

The Complexity of Decision Problems
in Automata Theory and Logic

by
Larry J. Stockmeyer

ABSTRACT

The inherent computational complexity of a variety of decision problems in mathematical logic and the theory of automata is analyzed in terms of Turing machine time and space and in terms of the complexity of Boolean networks.

The problem of deciding whether a star-free expression (a variation of the regular expressions of Kleene used to describe languages accepted by finite automata) defines the empty set is shown to require time and space exceeding any composition of functions exponential in the length of expressions. In particular, this decision problem is not elementary-recursive in the sense of Kalmar.

The emptiness problem can be reduced efficiently to decision problems for truth or satisfiability of sentences in the first order monadic theory of $(\mathbb{N}, <)$, the first order theory of linear orders, and the first order theory of two successors and prefix, among others. It follows that the decision problems for these theories are also not elementary-recursive.

The number of Boolean operations and hence the size of logical circuits required to decide truth in several familiar logical theories of sentences only a few hundred characters long is shown to exceed the number of protons required to fill the known universe.

The methods of proof are analogous to the arithmetizations and reducibility arguments of recursive function theory.

Keywords: computational complexity, decision procedure
star-free, Turing machine

AMS (MOS) Subject Classification Scheme (1970)

primary 68A20, 02G05
secondary 68A40, 94A20

Table of Contents

1. Introduction	7
2. The Model of Computation	17
2.1 The Basic Model	18
2.2 A Technically Useful Model	34
3. Efficient Reducibility	41
3.1 Definitions	42
3.2 Applications to Complexity Bounds	46
3.3 Other Applications	53
4. Regular-Like Expressions	67
4.1 Expressions With Squaring	79
4.2 Expressions With Complementation	103
4.3 (deleted)	
4.4 Expressions Over a One-Letter Alphabet	157
5. Nonelementary Logical Theories	161
6. Complexity of Finite Problems	179
6.1 Second Order Theory of Successor	186
6.2 First Order Integer Arithmetic	205
7. Conclusion	213
Bibliography	214
Appendix I, Notation	221
Appendix II, Some Properties of logspace	223

List of Figures

Figure 4.1:	E_2 "matches" a word ω	82
Figure 6.1:	P, B, and d	194
Figure 6.2:	Illustrating the proof of Lemma 6.5.2 (i) and (ii)	196
Figure 6.3:	I and J "code" a circuit	198
Figure 6.4:	The circuit C_0	204

Chapter 1. Introduction

One major goal of computational complexity is to achieve the ability to characterize precisely the amount of computational resource needed to solve given computational **problems** or classes of problems. **Two** important kinds of computational resource are time and space, respectively the number of basic computational steps and the amount of memory used in solving the problem. The complexity of a particular problem **can** be characterized by upper and lower bounds on **computational** resources sufficient to solve the problem.

Upper bounds are usually established by exhibiting a specific algorithm which solves the problem and whose time **and/or** space complexity can be bounded from above. **Much** progress has been made on this positive side of the complexity question. **Many** clever and efficient algorithms have been devised for performing a wide variety of computational tasks (cf. D.E. Knuth, The Art of Computer Programming).

However the progress made on the negative side of the question has been less striking. In order to establish a lower bound **on** the complexity of a particular problem, one must show that some **minimum** amount of resource (time or space) is always required no matter which of the infinitely many possible algorithms is used or **how** cleverly one writes the algorithm to solve the problem. It is this latter side of the complexity question which we address in **this paper** . Although lower bound **results are** negative in nature, they have the value that they enable one to cease **looking** for efficient **algorithms when** none exist.

Also, the exhibition of specific problems or classes of problems which are provably difficult may give insight into the "reasons" for their difficulty, and these "reasons" and proofs of difficulty may provide clues for reformulating the problems so that in revised form they become tractable.

Let us now sketch a bit more precisely what we mean by "computational problem" and "algorithm"[†]. Many computational problems can be viewed as problems of function evaluation. In particular, consider functions mapping strings of symbols to strings of symbols. As a concept of "algorithm" we could choose any one of a variety of universal computer models. For definiteness we choose the well-known Turing machine model.

A Turing machine M computes the function f if M , when started with any string x on its tape, eventually halts with $f(x)$ on its tape. The time and space used by M on input x are respectively the number of basic steps executed and the number of tape squares visited by M before halting when started on input x . In general, the time and space will vary depending on the particular input x . One simplification which is commonly made is to measure the time and space solely as a function of the length of the input string.

Note that some functions can be complex for a reason which sheds little light on the question of inherent difficulty; namely, a function can be computed no faster than the time required to print the value of

[†]Complete definitions appear in the main text.

the function. For example, consider the function which, for any positive integer m , maps the binary representation of m to the binary representation of 2^m . Any algorithm which computes this function uses at least 2^n steps on many inputs of length n for all n , these steps being required to print the answer consisting of a one followed by as many as $2^n - 1$ zeroes.

We avoid these cases by considering only functions whose value is always 0 or 1. The problem of computing such a 0-1 valued function f can be viewed as the problem of recognizing the set of inputs which f maps to 1. For example, we may wish to recognize the set of all strings which code true sentences of some decidable logical theory. When such a "set recognition" or "decision" problem is shown to require time 2^n on inputs of length n for infinitely many n , we conclude that there is something inherently complex about the set itself; that is, 2^n steps must be spent in deciding what to answer, not in printing the answer.

Some information is known concerning the complexity of set recognition problems. There are known to be sets whose recognition problems are recursive yet "arbitrarily" complex [Rab60]. Let $T(n)$ and $S(n)$ be any recursive functions from positive integers to positive integers. Well-known diagonalization arguments imply the existence of a recursive set A_{hard} such that any algorithm recognizing A_{hard} requires at least time $T(n)$ and space $S(n)$ on all inputs of length n for all sufficiently large n .

It is also possible to construct arbitrarily difficult recursive problems by considering "bounded" versions of undecidable problems. The "bound" implies decidability, but the problem can be made arbitrarily complex by making the "bound" arbitrarily large. For example, Blum [Bl66] and Jeroslow [Jer72] consider a bounded version of the halting problem, and Ehrenfeucht [Ehr72] considers a bounded version of the first order theory of integer arithmetic.

One might animadvert that sets such as A_{hard} above are not "natural" in the sense that they were explicitly constructed to be difficult to recognize. Informally, by "natural" computational problem we mean one which has arisen previously in the mathematical literature (excluding complexity theory); for example, decision problems drawn from logic and automata theory, word problems in algebra, etc.

Under even this weak view of "natural", there are few examples of natural recursive set recognition problems whose time complexity has been shown to necessarily grow faster than linearly in the length of the input. Excluding "diagonalization" and "bounded undecidable" problems, then prior to the research described here (and related work by Meyer [Mey73], Fischer and Rabin [FR74], and Hunt [Hun73b]) we know of no examples of natural recursive set recognition problems whose time complexity had been shown to necessarily grow more than polynomially or whose space complexity had been shown to grow more than linearly in the length of the input.

We now outline the remainder of this paper. Chapters 2 and

3 are devoted mainly to definitions of key concepts and descriptions of the technical machinery to be used in proving the results of Chapters 4 and 5. Chapter 2 defines our formal model of "algorithm" for set recognition and function computation. This model is a slight variant of the well-known Turing machine. Known facts concerning the model which are relevant to the sequel are also stated.

Chapter 3 defines the concept of "efficient reducibility". This concept was first formally defined by Cook [Co71a], though its significance was emphasized earlier by Meyer and McCreight [MM71]. Speaking informally for the moment, we say that a set A is efficiently reducible to a set B , written $A \leq_{\text{eff}} B$, if there is an efficiently computable function f such that any question of the form "Is x in A ?" has the same answer as the question "Is $f(x)$ in B ?". Instead of being precise about what is meant by f being "efficiently computable", let us for the moment just assume that the time and space **required** to compute f is very small compared to the minimum time required to recognize A or B . Now given an algorithm M which recognizes B , one **can** construct an algorithm M' which recognizes A as follows. Given input x , M' first computes $f(x)$ and then simulates M on input $f(x)$. Since $x \in A$ iff $f(x) \in B$, M' recognizes A correctly. Moreover, the resources used by M' are roughly the same as those used by M because the resources used in computing f are negligible. Therefore an upper bound on the complexity of B implies an upper bound on that of A . Contrapositively, a lower bound on the complexity of A implies a

lower bound on that of B.

In Chapter 4, this reducibility technique is applied to several specific problems. This chapter deals with problems of recognizing equivalence of expressions similar to the Kleene regular expressions of **finite** automata theory [cf. Har65]. For example, consider regular expressions which may use, as well as the usual operations \cup , \cdot , and * , a new unary operation on sets of words, "**squaring**", defined by $S^2 = S \cdot S$. Let B_{sq} denote the set of all pairs of inequivalent such expressions.

The major technical portion of most applications of the reducibility technique involves a proof that any one of a large class of sets is efficiently reducible to a particular set of interest. We always choose the large class to be the class of all sets whose time or space complexity is bounded above by some function or familiar family of functions such as the polynomial or exponential functions.

In the case of B_{sq} , this class, called **EXPSPACE**, is the class of all sets recognizable within space which grows at most exponentially in the length of the input. We show that if $A \in \text{EXPSPACE}$ then $A \leq_{\text{eff}} B_{sq}$. Now diagonalization **arguments** imply the existence of a set A_{hard} in **MPSPACE** which requires exponential space for recognition by any algorithm. Thus $A_{\text{hard}} \leq_{\text{eff}} B_{sq}$ and so B_{sq} also requires exponential space (and hence also requires exponential time).

Similarly we characterize the space complexity of recognizing equivalence of regular expressions involving only the operations of \cup , \cdot , and * . We also consider other variants such as expressions with

only \cup and \cdot , and expressions over a one-letter alphabet.

If the expressions are allowed to use the operation of set complementation (\sim), a drastic increase in the complexity of the equivalence problem results. We show that the equivalence problem for "star-free" expressions [cf. MP71] (using only the operations \cup , \cdot , and \sim) is not elementary-recursive [cf. Pet67]; that is, for no constant

k is its time or space complexity bounded above by $2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}$ for all inputs of length n and all n .

Chapter 5 gives several corollaries about the complexities of decidable theories of formal logic. The equivalence problem for star-free expressions is efficiently reducible to the decision problems for several decidable logical theories; thus these decision problems are not elementary-recursive. Our main corollary states that the first order theory of any infinite linear order with a single monadic predicate is not elementary-recursive. In particular, we obtain

the result that the weak monadic second order theory of successor is not elementary-recursive [cf. Mey73].

For convenience, we are content in Chapters 4 and 5 to give a lower bound on the complexity of a particular set by proving that the resources used by any algorithm in recognizing the set must exceed the lower bound on infinitely many inputs. Section 3.3 points out that a given result can usually be strengthened to state that the lower bound must hold on some input of length n for all but finitely many n .

Even so, one might reasonably question the significance of our results and methods on the grounds that the "difficult" inputs might be so large as to never occur in practice. This is indeed an important issue. Closer examination of our proofs can **determine** the point at which the lower bounds take effect, though we do not in general elaborate such results here.

However, in Chapter 6 we investigate two examples in detail.[†] Our methods do yield astronomical lower bounds on the complexities of finite decision problems about words of only a few hundred characters. The notion of "algorithm" used here is Boolean circuits similar to those studied in [Win65] and [Sav72]. For two logical theories, the number of Boolean operations required by a circuit which recognizes the true sentences only a few hundred characters long is shown to exceed the number of protons required to fill the known universe.

In Chapters 4 and 5 we also give upper bounds on the complexities of recognizing the particular sets considered. In most cases, the upper bound given for a set is reasonably close to the proven lower bound. The verifications of upper bounds involve only standard techniques from automata theory.

In summary, the main contribution of this paper is the demonstration that efficient reducibility techniques can be used to

[†]The major portion of Chapter 6 can be read independently of Chapters 2 through 5.

prove non-trivial lower bounds on the time, space, or circuit complexities of certain natural recursive decision problems. The main technical contribution lies in the various reducibility constructions and "arithmetizations" of Turing machines and circuits. These constructions are of an essentially different character than those **commonly** found in recursion theory, due to the added condition that reducibilities must be efficiently computable.

Chapter 2. The Model of Computation

In order to prove that certain problems require a certain minimum amount of computational resource no matter how one writes algorithms to solve the problems, it is essential to have a **formal definition** of an algorithm or computer. There are many formulations of the notions of algorithm which are equivalent in the sense that the functions computable within any of the formulations are precisely the recursive functions.

We shall choose our model of computer to be Turing machines [HU69], partly because this model is well-known and has been the subject of much previous investigation, but more importantly because its simplicity will ease the technical task of showing that the model cannot solve certain problems quickly. It might seem that the simplicity of the model itself implies its inefficiency and ~~that~~ it would be more realistic to choose a more powerful formulation such as random access register machines or iterative arrays [Col69]. However Turing machines can simulate the more powerful models "efficiently enough" (in a sense to be made precise shortly) for our purposes, so that if a Turing machine cannot compute something "quickly" neither will either of the more powerful models. In fact, all of the results in this **paper** giving upper or lower bounds on the complexities of particular problems remain true without modification if the Turing machine model is replaced by either of the more powerful models mentioned above.

2.1 The Basic Model.

First we assume the reader is familiar with the basic concepts of set theory and formal language theory. A discussion of the necessary concepts can be found in the introductory portions of most **formal** language theory texts, for example [HU69], [AU72].

In particular, we let Σ^* denote the set of all words over Σ , including the empty word λ ; Σ^+ denotes the set $\Sigma^* - \{\lambda\}$.

$|\omega|$ denotes the length of the word ω ; $|\lambda| = 0$.

, where k is a nonnegative integer, denotes repeated concatenation, that is, $\Sigma^k = \{ \omega \in \Sigma^* \mid |\omega| = k \}$.

If σ is a symbol, σ^k denotes the word $\sigma\sigma\sigma\cdots\sigma$ of length k .

Since this notation is **commonly** used for repeated Cartesian product, we let $\Sigma^{xk} = \Sigma \times \Sigma \times \cdots \times \Sigma$ (k times).

This and other notation is collected in Appendix I.

Our basic model of computation is **input/output** Turing machines (**IOTM**'s). IOTM's are multi-tape Turing machines in which the tapes which handle the **input/output** processes are separated from the tapes which serve as memory for the computation. Every IOTM consists of a finite state control and $k + 2$ tapes (where k is a positive integer): an input tape, k work tapes, and an output tape. Single heads scanning each tape are called respectively the input **head** (2-way, read-only), the work heads (2-way, **read/write**), and the output head (right-moving, write-only).

We now give precise informal definitions of the IOTM model, its

computations, the time and space used by a computation, etc. Turing machines (of which IOTM's are a minor variant) are formally defined in many standard reference texts (e.g. [HU69]). Since our results are **invariant** under the various differences in conventions normally used in making these definitions, the reader can supply his own formal definitions by choosing any consistent set of conventions.

One important distinction we must make is the difference between nondeterministic and deterministic machines. We first define nondeterministic IOTM's; deterministic IOTM's are then defined as a restricted form of nondeterministic IOTM's.

A particular nondeterministic IOTM, M , is specified by finite sets Q (the set of states), I (the input alphabet), Γ (the work tape alphabet), and Δ (the output alphabet); a transition function δ ; and designated states $q_0 \in Q$ (the initial state) and $q_a \in Q$ (the accept state). M operates in steps. The action taken at a given step depends on the current state of the control and the symbols being scanned by the input and work tape heads. M performs a particular action by changing state, printing new symbols on the work tapes and possibly on the output tape, and shifting the heads.

*

We now describe the computations of M on input $x \in I^*$. M is started with the word $\$x\$$ written on the input tape with the input head scanning the **leftmost** $\$$. ($\$ \in I$ is an endmarker. Let $I' = I \cup \{\$\}$.) The control is placed in state q_0 , and the work and output tapes are initially blank.

The total state of the machine at some step is given by an

instantaneous description (i.d.). An i.d. consists of (1) the state of the control, (2) the input word x , (3) the position of the input head in the word $\$x\$$, (4) for each $i = 1, 2, \dots, k$, the word $\omega_i \in \Gamma^*$ written on the nonblank portion of the i^{th} work tape, (5) for each i such that $\omega_i \neq A$, the position of the i^{th} work head in the word ω_i , and (6) the word written on the **nonblank** portion of the **output** tape.

For example, the initial i.d. of M on input x described above is given by: (1) the initial state; (2) $\$x\$$; (3) the input head is scanning the **leftmost** symbol $\$$; (4) $\omega_i = h$ for $i = 1, 2, 3, \dots, k$; (5) \emptyset ; (6) λ .

If r is an i.d., then display(r) is $(q, \sigma, s_1, s_2, \dots, s_k) \in Q \times \Gamma' \times \Gamma^{xk}$, where q is the current state of the control, and σ, s_1, \dots, s_k are the symbols being scanned by the input head and the k work heads respectively.

The function δ maps each element in $Q \times \Gamma' \times \Gamma^{xk}$ to a (possibly empty) set of moves. A move is of the form

$$\mu = (q', s_1', s_2', \dots, s_k', m_0, m_1, m_2, \dots, m_k, p) \in Q \times \Gamma^{xk} \times \{\underline{l}, \underline{r}, \underline{n}\}^{xk+1} \times (\Delta \cup \{\lambda\})$$

If M is currently in a situation described by i.d. r , M may execute any move in $\delta(\text{display}(r))$. M executes move μ above as follows: the finite state control enters state q' ; for each $i = 1, 2, \dots, k$, the i^{th} work head prints symbol s_i' and shifts one square in direction m_i (**left**, **right**, or **nomove**); the input head shifts in direction m_0 ; if $p \neq h$, the output head prints p and shifts right one square; if $p = \lambda$, the output head does not print or shift.

If the execution of any move in $\delta(\text{display}(r))$ causes M to enter

i.d. r' , we say $r \xrightarrow{M} r'$.

A computation of M on input x , c , is any sequence of **i.d.**'s

$$c = \mathbf{i.d.}_1, \mathbf{i.d.}_2, \dots, \mathbf{i.d.}_\ell \quad \text{such that:}$$

- (1). $\mathbf{i.d.}_1$ is the initial **i.d.** of M on input x ,
- (2). $\mathbf{i.d.}_j \xrightarrow{M} \mathbf{i.d.}_{j+1}$ for all $j = 1, 2, 3, \dots, \ell-1$,
- (3). $\delta(\text{display}(\mathbf{i.d.}_\ell)) = \emptyset$; that is, M halts on $\mathbf{i.d.}_\ell$.

The length of the computation $c = \mathbf{i.d.}_1, \mathbf{i.d.}_2, \dots, \mathbf{i.d.}_\ell$ is ℓ .

The space used by the computation c is the **number** of work tape squares visited by heads of M during the computation. It is technically convenient to make one exception to this definition of space; namely, if $c = \mathbf{i.d.}_1, \mathbf{i.d.}_2, \dots, \mathbf{i.d.}_\ell$ and if for all $j = 1, 2, 3, \dots, \ell$, $\mathbf{i.d.}_j$ describes a situation in which all work tapes are entirely blank, then the space used by c is defined to be 0.

The output produced by the computation c is the word written on the nonblank portion of the output tape in $\mathbf{i.d.}_\ell$.

If $c = \mathbf{i.d.}_1, \mathbf{i.d.}_2, \dots, \mathbf{i.d.}_\ell$ as above, and also $\text{display}(\mathbf{i.d.}_\ell) \in \{q_a\} \times I' \times \Gamma^{xk}$, then c is an accepting computation of M on input x . (We assume q_a is a halting state; that is,

$$\delta(q_a, \sigma, s_1, \dots, s_k) = \emptyset \quad \text{for all } \sigma \in I', s_1, \dots, s_k \in \Gamma.)$$

Let $\text{AccComp}_M(x)$ denote the set of all accepting computations of M on input x . Note that $\text{AccComp}_M(x)$ may contain many computations corresponding to the different choices of moves from δ taken at each

step. $\text{AccComp}_M(x)$ may also be empty if M does not enter state q_a regardless of what choices are made.

If $x \in I^*$ and $\text{AccComp}_M(x) \neq \emptyset$, define

$$\underline{\text{Time}}_M(x) = \min\{ \ell \mid \text{there is an accepting computation } c \in \text{AccComp}_M(x) \text{ of length } \ell \},$$

and

$$\underline{\text{Space}}_M(x) = \min\{ m \mid \text{there is a } c \in \text{AccComp}_M(x) \text{ which uses space } m \}.$$

We leave $\text{Time}_M(x)$ and $\text{Space}_M(x)$ undefined if $\text{AccComp}_M(x) = \emptyset$.

Nondeterministic IOTM's are a technical construct and do not correspond to the notion of algorithm in which each step is uniquely determined. Deterministic IOTM's do correspond to this step-by-step notion of algorithm.

A deterministic IOTM is a nondeterministic IOTM with the property that its transition function δ maps each element in $Q \times I' \times \Gamma^{Xk}$ to a set containing at most one move. Thus the computation of a deterministic IOTM on an input x is uniquely **determined** (provided that it exists). Deterministic IOTM's are a special case of **nondeterministic IOTM's**; the definitions of $\text{AccComp}_M(x)$, $\text{Time}_M(x)$, $\text{Space}_M(x)$ given above also define these concepts for deterministic IOTM's.

IOTM's serve as our model of algorithm for set recognition.

Definition 2.1. Let M be a nondeterministic (or deterministic) IOTM^{*} with input alphabet I , and let $x \in I$.

M accepts x iff $\text{AccComp}_M(x) \neq \emptyset$.

M rejects x iff M does not accept x .

Let $A \subseteq I^+$. M accepts A iff

$$M \text{ accepts } x \Leftrightarrow x \in A \quad \text{for all } x \in I^+.$$

Definition 2.2. Let M be a nondeterministic IOTM, let $A \subseteq I^+$, and let T and S both map \mathbb{N} into the nonnegative rational numbers.[†]

M accepts A within time $T(n)$ (within space $S(n)$) iff

- (1). M accepts A
 and
 (2). for all but finitely many $x \in A$,
- $$\text{Time}_M(x) \leq T(|x|)$$
- $$(\text{Space}_M(x) \leq S(|x|)).$$

Remark. Note that Definition 2.2 only requires the time and space used by M to be bounded on almost all inputs $x \in A$. A stronger definition would require the time and space to be bounded for all inputs $x \in I^+$. However if we show, for a certain set A and functions $T(n)$ and $S(n)$, that no IOTM accepts A within time $T(n)$ or space $S(n)$ under the given definition, certainly the same result is true under the stronger definition.

[†] \mathbb{N} denotes the nonnegative integers.

In particular we require only "for all but finitely many x " to emphasize the fact that, with respect to Turing machines, the inherent complexity of a set is insensitive to finitely many exceptions.

Lemma 2.3. Suppose a nondeterministic (deterministic) IOTM M accepts

A. Let $X \subseteq A$ with X finite. Then there are **nondeterministic** (deterministic) IOTM's M' and M'' which accept A such that:

- (1). and $\text{Time}_{M'}(x) \leq \text{Time}_M(x)$ for all $x \in A$
 $\text{Time}_{M'}(x) \leq |x| + 2$ for all $x \in X$.
- (2). and $\text{Space}_{M''}(x) \leq \text{Space}_M(x)$ for all $x \in A$
 $\text{Space}_{M''}(x) = 0$ for all $x \in X$.

Proof sketch. Let \mathcal{Q} be a finite state acceptor (cf. [HU69]) for X .

(1). M' runs two procedures in parallel. The first procedure runs \mathcal{Q} on the input, at the same time copying the input onto the first work tape. The second procedure simulates M on the **input** by viewing the first work tape as the input tape. M' accepts when either procedure accepts. M' as described requires two heads on the first work tape.

However Fischer, Meyer, and Rosenberg [FMR72] show how to **replace** many heads per tape by several single-headed tapes with no time loss.

(2). M'' first runs \mathcal{Q} on the input; blanks are reprinted on the work tapes at each step. M'' accepts if \mathcal{Q} does, or simulates M on the input otherwise. □

Similarly our lower bound results are strengthened by using the nondeterministic model. If no nondeterministic IOTM can accept a certain set within time $T(n)$ or space $S(n)$, then neither can any deterministic IOTM. We discuss this further below.

IOTM's also serve as our model of function computation.

Definition 2.4. Let M be a deterministic IOTM and f be a total function, $f: I \rightarrow A$, where I, A are finite alphabets.

M computes f within time $T(n)$ (within space $S(n)$) iff
for all $x \in I$

- (1). $\text{AccComp}_M(x) \neq \emptyset$ and the (necessarily unique)
 $c \in \text{AccComp}_M(x)$ produces output $f(x)$,
and
(2). $\text{Time}_M(x) \leq T(|x|)$
($\text{Space}_M(x) \leq S(|x|)$).

Our motivation in separating the **input/output** processes from the computation process is so that it makes sense to consider a set being accepted within space $S(n)$ where $S(n)$ grows more slowly than linearly in n . The usual convention of writing the input initially on some work tape requires the machine to use space $|x|$ just to read the entire input x . Similarly, we may consider a function f being computed within space $S(n)$ where $|f(x)|$ is much larger than $S(|x|)$.

It is convenient to have notation for certain classes of all sets which can be accepted within a given resource bound.

Definition 2.5. $\text{NTIME}(T(n))$ ($\text{DTIME}(T(n))$)
 $= \{ A \mid \text{there is a nondeterministic (deterministic)}$
 $\text{IOTM which accepts } A \text{ within time } T(n) \}.$
 $\text{NSPACE}(S(n))$ ($\text{DSPACE}(S(n))$)
 $= \{ A \mid \text{there is a nondeterministic (deterministic)}$
 $\text{IOTM which accepts } A \text{ within space } S(n) \}.$

Here the sets A are also assumed to satisfy $A \subseteq I^+$ for some finite alphabet I .

In particular define:

$$P = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k) ; \quad NP = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k) ;$$

$$\text{CSL} = \text{NSPACE}(n) \quad (= (\text{context sensitive languages}), \text{ cf. [HU69]});$$

$$\text{POLYSPACE} = \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k) ;$$

$$\text{EXPNTIME} = \bigcup_{c \in \mathbb{N}} \text{NTIME}(c^n) ; \quad \text{EXPSPACE} = \bigcup_{c \in \mathbb{N}} \text{NSPACE}(c^n) .$$

• For a particular set B , a lower bound on the complexity of B will be given as the statement that B does not belong to some class $\text{NTIME}(T(n))$ or $\text{NSPACE}(S(n))$ for some particular $T(n)$ or $S(n)$. By Definitions 2.2 and 2.5, such a statement implies that $T(n)$ or $S(n)$ is an i.o. (infinitely often) lower bound on the nondeterministic time or space complexity of B .

If $B \notin \text{NTIME}(T(n))$ ($B \notin \text{NSPACE}(S(n))$) and M is a nondeterministic IOTM which accepts B , then

$$\begin{aligned} \text{Time}_M(x) &> T(|x|) \\ (\text{resp., } \text{Space}_M(x) &> S(|x|)) \end{aligned} \quad \text{for infinitely many } x \in B.$$

We now make more precise our earlier statement that the IOTM model is not restrictive and that our results have genuine significance independent of which **formal** notion of algorithm we adopt. In **particular**, consider two "more powerful" models of algorithm: random access machines (RAM's) [SS63], [CR72], and d -dimensional iterative **arrays** of finite state machines (**d-IA's**) [Co169].

