\text{PS}(S) \subseteq \text{PS}(T)$, $\text{DS}(S) \subseteq \text{DS}(T)$, and $F_0$ is executable in $T$.

3. **AN EXPONENTIAL TIME LOWER BOUND FOR THE EP**

In this section we prove that a restricted version of the executability problem for a class $\mathcal{C}$ of very simple monadic recursion schemes requires deterministic exponential time. In Section 4 this exponential time lower bound and simple reducibility arguments are used to prove deterministic exponential time lower bounds for a number of decision problems for $\mathcal{M}$.

Before proving that the executability problem requires exponential time, we present an example that illustrates, in a simple setting, some of the techniques used in the proof.

**Example 3.1.** Consider the monadic recursion scheme $S$ with the defining statements given in Figure 3. Note that $f$ is the only basis function symbol of $S$, and that $F$ is the initial defined function symbol.

The values of the predicates $p_1$, $p_2$, $p_3$, and $p_4$ at any time during a computation of $S$ can be viewed as encoding a tape with four cells, each of which can contain a 0
or 1; the value of $p_i$ encodes the leftmost cell. Each defined function $P_i$ can be viewed as a verifier for confirming that cell $i$ of the current encoded tape contains 1. Defined function $P_i$ performs the verification by returning if $p_i$ is true for the current value of the parameter, and looping otherwise. Similarly, defined function $P_i$ is a verifier for confirming that cell $i$ contains 0. When $A$ is called for the first time (from $F$), the predicates have been verified to encode the tape 1000.

When $A$ is called with a parameter whose predicate values encode a tape whose rightmost cell contains 1, $A$ immediately returns. Otherwise, the expansion of $A$ either diverges without calling $A$ again (because the call of $R_1$ diverges) or $A$ calls itself recursively with the parameter value returned by $R_1$. If $R_1$ returns, the tape encoded by the value of the parameter returned is a right circular shift of the tape encoded by the value of the parameter with which $R_1$ is called. Thus $R_1$ either diverges or performs a right circular shift of the encoded tape. $R_1$, $R_2$, $R_3$, and $R_4$ each provide for one cell of the shifted tape. For instance, $R_2$ tests cell 2 of the current encoded tape, and provides for the calling of a verifier to confirm that cell 3 of the next encoded tape will equal cell 2 of the current encoded tape. In addition to providing for checking cell 1 of the next tape, $R_4$ calls basis function $f$ to produce the new encoded tape. If the new encoded tape is indeed a right circular shift of the old encoded tape, the verifiers will all return, and $A$ will be expanded again. Thus, either the computation diverges or $A$ keeps calling itself recursively, each time with a parameter value that encodes a right circular shift of the previously encoded tape, until the rightmost cell of the encoded tape contains 1.

The proof of the next theorem uses the techniques of verifiers, encoding each cell of a tape as the value of a predicate, and ensuring that the computation will diverge if the subsequent encoded tape is not correct.

**Theorem 3.2.** There exists $c > 0$ such that the executability problem for $\mathcal{M}$ requires more than $2^{c^{15\frac{1}{2}}}$ steps infinitely often on any deterministic Turing machine.
Proof. We show that there exists a class \( \mathcal{C} \) of monadic recursion schemes such that

1. for all schemes \( S \in \mathcal{C} \), \( B \) is a defined function symbol of \( S \);
2. for all schemes \( S \in \mathcal{C} \), \( |\text{VAL}(S)| \leq 1 \); and
3. there exists \( c > 0 \) such that the problem of determining for \( S \in \mathcal{C} \), if the statement labeled \( B \) is executed during some computation of \( S \), requires more than \( 2^{c|S|} \) steps infinitely often on any deterministic Turing machine.

The proof is by explicit construction.

Let \( M \) be a deterministic linearly space-bounded auxiliary pushdown machine that halts for all inputs and that has input tape alphabet \( \{0, 1\} \). We show that there exists a constant \( k \) and an \( O(n \log n) \) time-bounded deterministic Turing machine that, given input \( y \in \{0, 1\}^+ \), outputs a monadic recursion scheme \( M[ y] \) satisfying the conditions:

\( \alpha \). \( ||M[ y]|| \leq k \cdot |y| \);
\( \beta \). the string \( y \) is accepted by \( M \) if and only if the defined function symbol \( B \) is executable in \( M[y] \); and
\( \gamma \). \( M[y] \) satisfies conditions 1 and 2 given above.

The proof consists of three parts.

Part 1. Construction of the scheme \( M[y] \).
Without loss of generality, we assume that

i. \( M \)'s state set is \( \{s_1, s_2, \ldots, s_m\} \), where \( s_1 \) is \( M \)'s start state and \( s_m \) is \( M \)'s accepting or final state;
ii. \( M \)'s pushdown store alphabet is \( \Gamma \), where \( Z_0 \subset \Gamma \) is the bottom of stack marker of \( M \);
iii. \( M \)'s transition function is \( \delta \); and
iv. \( M \) accepts by final state and empty pushdown store.