The time and space of RAM and **d-IA** computations can reasonably be defined as follows. The time of a RAM computation is the sum of the costs of all steps; a step which manipulates (stores, fetches, adds) numbers of magnitude z is charged cost $\lceil \log(z+1) \rceil^\dagger$ (this being the length of the binary representation of integer z). **The** space of a

[†]Logarithms with no specified base are taken to base 2.

RAM computation is the ~~sum~~ over all registers of $\lceil \log(z+1) \rceil$ where **z** is the largest integer stored in the register at some step during the computation. The time of a d-IA computation is the number of steps executed. The space of a d-IA computation is the total number of cells which do not remain quiescent throughout the entire computation.

The fact stated below follows by simulations of the other models by IOTM's. See for example [CR72] for the simulation of RAM's.

Fact 2.6. Let **A** be a set which can be accepted by a nondeterministic (deterministic) RAM or d-IA within time **T(n)** and space **S(n)**. Then there is an integer **k** such that

$$\begin{aligned} & \mathbf{A} \in \mathbf{NTIME}((\mathbf{T(n)})^k) \quad \text{and} \quad \mathbf{A} \in \mathbf{NSPACE}(\mathbf{S(n)}) \\ & (\mathbf{A} \in \mathbf{DTIME}((\mathbf{T(n)})^k) \quad \text{and} \quad \mathbf{A} \in \mathbf{DSpace}(\mathbf{S(n)})) . \end{aligned}$$

Moreover, we can always choose **k = 2** for the case of RAM's.

Thus any lower bound on space complexity applies equally well to either of the more powerful models. Lower bounds on time complexity may suffer a decrease with respect to the other models, but this decrease is polynomial bounded which will be negligible in the cases to be considered. For example, if we show that a set **B** requires time c^n (i.o.) for acceptance by any IOTM, it follows that **B** requires time d^n (i.o.) for acceptance by any RAM, where $d = \sqrt{c}$.

The remainder of section 2.1 gives some known facts and open

questions concerning the classes NTIME, DTIME, NSPACE, DSPACE

All the particular functions we give bounding time or space complexity are of a special type defined next.

Definition 2.7. A function $T(n)$ ($S(n)$) is said to be countable (constructable) iff for any finite I there is a deterministic IOTM M such that

$$\begin{aligned} \text{Time}_M(x) &= T(|x|) & \text{for all } x \in I^+ \\ (\text{Space}_M(x) &= S(|x|) & \text{for all } x \in I^+). \end{aligned}$$

The countable and constructable functions are rich classes. The countable functions include in particular $\max(n^k, n+2)$, $\max(\lceil c^n \rceil, n+2)$, for all $k \in \mathbb{N}^+$, $c \in \mathbb{Q}^+$.[†] The constructable functions include n^k , $\lceil c^n \rceil$, and $(\lceil \log n \rceil)^k$ for all $k \in \mathbb{N}^+$, $c \in \mathbb{Q}^+$. Both classes are closed under addition, multiplication, and composition [Yam62].

The following **notation** is useful for comparing the growth rates of functions. Let $F(n)$ and $G(n)$ be functions from \mathbb{N}^+ to $\mathbb{Q}^+ \cup \{0\}$.

$$\begin{aligned} \underline{F(n) = O(G(n))} \quad \text{iff} \quad & \text{there is a } c \in \mathbb{Q}^+ \text{ such that} \\ & F(n) \leq c \cdot G(n) \quad \text{for all } n. \end{aligned}$$

$$\underline{F(n) = o(G(n))} \quad \text{iff} \quad \lim_{n \rightarrow \infty} F(n)/G(n) = 0.$$

The next fact states that any computation can be "sped-up" by any constant factor. The proof is implicit in [SHL65] and [HS65], (see also [HU69]). Part (2) also uses the main result in [FMR72].

[†] \mathbb{Q}^+ denotes the positive **rational**s. \mathbb{N}^+ denotes the positive integers.

Fact 2.8. Let $c \in \mathbb{Q}^+$ be arbitrary.

(1). Given a deterministic IOTM M with input alphabet I which computes a function f , we can effectively find a deterministic IOTM M' which **computes** f such that

$$\text{Space}_{M'}(x) \leq c \cdot \text{Space}_M(x) \quad \text{for all } x \in I. \quad *$$

(2). Given a nondeterministic (deterministic) IOTM M which accepts a set A , we can effectively find nondeterministic (deterministic) IOTM's M' and M'' which accept A such that

$$\begin{aligned} \text{Time}_{M'}(x) &\leq \max(c \cdot \text{Time}_M(x), |x| + 2) \quad \text{for all } x \in A \\ \text{and} \\ \text{Space}_{M''}(x) &\leq c \cdot \text{Space}_M(x) \quad \text{for all } x \in A. \end{aligned}$$

(3). Assume $n = o(T(n))$. Then

$$\begin{aligned} A \in \text{NTIME}(T(n)) &\Rightarrow A \in \text{NTIME}(c \cdot T(n)) \\ \text{and} \\ A \in \text{NSPACE}(S(n)) &\Rightarrow A \in \text{NSPACE}(c \cdot S(n)). \end{aligned}$$

Thus the inherent complexity of a particular problem is insensitive to constant factors and can at best be determined as an asymptotic growth rate (exponential, quadratic, etc.). Fact 2.8 is also used implicitly in several upper bound results. For example, we may describe an algorithm which accepts a set B within space $17n$, and then claim BECSL.

The next fact gives several known relationships among the complexity classes.

Fact 2.9. Let $T(n)$, $S(n)$ be arbitrary.

A. Nondeterministic versus deterministic time.

- (a). $\text{DTIME}(T(n)) \subseteq \text{NTIME}(T(n))$.
- (b). $\text{NTIME}(T(n)) \subseteq \bigcup_{c \in \mathbb{N}} \text{DTIME}(cT(n))$.

B. Nondeterministic versus deterministic space.

- (a). $DSPACE(S(n)) \subseteq NSPACE(S(n))$.
 (b). $NSPACE(S(n)) \subseteq DSPACE((S(n))^2)$.

C. Time versus space.

- (a). $DTIME(T(n)) \subseteq DSPACE(T(n))$.
 $NTIME(T(n)) \subseteq NSPACE(T(n))$.
 (b). $NSPACE(S(n)) \subseteq \bigcup_{c \in \mathbb{N}} DTIME(c^{S(n)})$, provided $\log n = O(S(n))$.

The statements (a) all follow directly from definitions and constant factor speedup (Fact 2.8). A.(b) follows from the fact that, if M is nondeterministic and accepts a set **within** time $T(n)$, $AccComp_M(x)$ contains at most $c^{T(|x|)}$ computations which could conceivably accept x , for some c and all x . A deterministic machine can try each of these **computations** in sequence and accept the input if any such computation accepts. B.(b) is proved by Savitch [Sav70]. Note that B.(b) implies that the definitions of POLYSPACE and EXPSPACE could have been made equivalently in terms of $DSPACE()$. C.(b) is true because a space $S(n)$ bounded IOTM can enter at most $c^{S(|x|)}$ different 'i.d.'s when computing on input x . A complete proof of C.(b) appears in [Co71b].

The "gaps" between (a) and (b) in each of A, B, and C represent major open questions of complexity theory.

Open Questions 2.10.

- A. (i). Is there a class of functions \mathcal{F} all of which grow slower than exponentially for which

$$NTIME(T(n)) \subseteq \bigcup_{F \in \mathcal{F}} DTIME(F(T(n))) ?$$

- (ii). May we take \mathcal{F} to be the class of polynomials ?
- (iii). In particular, does $\mathcal{P} = \text{NP}$?
- B. (i). Does $\text{NSPACE}(S(n)) = \text{DSpace}(S(n))$?
- (ii). In particular, does $\text{CSL} = \text{DSpace}(n)$?
- C. (i). Is there a class of functions \mathcal{F} as in A.(i) above for which

$$\text{NSPACE}(S(n)) \subseteq \bigcup_{F \in \mathcal{F}} \text{DTIME}(F(S(n))) ?$$
- (ii). May we take \mathcal{F} to be the class of polynomials ?
- (iii). In particular, is $\text{CSL} \subseteq \mathcal{P}$?

These open questions are stated to point out that, for most particular problems we consider, the upper and **lower** bounds we give are "tight" in the sense that any substantial improvement of either bound would close the gap implicit in some open question. For example, in section 4.1 we consider a set B (the set of all regular expressions over alphabet $\{0,1\}$ which do not describe $\{0,1\}^*$) and show $B \in \text{NSPACE}(n)$ but $B \notin \text{NSPACE}(n^r)$ if $r < 1$. Even though these space bounds are tight, they do not translate into tight bounds on deterministic time complexity. The best we can conclude (given present knowledge) is $B \in \text{DTIME}(d^n)$ for some $d \in \mathbb{Q}^+$ (by Fact 2.9C(b)); but $B \notin \text{DTIME}(n^r)$ if $r < 1$, which is a trivial lower bound on time. However it will be seen that this gap (d^n versus n) is closely related to Open Question 2.10C. For example, if one succeeds in **raising** the lower bound, say to $c^{\sqrt{n}}$ for some $c > 1$, then Open Question 2.10C(iii) would be settled in the negative. On the other hand, if one shows that $B \in \mathcal{P}$, then this question would be settled in the affirmative. See Remark 4.20 for further discussion of

the relevance of these open questions to this work.

Finally we give a fact which states that the complexity classes $\text{NTIME}(T(n))$, $\text{NSPACE}(S(n))$ describe fine complexity hierarchies; that is, for small increases in the growth rate of $T(n)$ or $S(n)$, new sets can be accepted that could not be accepted before. The following deep results, which are used several times in the sequel, are due to Seiferas, Fischer, and Meyer [SFM73], and are refinements of earlier work by Ibarra [Ib72] and Cook [Co73].[†]

Fact 2.11.

(1). Let $T_2(n)$ be countable. There is a set $A \subseteq \{0,1\}^+$ such that $A \in \text{NTIME}(T_2(n))$ and for all $T_1(n)$

$$T_1(n+1) = o(T_2(n)) \text{ implies } A \notin \text{NTIME}(T_1(n)).$$

(2). Let $S_2(n)$ be constructable and satisfy $\log n = o(S_2(n))$. There is a set $A \subseteq \{0,1\}^+$ such that $A \in \text{NSPACE}(S_2(n))$ and for all $S_1(n)$

$$S_1(n+1) = o(S_2(n)) \text{ implies } A \notin \text{NSPACE}(S_1(n)).$$

Diagonalization arguments give similar hierarchies [SHL65], [HS65] for the deterministic complexity classes, although the known time hierarchy is slightly coarser in the deterministic case.

[†]Fact 2.11 is not essential to our proofs, although we shall use it for convenience. See Remark 4.21 for an alternative to the use of Fact 2.11.

2.2 A Technically Useful Model

Having defined the basic model of algorithm, we now define a more restricted model called simple Turing machines (**STM's**). **STM's** serve only as a technical tool within the proofs of certain results, and are used only for set recognition. **STM's** are similar to **IOTM's**; the major differences are the following.

An **STM** has one tape and one head. The single tape is one-way infinite to the **right** and serves as both input tape and work tape. An **STM** is started on input x by writing x left justified on the otherwise blank tape with the head scanning the **leftmost** symbol of x . The moves of **STM's** are similar to those of **IOTM's**. Any move which shifts the head off the left end of the tape causes the **STM** to halt and reject the input. We also require **STM's** to have a unique accepting configuration; this configuration occurs when the control is in a designated state q_a , the entire **tape** is blank, and the head is scanning the **leftmost** tape square. q_a **must** be a halting state. Also the **STM** cannot enter state q_a when computing on a word which is not to be accepted. **STM's** and their related computational concepts are now made precise by a series of definitions.

A (nondeterministic)[†] **STM** is a six-tuple $M = (I, \Gamma, Q, \delta, q_0, q_a)$ consisting of a finite set Γ (the tape alphabet), a set $I \subseteq \Gamma$ (the input alphabet), a finite set Q (the set of states), a transition

[†] The adjective "**nondeterministic**" will sometimes be omitted.

function

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{-1, 0, 1\}},^{\dagger}$$

and designated states $q_0 \in Q$ (the initial state) and $q_a \in Q$ (the accept state). δ must satisfy the constraint $\delta(q_a, s) = \emptyset$ for all $s \in \Gamma$.

M is deterministic if $\text{card}(\delta(q, s)) \leq 1$ for all $q \in Q, s \in \Gamma$.

An instantaneous description (i.d.) of M is any word in $\Gamma^* \cdot Q \cdot \Gamma^*$.

Informally, if d is an i.d. of M , say

$$d = yqsz \quad \text{where } y, z \in \Gamma^*, s \in \Gamma, q \in Q,$$

we treat d as describing the symbols on the tape squares in an interval around the head, with q being the state of the control, and q being positioned in d **immediately** to the left of the symbol s being scanned.

We associate with M a function

$$\text{Next}_M : \Gamma^* \cdot Q \cdot \Gamma^* \rightarrow 2^{\Gamma^* \cdot Q \cdot \Gamma^*}.$$

$\text{Next}_M(d)$ is the set of i.d.'s that can occur one step after the situation described by i.d. d .

We first define $\text{Next}_M^1(d, \mu)$, an empty or singleton set containing the next i.d. reached from d by a particular move μ .

Let $\mu = (q', s', m) \in Q \times \Gamma \times \{-1, 0, 1\}$ and let $d_0 = yqsz$ as above.

$$\text{Next}_M^1(d_0, \mu) = \begin{cases} \{ yq's'z \} & \text{if } m = 0 \\ \{ ys'q'z \} & \text{if } m = 1 \\ \{ wq'ts'z \} & \text{if } m = -1 \text{ and } y = wt \text{ for} \\ & \text{some } w \in \Gamma^* \text{ and } t \in \Gamma \\ \emptyset & \text{if } m = -1 \text{ and } y = \lambda \end{cases}$$

[†] 2^S denotes the set of all subsets of the set S .

Now

$$\underline{\text{Next}_M(d)} = \begin{cases} \bigcup_{\mu \in \delta(q,s)} \text{Next}_M^1(d,\mu) & \text{if } d = yqs \text{ as above} \\ \emptyset & \text{if } d = yq \text{ for some } y \in \Gamma^*, q \in Q. \end{cases}$$

Note that $d' \in \text{Next}_M(d)$ implies $|d'| = |d|$. This differs from the usual definitions of "i.d." and "next i.d." in the literature.

The set of i.d.'s occurring ℓ steps after d , $\underline{\text{Next}_M(d,\ell)}$, is defined by induction:

$$\begin{aligned} \text{Next}_M(d,0) &= \{d\}, \\ \text{Next}_M(d,\ell+1) &= \{ d'' \mid d'' \in \text{Next}_M(d') \text{ for some} \\ &\quad d' \in \text{Next}_M(d,\ell) \} \end{aligned}$$

Definition 2.12. Let $M = (I, \Gamma, Q, \delta, q_0, q_a)$ be an STM, and let $A \subseteq I^+$. Let \emptyset denote the blank tape symbol.

M accepts A within time $T(n)$ (within space $S(n)$; here we assume $S(n) \geq n$) iff:

(1). For all $x \in A$, there exist $k \in \mathbb{N}$ with $k \leq T(|x|)$ and $k \geq |x|$ (resp., with $|x| \leq k \leq S(|x|)$) such that

$$q_a \emptyset^k \in \text{Next}_M(q_0 x \emptyset^{k-|x|}, \ell), \text{ and}$$

(2). for all $x \in I^+ - A$, there do not exist $\ell, k \in \mathbb{N}$ and $y, z \in \Gamma^*$ such that

$$y q_a z \in \text{Next}_M(q_0 x \emptyset^{k-|x|}, \ell).$$

We require $S(n) \geq n$ for STM's because this amount of space is required just to read the entire input. The following lemma states that STMs can simulate IOTM's efficiently enough for our purposes.

Lemma 2.13. If $A \in \text{NTIME}(T(n))$ where $T(n) \geq n+1$ (if $A \in \text{NSPACE}(S(n))$) then there is an STM which accepts A within time $(T(n))^2$ (resp., within space $\max(S(n), n+1)$).

Proof. The proof follows by straightforward simulation of a multi-tape Turing machine by a one tape Turing machine [HS65] (see also [HU69]). Note that STM's possess "constant factor speedup" similar to Pact 2.8.

The simulated IOTM may not operate within the given resource bound $T(n)$ or $S(n)$ on a finite subset of A . However the simulating STM can handle these finite exceptions by table look-up in its finite state control (cf. Lemma 2.3).

The one tape machine can be easily modified to operate on a one-way infinite tape [HU69]. This modification is usually implemented by keeping a marker $\#$ on the leftmost tape square. The simulating STM can fulfill the acceptance convention by always keeping another marker $\#'$ on the rightmost tape square thus far visited. If the simulated IOTM ever enters its accepting state, the simulating STM can erase its tape in a left sweep from $\#'$ to $\#$ and enter state q_a without moving after $\#$ has been erased. Moreover, this is the only situation in which q_a is entered. \square

The remainder of section 2.2 treats a portion of the technical machinery to be used in describing the computations of STM's. We wish to formalize the statement that, given i.d.'s d_1 and d_2 of M , one can determine if $d_2 \in \text{Next}_M(d_1)$ or not by making "local checks". A "local check" consists of comparing the $(j-1)^{\text{th}}$, j^{th} , and $(j+1)^{\text{th}}$

symbols of \mathbf{d}_1 and \mathbf{d}_2 for some j , $2 \leq j \leq |\mathbf{d}_1| - 1$. We can conclude $\mathbf{d}_2 \in \text{Next}_M(\mathbf{d}_1)$ if and only if all local checks succeed. This is now formalized in a useful technical lemma.

Lemma 2.14. Let $M = (I, \Gamma, Q, \delta, q_0, q_a)$ be an STM. Assume $\$ \notin \Gamma \cup Q$. Let $\Sigma = \Gamma \cup Q \cup \{\$\}$. There is a function $N_M: \Sigma^3 \rightarrow \Sigma^3$ with the following properties.

- (1). Let \mathbf{d}_1 be any i.d. of M , let $k = |\mathbf{d}_1|$, and write
 $\$ \mathbf{d}_1 \$ = d_{10} d_{11} d_{12} \cdots d_{1k} d_{1,k+1}$ where $d_{1j} \in \Sigma$ for $0 \leq j \leq k+1$.
Let $\$ \mathbf{d}_2 \$ = d_{20} d_{21} d_{22} \cdots d_{2k} d_{2,k+1}$ where $d_{2j} \in \Sigma$ for $0 \leq j \leq k+1$.
Then
 $\mathbf{d}_2 \in \text{Next}_M(\mathbf{d}_1)$ iff $d_{2,j-1} d_{2,j} d_{2,j+1} \in N_M(d_{1,j-1} d_{1,j} d_{1,j+1})$
for all j , $1 \leq j \leq k$.
- (2). For all $\sigma_1, \sigma_2, \sigma_3, \sigma_1', \sigma_2', \sigma_3' \in \Sigma$, if $\sigma_1' \sigma_2' \sigma_3' \in N_M(\sigma_1 \sigma_2 \sigma_3)$,
then $\sigma_i' = \$ \Leftrightarrow \sigma_i = \$$ for $i = 1, 2, 3$.

Proof. Four cases are involved in the specification of N_M .

- (i). N_M must satisfy condition (2) of the lemma.
- (ii). If $\sigma_1, \sigma_2, \sigma_3 \notin Q$, then σ_2 cannot change in going to some next i.d.
- (iii). If $\sigma_2 \in Q$ and $\sigma_3 \in \Gamma$, then each move in $6(\sigma_2, \sigma_3)$ uniquely determines one word in $N_M(\sigma_1 \sigma_2 \sigma_3)$.
- (iv). If $\sigma_2 \in Q$ and $\sigma_3 = \$$ then $N_M(\sigma_1 \sigma_2 \sigma_3) = \emptyset$.

N_M is precisely specified as follows. For each $\sigma_1 \sigma_2 \sigma_3 \in \Sigma^3$,

$N_M(\sigma_1\sigma_2\sigma_3) = \{ \sigma_1'\sigma_2'\sigma_3' \in \Sigma^3 \mid \sigma_1,\sigma_2,\sigma_3,\sigma_1',\sigma_2',\sigma_3' \text{ satisfy all conditions (i),(ii),(iii), and (iv) below } \}.$

- (i). $\sigma_i = \$$ iff $\sigma_i' = \$$ for $i = 1,2,3$.
- (ii). If $\sigma_1,\sigma_2,\sigma_3 \notin Q$ then $\sigma_2 = \sigma_2'$.
- (iii). If $\sigma_2 \in Q$ and $\sigma_3 \in \Gamma$ then

$$\sigma_1'\sigma_2'\sigma_3' \in \bigcup_{\mu \in \delta(\sigma_2,\sigma_3)} N_M^1(\sigma_1,\mu),$$

where for arbitrary $\sigma \in \Sigma$ and $\mu = (q',s',m) \in Q \times \Gamma \times \{-1,0,1\}$

$$N_M^1(\sigma,\mu) = \begin{cases} \{ \sigma q' s' \} & \text{if } m = 0 \\ \{ q' \sigma s' \} & \text{if } m = -1 \\ \{ a s' q' \} & \text{if } m = 1 \end{cases}$$

- (iv). $\sigma_2 \notin Q$ or $\sigma_3 \neq \$$.

The proof that N_M satisfies condition (1) of the lemma is straightforward and is left as an exercise. \square



Chapter 3. Efficient Reducibility

In this section we introduce a concept which will play a key role in the remainder of the paper. This is the concept of efficient reducibility.

Reducibility techniques have for some time been standard tools of recursive function theory (cf. [Rog67]). Set A is reducible to set B if the ability to answer questions about B enables one to answer questions about A by various effective methods. Then, for example, the undecidability of A implies the undecidability of B. However in order to get more detailed information about computational complexity, one must also show that the reducibility of A to B can be done "efficiently". Then if questions about A are known to be computationally complex, so must corresponding questions about B. See the Introduction for a further informal discussion of efficient reducibility.

3.1 Definitions.

There are a variety of inequivalent technical formulations of efficient reducibilities, differing not only in the degree of efficiency but also in the methods by which questions about A are reduced to questions about B. Many of these distinctions among efficient **reducibilities** are analyzed in [LLS74]. The distinctions are analogous to the differences among various reducibilities of recursion theory such as many-one, truth-table, Turing reducibility, etc. (cf. [Rog67]).

We shall use essentially one kind of efficient reducibility corresponding to the "**strong**" reducibility (many-one or one-one) of recursion theory. However we do use several different bounds on the efficiency in terms of time or space to obtain four different **reducibilities** of this kind.

Following a definitional suggestion of Knuth [Knu74], we henceforth refer to these particular reducibilities as "**transformations**".

Definition 3.1. Let $\left\{ \begin{array}{c} \underline{\text{logspace}} \\ \underline{\text{polylin}} \\ \underline{\text{poly}} \end{array} \right\}$ denote the class of functions $\{ f \mid f: I^* \rightarrow A^* \text{ for some finite alphabets } I, A, \text{ and there is a deterministic IOTM which computes } f$

$$\left\{ \begin{array}{l} \text{within space } \log n \\ \text{within time } p(n) \text{ and space } n \\ \text{within time } p(n) \end{array} \right\}$$

for some polynomial $p(n)$ }.

Definition 3.2. Let $L: \mathbb{N}^+ \rightarrow \mathbb{N}^+$. A function $f: I^+ \rightarrow A$ is said to be

length $L(n)$ bounded iff $|f(x)| \leq L(|x|)$ for all $x \in I^+$.

f is linear bounded iff there is a $c \in \mathbb{N}^+$ such that

$$|f(x)| \leq c|x| \quad \text{for all } x \in I^+.$$

Definition 3.3. (Efficient transformations).

Let $A \subseteq I^+$, $B \subseteq \Delta^+$ for some finite alphabets I, Δ .

($A \leq_{\log} B$; $A \leq_{\log\text{-lin}} B$; $A \leq_{\text{pl}} B$; $A \leq_{\text{eff}} B$) via f

iff f is a function, $f: I^+ \rightarrow \Delta^+$, such that

$$x \in A \text{ iff } f(x) \in B \quad \text{for all } x \in I^+, \text{ and}$$

($(\leq_{\log}) \quad f \in \text{logspace}$;

($\leq_{\log\text{-lin}}$) $f \in \text{logspace}$ and f is linear bounded ;

(\leq_{pl}) $f \in \text{polylin}$ and f is linear bounded ;

(\leq_{eff}) $f \in \text{poly}$).

Also, if $\text{eff} \in \{ \log, \log\text{-lin}, \text{pl} \}$ then

$$\underline{A \leq_{\text{eff}} B} \text{ iff } A \leq_{\text{eff}} B \text{ and } B \leq_{\text{eff}} A.$$

Note: The transformations defined above do not change if we require the function f to be computed by an IOTM with one work tape. Thus our definitions are equivalent to previous definitions of \leq_{\log} , $\leq_{\log\text{-lin}}$ [SM73], and \leq_{pl} [MS72].

Remark. It can be seen (by counting the number of possible i.d.'s)

that an IOTM which computes within space $\log n$ also computes within

polynomial time. Therefore $A \leq_{\log} B \Rightarrow A \leq_{\text{pl}} B$,

and $A \leq_{\log\text{-lin}} B \Rightarrow A \leq_{\text{pl}} B$.

The next lemma is **immediate** from the facts that logspace, polylin, and poly are each closed under functional composition. It should be obvious that polylin and poly are closed under composition. Lind and Meyer [LM74] prove that **logspace** is closed under composition; this proof is very similar to the proof of Lemma 3.6 to follow.

Lemma 3.4. Let $\leq_{\text{eff}} \in \{ \leq_{\text{log}}, \leq_{\text{log-lin}}, \leq_{\text{pl}}, \leq \}$. Let $A \leq_{\text{eff}} B$ and $B \leq_{\text{eff}} C$ via length $L_1(n)$, $L_2(n)$ bounded f_1 , f_2 respectively where $L_2(n)$ is monotone nondecreasing.

Then $A \leq_{\text{eff}} C$ via length $L_2(L_1(n))$ bounded $f_2 \circ f_1$.

The following definition is of central importance.

Definition 3.5. Let \mathcal{C} be a class of sets, B be a set, and \leq be a transformation.

- (1). $\mathcal{C} \leq B$ iff $A \leq B$ for all $A \in \mathcal{C}$.
- (2). B is I-complete in \mathcal{C} iff
 - (i). $\mathcal{C} \leq B$, and
 - (ii). $B \in \mathcal{C}$.
- (3). $\mathcal{C} \leq B$ via length order $L(n)$ iff for all $A \in \mathcal{C}$ there is a $c \in \mathbb{N}^+$ such that $A \leq B$ via some length $c \cdot L(n)$ bounded function.

All of the particular transformations described in the sequel are members of logspace. Lind and Meyer [LM74] give a machine independent characterization of **logspace** (which is similar in flavor to Ritchie's characterizations of other subrecursive classes [Rit63])

by which one can prove rigorously that our transformations do indeed belong to logspace. However such proofs are tedious and shed no new **light** on the main issues.

For this reason, we use \leq_{\log} and $\leq_{\log\text{-lin}}$ only in section 4.1 where our transformations are simple enough that their membership in **logspace** should be obvious. In some cases we sketch a verification that a particular transformation belongs to logspace, omitting many of the details by appeal to the reader's intuition about space bounded Turing machines. For convenience, Appendix II collects those closure properties and particular members of **logspace** which are used either explicitly or implicitly in these verifications.

In other sections, we claim only that transformations are of the types $\leq_{p\ell}$ or \leq ; closer examination reveals that these transformations also belong to logspace.

It is interesting to note that a few of our particular transformations can be easily modified to be computable within space zero, that is, computable by a deterministic finite state transducer with 2-way input. Aho and Ullman [AU70] prove that the class of zero-space computable functions is closed under composition, and hence that "0-space-transformable" is a transitive relation.

The notion of efficient reducibility was first formally defined by Cook [Co71a] (as a "Turing" version of \leq). Efficient reducibility was used as a proof technique earlier in [MM71]. Karp [Kar72] and others have used \leq as a means of relating the complexities of various **combinatorial problems**. It is noted in [SM73] and [Jon73]

that many of the particular polynomial time reducibilities presently in the literature can actually be done within space $\log n$, (although it would be suprising if $\text{poly} = \text{logspace}$ in general, cf. Open Question 2.10.C.).

3.2 Applications to Complexity Bounds.

We shall use efficient transformations as a means of relating the computational complexities of problems. Informally, if \leq_{eff} is a transformation, and $A \leq_{\text{eff}} B$ via f , then one can conclude

$$\text{"Complexity of } A \text{"} \leq \text{"Complexity of } B \text{"} + \text{"Complexity of } f \text{"} .$$

Thus the **computational** resources required to accept B are "no less than" the resources required to accept A provided that the resources used in computing f are low order compared to those used in accepting B .

This is made precise by a **lemma** for the case $\leq_{\text{eff}} = \leq_{\log}$. The technical details involved in proving such a result for the case \leq_{\log} are presented in [SM73] and [Jon73] . We reproduce a proof sketch for this **lemma** here because minor modifications to the proof are used implicitly in section 3.3.

Lemma 3.6. Suppose $A \leq_{\log} B$ via f where f is length $L(n)$ bounded, and M is a nondeterministic (deterministic) IOTM which accepts B within time $T(n)$ and within space $S(n)$ where $T(n)$ and $S(n)$ are monotone nondecreasing.

Then there is a polynomial $p(n)$ and nondeterministic (deterministic) IOTM's M' and M'' such that:

M' accepts A within time $T'(n) = p(n) \cdot T(L(n))$ and within space $S'(n) = S(L(n)) + \log n$;

M'' accepts A within time $T''(n) = T(L(n)) + p(n)$.

Therefore:

$$\begin{aligned} & B \in \begin{Bmatrix} \text{NTIME} \\ \text{DTIME} \end{Bmatrix}(T(n)) \Rightarrow A \in \begin{Bmatrix} \text{NTIME} \\ \text{DTIME} \end{Bmatrix}(T(L(n)) + p(n)) \\ \text{and} \\ & B \in \begin{Bmatrix} \text{NSPACE} \\ \text{DSpace} \end{Bmatrix}(S(n)) \Rightarrow A \in \begin{Bmatrix} \text{NSPACE} \\ \text{DSpace} \end{Bmatrix}(S(L(n)) + \log n). \end{aligned}$$

Proof. The obvious M'' , given an input x , first computes $f(x)$ and writes $f(x)$ on some work tape. As was noted before, $f \in \text{logspace}$ implies that f can be computed deterministically within polynomial time. M'' then simulates M on input $f(x)$. M is time $T(n)$ bounded (on accepted words) and is computing on the input $f(x)$ of length at most $L(|x|)$. Recall $T(n)$ is nondecreasing. M'' clearly accepts A within time $T''(n)$.

This obvious approach may not work for M' . The difficulty is that M' cannot write $f(x)$ on a work tape because $|f(x)|$ might be much larger than $\log|x| + S(L(|x|))$; however M' must operate within space $S(n)$. Instead, M' with input x can simulate the computation of M on input $f(x)$ by recording on its work tape an instantaneous description of the computation of M , including the position j in $f(x)$ which the input head of M would occupy if the input to M were actually $f(x)$.

$f \in \text{logspace}$ implies $f \in \text{poly}$, and therefore

$$j \leq |f(x)| \leq p'(|x|) \quad \text{for some polynomial } p'(n);$$

only $c \cdot \log|x|$ extra work tape squares are required to record j in binary. To simulate another step in the computation of M on input $f(x)$, M' computes the j^{th} digit of $f(x)$ within space $\log|x|$ and time

$p'(|x|)$, and updates the **i.d.** of M accordingly.

After an application of speedup (Fact 2.8), it is easy to see that M' accepts A within time $T(n)$ and space $S(n)$. \square

For completeness, similar results for the other transformations are stated next, even though we shall not have occasion to use **Lemma 3.7** in its entirety.

Lemma 3.7. Assume $T(n)$ and $S(n)$ are nondecreasing.

(1). If $A \leq_{p\ell} B$ then

$$B \in \left\{ \begin{array}{c} \text{NTIME} \\ \text{DTIME} \end{array} \right\}(T(n)) \Rightarrow A \in \left\{ \begin{array}{c} \text{NTIME} \\ \text{DTIME} \end{array} \right\}(p(n) + T(cn))$$

$$B \in \left\{ \begin{array}{c} \text{NSPACE} \\ \text{DSPACE} \end{array} \right\}(S(n)) \Rightarrow A \in \left\{ \begin{array}{c} \text{NSPACE} \\ \text{DSPACE} \end{array} \right\}(n + S(cn))$$

for some constant $c \in \mathbb{N}^+$ and polynomial $p(n)$.

(2). If $A \leq B$ then

$$B \in \left\{ \begin{array}{c} \text{NTIME} \\ \text{DTIME} \end{array} \right\}(T(n)) \Rightarrow A \in \left\{ \begin{array}{c} \text{NTIME} \\ \text{DTIME} \end{array} \right\}(p(n) + T(p(n)))$$

$$B \in \left\{ \begin{array}{c} \text{NSPACE} \\ \text{DSPACE} \end{array} \right\}(S(n)) \Rightarrow A \in \left\{ \begin{array}{c} \text{NSPACE} \\ \text{DSPACE} \end{array} \right\}(p(n) + S(p(n)))$$

for some polynomial $p(n)$.

The proof of **Lemma 3.7** is by the obvious approach used to construct M' in the proof of **Lemma 3.6**.

Our next objective is to give the basic outline which the majority of results herein will follow. We give the outline for a space result; a time result is analogous.

Outline 3.8. Let B be a particular set of interest.

(1). Choose a class \mathcal{F} of nondecreasing functions from \mathbb{N}^+ to \mathbb{Q}^+ .

\mathcal{F} will in general depend on B . Let

$$\mathcal{G} = \bigcup_{S(n) \in \mathcal{F}} \text{NSPACE}(S(n)).$$

For example, we may take $\mathcal{G} = \text{EXPSPACE}$ or $\mathcal{G} = \text{POLYSPACE}$ in particular cases.

(2). Prove that $\mathcal{G} \leq_{\text{eff}} B$ (via length order $L(n)$),

where \leq_{eff} is an appropriate efficient transformation.

In many of our examples, the proof is analogous to an "arithmetization" of Turing machines so that questions about Turing machines accepting sets in \mathcal{G} can be transformed into questions about B . This of course is the main portion of most of our proofs.

(3). (Deduce a lower bound on the complexity of B).

Since the majority of our particular transformations are linear bounded, assume here that $L(n) = n$.

By Fact 2.11 (the nondeterministic hierarchy theorem), find a "hard" set $A \in \mathcal{G}$ such that $S(n)$ is a large lower bound on the space complexity of A ; that is, $A \notin \text{NSPACE}(S(n))$. Also choose $S(n)$ to be nondecreasing.

Now by part (2) above, $A \leq_{\text{eff}} B$ via f , where f is length bn bounded for some $b \in \mathbb{N}^+$.

We claim that $S(\lceil n/b \rceil)$ is a lower bound on the space complexity of B . For suppose $B \in \text{NSPACE}(S(\lceil n/b \rceil))$. Lemma 3.6 or 3.7 then implies $A \in \text{NSPACE}(S(n) + F(n))$ where $F(n)$ is the space required to compute f . Assuming $F(n) \leq S(n)$ because f is an efficient transformation,

$A \in \text{NSPACE}(2 \cdot S(n)) = \text{NSPACE}(S(n))$ by Fact 2.8 (constant factor speedup).

This contradicts one condition A was chosen to satisfy, and therefore

$$B \notin \text{NSPACE}(S(\lceil n/b \rceil)).$$

For example, in the proof of Theorem 4.12 we have

$\text{EXPSPACE} \leq_{\log\text{-lin}} B$. We can then choose $A \in \text{NSPACE}(2^n)$ but

$A \notin \text{NSPACE}((2-\epsilon)^n)$ if $\epsilon > 0$, and conclude

$$B \notin \text{NSPACE}(c^n) \quad \text{where } c = (2-\epsilon)^{1/b},$$

and b is such that $A \leq_{\log\text{-lin}} B$ via some length bn bounded function.

(4). In ~~some~~ ^{some} cases, we also show $B \in \mathfrak{G}$; thus B is $\leq_{\text{eff}}\text{-complete}$ in \mathfrak{G} . A completeness result in a sense pins down the complexity of B . $B \in \mathfrak{G}$ implies an upper bound; $\mathfrak{G} \leq_{\text{eff}} B$ usually provides a lower bound as in (3).

Remark. Step (3) only requires $A \leq_{\text{eff}} B$ for the particular "hard" set A , rather than $\mathfrak{G} \leq_{\text{eff}} B$. However the latter general **statement** is no harder to prove than the former particular statement in the cases we consider. Also, the general statement may have other implications for B . (See for example section 3.3.)

As noted above, the main part of the proofs which follow the preceding outline will consist in the proof of (2). The details involved in (3) will be given for a few results and left as simple exercises for others. The upper bound required for (4) will be verified by giving an informal description of an algorithm which accepts B .

For most examples there remain gaps between known lower and upper bounds on their deterministic time complexity. As was mentioned earlier, **these** gaps correspond to the gaps stated in Open Questions 2.10,

A particular instance of this relationship is the following. Several workers [Edm65], [Kar72] have proposed that a problem can be considered computationally "**tractable**" only if it can be solved by a deterministic algorithm within polynomial time, that is, only if it is a member of \mathcal{P} . The following **lemma** can be used to relate the tractability of various particular problems to the open questions " $\mathcal{P} = \text{NP}?$ " and " $\text{CSL} \subseteq \mathcal{P}?$ ". A result of this flavor was first noted in [Co71a].

Lemma 3.9. Let $\leq_{\text{eff}} \in \{ \leq_{\log}, \leq_{\log\text{-lin}}, \leq_{\text{pol}}, \leq_{\text{poly}} \}$. Let B be a set, and \mathcal{C} be a class of sets. If B is \leq_{eff} -complete in \mathcal{C} then

$$B \in \mathcal{P} \Leftrightarrow \mathcal{C} \subseteq \mathcal{P}.$$

Proof. Immediate from definitions and Lemmas 3.6 and 3.7. \square

Following the original work of Cook [Co71a] and Karp [Kar72], a large number of common combinatorial problems have been shown to be \leq_{\log} -complete in NP (see for example [Sah72], [Set73], [U1173], [GJS74]); such problems are called NP-complete. By Lemma 3.9, either all or none of the NP-complete problems are members of \mathcal{P} ; moreover, the former case holds if and only if $\mathcal{P} = \text{NP}$.

We shall make a few additions to the list of NP-complete problems. In these cases, where we show that some particular B is \leq_{\log} -complete in

NP, it will be seen that an application of step(3) of the outline yields only a trivial bound on the nondeterministic time complexity of \mathcal{B} . (One could show that \mathcal{B} requires time \sqrt{n} in certain cases, but this is trivial because time n is required just to read the entire input.) In these cases, step(3) of the outline can simply be replaced by the statement that $B \in \mathcal{P}$ iff $\mathcal{P} = \text{NP}$.

3.3 Other Applications.[†]

Lemma 3.6 or 3.7 can be loosely interpreted as stating that the property "i.o. lower complexity bound" of sets translates through an efficient transformation. For example, as Outline 3.8. (3) shows, if $A \leq_{\log\text{-lin}} B$ and A possesses the i.o. lower bound $S(n)$ on space complexity, then B possesses the i.o. lower bound $S(\lceil cn \rceil)$ on space complexity for some $c \in \mathbb{Q}^+$, (provided $\log n = o(S(n))$).

The field of axiomatic complexity theory (initiated by Blum [B167]) has considered many other interesting computational properties. For example: (A). There are known to exist sets which possess no optimal acceptance algorithm in the sense that any algorithm accepting the set can be effectively sped up on infinitely many inputs; (B). There are known to exist sets for which any acceptance algorithm consumes large amounts of time and space on some input of length n for all sufficiently large n (rather than just infinitely many n). However these properties have previously been known to hold only for 'sets constructed by diagonalizations or other esoteric methods.

The purpose of this section is to show that these two properties also "translate through" an efficient transformation and can therefore be shown to hold for natural sets. Our aim is only to prove particular results indicative of the types of results one can obtain rather than to give a general treatment. We concentrate attention on the space

[†]The material of §3.3 is not used directly in the sequel.

measure; analogous results for the time measure can be obtained similarly.

For the purposes of this section, assume all transformations f mentioned satisfy $|f(x)| \geq |x|$ for all x .

A. Effective i.o. speedup.

Definition 3.10. Let $A \subseteq \Sigma^+$ be a set of words. A possesses $S(n)$ -to-log effective i.o. speedup iff given any deterministic IOTM M which accepts A one can effectively find a deterministic IOTM M' which accepts A such that:

- (1). $\text{Space}_{M'}(x) \leq \text{Space}_M(x)$ for all $x \in A$;
 and
 (2). There exist infinitely many $x \in A$ such that

$$\begin{aligned} & \text{Space}_M(x) > S(|x|) \\ \text{and} \\ & \text{Space}_{M'}(x) = \log |x| . \end{aligned}$$

Thus the new algorithm M' never uses more space than the old M (on accepted words), but in general uses much less space than M on infinitely many inputs.

Remark. For deterministic M , we can extend the definition of $\text{Space}_M(x)$ in the obvious way to include also those inputs x which M rejects. (In §2.1 $\text{Space}_M(x)$ is defined only if M accepts x). Then one can

replace (1) of Definition 3.10 by " $\text{Space}_M(x) \leq \text{Space}_M(x)$ for all $x \in \Sigma^+$." The main result (Theorem 3.13) of this section is true with respect to this modified definition of effective **i.o.** speedup, although the proof requires minor changes.

Within the framework of axiomatic complexity theory, Blum [B171] first proved the existence of sets with effective **i.o.** speedup. By combining Blum's techniques with methods for constructing sets with tight upper **and** lower bounds on space complexity, ^[cf. MM71] one can prove the following.

Fact 3.11. Let $S_1(n)$, $S_2(n)$ be such that $S_2(n)$ is constructable, $S_1(n) \geq \log n$, and $S_1(n) = o(S_2(n))$. Then there is a set $A \in \text{DSPACE}(S_2(n))$ such that A possesses $S_1(n)$ -to-log effective **i.o.** speedup.

^{We omit a proof, but remark}
~~The unpublished proof of Fact 3.11 is due to A.R. Meyer. We remark~~
 that the proof actually shows that A possesses " $S_1(n)$ -to-zero effective **i.o.** speedup"; this notion is defined as in Definition 3.10, where 0 replaces $\log |x|$.

To complete the proof that the speedup property translates through **an** efficient transformation, we need an additional "efficient invertibility" condition on the transformation.

Definition 3.12. Let $f : \Sigma^+ \rightarrow A$. f is logspace-invertible iff f is one-to-one, and the function $f^{-1} : A \rightarrow \Sigma^+ \cup \{u\}$ defined by

$$f^{-1}(y) = \begin{cases} x & \text{if } f(x) = y \text{ for some } x \in \Sigma^+ \\ u & \text{otherwise} \end{cases} \quad (\text{where } u \notin \Sigma)$$

is a member of logspace.

We now show that the speedup property translates through "invertible" $\leq_{\log\text{-lin}}$.

Theorem 3.13. Assume $A \leq_{\log\text{-lin}} B$ via f , where f is **logspace-invertible** and $|f(x)| \geq |x|$ for all x . Let $S(n)$ be nondecreasing and satisfy $S(n) \geq \log n$. If A possesses **$S(n)$ -to-log effective i.o. speedup**, then B possesses **$S(\lceil cn \rceil)$ -to-log effective i. o. speedup** for some $c \in \mathbb{Q}^+$.

Proof. Let $A \subseteq \Sigma^+$, and $B \subseteq \Delta^+$ for finite alphabets Σ, Δ .

Let M_1 be any deterministic IOTM which accepts B . Effectively find a deterministic IOTM M_2 which accepts A such that:

$$(1). \quad \text{Space}_{M_2}(x) \leq (1/2)(\text{Space}_{M_1}(f(x)) + \log|x|) \quad \text{for all } x \in A.$$

M_2 operates like the procedure M' in the proof of Lemma 3.6, after this procedure has been sped-up by a factor of $1/2$ à la Fact 2.8.

Since A possesses **$S(n)$ -to-log effective i.o. speedup**, effectively find M_3 accepting A where:

(2). $\text{Space}_{M3}(x) \leq \text{Space}_{M2}(x)$ for all $x \in A$, and

(3). There is an infinite set $X \subseteq A$ such that:

(3.1). $\text{Space}_{M2}(x) > S(|x|)$ for all $x \in X$,

and

(3.2). $\text{Space}_{M3}(x) \leq \log|x|$ for all $x \in X$.

Let $f^{-1} \in \text{logspace}$ be as in Definition 3.12.

We describe a deterministic IOTM $M4$ which accepts B . $M4$ runs two procedures $M1$ and $P1$ in parallel. Procedure $P1$ is procedure $P1'$ sped-up (Fact 2.8) by a factor of $1/3$. $P1'$ operates as follows.

$P1'$. Given input $y \in \Delta^+$:

Begin a computation of $f^{-1}(y)$. If $f^{-1}(y) = u$, then halt.

If $f^{-1}(y)$ produces an output symbol other than u , stop computing $f^{-1}(y)$ and simulate $M3$ on input $f^{-1}(y)$ as in the proof of Lemma 3.6(M'). (Recall $f^{-1} \in \text{logspace}$).

END $P1'$.

Therefore:

(4). $\text{Space}_{P1}(y) \leq (1/3)(\text{Space}_{M3}(f^{-1}(y)) + \log|y| + \log|f^{-1}(y)|)$
for all $y \in B$.

Given input $y \in \Delta^+$, $M4$ can run $M1$ and $P1$ in "parallel" in such a way that $M4$ accepts y iff either $M1$ or $P1$ accepts y , and

(5). $\text{Space}_{M4}(y) \leq \min(\text{Space}_{M1}(y), \text{Space}_{P1}(y))$ for all $y \in B$.

(Informally, $M4$ uses a "new" tape square iff both $P1$ and $M1$ require another tape square).

Now if $f^{-1}(y) \neq u$, then $f^{-1}(y) \in A \Leftrightarrow y \in B$. Thus $M4$ accepts

B correctly.

Let $b \in \mathbb{N}^+$ be such that $|x| \leq |f(x)| \leq b|x|$ for all $x \in \Sigma^+$.

Let $c = 1/b$.

We now verify that M_4 satisfies the conditions of Definition 3.10 to be a $S(\lceil cn \rceil)$ -to-log "sped-up" version of M_1 . First, by (5),

$$\text{Space}_{M_4}(y) \leq \text{Space}_{M_1}(y) \quad \text{for all } y \in B.$$

Let $Y = f(X) = \{ f(x) \mid x \in X \}$. Note Y is infinite because f is one-to-one. Also, $Y \subseteq B$ because f transforms A to B and $X \subseteq A$. Y is the set of inputs on which M_4 uses space $\log n$ while M_1 requires space $S(\lceil cn \rceil)$.

To verify this, let $y \in Y$ be arbitrary and let $x = f^{-1}(y)$, so $x \in X \subseteq A$. Recall $c|y| \leq |x| \leq |y|$. First:

$$\begin{aligned} \text{Space}_{M_4}(y) &\leq (1/3)(\text{Space}_{M_3}(x) + \log|y| + \log|x|), \quad \text{by (4) and (5),} \\ &\leq (1/3)(\log|x| + \log|y| + \log|x|), \quad \text{by (3.2),} \\ &\leq \log|y|, \quad \text{because } |x| \leq |y|. \end{aligned}$$

Now suppose that $\text{Space}_{M_1}(y) \leq S(\lceil c|y| \rceil)$. Then:

$$\begin{aligned} \text{Space}_{M_2}(x) &\leq (1/2)(S(\lceil c|y| \rceil) + \log|x|), \quad \text{by (1) and by assumption,} \\ &\leq S(|x|), \quad \text{by } c|y| \leq |x|, S \text{ is nondecreasing, and} \end{aligned}$$

$$S(n) \geq \log n.$$

Since $x \in X$, this contradicts (3.1) and therefore

$$\text{Space}_{M_1}(y) > S(\lceil c|y| \rceil).$$

Since M_1 was arbitrary, we are done. \square

Corollaries like the one below follow **immediately** from Fact 3.11 and Theorem 3.13. For example, Fact 3.11 implies that MPSPACE contains **some** set with 2^n -to-log effective **i.o.** speedup.

Corollary 3.14. Let B be a set such that $\text{EXPSPACE} \leq_{\log\text{-lin}} B$.

Assume furthermore that for all $A \in \text{EXPSPACE}$, $A \leq_{\log\text{-lin}} B$ via some lagspace-invertible function f such that $|f(x)| \geq |x|$.

There is a rational $c > 1$ such that B possesses c^n -to-log effective **i.o.** speedup.

B. Lower bounds which hold for almost all input lengths.

As was mentioned before, we shall be content to show that lower complexity bounds hold infinitely often. However, given any recursive $S(n)$, there is known to exist a set A such that any deterministic algorithm accepting A uses more than space $S(|x|)$ on all sufficiently long inputs x. (Here we count space on all inputs rather than just those $x \in A$).

It would be suprising to find an uncontrived example of a set with this property since the natural examples all seem to have "easy subcases" which occur infinitely often. For example, let TAUT denote the set of all Boolean formulas in disjunctive **normal** form which are tautologies. It has been conjectured [Co71a] that $\text{TAUT} \notin P$. Let $X \subset \text{TAUT}$ denote the(infinite) set of such formulas of the **form** $F \vee x_1 \vee \neg x_1 \vee G$, where F and G are formulas and x_1 is a Boolean variable. A deterministic

algorithm M accepting TAUT can first check within polynomial time if the input x is in X . M accepts **immediately** if $x \in X$, or applies a resolution procedure if $x \notin X$. Therefore we cannot show that TAUT is difficult on all sufficiently large inputs.

However, we can show that certain natural sets are difficult on some input of length n for all sufficiently large n . We would then say the set is difficult ~~a.e.~~ (almost everywhere) with respect to input lengths. This question of "frequency of difficult inputs" is important, and there are **some** obvious directions for further inquiry which we have not had time to pursue. For example, although we can show that the number of difficult inputs of length n **grows** unboundedly with n , we have not been able to show that a nonzero fraction of the length n inputs are difficult.

Definition 3.15. Let A be a set of words. A requires space $S(n)$ a.e. n iff for each deterministic IOTM which accepts A there is a $n_0 \in \mathbb{N}$ such that

$$(\forall n \geq n_0) (\exists x \in A) [|x| = n \text{ and } \text{Space}_M(x) \geq S(n)].$$

Fact 3.16 (Stearns, Hartmanis, Lewis [SHL65]). Let $S_1(n)$, $S_2(n)$ be such that $S_2(n)$ is constructable, $S_1(n) \geq \log n$, and $S_1(n) = o(S_2(n))$.

Then there is a set $A \in \text{DSPACE}(S_2(n))$ such that A requires space $S_1(n)$ **a.e.** n .

Remark. The proof of Fact 3.17 is by a fairly straightforward **diagonalization**. The reader should be aware that by using more subtle techniques one can construct sets A as in Fact 3.16 such that any IOTM M accepting

A satisfies $\text{Space}_M(x) \geq S_1(|x|)$ for all but finitely many x (rather than just one x of each length). We would then say that A requires space $S_1(n)$ **a.e.** ^(as opposed to 'a.e.n'). For arbitrary recursive $S_1(n)$, Rabin [Rab60] first exhibited sets which require space $S_1(n)$ **a.e.** Blum [Bl67] shows that the complexity of these sets can be "compressed", that is, one can also place tight upper bounds ($S_2(n)$) on their complexity. Trachtenbrot [Tra70] and Meyer and McCreight [MM71] show that the two bounds can be compressed as tightly as $S_1(n) = o(S_2(n))$.

Definition 3.17. Let $B \subseteq A^+$. B is invariant under padding iff there is a symbol $\# \in \Delta$ such that $y \in B \implies y\# \in B$, for all $y \in \Delta^+$.

Theorem 3.18. Assume $A \leq_{\log\text{-lin}} B$ via f , where B is invariant under padding and $|f(x)| \geq |x|$. Let $S(n)$ be nondecreasing and satisfy $S(n) \geq \log n$. If A requires space $S(n)$ **a.e.** n , then B requires space $S(\lceil cn \rceil)$ **a.e.** n for some $c \in \mathbb{Q}$

Proof. Let $A \leq_{\log\text{-lin}} B$ via f , where $|x| \leq |f(x)| \leq b|x|$ for some $b \in \mathbb{N}^+$ and all x .

Let M be an arbitrary deterministic IOTM which accepts B. We describe an IOTM M' which accepts A.

M' . Given input x :

For $s = 0, 1, 2, 3, \dots$ do:

For $j = 0, 1, 2, 3, \dots, b|x|$ do:

Simulate M on input $f(x) \cdot \#^j$,

(A trivial modification of the proof of Lemma 3.6 shows that this can be done within space at most

$$\text{Space}_M(f(x) \cdot \#^j) + \log |x| \text{ ;}$$

If during this simulation M' detects that $\text{Space}_M(f(x) \cdot \#^j) > s$, then erase everything on the work tapes except the counters s and j , and continue ;

If M accepts $f(x) \cdot \#^j$, then accept x .

EM)

~~EM)~~ M' .

M' obviously accepts A .

Define $\text{Reduce}(x) = \{ f(x) \cdot \#^j \mid 0 \leq j \leq b|x| \}$. In a computation on input x , M' considers all words in $\text{Reduce}(x)$ as inputs to M .

Two facts about $\text{Reduce}(x)$ are useful. The first is obvious. The second follows from $|x| \leq |f(x)| \leq b|x|$.

- (1). $(x \in A \text{ and } y \in \text{Reduce}(x)) \implies (y \in B \text{ and } |y| \leq 2b|x|)$.
- (2). For $n \in \mathbb{N}^+$, define the interval $I_n = \{ m \in \mathbb{N} \mid bn \leq m \leq bn + n \}$. Then for all $n \in \mathbb{N}^+$, for all $x \in A$ with $|x| = n$, for all $m \in \mathbb{N}^+$ with $m \in I_n$, there is some $y \in \text{Reduce}(x)$ with $|y| = m$.