Let \( y = a_1 a_2 \ldots a_n \) be an input to \( M \) where each \( a_i \in \{0, 1\} \). The scheme \( M[y] \) has a single basis function symbol \( f \), has the predicate symbols

a. \( \text{state}_i \), for \( 1 \leq i \leq m \),

b. \( \text{loc}_k \), for \( 0 \leq k \leq n + 1 \), and

c. \( \text{tape}_j \), for \( 1 \leq j \leq n \),

and has the defined function symbols

d. \( \text{STATE}_i \) and \( \text{STATE}_i \), for \( 1 \leq i \leq m \),
e. \( \text{LOC}_k \) and \( \text{LOC}_k \), for \( 0 \leq k \leq n + 1 \),
f. \( \text{TAPEx} \) and \( \text{TAPEx} \), for \( 1 \leq j \leq n \),
g. \( B \),
The starting defined function symbol of $M[y]$ is INIT.

The scheme $M[y]$ will simulate the computation of $M$ on $y$. In this simulation, predicate state$_i(x)$ will be true if and only if $M$ is in state $s_i$, predicate loc$_k(x)$ will be true if and only if $M$ is scanning the $k$th symbol on its input tape, and predicate tape$_j(x)$ will be true if and only if the contents of the $j$th input tape cell of $M$ is 1. The predicate symbols loc$_0$ and loc$_{n+1}$ are introduced to handle the case when $M$ is scanning an endmarker. The defining statements for the defined function symbols of d–f either

- verify that the corresponding predicate is true or cause $M[y]$ to diverge without calling $B$, or
- verify that the corresponding predicate is false or cause $M[y]$ to diverge without calling $B$.

The defining statements for the defined function symbols of d–f are:

\[ d1. \quad \text{for } 1 \leq i \leq m, \]
\[ \text{STATE}_i x := \begin{cases} \text{if} \text{ state}_i x \text{ then } x \text{ else } \text{STATE}_i x \\
\end{cases} \]

and

\[ \text{STATE}_i x := \begin{cases} \text{if} \text{ state}_i x \text{ then } \text{STATE}_i x \text{ else } x, \\
\end{cases} \]

\[ e1. \quad \text{for } 0 \leq k \leq n+1, \]
\[ \text{LOC}_k x := \begin{cases} \text{if} \text{ loc}_k x \text{ then } x \text{ else } \text{LOC}_k x \\
\end{cases} \]

and

\[ \text{LOC}_k x := \begin{cases} \text{if} \text{ loc}_k x \text{ then } \text{LOC}_k x \text{ else } x, \\
\end{cases} \]

and

\[ f1. \quad \text{for } 1 \leq j \leq n, \]
\[ \text{TAPE}_j x := \begin{cases} \text{if} \text{ tape}_j x \text{ then } x \text{ else } \text{TAPE}_j x \\
\end{cases} \]

and

\[ \text{TAPE}_j x := \begin{cases} \text{if} \text{ tape}_j x \text{ then } \text{TAPE}_j x \text{ else } x. \\
\end{cases} \]

The defining statements for the defined function symbols $B$ and INIT are:

\[ g1. \quad B x := x \]
and

h1. \texttt{INIT } x := B \texttt{ FSTATE } Z_0 T_n T_{n-1} \cdots T_1 \texttt{ LOC}_n \texttt{ LOC}_{n-1} \cdots \texttt{ LOC}_2 \texttt{ LOC}_1 \texttt{ LOC}_0 \texttt{ STATE}_m \cdots \texttt{ STATE}_2 \texttt{ STATE}_1 x,

where, for \( 1 \leq i \leq n \), \( T_i \) is \texttt{TAPE}_i if \( a_i = 1 \), and \( T_i \) is \texttt{TAPE}_i if \( a_i = 0 \).

By direct observation of the defining statements d1, e1, f1, and h1, there exists a free interpretation \( I \) such that \( Z_0 \) is called during the computation of \( M[y] \) under \( I \) if and only if the predicates satisfied by \( x \) correspond to the initial configuration of \( M \) on \( y \), i.e.,

for \( 1 \leq i \leq m \), \( (\text{state}_i)_T(x) = \text{True} \) if and only if \( i = 1 \),

for \( 0 \leq j \leq n + 1 \), \( (\text{loc}_j)_T(x) = \text{True} \) if and only if \( j = 1 \), and

for \( 1 \leq k \leq n \), \( (\text{tape}_k)_T(x) = \text{True} \) if and only if \( a_k = 1 \).

Otherwise, the scheme \( M[y] \) diverges without calling \( B \).

For all \( Z \in \Gamma \), the defined function symbol \( Z \) of \( M[y] \) is called during the simulation of \( M \)'s computation on \( y \) when \( M \)'s top pushdown store symbol is \( Z \). The defining statement for \( Z \) initiates a series of tests to determine the current state of \( M \) and the input symbol scanned by \( M \)'s input tape head. The information needed to simulate the next state transition of \( M \) is encoded in the names of the defined function symbols \( F[-,-,Z], F[i,-,Z] \), and \( F[i,a,Z] \). The defining statements for the defined function symbols \( Z, F[-,-,Z], \) and \( F[i,-,Z] \) are:

i1. \( Zx := F[-,-,Z]x \), for each \( Z \in \Gamma \),

j1. \( F[-,-,Z]x := \text{if } \text{state}_1 x \text{ then } F[1,-,Z]x \)
   \( \text{ else if } \text{state}_2 x \text{ then } F[2,-,Z]x \)
   \( \vdots \)
   \( \text{ else if } \text{state}_{m-1} x \text{ then } F[m-1,-,Z]x \)
   \( \text{ else } F[m,-,Z]x, \)