It is helpful to picture (2) as stating that all x of length n are mapped onto the entire interval I_n . If any $m \in I_n$ has the property that M is "efficient" on all inputs $y \in B$ of length m , then M' is "efficient" on all inputs $x \in A$ of length n . This is true because (if the counters s and j are represented in radix notation),

- (3). $\text{Space}_{M'}(x) \leq F(x) + \log F(x) + k \cdot \log |x|$ for all $x \in A$,
 where $F(x) = \min\{ \text{Space}_M(y) \mid y \in \text{Reduce}(x) \}$, $\log F(x)$ = space
 for counter s , and $k \cdot \log |x|$ = space for counter j and simulation
 overhead where $k \in \mathbb{N}^+$.

Let $c = 1/2b$.

Suppose the conclusion of the theorem is false. That is, assume
 there is a deterministic IOTM M which accepts B and an infinite set
 $E \subseteq \mathbb{N}^+$ of "easy lengths" such that

- (4). $\text{Space}_M(y) < S(\lceil c|y| \rceil)$ for all $y \in B$ with $|y| \in E$.

Let E' be the corresponding set of "easy lengths" for M' .

$$E' = \{ n \in \mathbb{N}^+ \mid m \in I_n \text{ for some } m \in E \}.$$

E' is infinite because $I_n \cap I \neq \emptyset$ for all $n \geq b$.

We claim that

- (5). $\text{Space}_{M'}(x) < (k+2) \cdot S(|x|)$ for all $x \in A$ with $|x| \in E'$.

This, combined with constant factor speed-up (Fact 2.8), contradicts
 the fact that A requires space $S(n)$ a.e. n . It remains only to prove (5).

Let $x \in A$ with $|x| \in E'$ be arbitrary. By the definition of E' ,
 together with fact (2), there is some $y \in \text{Reduce}(x)$ with $|y| \in E$.
 Also, by (1), $y \in B$ and $c|y| \leq |x|$. Now,

$$\begin{aligned} F(x) &\leq \text{Space}_M(y), \text{ by definition of } F(x), \\ &< S(\lceil c|y| \rceil), \text{ by assumption (4) because } |y| \in E, \\ &\leq S(|x|), \text{ because } S \text{ is nondecreasing.} \end{aligned}$$

Now by (3),

$$\text{Space}_{M'}(x) < (k+2) \cdot S(|x|) \text{ because } S(n) \geq \log n.$$

Therefore (5) is proved. \square

. As in part (A) above, corollaries now follow immediately from Fact 3.16 and Theorem 3.18. For example, if $\text{EXPSPACE} \leq_{\log\text{-lin}} B$, and B is invariant under padding, then B requires space c^n a.e. n for some rational $c > 1$.

The particular method of padding (Definition 3.17) was chosen mainly for simplicity. It illustrates the point that more information about "frequency of difficult inputs" can be obtained.

Many natural examples already possess, even without the artificially added $\#$ symbol, a slightly weaker kind of padding property defined below. This weaker property is also sufficient to imply Theorem 3.18 by a very similar proof which we omit.

Definition 3.19. Let $B \subseteq A^+$. B is naturally padded iff there is a symbol $d \notin A$, a $j_0 \in \mathbb{N}$, and a function $p \in \text{logspace}$, $p: \Delta^+ d^* \rightarrow \Delta^+$, such that:

- and
- (1). $B \cdot d^* \log B \text{ via } p$;
 - (2). $|p(y \cdot d^j)| = |y| + j$, for all $y \in \Delta^+$ and all integers $j \geq j_0$.

However, the condition that B be invariant under some notion of "padding" is necessary to reach the conclusion of Theorem 3.18. For any large recursive $S(n)$, let $A \subseteq \{0,1\}^+$ be a recursive set which requires space $S(n)$ a.e. n . Define the set B by

$$B = \{ x \cdot 0^{|x|} \mid x \in A \} \cup \{ x \in \{0,1\}^+ \mid |x| \text{ is odd} \}.$$

Clearly $A \leq_{\log\text{-lin}} B$, but it is easy to design an IOTM M which accepts B and for which $\text{Space}_M(x) = 0$ for all x such that $|x|$ is odd.



Chapter 4. Regular-Like Expressions

Regular expressions are a family of notations for describing sets of words. They were first introduced in automata theory as an alternative characterization of the languages (sets of words) accepted by finite state machines [Kle 56], [CEW58], [MY60]. A treatment of regular expressions can be found in most automata theory texts, for example [Har65], [Sal69]. [Brz62] is an early survey paper. More recently, regular expressions have been used to define the lexical analysis phase of compilers [Gri71], and to specify patterns for pattern matching algorithms [AHU74] and text editors.

Given two regular expressions, one might want to determine if they are equivalent, that is, if they describe the same set of words. Several workers, for example [Gin67], [Brz64], have given algorithms which solve this equivalence problem. However no deterministic algorithm has been found which runs within time bounded by a polynomial in the input length.

In this chapter, ~~inter alia~~, we show (Theorem 4.13) that the problem of recognizing equivalence of regular expressions has the same time and space requirements as the problem of deciding membership of words in context sensitive languages. Theorem 4.13 provides strong evidence that there is no deterministic polynomial time algorithm for this equivalence problem, or for the related problem of minimizing the size of nondeterministic finite state automata [cf. KW70].

There is reason to believe that the general membership problem for context sensitive languages cannot be solved in deterministic polynomial time. In particular, $P \neq NP$ implies $P \neq POLYSPACE$ iff $CSL = P \neq \emptyset$ [cf. Bo72]. (See the discussion following Lemma 3.9 concerning the P versus NP question.) Because this question whether $CSL = P \neq \emptyset$ is open, we cannot actually prove that the equivalence problem for regular expressions is not in \mathcal{O}' . However we can prove that

$CSL = \emptyset \neq \emptyset$ iff the equivalence problem for regular expressions is not in P ; we also obtain a nontrivial linear lower bound on the space required for the equivalence problem.

The succinctness of regular expressions is increased by allowing the use of operations other than \cup , \cdot , and $*$ in writing expressions. For example, the additional set operations of intersection (\cap) and complementation (\sim) relative to Σ^* are sometimes helpful. Brzozowski [Brz64] has developed methods for handling regular expressions extended by \cap and \sim ; in particular, his methods yield an algorithm for checking equivalence of such extended regular expressions. However a priori analysis of his algorithm shows that for no fixed k is the running time

bounded above by $2^{2^{\cdot^{\cdot^{\cdot 2^n}}}} \}^k$ on all inputs of length n and all n . In section 4.2 we show that such complexity growth is inherent in the problem. The equivalence problem for star-free expressions [MP71] (which may use only the operations of \cup , \cdot , and \sim) can be solved by

no algorithm which runs within time and space $2^{2^{\cdot^{\cdot^{\cdot 2^n}}}} \}^{\log_b n}$ if $b > 3$.

It immediately follows that the equivalence problem for star-free expressions is not elementary-recursive in the sense of Kalmar [cf. Pet67]. Ritchie [Rit63] has shown that (the characteristic function of) a set is elementary recursive iff the set can be accepted within space

$$2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}} \text{ for some fixed } k.$$

Apart from providing a nonelementary lower bound on a simple explicit word problem, this result yields several interesting corollaries about the complexity of decidable theories of formal logic. Chapter 5 is devoted to these corollaries, each of which follows by an efficient transformation from the equivalence problem for star-free expressions to the decision problem for a particular logical theory. Thus these theories are not elementary-recursive.

In section 4.1, lower bounds of exponential space and exponential time are obtained for the equivalence problem if the unary operation "squaring" (defined by $L^2 = L \cdot L$) may occur in expressions, even if \cap and \sim may not occur.

Regular-like expressions are regular expressions generalized by allowing sets of operations other than the usual $\{ \cup, \cdot, * \}$. A particular class of regular-like expressions is specified by a finite set Σ of alphabet symbols and a finite set \wp of operations which may occur in expressions.

Definition 4.1. Let Σ be a finite alphabet and \wp be a finite set of symbols denoting operations on sets of words. Assume \wp contains only

unary and binary operations. Assume Σ , φ , and $\{\lambda, \wr, \wr\}$ are pairwise disjoint sets of symbols.

We inductively define the class of Σ - φ -expressions and simultaneously define the map L which maps the class of Σ - φ -expressions to subsets of C^* . If E is an expression, $L(E)$ is the language (set of words) described by E .

- (1). (i). $\wr(\lambda)$ is a Σ - φ -expression, and $L(\wr(\lambda)) = (X)^\dagger$.
 (ii). If $\sigma \in \Sigma$, $\wr(\sigma)$ is a Σ - φ -expression and $L(\wr(\sigma)) = \{\sigma\}$.
- (2). If E_1 and E_2 are Σ - φ -expressions, then:
 - (i). If $\wr \in \varphi$ denotes the binary operation \wr ,
 $\wr(E_1 \wr E_2)$ is a Σ - φ -expression and
 $L(\wr(E_1 \wr E_2)) = L(E_1) \wr L(E_2)$.
 - (ii). If $\wr \in \varphi$ denotes the prefix (postfix) unary operation \wr ,
 $\wr(\wr E_1)$ (resp., $\wr(E_1 \wr)$) is a Σ - φ -expression and
 $L(\wr(\wr E_1)) = \wr L(E_1)$ (resp., $L(\wr(E_1 \wr)) = L(E_1) \wr$).
- (3). That's all.

If E is a Σ - φ -expression, $|E|$ denotes the length of E viewed as a word in $(\Sigma \cup \varphi \cup \{\wr, \wr, \wr\})^*$.

In particular, we consider cases where $\varphi \subset \{\cup, \cap, \cdot, *, ^2, \sim\}$.

Binary operations \cup (union) and \cap (intersection) are familiar.

[†]Note: \wr is a formal symbol; X denotes the empty word itself. We allow \wr as an expression merely as a technical convenience. \wr can be removed from our proofs at the cost of minor awkwardness. See Remark 4.23.

Concatenation is extended to sets of words in the obvious way;

$$\underline{R_1 \cdot R_2} = \{ wy \mid w \in R_1 \text{ and } y \in R_2 \} \quad \text{for } R_1, R_2 \subseteq \Sigma^*.$$

If $R \subseteq \Sigma^*$, define $\underline{R}^0 = \{\lambda\}$ and $\underline{R}^{k+1} = R \cdot R^k$ for all $k \in \mathbb{N}$.

In particular, the unary "squaring" operation is $\underline{R}^2 = R \cdot R$.

Unary operation \star (Kleene star) is now defined as

$$\underline{R}^* = \bigcup_{k=0}^{\infty} R^k.$$

Unary operation \sim denotes set complementation relative to Σ^* ;

$$\underline{\sim R} = \Sigma^* - R.$$

The set Σ will always be clear from context.

To improve readability, several abbreviations are used in the text in describing expressions. These are as follows.

Having made clear the distinction between the formal symbol \sim and the metanotation \sim for an operation, " \sim " is usually ~~deleted~~. Similarly, we write $($ for \langle , etc.

Parentheses are used sparingly; the full parenthesization required by Definition 4.1 is not used. Any ambiguity can be resolved by two precedence rules: any unary operation takes precedence over any binary operation; concatenation takes precedence over both union and intersection.

If some character, say Σ , is defined within the text to denote a finite set of symbols, say $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_s\}$, then Σ may be used to abbreviate the Σ - $\{U\}$ -expression $(\sigma_1 U \sigma_2 U \dots U \sigma_s)$ which describes $\{\sigma_1, \sigma_2, \dots, \sigma_s\}$. Similarly (for example) $\Sigma - \{\sigma_1\}$ may be used to abbreviate the regular like expression $(\sigma_2 U \dots U \sigma_s)$.

Occasionally we let a **word** $\omega \in \Sigma^*$ abbreviate a $\Sigma\{-\cdot\}$ -expression which describes $\{\omega\}$.

.Iterated operations such as $\bigcup_{i=1}^k E_i$ are used to abbreviate $E_1 \cup E_2 \cup \dots \cup E_k$.

Two particular classes of expressions are used often enough to deserve special notation.

Recall $\Sigma^k = \{ \omega \in \Sigma^* \mid |\omega| = k \}$ for $k \in \mathbb{N}$.

Let $\Sigma^{\leq k} = \{ \omega \in \Sigma^* \mid |\omega| \leq k \}$ for $k \in \mathbb{N}$.

Within the context of regular-like expressions, $[\Sigma^k]$ ($[\Sigma^{\leq k}]$) is an abbreviation for the obvious $\Sigma\{-\cup, \cdot\}$ -expression of size bounded by $7k(\text{card}(\Sigma))$, namely,

$[\Sigma^k]$ abbreviates $\Sigma \cdot \Sigma \cdot \Sigma \cdot \dots \cdot \Sigma$ (k times)

$[\Sigma^{\leq k}]$ abbreviates $(\Sigma \cup \lambda) \cdot (\Sigma \cup \lambda) \cdot \dots \cdot (\Sigma \cup \lambda)$ (k times).

In most proofs involving regular-like expressions, a major concern is the lengths of expressions we write. **All abbreviations** must be taken into account when bounding the lengths of expressions.

Example 4.2. This example investigates two ways of writing a regular-like expression which describes $\{0,1\}^* - \{ (01)^k \mid k \geq 0 \}$.

(1). If \cdot , $*$, and \sim are available, a simple such expression is

$$E_0 = \sim((0 \cdot 1)^*)$$

(2). Such an expression can also be written using \cup , \cdot , and $*$.

Expression F_0 very simply illustrates a technique to be used throughout Chapter 4.

$$F_0 = 1 \cdot (0 \cup 1)^* \cup (0 \cup 1)^* \cdot 0 \cup (0 \cup 1)^* \cdot (0 \cdot 0 \cup 1 \cdot 1) \cdot (0 \cup 1)^*$$

F_0 describes the correct language because a word $\omega \in \{0,1\}^*$ is not in $(01)^k \mid k \geq 0$ iff ω "begins wrong" (i.e. begins with 1), or "ends wrong" (i.e. ends with 0), or "moves wrong" (i.e. contains 00 or 11 as a subword).

Given a predicate P on regular-like expressions, an alphabet Σ , and a set of operations φ , we may be interested to characterize the complexity of deciding P restricted to $\Sigma\text{-}\varphi\text{-expressions}$. The problem of "deciding" P is equivalent to the problem of accepting the set $P(\Sigma, \varphi)$ defined next.

Definition'4.3. Let P be an n -place predicate on regular-like expressions.

Define $P(\Sigma, \varphi) = \{ (E_1, E_2, \dots, E_n) \mid E_1, E_2, \dots, E_n \text{ are } \Sigma\text{-}\varphi\text{-expressions} \text{ and } P(E_1, E_2, \dots, E_n) \text{ is true} \}.$

For simplicity, we concentrate attention on the problem of checking inequivalence of expressions. Define the binary predicate INEQ by

$$\text{INEQ}(E_1, E_2) \text{ iff } L(E_1) \neq L(E_2).$$

In many cases we consider the special inequivalence predicate NEC (nonempty complement) defined by

$$\text{NEC}(E_1) \text{ iff } L(E_1) \neq \Sigma^*$$


where Σ is the smallest alphabet such that E_1 is a $\Sigma\text{-}\varphi\text{-expression}$ for some φ .

For example, if E_0 and F_0 are as in Example 4.2, then $(E_0, F_0) \notin \text{INEQ}(\{0,1\}, \{\sim, \cdot, *, \cup\})$ and $(F_0) \in \text{NEC}(\{0,1\}, \{\cup, \cdot, *\})$.

It is obvious that NEC is a special case of INEQ in the sense that,

*
if Σ is the language of some Σ - φ -expression, then an algorithm which accepts $\text{INEQ}(\Sigma, \varphi)$ immediately yields an algorithm which accepts $\text{NEC}(\Sigma, \varphi)$; formally $\text{NEC}(\Sigma, \varphi) \leq_{\log\text{-lin}} \text{INEQ}(\Sigma, \varphi)$. A lower bound on the complexity of NEC yields essentially the same lower bound on that of INEQ.

Hunt [Hun73a], [Hun73c] has extended our work to many other interesting predicates on expressions. He gives various criteria to determine if the generalization applies to a given predicate. For example, the unary predicates " $L(E_1)$ is cofinite", " $L(E_1) = R_0$ " where R_0 is any particular unbounded regular set, and " $L(E_1)$ is a non-counting event [MP71]" satisfy one criterion. For these predicates and others which satisfy the criterion, $P(\Sigma, \varphi)$ is as computationally difficult as $\text{NEC}(\Sigma, \varphi)$.

 The reader is referred to [Hun73a], [Hun73c], and [HR74] for further details.

Remark. We consider inequivalence (rather than equivalence) problems because such problems are more amenable to solution by nondeterministic algorithms; to determine that $L(E_1) \neq L(E_2)$, a nondeterministic algorithm can "guess" a word in the symmetric difference

$(L(E_1) - L(E_2)) \cup (L(E_2) - L(E_1))$. (See for example Proposition 4.11.)

It is then possible ^{some} in ~~most~~ cases to show that a particular inequivalence problem is complete in some nondeterministic complexity class, whereas it may not be immediate (or even true) that the corresponding equivalence problem is complete in the class because certain nondeterministic complexity classes such as NP and CSL are not known to be

closed under complementation.

Because deterministic time(space) classes are closed under **complement** for countable(constructable) bounds, it is clear that a lower bound on the deterministic complexity of a particular **inequivalence** problem immediately gives a lower bound on the deterministic complexity of the corresponding **equivalence** problem provided the time(space) bounds are countable(constructable). (The conditions of countability or constructability are required only because our definition of complexity bounded set acceptance (Definition 2.2) places no bounds on the resources used by the algorithm when computing on rejected words. Of course the countable or constructable conditions can be dropped if we adopt the **common** definition of acceptance in which the algorithm must halt within the resource bound on all inputs.) See also Remarks 4.20 and 4.21 for more discussion on the deterministic or nondeterministic complexity of equivalence problems.

In the following sections we characterize the complexity of accepting $NEC(\Sigma, \varphi)$ or $INEQ(\Sigma, \varphi)$ for various **choices** of Σ and φ . Sections 4.1 and 4.2 consider $\Sigma = \{0, 1\}$, which actually subsumes all choices of finite Σ with $\text{card}(\Sigma) \geq 2$. Section 4.4 contains two results for the case $\Sigma = \{0\}$ which show that restriction to a one letter alphabet can affect the complexity of the **inequivalence** problem. Section 4.1 considers several choices of φ from $\{U, \cdot, *, ^2\}$, and in particular considers regular expressions as usually defined ($\varphi = \{U, \cdot, *\}$). In **section** 4.2 we show that the inequivalence problem

with $\varphi = \{U, \cdot, \sim\}$ requires time and space exceeding $2^{2^{.2}} \}^{\lceil \log_b n \rceil}$ if $b > 3$. We also investigate how the depth of nesting of \sim operations affects the complexity, and find that each increase by one in ---depth causes an exponential jump in complexity.

As was described in section 3.2, we can obtain a lower bound on the complexity of a particular set B by showing $\mathcal{S} \leq_{\text{eff}} B$ where \leq_{eff} is an efficient transformation and \mathcal{S} is a suitably rich class of sets. There is one basic method used in section 4.1 to show $\mathcal{S} \leq_{\text{eff}} \text{NEC}(\Sigma, \varphi)$ or $\mathcal{S} \leq_{\text{eff}} \text{INEQ}(\Sigma, \varphi)$ for various particular \mathcal{S} , Σ , φ , and \leq_{eff} . In section 4.2 the technical details become more complicated but the basic method remains the same. The method utilizes the following formal notion of the **computations** of a STM.

Definition 4.4. Let $M = (I, \Gamma, Q, \delta, q_0, q_a)$ be a (nondeterministic) STM. Let d be an **i.d.** of M

$$\begin{aligned} \underline{\text{PartComp}}_M(d) &= \{ \omega \mid \omega = \$d_1\$d_2\$d_3\$ \cdots \$d_\ell\$ \text{ where } d_1 = d \text{ and} \\ &\quad d_{i+1} \in \text{Next}_M(d_i) \text{ (and hence } |d_{i+1}| = |d_i| \text{)} \\ &\quad \text{for all } i = 1, 2, 3, \cdots, \ell-1 \text{)} . \\ \underline{\text{Comp}}_M(d) &= \text{PartComp}_M(d) \cap \{ w \mid w = \alpha q_a \beta \text{ for some} \\ &\quad \alpha, \beta \in (Q \cup \Gamma \cup \{\$ \})^* \} . \end{aligned}$$

Recall the convention that state q_a is entered iff M is computing

on an input x which is to be accepted. The next fact is then obvious.

Fact 4.5. Let $M = (I, \Gamma, Q, \delta, q_0, q_a)$ be an STM which accepts a set A within space $S(n)$ ($S: \mathbb{N}^+ \rightarrow \mathbb{N}^+$). Then for all $x \in I^+$,

$$x \in A \text{ iff } \text{Comp}_M(q_0 x \text{ } \text{\textcircled{\tiny $}}^S(|x|) - |x|) \neq \emptyset$$

where $\text{\textcircled{\tiny $}}$ denotes the blank tape symbol.

The next lemma provides a useful equivalent **characterization** of $\text{Comp}_M(d)$.

Lemma 4.6. Let $M = (I, \Gamma, Q, \delta, q_0, q_a)$ be an STM. Let d be an i.d. of M ; assume q_a does not appear as a symbol in d . Let $k = |d|$ and $\Sigma = Q \cup \Gamma \cup \{\text{\textcircled{\tiny $}}\}$.

*

Then for all $\omega \in \Sigma^*$, $\omega \in \text{Comp}_M(d)$ iff

- (i). ("starts correctly") $\text{\textcircled{\tiny $}}d\text{\textcircled{\tiny $}}$ is a prefix of ω ;
 and
 (ii). ("moves correctly") If we write $w = \sigma_1 \sigma_2 \sigma_3 \cdots \sigma_m$ where $\sigma_j \in \Sigma$ for $1 \leq j \leq m$, then for all j with $2 \leq j \leq m-k-2$
- $$\sigma_{j+k} \sigma_{j+k+1} \sigma_{j+k+2} \in N_M(\sigma_{j-1} \sigma_j \sigma_{j+1})$$

where N_M is the function of Lemma 2.14;

and (iii). ("ends correctly")

- (iiia). $\text{\textcircled{\tiny $}}$ is the last symbol of ω
 and
 (iiib). q_a appears as a symbol in ω .

Proof. The "only if" direction of the proof is straightforward. We sketch the proof of the "if" direction.

Using Lemma 2.14 the following statement can be proved by induction

on the number of $\$$ symbols which appear in ω :

. For all $\omega \in \Sigma^*$, if $|\omega| \geq k+4$ and ω satisfies conditions (i), (ii), and (iiia), then $\omega \in \text{PartComp}_M(d)$.

Now assume ω satisfies all four conditions. If $|\omega| \leq k+2$ then q_a cannot appear in ω . If $|\omega| = k+3$ then ω cannot both end with $\$$ and contain q_a . Therefore $|\omega| \geq k+4$. Now $\omega \in \text{PartComp}_M(d)$ by the above, and q_a appears in ω by (iiib). Therefore $\omega \in \text{Comp}_M(d)$ by the definition of $\text{Comp}_M(d)$. \square

The proof of Lemma 4.8 soon to follow illustrates the general method and is a prototype for most results of Chapter 4 which show

$\mathcal{C} \leq_{\text{eff}} P(\Sigma, \varphi)$ for some \mathcal{C} , C , φ , \leq_{eff} and $P \in \{\text{NEC}, \text{INEQ}\}$.

4.1 Expressions With Squaring.

In **this** section we **show** that $\text{NEC}(\{0,1\}, \{U, \cdot, *, ^2\})$ is $\leq_{\log\text{-lin}}$ **complete** in EXPSpace (Theorem 4.12) and that $\text{INEQ}(\{0,1\}, \{U, \cdot, ^2\})$ is $\leq_{\log\text{-lin}}$ -complete in EXPTime (Theorem 4.18). It is then easy to deduce lower bounds of exponential space and exponential time respectively for these problems using the methods outlined in section 3.2. Also, $\text{NEC}(\{0,1\}, \{U, \cdot, *\})$ is $\leq_{\log\text{-lin}}$ -complete in CSL (Theorem 4.13) and $\text{INEQ}(\{0,1\}, \{U, \cdot\})$ is \leq_{\log} -complete in NP (Theorem 4.19).

First, the following fact is useful in the proofs of Lemmas 4.8 and 4.15: for any $k \in \mathbb{N}^+$, using squaring and concatenation we can write an expression $[\Sigma^k]_{\text{sq}}$ of length $O(\log k)$ which describes Σ^k ; moreover $[\Sigma^k]_{\text{sq}}$ is computable from $\text{bin}(k)$ by a function in logspace.

$\text{bin}(k)$ is defined as the binary representation of $k \in \mathbb{N}$ without leading zeroes unless $k = 0$.

Lemma 4.7. Let Σ be a finite alphabet. There is a constant $\alpha = \alpha(\Sigma)$ such that for any $k \in \mathbb{N}^+$ there is a $C(\{U, \cdot, ^2\})$ -expression $[\Sigma^k]_{\text{sq}}$ such that

- (1). $L([\Sigma^k]_{\text{sq}}) = \Sigma^k$
and
(2). $|[\Sigma^k]_{\text{sq}}| \leq \alpha \log k.$

Moreover, there is a function $f_{\Sigma} \in \text{logspace}$ with domain $\{0,1\}^+$ such that $f_{\Sigma}(\text{bin}(k)) = [\Sigma^k]_{\text{sq}}$ for all $k \in \mathbb{N}^+$.

Proof. Define $[\Sigma^k]_{\text{sq}}$ inductively as follows:

$$[\Sigma^1]_{\text{sq}} = \Sigma ; \quad [\Sigma^{2k}]_{\text{sq}} = ([\Sigma^k]_{\text{sq}})^2 ; \quad \text{and} \quad [\Sigma^{2k+1}]_{\text{sq}} = ([\Sigma^k]_{\text{sq}})^2 \cdot \Sigma$$

for all $k \in \mathbb{N}^+$,

It is obvious that (1) and (2) hold for some constant $\alpha(\Sigma)$.

The structural similarity between $\text{bin}(k)$ and $[\Sigma^k]_{\text{sq}}$ should be obvious from the inductive definition above. We let the reader convince himself that a suitable $f_\Sigma \in \text{logspace}$ exists. (Alternatively, f_Σ can be defined by 2 sided recursion of concatenation (cf. Appendix II) from functions which are trivially members of logspace.) \square

Notation. Note that $L([\Sigma \cup \lambda]^k_{\text{sq}}) = \Sigma^{\leq k}$.

We use the notation $\underline{[\Sigma^{\leq k}]_s}$ for $[\Sigma \cup \lambda]^k_{\text{sq}}$.

Lemma 4.8 is the first result which shows that a regular-like expression can "simulate" a complexity bounded Turing machine. The proof of this lemma was first given in [MS72].

Lemma 4.8. Let $A \in \text{EXPSPACE}$. There is a finite alphabet Σ such that $A \leq_{\log\text{-lin}} \text{NEC}(\Sigma, \{U, \cdot, *, ^2\})$.

Proof. Let $A \in \text{EXPSPACE}$. By Lemma 2.13 we can choose $d \in \mathbb{N}$ such that some (nondeterministic) STM $M = (I, \Gamma, Q, \delta, q_0, q_a)$ accepts A within space $S(n) = 2^{dn}$. Let $x \in I^+$ be arbitrary, let $n = |x|$, and $\Sigma = \Gamma \cup Q \cup \{\$ \}$ (where $\$ \notin \Gamma \cup Q$).

We construct a Σ - $\{U, \cdot, *, ^2\}$ -expression $E_M(x)$ such that

$$L(E_M(x)) = \Sigma^* - \text{Comp}_M(q_0 x \$^{2^{dn}-n}).$$

Therefore, $(E_M(x)) \in \text{NEC}(\Sigma, \{U, \cdot, *, ^2\})$ iff $L(E_M(x)) \neq \Sigma^*$
iff $\text{Comp}_M(q_0 x \$^{2^{dn}-n}) \neq \emptyset$ iff $x \in A$.

Letting f_M be the function mapping x to $E_M(x)$ for all $x \in I^+$,

we shall see that $f_M \in \text{logspace}$ and f_M is linear bounded. Thus

$$A \leq_{\text{log-lin NEC}} (\Sigma, \{U, \cdot, *, 2\}) \text{ via } f_M.$$

By Lemma 4.6, words in $L(E_M(x))$ can be characterized as follows:

$$\omega \in \Sigma^* - \text{Comp}_M(q_0 x \tau 2^{dn-n}) \text{ iff}$$

- or
- (1). ("starts wrong") ω does not begin with $\$q_0 x \tau 2^{dn-n}\$$;
- or
- (2). ("moves wrong") ω is of the form $\alpha \sigma_1 \sigma_2 \sigma_3 \beta \sigma'_1 \sigma'_2 \sigma'_3 \gamma$ where $\alpha, \gamma \in \Sigma^*$, $\beta \in \Sigma^{2^{dn-1}}$, and $\sigma'_1 \sigma'_2 \sigma'_3 \notin N_M(\sigma_1 \sigma_2 \sigma_3)$;
- or
- (3). ("ends wrong") ω does not contain q_a or does not end with $\$$.

We now write expressions E_1, E_2, E_3 which formally describe the sets of words (1), (2), (3) above.

If $a \in \Sigma$, let $\bar{\sigma}$ denote $\Sigma - \{\sigma\}$.

(1) Words may satisfy (1) for three reasons.

First E_{11} describes all words which are "too short", that is

$$E_{11} = [\Sigma^{\leq 2^{dn}+2}]_{sq}.$$

Recall from Lemma 4.7 that $|[\Sigma^m]_{sq}| \leq |[\Sigma^{\leq m}]_{sq}| \leq \alpha \log m$ for all m , where α depends only on Σ . Therefore, viewing $|E_{11}|$ as a function of n , $|E_{11}| = O(n)$.

Second, E_{12} describes all words which do not begin with $\$q_0 x$.

Let $x = x_1 x_2 x_3 \cdots x_n$.

$$E_{12} = (\$ \cup \$ \cdot (\bar{q}_0 \cup q_0 \cdot (\bar{x}_1 \cup x_1 \cdot (\bar{x}_2 \cup \cdots \cup x_{n-2} \cdot (\bar{x}_{n-1} \cup x_{n-1} \cdot \bar{x}_n))) \cdots) \cdot \Sigma).$$

Viewing $|E_{12}|$ as a function of n , note that $|E_{12}| = O(n)$.

Finally, E_{13} describes all words longer than $n+2$ which do not begin with $\tau \tau 2^{dn-n}\$$ for some τ of length $n+2$.

$$E_{13} = [\Sigma^{n+2}] \cdot [\Sigma^{\leq 2^{dn}-n-1}]_{sq} \cdot \bar{\psi} \cdot \Sigma^* \cup [\Sigma^{2^{dn}+2}]_{sq} \cdot \bar{\psi} \cdot \Sigma^* .$$

Note $|E_{13}| = O(n)$ for the same reasons given for $|E_{11}|$.

$$\text{Now let } E_1 = E_{11} \cup E_{12} \cup E_{13} .$$

(2'). Words (2) are described by an expression E_2 . Subexpression $[\Sigma^{2^{dn}-1}]_{sq}$ serves as a "ruler" to measure the distance between pairs of words $\sigma_1 \sigma_2 \sigma_3$ and $\sigma'_1 \sigma'_2 \sigma'_3$ which are inconsistent in the sense that $\notin N_M(\sigma_1 \sigma_2 \sigma_3)$. See also Figure 4.1 where $k = 2^{dn}+1$ is the length of each i.d.

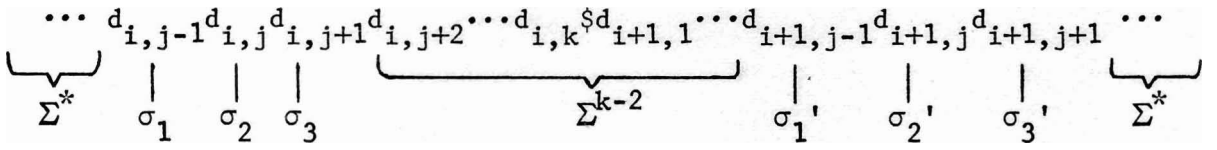


Figure 4.1: E_2 "matches" a word ω .

$$E_2 = \Sigma^* \cdot \left(\bigcup_{\sigma_1 \sigma_2 \sigma_3 \in \Sigma^3} \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdot [\Sigma^{2^{dn}-1}]_{sq} \cdot (\Sigma^3 - N_M(\sigma_1 \sigma_2 \sigma_3)) \right) \cdot \Sigma^* .$$

Note $|E_2| = O(n)$.

$$(3'). \quad E_3 = (\Sigma - \{q_a\})^* \cup \Sigma^* \cdot \bar{\psi} .$$

$|E_3|$ is fixed independent of n .

$$\text{Now let } E_M(x) = E_1 \cup E_2 \cup E_3 .$$

For $\omega \in \Sigma^*$ we have

$$\begin{aligned} \omega \notin L(E_M(x)) & \text{ iff } \omega \notin L(E_1) \text{ and } \omega \notin L(E_2) \text{ and } \omega \notin L(E_3) \\ & \text{ iff } \omega \in \text{Comp}_M(q_0 x \psi^{2^{dn}-n}) \text{ by Lemma 4.6.} \end{aligned}$$

Therefore $L(E_M(x)) = \Sigma^* - \text{Comp}_M(q_0 x^y 2^{dn-n})$ as required.

Let f_M be the function with domain I^+ defined by

$$f_M(x) = (E_M(x)) \text{ for all } x \in I^+.$$

To complete the proof, we must show $f_M \in \text{logspace}$ and f_M is linear bounded. The latter fact is immediate from our observation that

$|E_1|$, $|E_2|$, and $|E_3|$ (viewed as functions of n) are all $O(n)$.

Assuming $f_M \in \text{logspace}$, $A \leq_{\text{log-lin NEC}} (\Sigma, \{U, \cdot, *, 2\})$ via f_M .

We now outline how one might formally prove that $f_M \in \text{logspace}$, using a number of facts from Appendix II. Those readers familiar with space bounded Turing machines may wish to skip the next paragraph.

First by Lemma 4.7 there is a function $f_\Sigma \in \text{logspace}$ mapping $\text{bin}(m)$ to $[\Sigma^m]_{\text{sq}}$ for all m . The functions mapping x to $\text{bin}(|x|)$ and x to $\text{bin}(2^{d|x|})$ belong to logspace . Now the functions $a_1(x) = \text{bin}(2^{d|x|} - |x| - 1)$, $a_2(x) = \text{bin}(2^{d|x|} + 2)$, $a_3(x) = \text{bin}(2^{d|x|} - 1)$ belong to logspace because addition and minus belong to logspace and logspace is closed under composition. Therefore, by another application of closure under composition, the functions mapping x to $[\Sigma^{2^{d|x|} - |x| - 1}]_{\text{sq}}$, etc. belong to logspace . Finally E_{12} is definable from x by two sided recursion of concatenation. Thus, all the components of $E_M(x)$ can be computed by functions in logspace . These components can be combined appropriately by concatenation ($\in \text{logspace}$) to give $E_M(x)$. \square

Lemma 4.9.

Let $A \in \text{CSL}$. There is a finite

alphabet Σ such that $A \leq_{\log\text{-lin}} \text{NEC}(\Sigma, \{U, \cdot, *\})$.

Proof. The proof is essentially the same as Lemma 4.8; only the differences are sketched.

Let $A \in \text{CSL}$ and let $M = (I, \Gamma, Q, \delta, q_0, q_a)$ be a (nondeterministic) STM which accepts A within space $S(n) = n+1$. Let $x \in I^+$, $n = |x|$, and $\Sigma = Q \cup \Gamma \cup \{\$$ as before.

Since $x \in A$ iff $\text{Comp}_M(q_0 x \$) \neq \emptyset$, $E_M(x)$ is constructed such that

$$L(E_M(x)) = \Sigma^* - \text{Comp}_M(q_0 x \$).$$

$E_M(x)$ is constructed as in the proof of lemma 4.8 where 2^{dn} is replaced by $n+1$, and $[\]_{sq}$ is replaced by $[\]$ (without the use of "squaring"). For example, subexpression E_2 is now

$$E_2 = \Sigma^* \cdot \left(\bigcup_{\sigma_1 \sigma_2 \sigma_3 \in \Sigma^3} \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdot [\Sigma^n] \cdot (\Sigma^3 - N_M(\sigma_1 \sigma_2 \sigma_3)) \right) \cdot \Sigma^*.$$

Recall that $[\Sigma^n]$ abbreviates $\Sigma \cdot \Sigma \cdot \Sigma \cdot \dots \cdot \Sigma$ (n times). Therefore $|E_2| = O(n)$. Similarly one can check that the lengths of E_1 and E_3 (after modification) are also $O(n)$.

Let f_M' be the function mapping x to $E_M(x)$ for all x . Then one can prove $f_M' \in \text{logspace}$ just as one proves $f_M \in \text{logspace}$ in Lemma 4.8.

$$A \leq_{\log\text{-lin}} \text{NEC}(\Sigma, \{U, \cdot, *\}) \text{ via } f_M'. \quad \square$$

In Lemmas 4.8 and 4.9, the alphabet Σ depends on the set A . However we would like to show that (for example) $\text{NEC}(\Sigma, \{U, \cdot, *\})$ is complete in CSL for a fixed alphabet Σ . The next lemma shows that alphabet symbols can be coded into binary. Therefore Lemmas 4.8 and

4.9 are true with $\Sigma = \{0,1\}$, **always**. For convenience, many results to follow are stated only for the case $\Sigma = \{0,1\}$; these results are actually true for any finite Σ with $\text{card}(\Sigma) \geq 2$.

Lemma 4.10. Let Σ be a finite alphabet with $\text{card}(\Sigma) \geq 2$, and let $\varphi \subseteq \{U, \cdot, *, \cap\}$.

$$(1). \text{ INEQ}(\Sigma, \varphi) \equiv_{\log\text{-lin}} \text{INEQ}(\{0,1\}, \varphi).$$

(2). If also $U, \cdot, * \in \varphi$, then

$$\text{NEC}(\Sigma, \varphi) \equiv_{\log\text{-lin}} \text{NEC}(\{0,1\}, \varphi).$$

Proof. (1). The transformation $\text{INEQ}(\{0,1\}, \varphi) \leq_{\log\text{-lin}} \text{INEQ}(\Sigma, \varphi)$ is trivial. We only show $\text{INEQ}(\Sigma, \varphi) \leq_{\log\text{-lin}} \text{INEQ}(\{0,1\}, \varphi)$.

Let $k = \lceil \log_2(\text{card}(\Sigma)) \rceil$. Let h be any one-to-one map, $h : \Sigma^* \rightarrow \{0,1\}^k$. Extend h as a map from Σ^* to $2^{\{0,1\}^k}$ in the obvious way: $h(\lambda) = h$; $h(\omega\sigma) = h(\omega)h(\sigma)$ for all $\omega \in \Sigma^*$, $\sigma \in \Sigma$; and $h(R) = \{ h(\omega) \mid \omega \in R \}$ for $R \subseteq \Sigma^*$.

If E is a Σ - φ -expression, let $h(E)$ be the $\{0,1\}$ - φ -expression obtained from E by replacing each occurrence of a symbol $\sigma \in \Sigma$ in E by the word $h(\sigma)$. A simple inductive proof shows that $L(h(E)) = h(L(E))$ for all Σ - φ -expressions E .

$$(E_1, E_2) \in \text{INEQ}(\Sigma, \varphi) \text{ iff } (h(E_1), h(E_2)) \in \text{INEQ}(\{0,1\}, \varphi).$$

The function mapping (E_1, E_2) to $(h(E_1), h(E_2))$ is obviously linear bounded and a member of logspace. The conclusion follows.

(2). As in (1), we only show $\text{NEC}(\Sigma, \varphi) \leq_{\log\text{-lin}} \text{NEC}(\{0,1\}, \varphi)$. Given an expression E_1 , let Σ be the set of alphabet symbols which

actually occur in E_1 . Let $C = h(\Sigma) \cup \{x\} = \{h(\sigma) \mid \sigma \in \Sigma\} \cup \{x\}$ be the set of code words. As in part (1), for all Σ -expressions E ,
 $L(h(E)) \subseteq C^*$ and $L(h(E)) = C^*$ iff $L(E) = \Sigma^*$.

Let F be the $\{0,1\}$ - $\{U, \cdot, *\}$ -expression

$$F = ((0 \cup 1)^k)^* \cdot (\{0,1\}^{\leq k} \cdot C) \cdot ((0 \cup 1)^k)^*.$$

Note $L(F) = \{0,1\}^* \cdot C^*$. Therefore

$$L((h(E_1) \cup F)) = \{0,1\}^* \text{ iff } L(E_1) = \Sigma^*.$$

The reduction (2) is via the function mapping E_1 to $(h(E_1) \cup F)$. \square

The next result gives an upper bound on the space complexity of $INEQ(\Sigma, \{U, \cdot, *\})$. Essentially the same algorithm was discovered independently by Aho, Hopcroft, and Ullman [AHU74]. Since lower bounds are our main interest, we only outline the algorithm.

Proposition 4.11. Let Σ be a finite alphabet.

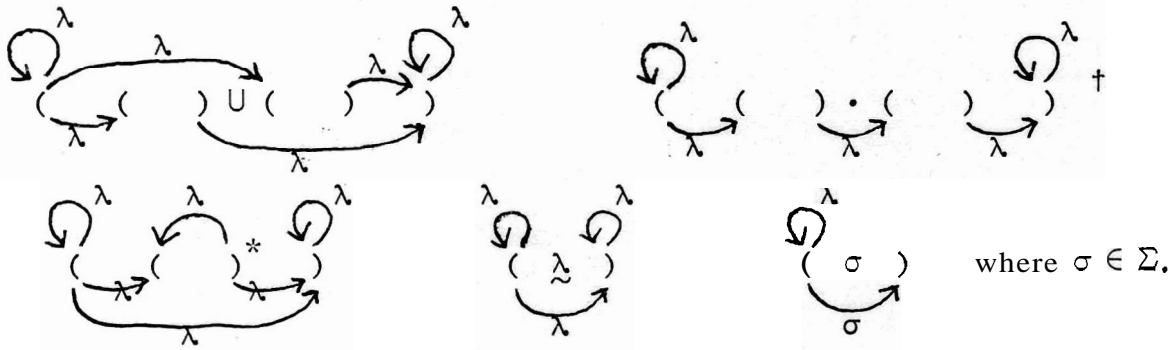
$$INEQ(\Sigma, \{U, \cdot, *\}) \in \text{CSL}.$$

Proof. Given two Σ - $\{U, \cdot, *\}$ -expressions E_1 and E_2 , an IOTM M tries to **nondeterministically "guess"** a word ω in $L(E_1) \oplus L(E_2)$

$= (L(E_1) \cdot L(E_2)) \cup (L(E_2) \cdot L(E_1))$. ω is guessed one symbol at a time. E_1 and E_2 will be viewed as **nondeterministic** finite automata (NFA's, cf. HU69) which accept $L(E_1)$ and $L(E_2)$ respectively. M can simulate these NFA's as though they were receiving ω as input, and thus determine if ω belongs to $L(E_1) \oplus L(E_2)$.

An expression, say E_1 , is viewed as an NFA as follows. The

parentheses of E_1 serve as the "states" of the NFA. If F is **any** subexpression of E_1 , the **leftmost** (rightmost) parenthesis of F is the initial (final) state of an NFA which accepts $L(F)$. Implicitly the following transitions exist between "states" of E_1 .



These transitions need not appear explicitly on a work tape because given two designated parentheses p_1 and p_2 (designated say by being marked **insome** way), an IOTM can check whether or not there is an arc from p_1 to p_2 by **counting** parentheses. Such a check can be performed within time polynomial in $|E_1|$ and space logarithmic in $|E_1|$.

The simulation of these NFA's, E_1 and E_2 , on ω works as **follows**. A parenthesis state in E_1 (or E_2) will be marked at some time iff the portion of ω received up to that time could lead the NFA E_1 (or E_2) to that state. The following procedure **update**(σ) is used to update the subset of marked states. **update**(σ) should perform as follows for $a \in \Sigma \cup \{\lambda\}$. At the completion of a call on **update**(σ), a state p_2 is marked iff there is a state p_1 (possibly $p_1 = p_2$) such that

† The h-self-loops are redundant, but are included for purposes of exposition.

(i) there is an arc labelled σ from p_1 to p_2 , and (ii) p_1 was marked before the call on $\text{update}(\sigma)$. Note $\text{update}(\sigma)$ can be programmed to run deterministically in polynomial time and linear space.

M operates as follows. Given input x (with $n = |x|$):

- (1). Note that for any Σ and φ , the set of Σ - φ -expressions is a context free language. Within space $(\log n)^2$ [cf. LSH65] check that x is of the form (E_1, E_2) where E_1 and E_2 are Σ - $\{U, \cdot, *\}$ -expressions. Reject if x is not of this form.
- (2). Write E_1 and E_2 on some work tape;
Mark the leftmost parenthesis of E_1 and E_2 .
- (3). Call $\text{update}(\lambda)$ n times.
- (4). If exactly one of the rightmost parentheses of E_1 and E_2 are marked, then accept.
- (5). Nondeterministically guess a symbol $\sigma \in \Sigma$;
Call $\text{update}(\sigma)$.
- (6). Go to (3).

\bar{M} operates within space cn for some constant c . The conclusion follows by Fact 2.8 (constant factor speedup). \square

Completeness results now follow easily for the two cases considered thus far.

Theorem 4.12. -

(1). $NEC(\{0,1\},(U,\cdot,*,^2))$ is $\leq_{\log\text{-lin}}$ -complete in EXPSPACE

(2). In particular:

(2i). There is a rational $c > 1$ such that

$$NEC(\{0,1\},(U,\cdot,*,^2)) \notin NSPACE(c^n) ;$$

(2ii). $NEC(\{0,1\},(U,\cdot,*,^2)) \in NSPACE(2^n)$.

Proof. First, for all $A \in \text{EXPSPACE}$,

$$A \leq_{\log\text{-lin}} NEC(\Sigma, (U, \cdot, *, ^2)) \leq_{\log\text{-lin}} NEC(\{0,1\}, (U, \cdot, *, ^2))$$

for some Σ by Lemmas 4.8 and 4.10. Therefore

$$\text{EXPSPACE} \leq_{\log\text{-lin}} NEC(\{0,1\}, (U, \cdot, *, ^2)) \text{ by transitivity of } \leq_{\log\text{-lin}}.$$

(2ii) is true because an IOTM, given a $\{0,1\}$ - $(U, \cdot, *, ^2)$ -expression E of length n , can first expand the squaring operations; that is, replace F^2 by $F \cdot F$ if F is some subexpression of E . This produces the $\{0,1\}$ - $(U, \cdot, *)$ -expression E' , where $|E'| \leq 2^n$ and $L(E') = L(E)$.

The IOTM now applies the procedure of Proposition 4.11 to the pair $(E', (0 \cup 1)^*)$. The entire procedure operates within space $O(2^n)$ and (2ii) then follows.

(1) is now immediate by the definition of $\leq_{\log\text{-lin}}$ -complete.

The proof of (2i) follows step (3) of Outline 3.8. That is, for $\epsilon > 0$ let $A \in NSPACE(2^n) - NSPACE((2-\epsilon)^n)$, and deduce (2i) where $c \leq (2-\epsilon)^{1/b}$ and $A \leq_{\log\text{-lin}} NEC(\{0,1\}, (U, \cdot, *, ^2))$ via some length bn bounded function. See Outline 3.8 for further details. \square

Theorem 4.13.

- (1). $NEC(\{0,1\}, \{U, \cdot, *\})$ is $\leq_{\log\text{-lin}}$ -complete in CSL.
- (2). If a **nondeterministic** IOTM accepts $NEC(\{0,1\}, \{U, \cdot, *\})$ within space $S(n)$, then there is a rational $c > 0$ such that $S(n) \geq cn$ for infinitely many integers n .

Proof. (1) is **immediate** from Lemmas 4.9 and 4.10 and Proposition 4.11.

(2). Let $B = NEC(\{0,1\}, \{U, \cdot, *\})$. Suppose a nondeterministic IOTM accepts B within space $S(n)$ where for all $c \in \mathbb{Q}^+$, $S(n) < cn$ for all but finitely many n .

Let $S'(n) = \max\{S(m) \mid m \leq n\}$. Then $B \in NSPACE(S'(n))$ and $S'(n)$ is nondecreasing.

By Fact 2.11 let the set A be such that $A \in CSL$; and for all $S_1(n)$, $S_1(n+1) = o(n)$ implies $A \notin NSPACE(S_1(n))$.

By part (1) above, $A \leq_{\log\text{-lin}} B$ via some length bn bounded function for some positive integer b . Therefore, by Lemma 3.6, $A \in NSPACE(S'(bn) + \log n)$. However, by definition of $S'(n)$ and our **assumption** on $S(n)$, $S'(b(n+1)) + \log(n+1) = o(n)$. This contradiction proves (2). □

Remark 4.14.

(1). As was mentioned earlier (following the definitions of **NEC** and **INEQ**), we can immediately replace **NEC** by **INEQ** in Theorems 4.12 and 4.13. [Hun73a], [Hun73c], and [HR74] give many other predicates which are as complex to decide as **NEC**.

(2). The proofs of **Lemmas** 4.8 (4.9) and 4.10 actually show that MPSPACE (resp., CSL) is log-lin reducible to the inequivalence problem for $\{0,1\}$ - $\{U, \cdot, *, ^2\}$ -expressions (resp., $\{0,1\}$ - $\{U, \cdot, \cdot^*\}$ -expressions) of star-height one [cf. **MP71**]. The expression $E_M(x)$ constructed in **Lemma** 4.8 (4.9) is of star-height one. Binary coding by **Lemma** 4.10 does not increase star-height above one. Therefore the lower bounds of **Theorems** 4.12 and 4.13 also hold for the respective **NEC** or **INEQ** problems restricted to expressions of star-height one.

(3). Using padding techniques of Ruby and Fischer [**RF65**], one can show that $CSL \leq_{\log} B$ implies $POLYSPACE \leq_{\log} B$ for any set B . (Hunt [**Hun73a**] has observed this fact using \leq_{\log}^* in place of \leq_{\log} .) Thus, immediate from **Theorem** 4.13, **NEC**($\{0,1\}, \{U, \cdot, \cdot^*\}$) is \leq_{\log} -complete in **POLYSPACE**.

We now investigate how removal of the $*$ operation affects the complexities of these problems. First consider **INEQ**($\Sigma, \{U, \cdot, ^2\}$). Note that this is a purely finite word problem: if E is a Σ - $\{U, \cdot, ^2\}$ -expression then $L(E)$ is a finite set of words. In fact, if $|E| = n$, then $\omega \in L(E)$ implies $|\omega| \leq 2^n$. This suggests that a Σ - $\{U, \cdot, ^2\}$ -expression cannot "simulate" a space 2^{dn} bounded STM as was done in **Lemma** 4.8 unless the expression itself is of length roughly 2^{dn} ; a STM which operates within space 2^{dn} may run for time $2^{2^{dn}}$ and thus may admit computations of length $2^{2^{dn}}$.

However, a Σ - $\{U, \cdot, ^2\}$ -expression can "simulate" a time 2^{dn} bounded STM. The computations (in the sense of $Comp_M(\)$) of a

time 2^{dn} bounded STM are of length roughly $(2^{dn})^2 = 2^{2dn}$.

The next result, presented by us previously in [SM73], was stimulated by a remark of **Brzozowski** that our use of $*$ in Lemmas 4.8 and 4.9 was very restricted and might therefore be removable.

Lemma 4.15.

$$\text{EXPNTIME} \leq_{\log\text{-lin}} \text{INEQ}(\{0,1\}, \{U, \cdot, ^2\}).$$

Proof. The proof is very similar to that of Lemma 4.8. We need only find a substitute for all occurrences of Σ^* in the expression $E_M(x)$ constructed to prove Lemma 4.8.

Let $A \in \text{EXPNTIME}$. Choose $d \in \mathbb{N}^+$ such that a (nondeterministic) STM $M = (I, \Gamma, Q, \delta, q_0, q_a)$ accepts A within time 2^{dn} (and thus M accepts A within space 2^{dn}). Let $x \in I^+$, $n = |x|$, and $\Sigma = \Gamma \cup Q \cup \{\$$ as before.

We construct a $\Sigma - \{U, \cdot, ^2\}$ -expression $E_M(x)$ such that

$$L(E_M(x)) = \Sigma^{\leq b(n)} - \text{Comp}_M(q_0 x^b 2^{dn-n})$$

where $b(n)$ is specified below.

$$\begin{aligned} \text{Note that } \omega \in \text{Comp}_M(q_0 x^b 2^{dn-n}) \text{ implies } |\omega| &\leq 2^{dn}(2^{dn}+1) + (2^{dn}+1) \\ &\leq 2^{2dn+2}, \end{aligned}$$

because each **i.d.** in ω is of length $2^{dn}+1$ and there are at most 2^{dn} such **i.d.**'s because M is time 2^{dn} bounded; the markers $\$$ account for at most $2^{dn}+1$ more symbols.

$$\text{Define } a(n) = 2^{2dn+2}.$$

The role of Σ^* in Lemma 4.8 is played by the expression $[\Sigma^{\leq a(n)}]_{sq}$.

Construct E_1 and E_2 exactly as in the proof of Lemma 4.8, except replace **all** occurrences of Σ^* by $[\Sigma^{\leq a(n)}]_{sq}$.

Let $E_3 = [(\Sigma - \{q_a\})^{\leq a(n)}]_{sq} \cup [\Sigma^{\leq a(n)}]_{sq} \cdot \bar{s}$.