for each \( Z \in \Gamma \),

and

k1. \( F[i,-,Z]z := \text{if } \text{loc}_0 x \text{ then } F[i,-,Z]x \)
   \( \text{ else if } \text{loc}_1 x \text{ then } \)
   \( \text{ if } \text{tape}_1 x \text{ then } F[i,1,Z]x \)
   \( \text{ else } F[i,0,Z]x \)
   \( \text{ else if } \text{loc}_2 x \text{ then } \)
   \( \text{ if } \text{tape}_2 x \text{ then } F[i,1,Z]x \)
   \( \text{ else } F[i,0,Z]x \)
   \( \vdots \)
   \( \text{ else if } \text{loc}_n x \text{ then } \)
   \( \text{ if } \text{tape}_n x \text{ then } F[i,1,Z]x \)
   \( \text{ else } F[i,0,z]x \)
   \( \text{ else } F[i,-,Z]x. \)
By direct inspection, the defined function symbol $F[i, a, Z]$ is called during the simulation of $M$'s computation on $y$ when $M$ is in state $s_i$, is reading an input tape cell containing the symbol $a$, and has a pushdown store whose top symbol is $Z$. The defining statements for the defined function symbols $F[i, a, Z]$ simulate the transition function $\delta$ of $M$. The defining statements for the defined function symbols $F[i, a, Z]$ are:

11. $F[i, a, Z]x := \eta G[r, b, p]x$, where $\delta(s_i, a, Z) = (s_r, b, \eta, p)$, for
    $1 \leq i \leq m$, for $a \in \{0, 1, \_, \_\}$, and for $Z \in \Gamma$.

The defining statements for the defined function symbols $G[r, b, p]$ verify that the predicate symbols $state_j$, for $1 \leq j \leq m$, correctly simulate the state of $M$ that results from a single application of a transition $\delta(s_i, a, Z) = (s_r, b, \eta, p)$. The defining statements for the defined function symbols $G[r, b, p]$ are:

m1. $G[i, b, p]x := \overline{STATE_m} \cdots \overline{STATE_{i+1}} \overline{STATE_i} \overline{STATE_{i-1}} \cdots \overline{STATE_1} H[1; b, p]x$, for $1 \leq i \leq m$, for $b \in \{0, 1, \_, \_\}$, and for $p \in \{0, 1, 1\}$. 

The defining statements for the defined function symbols $H[j, b, p]$ verify that the predicate symbols $tape_i$, for $1 \leq i \leq n$, correctly simulate the contents of $M$'s input tape that results from a single application of a transition $\delta(s_i, a, Z) = (s_r, b, \eta, p)$. The defining statements for the defined function symbols $H[j, b, p]$ are:

n1. For $b \in \{0, 1, \_, \_\}$ and for $p \in \{0, 1, 1\}$, $H[1; b, p]x := \overline{LOC_1} I[0; 0]x \text{ else } TAPE_1 I[2, b, p]x$
    where $T_1$ is $TAPE_1$ if $b = 1$, and $T_1$ is $TAPE_1$ if $b = 0$;
    $H[2, b, p]x := \overline{LOC_2} I[3, b, p]x \text{ else } TAPE_2 H[3, b, p]x$
    where $T_2$ is $TAPE_2$ if $b = 1$, and $T_2$ is $TAPE_2$ if $b = 0$; ...
    and $H[n, b, p]x := \overline{LOC_n} I[0, p]x \text{ else } TAPE_n I[0, p]x$
    where $T_n$ is $TAPE_n$ if $b = 1$, and $T_n$ is $TAPE_n$ if $b = 0$. 

The defining statements for the defined function symbols $I[\epsilon, p]$ verify that the predicate $loc_\epsilon$, for $0 \leq k \leq n + 1$, correctly simulate the location of $M$'s input tape head that results from a single application of a transition $\delta(s_i, a, Z) = (s_r, b, \eta, p)$. The defining statements for the defined function symbols $I[\epsilon, p]$ are:

o1. For $p = 0$,
    for $0 \leq \epsilon \leq n$, $I[\epsilon, 0]x := \overline{LOC_\epsilon} I[\epsilon + 1, 0]x \text{ else } LOC_\epsilon I[\epsilon + 1, 0]x$
and
\[ I[n+1,0]x := \text{if } \text{loc}_{n+1}x \text{ then } \text{LOC}_{n+1}fx \text{ else } \overline{\text{LOC}}_{n+1}fx. \]
For \( p = 1 \),
\[ I[0,1]x := \text{LOC}_0I[1,1]x, \]
for \( 1 \leq i \leq n \),
\[ I[i,1]x := \text{if } \text{loc}_{c-1}x \text{ then } \text{LOC}_cI[i+1,1]x \text{ else } \overline{\text{LOC}}_cI[i+1,1]x, \]
and
\[ I[n+1,1]x := \text{if } \text{loc}_nx \text{ then } \text{LOC}_{n+1}fx \text{ else } \overline{\text{LOC}}_{n+1}fx. \]
For \( p = -1 \),
\[ I[i,-1]x := \text{loc}_{c+1}x \text{ if } \text{LOC}_cI[i+1,-1]x \text{ else } \overline{\text{LOC}}_cI[i+1,-1]x, \]
and
\[ I[n+1,-1]x := \overline{\text{LOC}}_{n+1}fx. \]

Finally, the defining statement for the defined function symbol FSTATE is:

**Part 2.** Verification that \( M[y] \) satisfies conditions \( \alpha, \beta, \) and \( \gamma \).

By direct observation of the defining statements of \( M[y] \), the scheme \( M[y] \) is constructible from \( y \) deterministically in \( O(n \log n) \) time, and there exists a constant \( k \) independent of \( y \) such that \( ||M[y]|| \leq k \cdot |y| \). Thus \( M[y] \) satisfies condition \( \alpha \).