Following the proof of Lemma 4.8, it can be checked that $L(E_1 \cup E_2 \cup E_3)$ contains all words in $\Sigma^{\leq a(n)}$ except those in $\text{Comp}_M(q_0 x b^{2^{dn}-n})$. $L(E_1 \cup E_2 \cup E_3)$ contains other words longer than $a(n)$; however no such word is longer than

$$b(n) = 2a(n) + 2^{dn} + 5.$$

(These longest words are in $L(E_2)$.)

Therefore, we add all words ω such that $a(n) < |\omega| \leq b(n)$.

$$E_4 = [\Sigma^{a(n)+1}]_{sq} \cdot [\Sigma^{\leq b(n)-a(n)-1}]_{sq}.$$

Now if $E_M(x) = E_1 \cup E_2 \cup E_3 \cup E_4$,

$$L(E_M(x)) = \Sigma^{\leq b(n)} - \text{Comp}_M(q_0 x b^{2^{dn}-n}),$$

and therefore $(E_M(x), [\Sigma^{\leq b(n)}]_{sq}) \in \text{INEQ}(\Sigma, \{U, \cdot, ^2\})$ iff $x \in A$.

Let f_M be the function mapping x to $(E_M(x), [\Sigma^{\leq b(n)}]_{sq})$ for all $x \in I^+$. Following the proof of Lemma 4.8, the reader can check that $f_M \in \mathbf{logspace}$ and f_M is linear bounded. Finally the binary coding lemma (4.10) implies the conclusion. \square

Again we see that removal of the operation 2 causes an exponential drop in complexity. The following lemma was discovered independently by Hunt [Hun73a] (with \preceq in place of \preceq_{\log}) using another proof.

Lemma 4.16.

$$\text{NP} \preceq_{\log} \text{INEQ}(\{0, 1\}, \{U, \cdot\}).$$

Proof. Lemma 4.16 is analogous to 4.15 in the same way that Lemma 4.9

is analogous to 4.8.

Given an STM M which accepts $A \in NP$ within polynomial time $p(n)$, and given input x with $n = |x|$, $E_M(x)$ is constructed as in Lemma 4.15 to describe $\Sigma^{\leq b(n)} = \text{Comp}_M(q_0 x^{p(n)-n})$ for some suitable polynomial $b(n)$. Σ^* is replaced by the expression $[\Sigma^{\leq (p(n)+1)^2}]$ in this case, and the "ruler" in E_2 is $[\Sigma^{p(n)-1}]$. Recall that $[\Sigma^m]$ is written as $\Sigma \cdot \Sigma \cdot \Sigma \cdots \Sigma$ (m times). By Fact AII.3 (Appendix 11), if $q(n)$ is a polynomial there are functions in **logspace** mapping x to $[\Sigma^{q(|x|)}]$ and to $[\Sigma^{\leq q(|x|)}]$. Further details are left to the reader. \square

An upper bound on the time complexity of $\text{INEQ}(\Sigma, \{U, \cdot\})$ follows by a minor modification to the procedure of Proposition 4.11.

Proposition 4.17. Let Σ be a **finite** alphabet.

$$\text{INEQ}(\Sigma, \{U, \cdot\}) \in NP.$$

Proof. The set of $\Sigma\text{-}\{U, \cdot\}$ -expressions is a context free language. Given input x of length n , an IOTM can check deterministically within time $O(n^3)$ [cf. You67] that x is of the form (E_1, E_2) where E_1 and E_2 are $\Sigma\text{-}\{U, \cdot\}$ -expressions.

Note that if E is a $\Sigma\text{-}\{U, \cdot\}$ -expression, $\omega \in L(E)$ implies $|\omega| \leq |E|$. Therefore

$$L(E_1) \neq L(E_2) \text{ iff } (\exists \omega) [\omega \in L(E_1) \oplus L(E_2) \text{ and } |\omega| \leq n].$$

The procedure of Proposition 4.11 (with step (1) modified as above) therefore accepts $\text{INEQ}(\Sigma, \{U, \cdot\})$ within nondeterministic polynomial time. \square

Theorem 4.18.

- (1). $\text{INEQ}(\{0,1\},\{U,\cdot,^2\})$ is $\leq_{\log\text{-lin}}$ -complete in EXPNTIME .
 (2). Therefore there are rational $c,d > 1$ such that
 (2i). $\text{INEQ}(\{0,1\},\{U,\cdot,^2\}) \not\in \text{NTIME}(c^n)$
 (2ii). $\text{INEQ}(\{0,1\},\{U,\cdot,^2\}) \in \text{NTIME}(d^n)$.

Proof. (2ii) follows by eliminating the squaring operations as in the proof of Theorem 4.12, and then applying the procedure of Proposition 4.11 and 4.17. (1) now follows by Lemma 4.15.

The proof of (2i) is exactly as in Theorem 4.12(2i) where NTIME replaces NSPACE . □

Theorem 4.19.

$\text{INEQ}(\{0,1\},\{U,\cdot\})$ is \leq_{\log} -complete in NP .

Proof. The proof is **immediate** from **Lemma** 4.16 and Proposition 4.17. □

This section concludes with several remarks on the material of section 4.1.

Remark 4.20. (Deterministic time complexities of these and related problems.)

Given present knowledge, Theorems 4.13 and 4.19 provide no interesting lower bounds on the deterministic time complexities of $\text{NEC}(\{0,1\},\{U,\cdot,*\})$ or $\text{INEQ}(\{0,1\},\{U,\cdot\})$. These results imply only exponential upper bounds. Theorem 4.13 implies $\text{NEC}(\{0,1\},\{U,\cdot,*\}) \in \text{DTIME}(d_1^n)$ for some constant d_1 by Fact 2.9C(b).

A deterministic simulation of the procedure of Proposition 4.17 yields $\text{INEQ}(\{0,1\},\{U,\cdot\}) \in \text{DTIME}(d_2^n)$ for some constant d_2 .

The exponential difference between the upper and lower bounds on deterministic time is closely related to two important open problems of complexity theory, namely " $P = NP?$ " and " $\text{CSL} \subseteq P?$ ".

Corollary 4.20.1.

- (1). $\text{NEC}(\{0,1\},\{U,\cdot,*\}) \in P$ iff $\text{CSL} \subseteq P$ iff $\text{CSL} \subsetneq P$.
- (2). $\text{INEQ}(\{0,1\},\{U,\cdot\}) \in \mathcal{B}$ iff $P = NP$.

Proof. The equivalence $\text{CSL} \subseteq P$ iff $\text{CSL} \subsetneq P$ follows by the result of Book [Bo72] that $\text{CSL} \neq \mathcal{B}$. (1) is now immediate from Lemma 3.9.

(2) is by Lemma 3.9 and $\mathcal{B} \subseteq NP$. \square

More generally, if a set B is $\leq_{\log\text{-lin}}$ -complete in CSL , e.g. $B = \text{NEC}(\{0,1\},\{U,\cdot,*\})$, then upper and lower bounds on the deterministic time complexity of B are related to bounds for CSL by:

$$\begin{aligned} B \in \text{DTIME}(T(n)) \quad \text{implies} \quad \text{CSL} &\subseteq \bigcup_{c,k \in \mathbb{N}} \text{DTIME}(T(cn) + n^k) \\ \text{and} \\ \text{CSL} \subseteq \text{DTIME}(T(n)) \quad \text{implies} \quad B &\in \text{DTIME}(T(n)), \end{aligned}$$

the first implication following from Lemma 3.6.

Corollary 4.20.1(1) provides evidence that the problem of checking equivalence of regular expressions(cf. [Gin67], [Brz64]) is computationally intractable. (By "regular expression" we mean a $\Sigma\text{-}\{U,\cdot,*\}$ -expression. Following [Edm65], [Kar72], we call a problem "intractable" if there is no deterministic algorithm which solves the problem within polynomial time.) If $\text{CSL} = P \neq \emptyset$, then the equivalence

problem for regular expressions is intractable, as are the problems of checking equivalence of nondeterministic finite state automata (NFA's, [cf. HU69]) and minimizing NFA's [cf. KW70]. Assuming $CSL \cdot P \neq \emptyset$, the equivalence problem for NFA's is intractable since there are well-known deterministic polynomial time procedures for converting any regular expression to an equivalent NFA (e.g. [Har65], [Sa169]).

To see that the minimization problem is intractable, suppose we have a deterministic polynomial time procedure \mathcal{Q} which, when given an NFA F , finds a smallest (in terms of number of states) NFA which accepts the same language as F . Let $A \in CSL$ and consider the following procedure for accepting A . Given input x , construct $E_M(x)$ as in Lemma 4.9 such that $L(E_M(x)) \neq \Sigma^*$ iff $x \in A$. Convert $E_M(x)$ to an equivalent NFA and minimize this NFA using \mathcal{Q} . Since it is trivial to check if a minimized NFA accepts Σ^* (it can have only one state), the entire procedure accepts A within deterministic polynomial time. If $CSL \cdot P \neq \emptyset$, then such an \mathcal{Q} cannot exist.

There are also "gaps" in the known deterministic time complexities of the problems with squaring. For example, Theorem 4.12 immediately gives a lower bound of $DTIME(c^n)$ for $NEC(\{0,1\},\{U,\cdot,*,^2\})$, but an upper bound of $DTIME(d^{d^n})$ for some $c,d > 1$. As in the cases above, any improvement in this gap would supply new information about Open Question 2.10C for the case $S(n) \in (c^n \mid c > 1)$, and vice versa.

Also, the deterministic space complexity of $NEC(\{0,1\},\{U,\cdot,*,^2\})$ is

related to the "lba problem".

Corollary 4.20.2. $NEC(\{0,1\}, \{U, \cdot, \}^*) \in DSPACE(n)$ iff $CSL = DSPACE(n)$.

Proof. Immediate from Theorem 4.13 and Lemma 3.6. □

Remark 4.21. (An alternative to the nondeterministic hierarchy theorems.)

If one desires lower bounds on only the deterministic time or space complexity of **problems**, the deterministic hierarchy theorems, [SHL65] and [HS65], can be used in place of Fact 2.11 to assert, for example, the existence of a **set** $A \in DTIME(2^n) - DTIME((2-\epsilon)^n)$.

The deterministic hierarchy theorems follow by fairly straightforward **diagonalizations**, while the known proof of Fact 2.11 requires additional deeper "translational" arguments. For this reason, it seems worth pointing out that the deeper results of Fact 2.11 are not always needed to deduce lower bounds on nondeterministic complexity. In particular we consider nondeterministic time complexity.

If A is a set of words and Σ is the smallest alphabet such that $A \subseteq \Sigma^+$, let \bar{A} denote the set $\Sigma^+ - A$.

The first **lemma** was brought to **my** attention by Paul Young.

Lemma 4.21.1 (Young). Let $T(n)$ be countable and satisfy $T(n) \geq n$.

There is a set $A \subseteq \{0,1\}^+$ such that

$$A \in NTIME(n \cdot T(n)) \quad \text{and} \quad \bar{A} \notin NTIME(T(n)) .$$

Proof. Let $(M(y) \mid y \in \{0,1\}^+)$ be an efficient effective enumeration of the nondeterministic IOTM's such that each IOTM in the list has two work tapes and has input alphabet $\{0,1\}$. By "efficient effective enumeration" we mean that there is a universal IOTM U and a constant c such that for all $x, y \in \{0,1\}^+$,

- (i). U accepts $x\#y$ iff $M(y)$ accepts x ,
and
(ii). $\text{Time}_U(x\#y) \leq c|y|\text{Time}_{M(y)}(x)$.

The standard methods of enumerating Turing machines (e.g. lists of quintuples [Min67]) are suitably efficient.

Now let

$$A = \{ y \in \{0,1\}^+ \mid M(y) \text{ accepts } y \text{ and } \text{Time}_{M(y)}(y) \leq T(|y|) \}.$$

Since $T(n)$ is countable, and $\{M(y)\}$ is an efficient enumeration in the above sense, it follows that $A \in \text{NTIME}(n \cdot T(n))$.

Now suppose $\bar{A} \in \text{NTIME}(T(n))$. First, implicit in [BGW70] is the result that if $B \in \text{NTIME}(T(n))$ then some IOTM with two work tapes accepts B within time $T(n)$. Therefore, for some y_0 , $M(y_0)$ has two work tapes and $M(y_0)$ accepts \bar{A} within time $T(n)$. Now

$$\begin{aligned} y_0 \in \bar{A} & \text{ iff } M(y_0) \text{ accepts } y_0 \text{ and } \text{Time}_{M(y_0)}(y_0) \leq T(|y_0|) \\ & \text{ (by definition of } M(y_0)) \\ & \text{ iff } y_0 \in A \text{ (by definition of } A). \end{aligned}$$

This contradiction proves $\bar{A} \notin \text{NTIME}(T(n))$. □

Lemma 4.21.2. Let $\leq_{\text{eff}} \in \{\leq_{\log\text{-lin}}, \leq_{\log}, \leq_{\text{poly}}, \leq\}$, let A and B be sets of words with $A \subseteq \Sigma^+$ and $B \subseteq A^+$, and let $A \leq_{\text{eff}} B$ via f , where $f(\Sigma^+) \subseteq D \subseteq \Delta^+$ for some set D . Then $\bar{A} \leq_{\text{eff}} D - B$ via f .

Proof. The proof is immediate from the definition of **transformation**
(Definition 3.3). \square

Define the predicate EQUIV as $\text{EQUIV}(E_1, E_2)$ iff $L(E_1) = L(E_2)$.
We illustrate the use of Lemmas 4.21.1 and 4.21.2 by proving an exponential lower bound on the nondeterministic time complexity of $\text{EQUIV}(\{0,1\}, \{U, \cdot, ^2\})$.

Corollary 4.21.3. There is a rational $c > 1$ such that
 $\text{EQUIV}(\{0,1\}, \{U, \cdot, ^2\}) \notin \text{NTIME}(c^n)$.

Proof. By Lemma 4.21.1 let the set $A \subseteq \{0,1\}^+$ satisfy
 $A \in \text{NTIME}(n2^n)$ but $\bar{A} \notin \text{NTIME}(2^n)$. Let

$$D = \{ (E_1, E_2) \mid E_1 \text{ and } E_2 \text{ are } \{0,1\}\text{-}\{U, \cdot, ^2\}\text{-expressions} \}.$$

Since $A \in \text{EXPNTIME}$, the proof of Lemma 4.15 (with Lemma 4.10) gives a function f such that $A \leq_{\log\text{-lin}} \text{INEQ}(\{0,1\}, \{U, \cdot, ^2\})$ via f ,
 f is length bn bounded for some $b \in \mathbb{N}^+$, and $f(\{0,1\}^+) \subseteq D$.

By Lemma 4.21.2,

$$\bar{A} \leq_{\log\text{-lin}} D = \text{INEQ}(\{0,1\}, \{U, *, ^1\}) = \text{EQUIV}(\{0,1\}, \{U, \cdot, ^2\}) \text{ via } f.$$

Now let $c \leq 2^{1/b}$ and conclude as usual (via Lemma 3.6) that
 $\text{EQUIV}(\{0,1\}, \{U, \cdot, ^2\}) \in \text{NTIME}(c^n)$ implies $\bar{A} \in \text{NTIME}(2^n)$ contrary
to assumption. \square

Corollary 4.21.3 is of value itself because Theorem 4.18 does not
imply Corollary 4.21.3 given present knowledge. **EXPNTIME** is not
known to be closed under complementation.

Remark 4.22. (Effective **i.o.** speedup and **a.e.** n lower bounds.)

It can be checked that the transformations described in section 4.1 are logspace-invertible (cf. Definition 3.12). In all cases, the expression $E_M(x)$ is syntactically simple enough that an IOTM can determine within space $\log|y|$ that $y = E_M(x)$ for **some** x . The word x **can** then be "read off" subexpression E_1 of $E_M(x)$. From the results of section 3.3A and Theorems 4.12 and 4.13 we immediately obtain:

Corollary 4.22.1. There is a rational $c > 1$ such that $NEC(\{0,1\},(U, \cdot, *, ^2))$ possesses c^n -to-log effective **i.o.** speedup.

Corollary 4.22.2. For all rational $r < 1$, $NEC(\{0,1\},(U, \cdot, *))$ possesses n^r -to-log effective **i.o.** speedup.

None of the sets $NEC(\Sigma, \varphi)$ or $INEQ(\Sigma, \varphi)$ described above possess a nontrivial lower bound on **a.e. n complexity** because our syntactic conventions imply that the length of any well-formed Σ - φ -expression is divisible by 3. However, Σ - φ -expressions can be "naturally padded" (cf. Definition 3.19) to any length divisible by 3. For **example, using** methods of section 3.3B we can prove the following:

Corollary 4.22.3. Let $B = NEC(\{0,1\},(U, \cdot, *, ^2))$. There is a rational $c > 1$ such that given any deterministic IOTM M which accepts B there is an integer n_0 such that $(\forall n \geq n_0 \text{ such that } 3 \text{ divides } n)(\exists x \in B) [|x| = n \text{ and } \text{Space}_M(x) > c^n]$.

Remark 4.23. (λ is not needed.)

The ability to write λ as a regular-like expression is not essential to our proofs. For example, for any $k \in \mathbb{N}^+$, an expression $[\Sigma^{+\leq k}]_{sq}$ which describes $\{ \omega \in \Sigma^+ \mid |\omega| \leq k \}$ can be constructed by the rules:

$$[\Sigma^{+\leq 2k}]_{sq} = ([\Sigma^{+\leq k}]_{sq})^2 \cup \Sigma$$

and

$$[\Sigma^{+\leq 2k+1}]_{sq} = ([\Sigma^{+\leq k}]_{sq})^2 \cdot \Sigma \cup \Sigma^2 \cup \Sigma.$$

$[\Sigma^{+\leq k}]_{sq}$ can be used in place of $[\Sigma^{\leq k}]_{sq}$ in the expressions $E_M(x)$ constructed to prove Lemmas 4.8 and 4.15. Further minor modifications to the expressions are required; the reader can easily supply these.

Remark. One can of course investigate the complexity of $NEC(\Sigma, \varphi)$ or $INEQ(\Sigma, \varphi)$ for sets of operations φ other than those discussed here. For example, Hunt [Hun73b] (see also [AHU74, Chapter 1]) considers regular expressions extended by intersection and proves that $NEC(\{0, 1\}, \{U, \cdot, *, \cap\}) \notin NSPACE(c^{\sqrt{n/\log n}})$ for some $c > 1$. Hunt and Hopcroft (personal communication) have also observed that $NEC(\{0, 1\}, \{U, \cdot, *, \cap\}) \in EXPSpace$. The above lower bound can be improved slightly for the inequivalence problem;

$INEQ(\{0, 1\}, \{U, \cdot, *, \cap\}) \notin NSPACE(c^{n/\log n})$ for some $c > 1$.

A proof can be found in [Sto74]. ^{We believe that a} combination of the techniques in [Hun73b] and [Sto74] ^{also} should yield the same lower bound $NSPACE(c^{n/\log n})$ ~~also~~ on the NEC problem.

4.2 Expressions With Complementation.

To simplify notation in this section (and later) let $g(k, r)$ be the function $2^{2^{\cdot^{2^r}}}$ for $k \in \mathbb{N}$ and real r . That is, $g(0, r) = r$ and $g(k+1, r) = 2^{g(k, r)}$ for all $k \in \mathbb{N}$.

This section considers regular-like expressions with the operation of set complementation. In particular this includes the class of "star-free" expressions containing only the operations \cup , \cdot , and \sim . The languages describable by star-free expressions have been extensively studied as an interesting subset of the regular languages [cf. MP71]. For example, it is known that star-free expressions cannot describe all regular languages; in particular $(00)^*$ is the language of no $\Sigma\{U, \cdot, \sim\}$ -expression.

Our interest in such expressions is to characterize the complexity of their equivalence problem. As was mentioned earlier, Brzowski [Brz64] gives an algorithm which checks equivalence of regular expressions extended by other Boolean operations including \sim .

Even though star-free expressions cannot describe all regular languages, we shall show that they can describe certain regular languages much more succinctly than can regular-like expressions which use only \cup , \cdot , and $*$. In particular, a star-free expression of length $O(n)$ can describe the computations of any given STM which uses space $g(\lceil \log_b n \rceil, 0)$ on any given input of length n . It follows that the inequivalence problem for star-free expressions is enormously difficult

to decide; $\text{NEC}(\{0,1\},\{\cup,\cdot,\sim\})$ is accepted by no IOTM which operates within space $g(\lceil \log_b n \rceil, 0)$ for $b > 3$ (Theorem 4.27).

It immediately follows that other decision problems concerning star-free expressions are also this complex. For example, the problem of finding a shortest star-free **expression** equivalent to a given star-free expression also requires space $g(\lceil \log_b n \rceil, 0)$, (cf. Remark 4.20). See also [Hun73a], [Hun73c], and [HR74] for other predicates which are as difficult to decide as NEC.

By examining a straightforward algorithm for deciding $\text{NEC}(\{0,1\},\{\cup,\cdot,\sim\})$, we see why such multiple exponential complexity might arise. Given a $\{0,1\}$ - $\{\cup,\cdot,\sim\}$ -expression E , we might first construct a nondeterministic finite automaton (NFA) which accepts $L(E)$ and then check that this NFA does not accept $\{0,1\}^*$. This NFA can be constructed inductively on the structure of E by well-known methods [cf. RS59]. However, given a NFA F with q states which accepts $L(E_1)$, to construct a NFA F' which accepts $L(\sim E_1)$, we may first have to transform F to an equivalent deterministic finite automaton (DFA) F'' , say by the Rabin-Scott "subset construction" [RS59]. F'' , and thus F' , might have as many as 2^q states. F' might then be incorporated into a larger NFA which later must be made deterministic, resulting in a DFA with 2^{2^q} states, and so on. This suggests that the number of exponential functions which must be composed to yield a complexity bound is closely related to the depth of nesting of \sim operations in the expressions being checked for equivalence.

The relation between " \sim -depth" and complexity is characterized by another result (Theorem 4.28) which states that, for any fixed integer k , there is a $c \in \mathbb{Q}^+$ such that the inequivalence Problem for $\{0,1\}$ - $\{U, \cdot, \sim, *\}$ -expressions of maximum \sim -depth k cannot be solved by an algorithm which uses less than space $g(k, cn)$; however this problem can be solved by an algorithm which uses space $g(k, dn)$ for some other constant d . If $*$ is not allowed, we show (Theorem 4.29) that the inequivalence problem for $\{0,1\}$ - $\{U, \cdot, \sim\}$ -expressions of maximum \sim -depth k requires space $g(k-3, c\sqrt{n})$ for some constant c .

Definition 4.24. Let E be a Σ - φ -expression and define depth(E) inductively as follows:

$$\begin{aligned} \text{depth}((\sigma)) &= 0 \quad \text{for } a \in \Sigma \cup \{\lambda\} ; \\ \text{and } \left. \begin{aligned} \text{depth}((E_1 @ E_2)) &= \max(\text{depth}(E_1), \text{depth}(E_2)) \\ \text{depth}((E_1 @)) &= \text{depth}((@E_1)) = \text{depth}(E_1) \end{aligned} \right\} \text{ if } @ \in \varphi - \{\sim\} ; \\ \text{depth}((\sim E_1)) &= \text{depth}(E_1) + 1 . \end{aligned}$$

If $k \in \mathbb{N}$, let $P(\Sigma, \varphi, \text{depth} \leq k)$ denote the set $P(\Sigma, \varphi)$ restricted to regular-like expressions of depth not exceeding k . That is, if P is an n -place predicate,

$$\begin{aligned} \underline{P(\Sigma, \varphi, \text{depth} \leq k)} &= P(\Sigma, \varphi) \cap \{ (E_1, E_2, \dots, E_n) \mid \text{depth}(E_i) \leq k \\ &\quad \text{for } 1 \leq i \leq n \} . \end{aligned}$$

We first obtain some rough upper bounds on the complexity of inequivalence problems with complementation. The algorithms utilize the "subset-construction" together with some ideas used in the

algorithm of Proposition 4.11.

Proposition 4.25.

- (1). $\text{INEQ}(\{0,1\}, \{U, \cdot, \sim, *\}) \in \text{NSPACE}(g(n,0))$.
 (2). For all $k \in \mathbb{N}^+$, $\text{INEQ}(\{0,1\}, \{U, \cdot, \sim, *\}, \text{depth} \leq k) \in \text{NSPACE}(g(k,2n))$.[†]

Proof sketch. Given $\{0,1\}$ - $\{U, \cdot, \sim, *\}$ -expressions E_1 and E_2 , construct NFA's which accept $L(E_1)$ and $L(E_2)$. Note that if $L(E_i')$ is accepted by an NFA with q_i states for $i = 1, 2$, then:

(i). $L((E_1' \cup E_2'))$ and $L((E_1' \cdot E_2'))$ are each accepted by an NFA with $q_1 + q_2 + 2$ states;

(ii). $L((E_1')^*)$ is accepted by an NFA with $q_1 + 2$ states;

(iii). $L((\sim E_1'))$ is accepted by an NFA with 2^{q_1} states.

See for example [RS59] or [HU69].

It is now easy to show by induction that if E is a $\{0,1\}$ - $\{U, \cdot, \sim, *\}$ -expression and $n = |E|$, then $L(E)$ is accepted by an NFA with $\leq g(n-1,0)$ states. If also $\text{depth}(E) \leq k$, then $L(E)$ is accepted by an NFA with $\leq g(k,n)$ states. A description of an NFA with q states can be coded onto an SIM tape within space $O(q^2)$ in a straightforward way:

Note $(g(k,n))^2 \leq g(k,2n)$ and $(g(n-1,0))^2 \leq g(n,0)$ for all $k, n \geq 1$.

Also, given two NFA's with q_1 and q_2 states, by using the method of Proposition 4.11, a nondeterministic IOTM can determine within

[†]By [Sav70], the distinction between $\text{NSPACE}(S(n))$ and $\text{DSpace}(S(n))$ is essentially negligible for $S(n) \geq 2^{2^n}$. For example, $\text{NSPACE}(2^{2^n}) \subseteq \text{DSpace}(2^{2^{n+1}})$. We consider NSPACE here for definiteness.

space $q_1 + q_2$ whether or not they accept different languages.

The conclusions follow. \square

The next lemma contains all the technical details required to obtain the lower bounds. The proof of the **lemma** shows how expressions using \sim can very succinctly "**simulate**" the computations of STM's.

Lemma 4.26. Let M be a (nondeterministic) STM which accepts a set $A \subseteq I^+$ within space $S(n)$. Assume $\# \notin I$. **There** are deterministic IOTM's \mathcal{M} and \mathcal{M}' which compute functions f and f' respectively, there is a constant $a \in \mathbb{Q}^+$ and a polynomial p (all depending on M) with the following properties.

For all $x \in I^+$ and all $m, z \in \mathbb{N}^+$ such that $S(|x|) \leq g(m, z)$:

- (1). $f(x\#0^m\#0^z) = E (= E_{\mathcal{M}}(x, m, z))$ where
 - (li). E is a $\{0, 1\}$ - $\{U, \cdot, \sim\}$ -expression ;
 - (lii). $|E| \leq a(3^m m^2 z^2 + |x|)$;
 - (liii). $\text{depth}(E) \leq m + 3$;
 - (liv). $L(E) \neq \{0, 1\}^*$ iff $x \in A$;
 - (lv). $\text{Time}_{\mathcal{M}}(x\#0^m\#0^z) \leq p(|E|)$ and $\text{Space}_{\mathcal{M}}(x\#0^m\#0^z) \leq |E|$.
- (2). $f'(x\#0^m\#0^z) = E' (= E_{\mathcal{M}'}(x, m, z))$ where
 - (2i). E' is a $\{0, 1\}$ - $\{U, \cdot, \sim, *\}$ -expression ;
 - (2) $|E'| \leq a(3^m m^2 z + m^2 |x|)$;
 - (2iif). $\text{depth}(E') = m$;
 - (2iv). $L(E') \neq \{0, 1\}^*$ iff $x \in A$;
 - (2v). $\text{Time}_{\mathcal{M}'}(x\#0^m\#0^z) \leq p(|E'|)$ and $\text{Space}_{\mathcal{M}'}(x\#0^m\#0^z) \leq |E'|$.

Before proving this lemma, we prove the main results which illustrate its use. The first result concerns the case of unlimited ---depth.

Theorem 4.27. For all rational $b > 3$:

- (1). $\text{NSPACE}(g(\lceil \log_b n \rceil, 0)) \leq_{\text{p}\ell} \text{NEC}(\{0,1\}, \{U, \cdot, \sim\})$;
and
(2). $\text{NEC}(\{0,1\}, \{U, \cdot, \sim\}) \not\leq \text{NSPACE}(g(\lceil \log_b n \rceil, 0))$.

Proof. (1). Given $b > 3$, let $A \in \text{NSPACE}(g(\lceil \log_b n \rceil, 0))$ and let M be an STM which accepts A within space $S(n) = g(\lceil \log_b c''n \rceil, 1)$ for some constant c'' chosen so that (in particular) $S(n) \geq n+1$.

We describe a deterministic algorithm which computes a transformation f'' such that $A \leq_{\text{p}\ell} \text{NEC}(\{0,1\}, \{U, \cdot, \sim\})$ via f'' . Given x with $n = |x|$, first compute $m = \lceil \log_b c''n \rceil$; note that this can be done in time polynomial in n and space linear in n . Let \mathfrak{M} be the IOTM of Lemma 4.26. Simulate \mathfrak{M} on input $x\#0^m\#0$, obtaining a $\{0,1\}$ - $\{U, \cdot, \sim\}$ -expression E . Finally produce E as output.

Since $S(n) = g(m, 1)$, E satisfies the conditions (1) of Lemma 4.26. First $|E| \leq a(3^m + n) \leq c'n$ for some constant c' which depends on a , b , and c'' , but not on n . Thus f'' is linear bounded. Also, \mathfrak{M} operates within time $p(c'n)$ and space $c'n$ on input $x\#0^m\#0$, where p is a polynomial. Therefore $f'' \in \text{polylin}$. Since $L(E) \neq \{0,1\}$ iff $x \in A$, f'' is the required transformation.

(2). Assume $\text{NEC}(\{0,1\}, \{U, \cdot, \sim\}) \in \text{NSPACE}(g(\lceil \log_b n \rceil, 0))$ for some $b > 3$. Choose rational b', b'' with $3 < b' < b'' < b$.

By Fact 2.11, there is a set A such that

$$A \in \text{NSPACE}(g(\lceil \log_b n \rceil, 0)) - \text{NSPACE}(g(\lceil \log_{b^c} n \rceil, 0)).$$

By part (1) above, by Lemma 3.7, and by **assumption**, it follows that

$$A \in \text{NSPACE}(g(\lceil \log_b cn \rceil, 0)) \text{ for some } c \in \mathbb{N}^+.$$

However, $\lceil \log_b cn \rceil < \lceil \log_{b^c} n \rceil$ for all but finitely many n .

Therefore $A \in \text{NSPACE}(g(\lceil \log_{b^c} n \rceil, 0))$, and this contradiction implies the conclusion. \square

Recall that Proposition 4.25 gives an upper bound of space $g(n, 0)$ for this problem versus the lower bound of $g(\lceil \log_b n \rceil, 0)$ just proven. Whether this gap can be decreased is an open question on which we will comment at the end of this section.

We obtain a tighter complexity characterization for the case $\{U, \cdot, \sim, *\}$ by holding \sim -depth fixed at some k while allowing the lengths of expressions to grow.

Theorem 4.28. For all integers $k \geq 1$:

- (1). $\text{NEC}(\{0, 1\}, \{U, \cdot, \sim, *\}, \text{depth} \leq k)$ is $\leq_{p\ell}$ -complete
in $\bigcup_{d \in \mathbb{N}} \text{NSPACE}(g(k, dn))$;

(2). In particular,

(2i). There is a $c \in \mathbb{Q}^+$ such that

$$\text{NEC}(\{0, 1\}, \{U, \cdot, \sim, *\}, \text{depth} \leq k) \notin \text{NSPACE}(g(k, cn)),$$

(2ii). $\text{NEC}(\{0, 1\}, \{U, \cdot, \sim, *\}, \text{depth} \leq k) \in \text{NSPACE}(g(k, 2n))$.

Proof. First, the upper bound (2ii) required for completeness (1) is given by Proposition 4.25.

To prove the other half of completeness, let $A \in \text{NSPACE}(g(k, dn))$ for **some** $k, d \in \mathbb{N}^+$, and let the STM M accept A within space $S(n) = g(k, dn)$.

We show how to compute f'' such that $A \leq_{\text{p}\ell}^* \text{NEC}(\{0, 1\}, \{U, \cdot, \sim\}, \text{depth} \leq k)$ via f'' . Given x with $n = |x|$, set $m = k$ and $z = dn$, then simulate the IOTM \mathfrak{M}' of Lemma 4.26 on input $x\#0^m\#0^z$, and produce the resulting expression E' as output. since $S(n) = g(m, z)$, E' satisfies the conditions (2) of Lemma 4.26. In particular, $\text{depth}(E') = k$ and $|E'| \leq a(3^m z + m^2 n) \leq c'n$ for a constant c' which is independent of n . As in the preceding proof, it is easy to see that $f'' \in \text{polylin}$ and that f'' transforms A correctly.

The lower bound (2i) follows from (1) in the usual way. \square

Theorem 4.29. For all integers $k \geq 4$:

$$(1). \quad \bigcup_{d \in \mathbb{N}} \text{NSPACE}(g(k-3, d\sqrt{n})) \leq_{\text{p}\ell} \text{NEC}(\{0, 1\}, \{U, \cdot, \sim\}, \text{depth} \leq k) ;$$

(2). There is a $c \in \mathbb{Q}^+$ such that

$$\text{NEC}(\{0, 1\}, \{U, \cdot, \sim\}, \text{depth} \leq k) \not\subseteq \text{NSPACE}(g(k-3, c\sqrt{n})).$$

Proof. Proceed as in the proof of Theorem 4.28 except set $m = k-3$. and $z = \lceil d\sqrt{n} \rceil$, and use \mathfrak{M} in place of \mathfrak{M}' . \square

We now turn to the proof of Lemma 4.26. The proof of course can be simplified if one is content to show only that space $g(k, n)$ is not **sufficient** for my fixed k , or operation $*$ is allowed, or one is content with weaker bounds on the length and depth of $E_M(x, m, z)$ and $E_M'(x, m, z)$. A version of our proof simplified in these ways is

sketched in Chapter 11 of [AHU74].

The proof of Lemma 4.26 is similar in spirit to the proof in [Mey73]

that the emptiness problem for " γ -expressions" is not elementary-recursive. It is instructive to review one essential idea of [Mey73] which is also used here: how regular-like expressions using \sim and y (y is defined below) can very succinctly describe the computations of STM's.

Let M be an SIM and let d be an i.d. of M with $|d| = k$. Recall from the proofs of section 4.1 that, given a regular-like expression E which describes Σ^k (that is, E is a "ruler" which measures distance k), by using E as a subexpression and using operations U , \cdot , and $*$, one can write an expression $E_M(d)$ which describes $\Sigma^* = \text{Comp}_M(d)$ for some alphabet Σ . If operation \sim can also be used, $\sim E_M(d)$ describes $\text{Comp}_M(d)$.

Now let G be a particular deterministic "counting" SIM. When started on an i.d. of the form $\&q_00^k\&$, G successively adds 1 to the binary representation on its tape until 1^k is obtained. G then halts. Note that the unique computation of G on input $\&q_00^k\&$ is longer than 2^k . Therefore, $\sim E_G(\&q_00^k\&)$ describes a single word of length exceeding 2^k . Now suppose an operation y is available where $y(\omega) = (w \in \Sigma \mid |w| = |\omega|)$ for $\omega \in \Sigma^*$. The expression $E' = y(\sim E_G(\&q_00^k\&))$ thus describes $\Sigma^{k'}$ for some $k' > 2^k$. Also, if E is an expression such that $L(E) = \Sigma^k$, it is not hard to see (cf. Lemmas 4.8 and 4.9) that $|E'| \leq c|E|$ for some constant c

independent of k . In **summary**, given an expression E (a "ruler") which describes Σ^k , one can write an expression E' (an exponentially longer "ruler") which describes $\Sigma^{k'}$ where $k' > 2^k$. Moreover, $|E'| \leq c|E|$ for some constant c .

Now starting with the "ruler" Σ^z for some z , and applying the above construction m times, we obtain a $\Sigma^*\{U, \cdot, \cdot, \cdot, \cdot, \cdot\}$ -expression E which describes Σ^ℓ for some $\ell > g(m, z)$. Moreover, $|E| \leq O(c^m z)$. As in section 4.1, E can now be used as a ruler to write an expression of length $O(c^m z)$ which simulates the computations of a given STM M , even if M uses space $g(m, z)$. This is a very succinct representation of the computations of M , since $c^m z$ grows much slower than $g(m, z)$ as a function of m . In particular, it follows that $NEC(\Sigma, \{U, \cdot, \cdot, \cdot, \cdot, \cdot\}) \not\subseteq NSPACE(g(k, n))$ for all $k \in \mathbb{N}$.

However, if y cannot be used, difficulties arise. $\sim_{E_G}(\& q_0 0^k \&)$ is a single word of length exceeding 2^k . However, to continue the construction, we need a "ruler" consisting of all words of some large length. The solution to this dilemma, described in detail shortly, is to write an expression which describes all cycles of a computation. This set of cycles can then serve as a "ruler".

A preliminary lemma is useful. In the proof of Lemma 4.26, it is convenient to represent i.d.'s in a slightly redundant form. The j^{th} symbol of the redundant form of an i.d. d contains the information in the $(j-1)^{\text{th}}$, j^{th} , and $(j+1)^{\text{th}}$ symbols of d .

Definition 4.30. Let $M = (I, \Gamma, Q, \delta, q_0, q_a)$ be an STM. Assume $\$ \notin \Gamma \cup Q$. Define the map $\rho: (\Gamma \cup Q)^+ \rightarrow ((\Gamma \cup Q \cup \{\$\})^{X3})^+$ as follows.

. If $d_1, d_2, \dots, d_k \in \Gamma \cup Q$, $\rho(d_1 d_2 d_3 \dots d_k) = d_1' d_2' d_3' \dots d_k'$ where

$$d_j' = \begin{cases} (\$, d_1, d_2) & \text{if } j = 1 \\ (d_{j-1}, d_j, d_{j+1}) & \text{if } 2 \leq j \leq k-1 \\ (d_{k-1}, d_k, \$) & \text{if } j = k \end{cases}$$

Note ρ is one-to-one so ρ^{-1} is a function on $\text{range}(\rho)$.

r is a redundant i.d. (r.i.d.) of M iff $\rho^{-1}(r)$ is defined and is an i.d. of M .

The function Next_M is extended to r.i.d.'s in the obvious way:

If r_1, r_2 are r.i.d.'s of M ,

$$r_2 \in \text{Next}_M(r_1) \quad \text{iff} \quad \rho^{-1}(r_2) \in \text{Next}_M(\rho^{-1}(r_1)).$$

The technical convenience gained by using r.i.d.'s is the following.

If r_1, r_2 are r.i.d.'s, a "local check" (cf. Lemma 2.14) consists of comparing the single j^{th} symbols of r_1 and r_2 for some j .

Furthermore, given an arbitrary word r_2 in $((\Gamma \cup Q \cup \{\$\})^{X3})^+$, one can check if $r_2 \in \rho((\Gamma \cup Q)^+)$ or not by checking each adjacent pair of symbols in r_2 for consistency. This is formalized in the following

Lemma 4.31 which is the analogue for r.i.d.'s of Lemma 2.14.

Lemma 4.31. Let $M = (I, \Gamma, Q, \delta, q_0, q_a)$ be an STM. Let $\$$ be the special endmarker as in Definition 4.30 above.

Let $\Sigma = (\Gamma \cup Q \cup \{\$, \$\})^{x3} - \{(\$, \$, \$)\}^\dagger$

There are functions $R_M: \Sigma \rightarrow 2^\Sigma$, $J_M: \Sigma \rightarrow 2^\Sigma$ with the following property.

Let $r_1 = r_{11}r_{12}r_{13}\cdots r_{1k}$ be an r.i.d. of M,

and let $r_2 = r_{21}r_{22}r_{23}\cdots r_{2k}$ be arbitrary,

where $r_{1j}, r_{2j} \in \Sigma$ for $1 \leq j \leq k$.

Then $r_2 \in \text{Next}_M(r_1)$ iff

- (1). $r_{2j} \in R_M(r_{1j})$ for all j , $1 \leq j \leq k$,
and
(2). $r_{2,j+1} \in J_M(r_{2j})$ for all j , $1 \leq j \leq k-1$.

Proof. $J_M((\sigma_1, \sigma_2, \sigma_3))$ contains all triples in Σ which could consistently follow $(\sigma_1, \sigma_2, \sigma_3)$ in my word in $\rho((\Gamma \cup Q)^+)$.

For all $(\sigma_1, \sigma_2, \sigma_3) \in \Sigma$,

$$J_M((\sigma_1, \sigma_2, \sigma_3)) = \{ (\sigma_2, \sigma_3, \sigma) \in \Sigma \mid \sigma \in \Gamma \cup Q \cup \{\$, \$\} \}$$

R_M is defined in the obvious way from the function N_M of Lemma 2.14:

$$R_M((\sigma_1, \sigma_2, \sigma_3)) = \{ (\sigma_1', \sigma_2', \sigma_3') \in \Sigma \mid \sigma_1' \sigma_2' \sigma_3' \in N_M(\sigma_1 \sigma_2 \sigma_3) \}.$$

The simple verification that J_M and R_M have the required property is left as an exercise. \square

[†]The triple $(\$, \$, \$)$ never appears in a word in $\text{range}(\rho)$. A technical condition within the proof of Lemma 4.26 requires that $(\$, \$, \$)$ be explicitly removed from Σ .

Proof of Lemma 4.26. Part (1) is done first in detail. (2) then follows by some minor modifications to (1).

(1). Let $M = (I_M, \Gamma_M, Q_M, \delta_M, q_0, q_a)$ be the given STM which accepts a set A within space $S(n)$. Let $x \in I_M^+$ and integers $m, z \in \mathbb{N}^+$ with $S(|x|) \leq g(m, z)$ be given. Let $n = |x|$.

The major portion of the proof describes the construction of a Σ - $\{\cup, \cdot, --\}$ -expression E_{m+1} such that $L(E_{m+1}) \neq \Sigma^*$ iff $x \in A$, where Σ is a large alphabet defined below. The $\{0, 1\}$ - $\{\cup, \cdot, \sim\}$ -expression $E_M(x, m, z)$ is then obtained from E_{m+1} by appropriately coding the symbols Σ into binary. We show that $E_M(x, m, z)$ satisfies conditions (li) - (liv) of Lemma 4.26. It will be clear from the description of the construction that there is an IOTM \mathcal{M} which computes $E_M(x, m, z)$ from $x \# 0^m \# 0^z$ within time polynomial in and space linear in $|E_M(x, m, z)|$, so that (lv) is also satisfied.

A particular deterministic "counting" STM \mathcal{C} used here differs in an important way from the one described earlier in the outline of the proof for y -expressions. Namely, the halting state is never entered. When started in an i.d. $\& q_0 0^a \&$, \mathcal{C} cycles forever through the 2^a i.d.'s $\{ \& \omega q_0 \& \mid \omega \in \{0, 1\}^a \}$ (with several steps taken between occurrences of these i.d.'s to perform the addition modulo 2^a). Also, letting D be the set of i.d.'s which occur in a computation of \mathcal{C} started in $\& q_0 0^a \&$, \mathcal{C} is programmed so that the particular word $\& q_0$ appears as a subword of precisely one i.d. in D . In the construction of E_{m+1} , $\& q_0$ is used to uniquely identify the initial i.d. $\& q_0 0^a \&$.

Definition of the "counting" machine \mathcal{G} .

$\mathcal{G} = (I, \Gamma, Q, \delta, q_0, q_2)$ where $I = \Gamma = \{0, 1, \&\}$ and $Q = \{q_0, q_1, q_2\}$.
 $\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{-1, 0, 1\}}$ is given by the following table. q_1 is a left-moving state which performs the addition. q_0 is a right-moving state which returns to $\&$ after the addition is completed.

δ	0	1	$\&$
q_0	$\{(q_0, 0, 1)\}$	$\{(q_0, 1, 1)\}$	$\{(q_1, \&, -1)\}$
q_1	$\{(q_0, 1, 1)\}$	$\{(q_1, 0, -1)\}$	$\{(q_0, \&, 1)\}$
q_2	\emptyset	\emptyset	\emptyset

Table 4.2.1. Transition table for "counting" machine \mathcal{G} .

Also for $\ell \in \mathbb{N}^+$ define:

$$\begin{aligned} \underline{\text{init}}(\ell) &= \rho(\&q_0 0^\ell \&), \\ \underline{\text{loop}}(\ell) &= \min\{j \in \mathbb{N}^+ \mid \text{Next}_{\mathcal{G}}(\text{init}(\ell), j) = \{\text{init}(\ell)\}\}, \\ \underline{D}(\ell) &= \bigcup_{j=0}^{\underline{\text{loop}}(\ell)-1} \text{Next}_{\mathcal{G}}(\text{init}(\ell), j). \end{aligned}$$

The next fact, which can be verified by inspection, states those

properties of \mathbb{G} to be used.

Fact 4.26.1. For all $\ell \in \mathbb{N}^+$:

- (1). $\text{loop}(\ell) \in \mathbb{N}$ and $\text{loop}(\ell) \geq 2 + 4$;
- (2). Assume $\omega \in \mathbb{D}(\ell)$. Then $(\$, \&, q_0)$ appears as a **symbol** in ω iff $\omega = \text{init}(\ell)$. Moreover, $(\$, \&, q_0)$ is the first symbol of $\text{init}(\ell)$, and $|\text{init}(\ell)| = \ell + 3$.

Now for $k = 1, 2, 3, \dots, m$, let $\mathbb{G}(k) = (I_k, \Gamma_k, Q_k, \delta_k, q_{k0}, q_{k2})$ be a "copy" of \mathbb{G} ; that is, $I_k = \Gamma_k = \{0_k, 1_k, \&_k\}$, $Q_k = \{q_{k0}, q_{k1}, q_{k2}\}$, and δ_k is given by Table 4.2.1 where all states and tape symbols are subscripted with k .

Also, let $\underline{\text{init}}_k(\ell) = \rho(\&_k q_{k0} 0_k^\ell \&_k)$.

The alphabet symbols used in writing E_{m+1} are the following.

$$\Sigma_k = (Q_k \cup \Gamma_k \cup \{\$\})^{\times 3} - \{(\$, \$, \$)\} \quad \text{for } 1 \leq k \leq m.$$

$$\Sigma_{m+1} = (Q_M \cup \Gamma_M \cup \{\$\})^{\times 3} - \{(\$, \$, \$)\}.$$

$$\Sigma_{m+2} = \{\#\}.$$

Note: For $k = 1, 2, 3, \dots, m$, ρ maps $(Q_k \cup \Gamma_k)^+$ into Σ_k^+ , and $R_{\mathbb{G}(k)}$ and $J_{\mathbb{G}(k)}$ both map Σ_k into 2^{Σ_k} (cf. Lemma 4.31).

Assume symbols are chosen so that $\Sigma_1, \Sigma_2, \Sigma_3, \dots, \Sigma_{m+2}$ are pairwise disjoint.

The entire alphabet is $\Sigma = \bigcup_{i=1}^{m+2} \Sigma_i$.

Also denote $\Sigma_{\leq k} = \bigcup_{i=1}^k \Sigma_i$

and $\Sigma_{\geq k} = \bigcup_{i=k}^{m+2} \Sigma_i$.

Let $s = \text{card}(\Sigma)$.

For the remainder of the proof, the "0-notation" has the following meaning. Let $f_1(n, m, z, s, k)$ be a function of the indicated parameters (not necessarily depending on **all** the parameters). Then $O(f_1)$ denotes an unspecified function f_2 with the property that

$$f_2(n, m, z, s, k) \leq c \cdot f_1(n, m, z, s, k) \text{ for all } n, m, z, s, k \in \mathbb{N}^+$$

where $c \in \mathbb{N}^+$ can be chosen independently of all parameters M, x, n, m, z, s, k .

Certain subexpressions occur often within E_{m+1} ; special notation is now given for these. Even though $*$ cannot be used explicitly, it is possible to write a Σ - $\{U, \cdot, \sim\}$ -**expression** which describes Θ where $\Theta \subseteq \Sigma$. First let

$$[\Sigma^*] = (\sim\# \cup \#).$$

By convention, \sim denotes complementation relative to Σ^* in this context. Therefore $I([\Sigma^*]) = \Sigma^*$. Also note that $|[\Sigma^*]| = O(1)$ and $\text{depth}([\Sigma^*]) = 1$.

If $\Theta \subsetneq \Sigma$, let

$$[\Theta^*] = \sim([\Sigma^*] \cdot (\Sigma - \Theta) \cdot [\Sigma^*])$$

where " $\Sigma - \Theta$ " as usual abbreviates an expression equal to the union of the symbols in $\Sigma - \Theta$. Note that $I([\Theta^*]) = \Theta$, $|[\Theta^*]| = O(s)$, and $\text{depth}([\Theta^*]) = 2$.

In using these expressions within the construction of E_{m+1} , the brackets [and] are deleted to improve readability. However we must ^{*} keep in mind the length and depth of the expression which Θ abbreviates.

As in the outline for y -expressions, we construct longer and longer "rulers" in stages. The expression constructed at the k^{th} stage describes a "ruler" which "measures" distance $d(k, z)$. The numbers $\underline{d(k, z)}$ are defined as follows.

$$d(1, z) = \text{loop}(z)$$

$$d(k+1, z) = \text{loop}(d(k, z) - 4) \quad \text{for } k \in \mathbb{N}^+.$$

Lemma 4.26.2. For all $k \in \mathbb{N}^+$, $d(k, z) \geq g(k, z) + 4$.

Proof. By induction on k , using Fact 4.26.1(1). \square

The sets of words which serve as "rulers" in this construction are more complicated than those used in section 4.1. For this reason, it is useful to have semantic descriptions of the rulers as well as regular-like expressions for them.

^{*}
We now define certain words in Σ which are used in these semantic descriptions. The words c_{ki} for $1 \leq k \leq m$ and $i \in \mathbb{N}$ are defined inductively. Informally, one should think of c_{ki} as the i^{th} r.i.d. of $G(k)$ started on $\text{init}_k(d(k-1, z) - 4)$ (although c_{ki} for $k > 1$ is slightly more complicated than this).

Definition of the words c_{ki} .

For $i \in \mathbb{N}$, c_{1i} is the unique word in $\text{Next}_{G(1)}(\text{init}_1(z), i)$.

For $1 \leq k \leq m-1$ and $i \in N$,

$$c_{k+1,i} = c_{k0} a_{i1} c_{k1} a_{i2} c_{k2} \sigma_{i3} c_{k3} \cdots c_{k,d(k,z)-2} \sigma_{i,d(k,z)-1} c_{k,d(k,z)-1}$$

where $\sigma_{ij} \in \Sigma_{k+1}$ for $1 \leq j \leq d(k,z)-1$

and $\sigma_i = a_{i1} a_{i2} a_{i3} \cdots a_{i,d(k,z)-1}$ is the unique word in

$$\text{Next}_{G(k+1)}(\text{init}_{k+1}(d(k,z) - 4, i)).$$

The next fact gives those properties of the $\{c_{ki}\}$ to be used.

The fact follows from the definitions of $d(k,z)$ and c_{ki} , and Fact 4.26.1.

Fact 4.26.3. (1). For all k , $1 \leq k \leq m-1$, write

$$c_{k+1,i} = c_{k0} \sigma_{i1} c_{k1} \sigma_{i2} c_{k2} \sigma_{i3} c_{k3} \cdots c_{k,d(k,z)-2} \sigma_{i,d(k,z)-1} c_{k,d(k,z)-1}$$

and $a_i = a_{i1} a_{i2} a_{i3} \cdots a_{i,d(k,z)-1}$ as above.

Then $\sigma_{i+1} \in \text{Next}_{G(k+1)}(\sigma_i)$ for all $i \in N$.

Also, for all k , $1 \leq k \leq m$:

- (2). $c_{ki} \in (\Sigma_{\leq k})^*$ for all $i \in N$;
- (3). For all $i, j \in N$, $c_{ki} = c_{kj}$ iff $i \equiv j \pmod{d(k,z)}$;
- (4). $(\$, \&_k, q_{k0})$ appears as a symbol in c_{ki} iff $c_{ki} = c_{k0}$.

We are now in a position to give semantic descriptions of the sets of words which serve as "rulers". Actually, two related sets of words SE_k and SF_k are required at a given stage k , for $1 \leq k \leq m$.

Semantic description of the "rulers" SE_k and SF_k .

For $1 \leq k \leq m$:

SE_k is the set of words of the form

$$\xi_{j'} c_{kj'} \xi_{j'+1} c_{k,j'+1} \xi_{j'+2} c_{k,j'+2} \xi_{j'+3} c_{k,j'+3} \xi_{j'+4} \dots \xi_{j''} c_{kj''} \xi_{j''+1}$$

where $j'' > j'$ and $j''+1 \equiv j' \pmod{d(k,z)}$, and where $\xi_i \in \Sigma_{\geq k+1}$

is arbitrary for $j' \leq i \leq j''+1$;

$$SF_k = SE_k \cap \{ \omega \mid c_{k0} \text{ appears exactly once as a subword of } \omega \}.$$

Again if we informally describe c_{ki} as the i^{th} r.i.d. of $G(k)$, then SE_k is the set of all computations of $G(k)$ which start on an arbitrary $c_{kj'}$, run for an arbitrary (≥ 1) number of cycles, and stop on $c_{kj''}$ such that (if the computation were continued one more step to $c_{k,j''+1}$) $c_{k,j''+1} = c_{kj'}$. Arbitrary single symbols from $\Sigma_{\geq k+1}$ occur between adjacent r.i.d.'s c_{ki} and $c_{k,i+1}$ in these computations, as well as at the beginning and end of these computations. SF_k is the set of words in SE_k which are computations (in the above sense) which run for exactly one cycle; that is, these words contain c_{k0} exactly once as a subword.

The major technical portion of the proof now follows.