By direct inspection of the defining statements of \( M[y] \), the scheme \( M[y] \) satisfies the following.

(3.2.1) There exists a free interpretation \( I \) such that

\[ (\text{INIT}, x, P) \models_I (B, w, Q) \]

if and only if

\[ P = \{ p \in \text{PS}(M[y]) \mid p = \text{state}_1, p = \text{loc}_1, \text{ or, for } 1 \leq j \leq n, \]
\[ p = \text{tape}_j \text{ and } a_j = 1 \} \]

and

\[ (\text{INIT}, x, P) \not\models_I (B \text{ FSTATE } Z_0, x, P) \not\models_I (B, w, Q). \]

We also claim that \( M, y, \) and \( M[y] \) satisfy the following.

(3.2.2) For all \( i \geq 0 \),

\[ (s_1, \vdash y \vdash, Z_0, 1) \models_M (s_j, \vdash z \vdash, \gamma, k), \]
where $z = z_1 z_2 \cdots z_n$, each $z_h \in \{0, 1\}$, $\gamma \in \Gamma^*$, and $0 \leq k \leq n + 1$, if and only if there exists a free interpretation $I$ such that

$$(B \text{ FSTATE } Z_0, x, P) \xrightarrow{\delta} (B \text{ FSTATE } \gamma, w, Q),$$

where $P$ is as above, $w = f^\beta x$, and

$Q = \{ q \in \text{ PS}(M[y]) \mid q = \text{ state}_j, q = \text{ loc}_k, \text{ or, for } 1 \leq h \leq n, q = \text{ tape}_h \text{ and } z_h = 1 \}.$

The correctness of assertion (3.2.2) follows easily by induction on $i$ from the defining statements of $M[y]$. Combining assertions (3.2.1) and (3.2.2), we have that

the string $\gamma$ is accepted by $M$ if and only if the function symbol $B$ is executable in $M[y]$.

Thus $M[y]$ satisfies condition $\beta$.

By inspection of the defining statement gl of $M[y]$, $B$ is a defined function symbol of $M[y]$. Thus $M[y]$ satisfies condition 1. By inspection of the defining statements for $M[y]$, the computation of $M[y]$ under a free interpretation $I$ halts if and only if the defined function symbol $B$ is executed during the computation of $M[y]$ under $I$.

By inspection of the defining statements of $M[y]$ and by assertions (3.2.1) and (3.2.2), the computation of $M[y]$ under a free interpretation $I$ halts if the computation of $M[y]$ under $I$ correctly "simulates" the computation of $M$ on $\gamma$ and $\gamma \in L(M)$. Otherwise, a call of one of the defined function symbols

$\text{STATE}_i, \overline{\text{STATE}}_i, \overline{\text{LOC}}_k, \overline{\text{LOC}}_k, \overline{\text{TAPE}}_j, \overline{\text{TAPE}}_j$, or $\text{FSTATE}$

causes the computation of $M[y]$ under $I$ to enter an infinite loop. Since the automaton $M$ is deterministic and halts for all inputs, it has at most one valid computation on $\gamma$, say, of length $k > 0$. If $\gamma \in L(M)$, then $\text{VAL}(M[y]) = \{ f^k x \}$. Otherwise, $\text{VAL}(M[y])$ is empty. Thus, $M[y]$ also satisfies condition 2. Hence, $M[y]$ satisfies condition $\gamma$.

Part 3. Definition of $\mathcal{E}$ and verification that it satisfies conditions 1, 2, and 3.

Let $N$ be any deterministic linearly space-bounded auxiliary pushdown machine such that $N$ halts for all inputs, $N$'s input tape alphabet is $\{0, 1\}$, $N$ accepts by accepting state and empty pushdown store, and such that

there exists $c > 0$ for which the recognition of the language $L'$ accepted by $N$ requires more than $2^{cn}$ steps infinitely often on any deterministic Turing machine.

By Propositions 1.3 and 1.4 such an automaton $N$ exists. Let $\mathcal{E} = \{ N[y] \mid y \in \{0, 1\}^+ \}$. Then by Parts 1 and 2, $\mathcal{E}$ satisfies conditions 1, 2, and 3. $\blacksquare$
4. Exponential Time Lower Bounds

We use Theorem 3.2, its proof, and efficient reductions of executability problems to prove deterministic exponential time lower bounds for a number of problems for \( \mathcal{M} \) and for several of \( \mathcal{M} \)'s subclasses.

**Theorem 4.1.** For each of the following problems for \( \mathcal{M} \), there exist \( c > 0 \) such that the problem requires more than \( 2^{c^{|S|}} \) steps infinitely often on any deterministic Turing machine:

1. for all binary relations \( \rho \) on \( \mathcal{M} \) between strong computational identity and weak equivalence, determining, for \( S, T \in \mathcal{M} \), if \( S \rho T \);
2. the divergence problem;
3. testing, for \( S \in \mathcal{M} \), if \( \text{VAL}(S) \) is a finite or regular set;
4. testing, for \( S \in \mathcal{M} \), if \( S \) is strongly equivalent to a monadic single-variable program scheme, to a linear monadic recursion scheme, or to a free monadic recursion scheme; and
5. translating a scheme \( S \in \mathcal{M} \) into a strongly equivalent executable monadic recursion scheme or into a strongly equivalent free monadic recursion scheme.

**Proof.**

i. **Proof of 1.** Let \( N, \gamma, \) and \( N[\gamma] \) be as in the proof of Theorem 3.2. Let \( P[\gamma] \) be the monadic recursion scheme that is identical to \( N[\gamma] \) except that the defining statement for the defined function symbol \( B \) is

\[
Bx := fx.
\]

The defined function symbol \( B \) in \( P[\gamma] \) is executable if and only if the defined function symbol \( B \) in \( N[\gamma] \) is executable. If \( B \) is not executable in \( N[\gamma] \) or in \( P[\gamma] \), then the schemes \( N[\gamma] \) and \( P[\gamma] \) are strongly computationally identical. Otherwise, the schemes \( N[\gamma] \) and \( P[\gamma] \) are not weakly equivalent. Thus for all binary relations \( \rho \) on \( \mathcal{M} \) between strong computational identity and weak equivalence, \( N[\gamma] \rho P[\gamma] \) if and only if \( B \) is not executable in \( N[\gamma] \).

ii. **Proof of 2.** Let \( N, \gamma, \) and \( N[\gamma] \) be as in the proof of Theorem 3.2. Then the scheme \( N[\gamma] \) is divergent if and only if \( B \) is not executable in it.

iii and iv. **Proof of 3 and 4.** Let \( S1, S2, \) and \( S3 \) be the monadic recursion schemes in Figure 4. Let \( N, \gamma, \) and \( N[\gamma] \) be as in the proof of Theorem 3.2. Without loss of generality, we assume that no predicate symbol or defined function symbol appears in both \( N[\gamma] \) and \( S1 \), in both \( N[\gamma] \) and \( S2 \), or in both \( N[\gamma] \) and \( S3 \). Let \( P[\gamma], Q[\gamma], \) and \( R[\gamma] \) be the monadic recursion schemes that result from \( N[\gamma] \) by replacing the defining statement for \( B \) by

\[
Bx := Fhx
\]
a. The scheme $S_1$
\[ Fx := \text{if } px \text{ then } fFgx \text{ else } x \]

b. The scheme $S_2$
\[ Fx := \text{if } px \text{ then } fGgf'Ggx \text{ else } x \]
\[ Gx := \text{if } px \text{ then } fGgx \text{ else } x \]

c. The scheme $S_3$
\[ Fx := \text{if } px \text{ then } GFfx \text{ else } x \]
\[ Gx := \text{if } qx \text{ then } fx \text{ else } x \]

and by adding the defining statements of $S_1$, of $S_2$, and of $S_3$, respectively. Then the following statements are equivalent:

a. the defined function symbol $B$ is not executable in $N[y]$;
b. $\text{VAL}(P[y])$ is finite;
c. $\text{VAL}(P[y])$ is regular;
d. the scheme $P[y]$ is strongly equivalent to a monadic single-variable program scheme;
e. the scheme $Q[y]$ is strongly equivalent to a linear monadic recursion scheme; and
f. the scheme $R[y]$ is strongly equivalent to a free monadic recursion scheme.

We prove that statement a is equivalent to each of the statements b–f. If statement a is true, then the schemes $N[y]$, $P[y]$, $Q[y]$, and $R[y]$ are divergent. Thus, statements b–f are trivially true. If statement a is false, then

g. $\text{VAL}(P[y])$ is not a regular set; and
h. $\text{VAL}(Q[y])$ is not a linear context-free language.

By known properties of value languages (e.g., Theorem 2.5 in [9]), statements g and h imply that

the scheme $P[y]$ is not strongly equivalent to a monadic single-variable program scheme;

and

the scheme $Q[y]$ is not strongly equivalent to a linear monadic recursion scheme.

By Theorem 2 in [2], the scheme $S_3$ is not strongly equivalent to a free monadic recursion scheme. But if $B$ is executable in $N[y]$, then $B$ and $F$ are both executable in $R[y]$, and so scheme $R[y]$ is not strongly equivalent to any free monadic recursion scheme. Thus, statements b–f are false.
v. Proof of 5. Let N, y, and N[y] be as in the proof of Theorem 3.2. Let P[y] be the monadic recursion scheme that results from N[y] by replacing the defining statement for B by

\[ Bx := hx. \]

Then B is executable in N[y] if and only if any (also each) executable monadic recursion scheme strongly equivalent to P[y] has a defining statement in which the basis function symbol h appears.

Additionally B is executable in N[y] if and only if any free monadic recursion scheme strongly equivalent to P[y] has an executable defining statement in which the basis function symbol h appears. But the executability problem for free monadic recursion schemes is easily seen to be decidable deterministically in polynomial time.

Binary relations \( \rho \) on \( \mathcal{M} \) between strong computational identity and weak equivalence include strong computational identity, weak computational identity, isomorphism, strong equivalence, containment, and weak equivalence. Thus 1 of Theorem 4.1 implies that testing any of these relations on \( \mathcal{M} \) requires deterministic exponential time. Moreover, the proofs of Theorems 3.2 and 4.1 also imply that deterministic exponential time lower bounds hold for restrictions of the problems of Theorem 4.1 to a number of subclasses of \( \mathcal{M} \), including subclasses of \( \mathcal{M} \) for which such problems as containment and weak equivalence are decidable. (The containment and weak equivalence problems for \( \mathcal{M} \) are known to be undecidable [6].)

**Theorem 4.2.** Let \( \mathcal{L} \) be one of the following subclasses of \( \mathcal{M} \):

1. the set of \( S \in \mathcal{M} \) such that \(|VAL(S)| \leq 1\);
2. the set of \( S \in \mathcal{M} \) such that \( VAL(S) \) is finite;
3. the set of \( S \in \mathcal{M} \) such that \( VAL(S) \) is a regular set;
4. the set of \( S \in \mathcal{M} \) such that \( S \) is strongly equivalent to a monadic single-variable program scheme;
5. the set of \( S \in \mathcal{M} \) such that \( S \) is strongly equivalent to a linear monadic recursion scheme; and
6. the set of \( S \in \mathcal{M} \) such that \( S \) is strongly equivalent to a free monadic recursion scheme.

Then for each of the problems of 1, 2, and 5 of the statement of Theorem 4.1, there exists \( c > 0 \) such that the restriction of the problem to \( \mathcal{L} \) requires more than \( 2^c |S| \) steps infinitely often on any deterministic Turing machine.

**Proof.** The schemes \( N[y] \) and \( P[y] \) in the proof of 1, \( N[y] \) in the proof of 2, and \( N[y] \) in the proof of 5 of Theorem 4.1 are members of each of the subclasses \( \mathcal{L} \) of \( \mathcal{M} \) in the statement of the theorem.
A number of additional deterministic exponential time lower bounds for problems for \(M\) and for some of its subclasses are implied by Theorems 3.2, 4.1, and their proofs. Here, we mention only three different kinds of such additional lower bounds. First, the deterministic exponential time lower bounds of 1 of Theorem 4.1 hold even for pairs \((S, T)\) of schemes such that \(S\) and \(T\) are known a priori to halt for the same interpretation. Second, the conclusions of Theorem 4.2 hold even when the schemes \(S \in \mathcal{L}\) are presented together with proofs that they are in \(\mathcal{L}\). That is, the problem requires more than \(2^m\) steps infinitely often where \(m\) equals the sum of the size of the scheme and the length of the proof that the scheme is in \(\mathcal{L}\). Third, the conclusions of 3 and 4 of Theorem 4.1 can be generalized to a metatheorem giving sufficient condition for a monadic recursion scheme problem to require exponential time. A sample generalization is the following.

Let \(\mathcal{A}\) be any subclass of \(M\) such that

i. \(|S \in \mathcal{A} \mid S\) is divergent| \(\subset \mathcal{A}\); and

ii. there exists a scheme \(S_0 \in \mathcal{A}\) such that, for any scheme \(S \in \mathcal{A}\) in which \(S_0\) is an executable subscheme of \(S\), \(S \notin \mathcal{A}\).

Then, there exists \(c > 0\) such that the problem of determining for \(S \in \mathcal{A}\), if \(S \in \mathcal{A}\), requires more than \(2^c m^m\) steps infinitely often on any deterministic Turing machine.

(Each of the classes mentioned in 3 and 4 of Theorem 4.1 satisfy conditions i and ii.)

Finally, we note that the decidability of the strong equivalence problem for \(M\) is open, and is equivalent to the decidability of the equivalence problem for deterministic pushdown automata [5]. The exponential time lower bound of Theorem 4.1 applies to the strong equivalence problem for \(M\), but not to the equivalence problem for deterministic pushdown automata. The reason is that the standard reduction of the strong equivalence problem for \(M\) to the equivalence problem for deterministic pushdown automata involves an exponential increase in problem size, i.e., construction of deterministic pushdown automata whose description is exponentially larger than the size of the given monadic recursion schemes.

5. SOME EXPONENTIAL TIME UPPER BOUNDS

In this section we derive deterministic exponential time upper bounds for several decision problems for \(M\). Recalling the results of Section 4, we show that both deterministic exponential time lower and upper bounds hold for the strong computational identity, weak computational identity, isomorphism, divergence, and executability problems for \(M\). To obtain these exponential time upper bounds, we use a construction from [2] involving context-free grammars and the efficient reductions between decision problems for \(M\) in [13].

First, we note the following simple observation about \(M\).
LEMMA 5.1. There exists a constant $c > 0$ and a deterministic polynomially time-bounded Turing machine $\mathcal{A}$ such that $\mathcal{A}$, given input $S \in \mathcal{M}$, outputs $\hat{S} \in \mathcal{M}$ such that

1. the schemes $S$ and $\hat{S}$ are strongly equivalent;
2. $PS(\hat{S}) = PS(S)$;
3. each embedded string of $\hat{S}$ is of length $\leq 2$;
4. $\|\hat{S}\| \leq c \cdot \|S\|; \text{ and}$
5. each $F \in DFS(S)$ is an element of $DFS(\hat{S})$ and is executable in $\hat{S}$ if and only if it is executable in $S$.

Proof. Obvious. $\square$

Next, as in $[2]$, we shown how to associate with each monadic recursion scheme $S$ a context-free grammar $G[S]$ such that

$L(G[S]) = VAL(S)$.

Let $S = (DFS(S), BFS(S), PS(S), F, DS(S))$ be a monadic recursion scheme. Then the associated context-free grammar $G[S]$ is defined as follows:

1. The set $N$ of nonterminals of $G[S]$ is

$$\{S_0\} \cup \{[Q, F, R], [Q, F, R], [Q, I, Q] \mid Q, R \in PS(S); F \in DFS(S); \text{ and } f \in BFS(S)\}.$$

2. The set $Σ$ of terminals of $G[S]$ is $BFS(S)$.
3. The start symbol of $G[S]$ is $S_0$.
4. The productions $P$ of $G[S]$ are:
   a. $S_0 \rightarrow [Q, F, R]$, for all $Q, R \in PS(S)$.
   b. $[Q, f, R] \rightarrow f$, for all $Q, R \in PS(S)$ and $f \in BFS(S)$.
   c. $[Q, I, Q] \rightarrow \lambda$, for all $Q \in PS(S)$.
   d. For each $Q, R \in PS(S)$ and for each defining statement of $S$

$$F(x) := \text{if } p(x) \text{ then } w_1 x \text{ else } w_2 x$$

i. if $p \in R$ and $w_1 \neq \lambda,$

$$[Q, F, R] \rightarrow [Q, w_{11}, Q_1][Q_1, w_{12}, Q_2] \cdots [Q_{k-1}, w_{1k}, R]$$

for all $Q_1, Q_2, \ldots, Q_{k-1} \in PS(S)$, where

$$w = w_{11} w_{12} \cdots w_{1k} \text{ and each } w_{1i} \in BFS(S) \cup DFS(S);$$

ii. if $p \in R$ and $w_1 = \lambda,$

$$[R, F, R] \rightarrow [R, I, R];$$
iii. if $p \notin R$ and $w_2 \neq \lambda$,

$$[Q, F, R] \rightarrow [Q, w_{21}, Q_1][Q_1, w_{22}, Q_2] \cdots [Q_{l-1}, w_{2l}, R]$$

for all $Q_1, Q_2, \ldots, Q_{l-1} \subset \text{PS}(S)$, where

$w = w_{21}w_{22} \cdots w_{2l}$ and each $w_{2i} \in \text{BFS}(S) \cup \text{DFS}(S)$; and

iv. if $p \notin R$ and $w_2 = \lambda$,

$$[R, F, R] \rightarrow [R, I, R].$$

We make the following observation about the construction of $G[S]$.

**Lemma 5.2.** There exists a constant $c > 0$ such that, for a monadic recursion scheme $S = (\text{DFS}(S), \text{BFS}(S), \text{PS}(S), F_0, \text{DS}(S))$, where each embedded string is of length at most 2, the size of $G[S]$ is bounded by $2^{cN_{PS}(S)} \cdot \|S\|$.  

**Proof.** Obvious from inspection of the construction of $G[S]$.  

We will subsequently use the fact that there is a polynomial time algorithm to test if any of a specified set of nonterminals can be the rightmost nonterminal in a string generated from the start symbol of a grammar.

**Proposition 5.3.** There is a deterministic polynomial time algorithm that, given context-free grammar $G = (N, \Sigma, S, P)$ and $M \subset N$, determines if there exists a $B$ in $M$, $x$ in $\Sigma^*$, and $y$ in $(N \cup \Sigma)^*$ such that $S \Rightarrow^* yBx$.

**Proof.** The algorithm constructs a context-free grammar $H = (N', \Sigma, S', P')$, where

$$N' = N \cup \{A' \mid A \in N\},$$

$$P = P_1 \cup P_2 \cup P_3,$$

where

$$P_1 = \{A \rightarrow \omega \mid A \rightarrow \omega \in P\},$$

$$P_2 = \{A' \rightarrow C\psi \mid C \in N, \psi \in (N \cup \Sigma)^* \text{ and for some } \phi \in (N \cup \Sigma)^*, A \rightarrow \phi C\psi \in P\},$$

and

$$P_3 = \{B' \rightarrow \lambda \mid B \in M\}.$$ 

It should be clear that induction on the length of derivations can be used to show that

$$L(H) = \left\{ x \mid S \Rightarrow^*_G yBx \text{ for some } B \in M \text{ and } y \in (N \cup \Sigma)^* \right\}.$$ 

Thus, the algorithm outputs "YES" if and only if $L(H)$ is nonempty.

Finally, we observe that $H$ is constructable in time polynomial in $\|G\|$; and we recall that the emptiness problem for context-free grammars is decidable deterministically in polynomial time.  

Next we note polynomial reductions from [13] of the strong computational identity, weak computational identity, and isomorphism problems for $\mathcal{M}$ to the negation of the executability problem for $\mathcal{M}$.

**Proposition 5.4 [13].** There exist polynomial time algorithms $R_1, R_2,$ and $R_3$ that given schemes $S$ and $T$ in $\mathcal{M}$ produce schemes $W_1[S,T], W_2[S,T], \text{ and } W_3[S,T]$ in $\mathcal{M}$, respectively, such that $S$ and $T$ are strongly computationally identical, weakly computationally identical, or isomorphic if and only if the defined function symbol $B$ is not executable in $W_1[S,T], W_2[S,T],$ and $W_3[S,T],$ respectively. Furthermore each constructed scheme has the same set of predicate symbols as $S$.

\[\| W_1[S,T] \| \leq c \cdot (\| S \| + \| T \|),\]
\[\| W_2[S,T] \| \leq c \cdot (\| S \| + \| T \|),\text{ and}\]
\[\| W_3[S,T] \| \leq c \cdot (\| S \| + \| T \|)^2,\]

where $c$ is a constant independent of $S$ and of $T$.

Finally, we combine the above grammatical construction of [2], Lemma 5.1, and the efficient reductions between problems for $\mathcal{M}$ to prove deterministic exponential time upper bounds for a number of problems for $\mathcal{M}$.

**Theorem 5.5.** For each of the following problems for $\mathcal{M}$, there exists $d > 0$ such that the problem requires no more than $2^d \| S \|$ steps almost everywhere on any deterministic Turing machine:

1. the divergence problem;
2. the executability problem;
3. the problem of testing for $S \in \mathcal{M}$, if $\text{VAL}(S)$ is finite;
4. the strong computational identity problem;
5. the weak computational identity problem; and
6. the isomorphism problem.

**Proof.** i. Proof of 1. Let $S \in \mathcal{M}$. Let $\tilde{S} \in \mathcal{M}$ be the scheme output by the Turing machine $\mathcal{A}$ (of Lemma 5.1), given input $S$. Then $S$ is divergent if and only if $\tilde{S}$ is divergent and if only if $L(G[\tilde{S}]) = \text{VAL}(\tilde{S}) = \emptyset$. Thus the following is an algorithm to decide the divergence problem for $\mathcal{M}$:

**Algorithm A1.**

Step 1. Given input $S \in \mathcal{M}$, apply the Turing machine $\mathcal{A}$ to $S$.

Step 2. Letting $\tilde{S}$ be the output of $\mathcal{A}$ at Step 1, construct the associated context-free grammar $G[\tilde{S}]$.

Step 3. If $L(G[\tilde{S}]) = \emptyset$, then output "$S$ is divergent." Otherwise, output "$S$ is not divergent."
Algorithm A1 requires at most deterministic time $2^{d \cdot \|S\|}$ almost everywhere for some $d > 0$. This follows by noting that:

i. Step 1 requires only deterministic polynomial time;

ii. $\|\hat{S}\| \leq c_1 \cdot \|S\|$ for some constant $c_1$ independent of $S$;

iii. $\|G[\hat{S}]\| \leq 2^{c_2 \cdot \|S\|}$ for some constant $c_2$ independent of $S$; and

iv. the emptiness problem for context-free grammars is decidable deterministically in polynomial time.

(Note that (iii) is not true for $G[S]$.)

Proof of 2. Let $S \in \mathcal{M}$. Let $\hat{S} \in \mathcal{M}$ be the scheme output by the Turing machine $\mathcal{M}$, given input $S$. Let $B \in$ DFS$(S)$. Then $B$ is executable in $S$ if and only if $B$ is executable in $\hat{S}$. Let $G[\hat{S}] = (N, \Sigma, S_0, P)$. Then $B$ is executable in $\hat{S}$ if and only if $S_0 \xrightarrow*{\gamma(Q, B, R)} x$ for some $Q, R \in$ PS$(S)$, $x \in \Sigma^*$, and $\gamma \in (N \cup \Sigma)^*$. Thus an algorithm similar to Algorithm A1 can be used for the executability problem. From Proposition 5.3, the time bound $2^{d \cdot \|S\|}$ applies.

For a finer time bound, observe from Lemma 5.1 that PS$(\hat{S}) =$ PS$(S)$, and so from Lemma 5.2, $\|G[\hat{S}]\| \leq 2^{c_3 \cdot \|S\|}$ for some $c_3$. Thus the time required is bounded by $2^{c_4 \cdot \|\hat{S}\| \cdot \|P(\|S\|)\|}$ for some constant $c_4$ and some polynomial function $p$.

Proof of 3. VAL$(S)$ is finite if and only if $L(G[\hat{S}])$ is finite, and the finiteness problem for context-free grammars is decidable deterministically in polynomial time.

Proof of 4. From Proposition 5.4 and part 2.

Proof of 5. From Proposition 5.4 and part 2.

Proof of 6. Let $S, T \in \mathcal{M}$. Let $W_3[S, T]$ be the scheme produced by algorithm $R_3$ of Proposition 5.4. Note that PS$(W_3[S, T]) =$ PS$(S)$. From the finer time bound given at the end of the proof of 2, determining if $B$ is executable in $W_3[S, T]$ requires deterministic time at most exponential in $\|P(\|S\|)\|$ and polynomial in $\|W_3[S, T]\|$. 

REFERENCES