Σ - $\{U, \cdot, \sim\}$ -expressions E_k and F_k for $1 \leq k \leq m$ are constructed inductively such that $L(\sim E_k) = SE_k$ and $L(\sim F_k) = SF_k$. Finally, using $\sim F_m$ as a "ruler", we construct E_{m+1} such that

$$L(E_{m+1}) \neq \Sigma^* \text{ iff } \text{Comp}_M(q_0 x b^{d(m,z)-n-2}) \neq \emptyset \text{ iff } M \text{ accepts } x.$$

The reader should recall that the alphabets $\Sigma_1, \Sigma_2, \Sigma_3, \dots, \Sigma_{m+2}$ are **pairwise** disjoint. This fact is used implicitly several times in the constructions below. Also note that many of the basic ideas used in the case $k = 1$ are also used in the induction step.

Base $k = 1$.

E_1 should describe precisely those words which are not in SE_1 . E_1 is written as a **union** of "mistakes" which could cause a word to be excluded from SE_1 ;

$$E_1 = \left(\bigcup_{i=1}^5 e_{1i} \right).$$

For each i , the length and depth of e_{1i} and a semantic description of $L(e_{1i})$ are given as **comments**.

First recall SE_1 is the set of words ω of the form (*) shown below, where also $w_{i'} = c_{10}$ for some i' with $1 \leq i' \leq A$, and $w_{i+1} \in \text{Next}_{Q(1)}(w_i)$ for all i with $1 \leq i \leq \ell-1$, and $w_1 \in \text{Next}_{Q(1)}(w_\ell)$.

$$\left. \begin{aligned} \omega &= t_0 w_1 t_1 w_2 t_2 w_3 t_3 \cdots t_{\ell-1} w_\ell t_\ell \\ \text{where } A \geq 2, \quad t_i &\in \Sigma_{\geq 2} \quad \text{for } 0 \leq i \leq A, \\ \text{and } w_i &\in \Sigma_1^{z+3} \quad \text{for } 1 \leq i \leq A. \end{aligned} \right\} (*)$$

Construction of e_{11} .

e_{11} is constructed so that, for all $\omega \in \Sigma^*$, $\omega \notin L(e_{11})$ iff ω is a word of the form (*).

The first term of e_{11} describes all words which are "too short", i.e. shorter than $z+6$. The last four terms together describe a language which includes all words longer than $z+5$ which are not in

$$(\Sigma_{\geq 2} \cdot \Sigma_1^{z+3})^* \cdot \Sigma_{\geq 2}.$$

$$e_{11} = \Sigma^{\leq z+5} \cup \Sigma_1 \cdot \Sigma^* \cup \Sigma^* \cdot \Sigma_1 \cup \Sigma^* \cdot \Sigma_{\geq 2} \cdot \Sigma^{z+3} \cdot \Sigma_1 \cdot \Sigma^* \\ \cup \Sigma^* \cdot \Sigma_{\geq 2} \cdot \Sigma^{\leq z+2} \cdot \Sigma_{\geq 2} \cdot \Sigma^* .$$

$|e_{11}| = 0(zs)$. (Recall " Σ " abbreviates an expression of length s , and thus $\Sigma^{z+3} = 0(zs)$.)

$\text{depth}(e_{11}) = 1$. (Recall " Σ^* " abbreviates an expression of depth 1.)

For the remainder of the construction of E_1 , assume $\omega \notin L(e_{11})$ and therefore that ω denotes a word of the form $(*)$.

Construction of e_{12} .

e_{12} is constructed so that $\omega \notin L(e_{12})$ iff $w_{i'} = \text{init}_1(z) = c_{10}$ for some i' , $1 \leq i' \leq \ell$.

Let $y_1 y_2 y_3 \cdots y_{z+3} = c_{10}$ where $y_j \in \Sigma_1$ for $1 \leq j \leq z+3$. Note $y_1 = (\$, \&_1, q_{10})$ is the special symbol which appears in c_{1i} iff $c_{1i} = c_{10}$. Let \bar{y}_j denote $(\Sigma - \{y_j\})$.

$L(e_{12})$ is completely described as the union of three mistakes:

- (i). y_1 does not appear in ω ; these words are described by
- $$(\Sigma - \{y_1\})^* ;$$
- or
- (ii). Some occurrence of y_1 is immediately preceded by some $\sigma \in \Sigma_1$; that is, y_1 appears in the wrong place;

$$\Sigma^* \cdot \Sigma_1 \cdot y_1 \cdot \Sigma^* ;$$

or

(iii). Some y_1 is not immediately followed by $y_2 y_3 y_4 \cdots y_{z+3}$;

these words are described by

$$\Sigma^* \cdot y_1 \cdot (\bar{y}_2 \cup y_2 \cdot (\bar{y}_3 \cup y_3 \cdot (\bar{y}_4 \cup \cdots y_{z+1} \cdot (\bar{y}_{z+2} \cup y_{z+2} \cdot \bar{y}_{z+3}))) \cdots) \cdot \Sigma^* .$$

e_{12} is now the union of the three expressions above.

$|e_{12}| = 0(zs)$; $\text{depth}(e_{12}) = 2$. (Recall " $(\Sigma - \{y_1\})^*$ " abbreviates an expression of depth 2)

Construction of e_{13} .

e_{13} is constructed so that $\omega \notin L(e_{13})$ iff ω contains no pair of adjacent triples $\sigma_1, \sigma_2 \in \Sigma_1$ which are inconsistent in the sense $\sigma_2 \notin J_{G(1)}(\sigma_1)$.

$$e_{13} = \Sigma^* \cdot \left(\bigcup_{\sigma \in \Sigma_1} \sigma \cdot (\Sigma_1 - J_{G(1)}(\sigma)) \right) \cdot \Sigma^* .$$

$$|e_{13}| = 0(1); \text{depth}(e_{13}) = 1.$$

Construction of e_{14} .

Assuming also that $\omega \notin L(e_{13})$, e_{14} is constructed so that $\omega \notin L(e_{14})$ iff ω "moves **correctly**", that is, $w_{i+1} \in \text{Next}_{G(1)}(w_i)$ for all i , $1 \leq i \leq \ell-1$, such that w_i is an r.i.d. of $G(1)$.

By Lemma 4.31, e_{14} can be written as

$$e_{14} = \Sigma^* \cdot \left(\bigcup_{\sigma \in \Sigma_1} \sigma \cdot \Sigma^{z+3} \cdot (\Sigma_1 - R_{G(1)}(\sigma)) \right) \cdot \Sigma^* .$$

$$|e_{14}| = 0(zs); \text{depth}(e_{14}) = 1.$$

Construction of e_{15} .

Assuming again that $\omega \notin L(e_{13})$ and also that w_R is an **r.i.d.** of $G(1)$, e_{15} is constructed so that $\omega \notin L(e_{15})$ iff ω "loops back correctly", that is, $w_1 \in \text{Next}_{G(1)}(w_\ell)$. Again by Lemma 4.31,

$$e_{15} = \bigcup_{j=1}^{z+3} \bigcup_{\sigma \in \Sigma_1} \Sigma^j \cdot (\Sigma_1 - R_{G(1)}(\sigma)) \cdot \Sigma^* \cdot \sigma \cdot \Sigma^{z+4-j}.$$

$$|e_{15}| = O(z^2_s); \text{depth}(e_{15}) = 1.$$

$$\text{Now } E_1 = \left(\bigcup_{i=1}^5 e_{1i} \right).$$

Comparing this construction with the definition of SE_1 in terms of the form (*), it should now be apparent that $L(\sim E_1) = SE_1$.

To construct F_1 , note that a word ω is not in SF_1 iff either $\omega \notin SE_1$ or ω contains two (or more) occurrences of c_{10} . Recall that $(\$, \&_1, q_{10})$ appears in c_{1i} iff $c_{1i} = c_{10}$. F_1 can thus be written as follows.

$$F_1 = (E_1 \cup \Sigma^* \cdot (\$, \&_1, q_{10}) \cdot \Sigma^* \cdot (\$, \&_1, q_{10}) \cdot \Sigma^*).$$

$$\text{Clearly } L(\sim F_1) = SF_1.$$

To summarize the length and depth of E_1 and F_1 :

$$(1\ell) \quad |E_1| < |F_1| = O(z^2_s)$$

$$(1d) \quad \text{depth}(E_1) = \text{depth}(F_1) = 2.$$

Induction step $k+1$ ($k < n$).

Assume we have the expressions E_k and F_k such that $L(\sim E_k) = SE_k$ and $L(\sim F_k) = SF_k$.

E_{k+1} is constructed first. The construction is similar to the base case; the details are slightly more involved. Again,

$$E_{k+1} = \left(\bigcup_{i=0}^5 e_{k+1,i} \right)$$

is written as a union of "mistakes".

Recall $(\$, \&_k, q_{k0})$ is the special symbol which appears in c_{ki} iff $i \equiv 0 \pmod{d(k,z)}$ iff $c_{ki} = c_{k0}$. Let $u = (\$, \&_k, q_{k0})$.

Construction of $e_{k+1,0}$.

$$e_{k+1,0} = E_k \cup \Sigma_{\geq k+1} \cdot (\Sigma_{\leq k} - \{u\}) \cdot \Sigma_{\geq k+1}^* \cdot \Sigma^*.$$

We claim that $\omega \notin L(e_{k+1,0})$ iff ω can be written in the form (**) below. (The portion of (**) preceding ";" denotes a single word formed by concatenating the rows in order.)

(**)

$$\begin{array}{cccccccccccccccc} t_0 & c_{k0} & r_{11} & c_{k1} & r_{12} & c_{k2} & r_{13} & c_{k3} & \cdots & c_{k,d(k,z)-2} & r_{1,d(k,z)-1} & c_{k,d(k,z)-1} & t_1 \\ & c_{k0} & r_{21} & c_{k1} & r_{22} & c_{k2} & r_{23} & c_{k3} & \cdots & c_{k,d(k,z)-2} & r_{2,d(k,z)-1} & c_{k,d(k,z)-1} & t_2 \\ & & & & & & & & \vdots & & & & \\ & c_{k0} & r_{i1} & c_{k1} & r_{i2} & c_{k2} & r_{i3} & c_{k3} & \cdots & c_{k,d(k,z)-2} & r_{i,d(k,z)-1} & c_{k,d(k,z)-1} & t_i \\ & & & & & & & & \vdots & & & & \\ & c_{k0} & r_{\ell 1} & c_{k1} & r_{\ell 2} & c_{k2} & r_{\ell 3} & c_{k3} & \cdots & c_{k,d(k,z)-2} & r_{\ell,d(k,z)-1} & c_{k,d(k,z)-1} & t_\ell ; \end{array}$$

where $\ell \geq 1$, and $t_i, r_{ij} \in \Sigma_{\geq k+1}$ for $0 \leq i \leq \ell$, $1 \leq j \leq d(k,z)-1$.

Assume $\omega \notin L(e_{k+1,0})$. First, $\omega \in L(\sim E_k)$ and therefore $\omega \in SE_k$. Therefore (in particular) $\sigma_1 c_{kj} \sigma_2$ is a prefix of ω for some j' and some $\sigma_1, \sigma_2 \in \Sigma_{\geq k+1}$. The second term of $\%_{+10}$ ensures that $c_{kj'} = c_{k0}$, by Fact 4.26.3(4), and because $c_{ki} \in (\Sigma_{\leq k})$ for all i , and $\Sigma_{\leq k}$ and $\Sigma_{\geq k+1}$ are disjoint.

$$|e_{k+1,0}| = |E_k| + O(s); \text{depth}(e_{k+1,0}) = \max(\text{depth}(E_k), 2).$$

Until further notice, we assume $\omega \notin L(e_{k+1,0})$ and therefore that ω denotes a word of the form $(**)$.

Construction of $e_{k+1,1}$.

$e_{k+1,1}$ is constructed so that $\omega \notin L(e_{k+1,1})$ iff $A \geq 2$, and $t_i \in \Sigma_{\geq k+2}$ for $0 \leq i \leq \ell$, and $r_{ij} \in \Sigma_{k+1}$ for all i, j , $1 \leq i \leq A$, $1 \leq j \leq d(k, z) - 1$.

The mistake " $\ell < 2$ " occurs iff ω contains only one occurrence of c_{k0} :

$$(\Sigma - \{u\})^* \cdot u \cdot (\Sigma - \{u\})^*.$$

The mistake " $t_i \in \Sigma_{k+1}$ " occurs iff, for some $\sigma \in \Sigma_{k+1}$, either σ immediately precedes an occurrence of c_{k0} or σ is the last symbol of ω :

$$\Sigma^* \cdot \Sigma_{k+1} \cdot (\Sigma_{\leq k})^* \cdot u \cdot (\Sigma_{\leq k})^* \cdot \Sigma_{\geq k+1} \cdot \Sigma^* \cup \Sigma^* \cdot \Sigma_{k+1}.$$

The mistake " $r_{ij} \in \Sigma_{\geq k+2}$ " occurs iff some symbol in $\Sigma_{\geq k+2}$ immediately precedes c_{kj} for some $j \neq 0$:

$$\Sigma^* \cdot \Sigma_{\geq k+2} \cdot (\Sigma_{\leq k} - \{u\})^* \cdot \Sigma_{\geq k+1} \cdot \Sigma^*.$$

Then $e_{k+1,1}$ is the union of the three expressions above,

$$|e_{k+1,1}| = O(s); \text{depth}(e_{k+1,1}) = 2.$$

For the remainder of the construction of E_{k+1} , assume $\omega \notin L(e_{k+1,0} \cup e_{k+1,1})$ and therefore that ω denotes a word of the form (***) below.

$$\omega = \quad \quad \quad (***)$$

$$\begin{array}{cccccccccccccccccccc} t_0 & c_{k0} & r_{11} & c_{k1} & r_{12} & c_{k2} & r_{13} & c_{k3} & \cdots & c_{k,d(k,z)-2} & r_{1,d(k,z)-1} & c_{k,d(k,z)-1} & t_1 \\ & c_{k0} & r_{21} & c_{k1} & r_{22} & c_{k2} & r_{23} & c_{k3} & \cdots & c_{k,d(k,z)-2} & r_{2,d(k,z)-1} & c_{k,d(k,z)-1} & t_2 \\ & & & & & & & & \vdots & & & & \\ & c_{k0} & r_{i1} & c_{k1} & r_{i2} & c_{k2} & r_{i3} & c_{k3} & \cdots & c_{k,d(k,z)-2} & r_{i,d(k,z)-1} & c_{k,d(k,z)-1} & t_i \\ & & & & & & & & \vdots & & & & \\ & c_{k0} & r_{\ell 1} & c_{k1} & r_{\ell 2} & c_{k2} & r_{\ell 3} & c_{k3} & \cdots & c_{k,d(k,z)-2} & r_{\ell,d(k,z)-1} & c_{k,d(k,z)-1} & t_\ell \end{array}$$

$$\text{and } r_i = r_{i1} r_{i2} r_{i3} \cdots r_{i,d(k,z)-1} \quad \text{for } 1 \leq i \leq \ell ;$$

where $A \geq 2$, $t_i \in \Sigma_{\geq k+2}$, and $r_{i,j} \in \Sigma_{k+1}$ for $0 \leq i \leq \ell$,
 $1 \leq j \leq d(k,z)-1$,

Construction of $e_{k+1,2}$.

$e_{k+1,2}$ is constructed to ensure that ω contains a copy of the initial r.i.d. of $G(k+1)$ started with $d(k,z)-4$ zeroes. That is,
 $\omega \notin L(e_{k+1,2})$ iff $r_{i'} = \text{init}_{k+1}(d(k,z)-4)$ for some i' , $1 \leq i' \leq \ell$.

The construction is analogous to that of e_{12} given above,

$$\begin{aligned} \text{Let } y_1 &= (\$, \&_{k+1}, q_{k+1}, 0), y_2 = (\&_{k+1}, q_{k+1}, 0, 0_{k+1}), \\ y_3 &= (q_{k+1}, 0, 0_{k+1}, 0_{k+1}), y_4 = (0_{k+1}, 0_{k+1}, 0_{k+1}), \\ y_5 &= (0_{k+1}, 0_{k+1}, \&_{k+1}), y_6 = (0_{k+1}, \&_{k+1}, \$), \end{aligned}$$

so that $\text{init}_{k+1}(d(k, z) - 4) \in y_1 y_2 y_3 y_4 y_5 y_6$.

For $1 \leq j \leq 6$, let \bar{y}_j denote $(\Sigma_{k+1} - \{y_j\})$.

$L(e_{k+1, 2})$ is described as the union of four mistakes:

(i). y_1 does not appear:

$$(\Sigma - \{y_1\})^* ;$$

(ii). Some y_1 is immediately preceded by c_{kj} for some $j \neq 0$,

that is, y_1 appears in the wrong place:

$$\Sigma^* \cdot \Sigma_{\geq k+1} \cdot (\Sigma_{\leq k} - \{u\})^* \cdot y_1 \cdot \Sigma^* ;$$

(iii). If $r_{ij} = y_1$ for some i, j , then

$$\begin{aligned} & r_{i, j+1} r_{i, j+2} \cdots r_{i, d(k, z)-4} r_{i, d(k, z)-3} \notin y_2 y_3 y_4 : \\ & \Sigma^* \cdot y_1 \cdot (\Sigma_{\leq k})^* \cdot (\bar{y}_2 \cup y_2 \cdot (\Sigma_{\leq k})^* \cdot (\bar{y}_3 \\ & \cup y_3 \cdot (\Sigma_{\leq k+1})^* \cdot \bar{y}_4 \cdot (\Sigma_{\leq k+1})^* \cdot \Sigma_{k+1} \cdot (\Sigma_{\leq k})^* \cdot \Sigma_{k+1} \cdot (\Sigma_{\leq k})^* \\ & \cdot \Sigma_{\geq k+2})) \cdot \Sigma^* ; \end{aligned}$$

(iv). If $r_{ij} = y_1$ for some i, j , then $r_{i, d(k, z)-2} r_{i, d(k, z)-1} \neq y_5 y_6$:

$$\Sigma^* \cdot y_1 \cdot (\Sigma_{\leq k+1})^* \cdot (\bar{y}_5 \cdot (\Sigma_{\leq k})^* \cdot y_6 \cup \bar{y}_6) \cdot (\Sigma_{\leq k})^* \cdot \Sigma_{\geq k+2} \cdot \Sigma^* .$$

Note that in (iii) and (iv), with ω in form $(***)$, if y_1 matches r_{ij} for some i, j , then $\Sigma_{\geq k+2}$ must match t_i . Also, each $(\Sigma_{\leq k})^*$ can only match $c_{kj'}$ for some j' .

Now $e_{k+1,2}$ is the union of the four expressions above.

The fact that ω is in form (***) verifies that

$$\begin{aligned} \omega \notin L(e_{k+1,2}) & \text{ iff } r_{i'} \in y_1 y_2 y_3 y_4^* y_5 y_6 \text{ for some } i' \\ & \text{ iff } r_{i'} = \text{init}_{k+1}(d(k,z)-4) \text{ for some } i'. \end{aligned}$$

$$|e_{k+1,2}| = O(s); \text{depth}(e_{k+1,2}) = 2.$$

Construction of $e_{k+1,3}$.

$e_{k+1,3}$ prevents inconsistent triples. That is, $\omega \notin L(e_{k+1,3})$ iff ω contains no "adjacent" triples $r_{ij}, r_{i,j+1}$, which are inconsistent in the sense $r_{i,j+1} \notin J_{G(k+1)}(r_{ij})$.

$$e_{k+1,3} = \Sigma^* \cdot \left(\bigcup_{\sigma \in \Sigma_{k+1}} \sigma \cdot (\Sigma_{\leq k})^* \cdot (\Sigma_{k+1} - J_{G(k+1)}(\sigma)) \right) \cdot \Sigma^*.$$

With ω in form (***), $(\Sigma_{\leq k})^*$ can only match some c_{kj} .

Therefore, $\omega \notin L(e_{k+1,3})$ iff $r_{i,j+1} \in J_{G(k+1)}(r_{ij})$ for all i, j ,
with $1 \leq i \leq \ell$ and $1 \leq j \leq d(k,z)-2$.

$$|e_{k+1,3}| = O(s); \text{depth}(e_{k+1,3}) = 2.$$

Construction of $e_{k+1,4}$.

$\%+14$ ensures that the moves of $G(k+1)$ are described correctly by successive r_i 's in ω .

First we need the following fact: If ω is in form (***) and $\omega = \alpha\beta\gamma$ for some $\alpha, \beta, \gamma \in \Sigma^*$, then $\beta \in SF_k$ iff either $\beta = r_i j^\tau r_{i+1, j}$ or $\beta = t_i^\tau t_{i+1}$ for some $i, j \in N$ and some $\tau \in \Sigma^*$.

This can be seen by inspection of form (***) and the semantic

description of SF_k , that is, SF_k is one complete cycle of the $\{c_{kj}\}$ starting arbitrarily.

We wish to write $e_{k+1,4}$ such that $\omega \notin L(e_{k+1,4})$ iff $r_{i+1,j} \in R_{G(k+1)}(r_{ij})$ for all i,j . We use the preceding fact about $\omega = \alpha\beta\gamma$ to locate and constrain "adjacent" symbols r_{ij} and $r_{i+1,j}$. The constraints forced by the expression below will not apply to

t_i, t_{i+1} since $t_i, t_{i+1} \in \Sigma_{\geq k+2}$ while $r_{ij}, r_{i+1,j} \in \Sigma_{k+1}$.

Since $L(\sim F_k) = SF_k$ by induction, $e_{k+1,4}$ could be written as:

$$e' = \Sigma^* \cdot (\sim F_k \cap G) \cdot \Sigma^*$$

$$\text{where } G = \left(\bigcup_{\sigma \in \Sigma_{k+1}} \sigma \cdot \Sigma^* \cdot (\Sigma - R_{G(k+1)}(\sigma)) \right).$$

By De Morgan's law, e' is equivalent to

$$e'' = \Sigma^* \cdot \sim (F_k \cup \sim G) \cdot \Sigma^*.$$

Now note the following two facts.

(i). Using only the definitions of the operations on words and the fact that $R_{G(k+1)}$ maps Σ_{k+1} into 2^Σ , the following expression can be shown to describe $L(\sim G)$.

$$\lambda \cup \Sigma \cup (\Sigma - \Sigma_{k+1}) \cdot \Sigma^* \cup \bigcup_{\sigma \in \Sigma_{k+1}} \sigma \cdot \Sigma^* \cdot R_{G(k+1)}(\sigma).$$

(ii). $\lambda \cup \Sigma \subset L(F_k)$.

Therefore e'' can be written equivalently as

$$e_{k+1,4} = \Sigma^* \cdot \sim (F_k \cup (\Sigma - \Sigma_{k+1}) \cdot \Sigma^* \cup \bigcup_{\sigma \in \Sigma_{k+1}} \sigma \cdot \Sigma^* \cdot R_{G(k+1)}(\sigma)) \cdot \Sigma^*.$$

Assuming also that $\omega \notin L(e_{k+1,3})$, by Lemma 4.31 we have that $\omega \notin L(e_{k+1,4})$ iff $r_{i+1} \in \text{Next}_{G(k+1)}(r_i)$ for all i , $1 \leq i \leq \ell-1$, such that r_i is an r.i.d. of $G(k+1)$. (Recall that $\omega \notin L(e_{k+1,2})$ ensures that r_i is an r.i.d. of $G(k+1)$ for some i' , $1 \leq i' \leq \ell$.)

$$|e_{k+1,4}| = |F_k| + O(s); \quad \text{depth}(e_{k+1,4}) = \max(\text{depth}(F_k), 1) + 1.$$

Construction of $e_{k+1,5}$.

$e_{k+1,5}$ ensures that ω "loops back correctly", that is, that $r_1 \in \text{Next}_{G(k+1)}(r_\ell)$.

First note another obvious fact: If $\omega = t_0 \alpha \beta \gamma t_a$ for some $\alpha, \gamma \in (\Sigma_{\leq k+1})^*$, then $\beta \in SE_k$ iff either $\beta = r_{1j}^\tau r_{\ell j}$ or $\beta = t_1^\tau t_{a-1}$ for some $j \in \mathbb{N}$ and some $\tau \in \Sigma^*$. This follows from the semantic description of SE_k , together with the facts that

$$t_1, t_{\ell-1} \in \Sigma_{\geq k+2} \quad \text{and} \quad \Sigma_{\leq k+1} \cap \Sigma_{\geq k+2} = \emptyset.$$

We wish to write $e_{k+1,5}$ such that $\omega \notin L(e_{k+1,5})$ iff $r_{1j} \in R_{G(k+1)}(r_{\ell j})$ for all j , $1 \leq j \leq d(k, z)-1$. Thus since $L(\sim E_k) = SE_k$, $e_{k+1,5}$ could be written as:

$$\Sigma \cdot (\Sigma_{\leq k+1})^* \cdot (\sim E_k \cap (\bigcup_{\sigma \in \Sigma_{k+1}} (\Sigma - R_{G(k+1)}(\sigma)) \cdot \Sigma^* \cdot \sigma)) \cdot (\Sigma_{\leq k+1})^* \cdot \Sigma.$$

As in the construction of $e_{k+1,4}$ above, this expression can be written equivalently as:

$$e_{k+1,5} = \Sigma \cdot (\Sigma_{\leq k+1})^* \cdot \sim (E_k \cup \Sigma^* \cdot (\Sigma - \Sigma_{k+1})) \cup \bigcup_{\sigma \in \Sigma_{k+1}} R_{G(k+1)}(\sigma) \cdot \Sigma^* \cdot \sigma \cdot (\Sigma_{\leq k+1})^* \cdot \Sigma.$$

Assuming also that $\omega \notin L(e_{k+1,3})$ and that r_ℓ is an r.i.d. of $G(k+1)$, $\omega \notin L(e_{k+1,5})$ iff $r_1 \in \text{Next}_{G(k+1)}(ra)$.

$$|e_{k+1,5}| = |E_k| + O(s); \text{depth}(e_{k+1,5}) = \max(\text{depth}(E_k), 1) + 1.$$

Finally,
$$E_{k+1} = \left(\bigcup_{i=0}^5 e_{k+1,i} \right).$$

To summarize the construction of E_{k+1} , assume $\omega \in \Sigma^*$ is now arbitrary. $\omega \notin L(E_{k+1})$ iff

ω is a word in form $(***)$,

and $r_{i'} = \text{init}_{k+1}(d(k,z)-4)$ for some i' , $1 \leq i' \leq \ell$,

and $r_{i+1} \in \text{Next}_{G(k+1)}(r_i)$ for all i , $1 \leq i \leq A-1$,

and $r_1 \in \text{Next}_{G(k+1)}(r_\ell)$.

But (ignoring the $\{t_i\}$), the rows of $(***)$ are therefore just $c_{k+1,j'}, c_{k+1,j'+1}, c_{k+1,j'+2}, \dots, c_{k+1,j''}$ for some j', j'' with $c_{k+1,j''+1} = c_{k+1,j'}$ (i.e. $j''+1 \equiv j' \pmod{d(k+1,z)}$).

It should now be apparent that $L(\sim E_{k+1}) = SE_{k+1}$.

The construction of F_{k+1} is analogous to that of F_1 .

$$F_{k+1} = (E_{k+1} \cup \Sigma^* \cdot (\$, \&_{k+1}, q_{k+1,0}) \cdot \Sigma^* \cdot (\$, \&_{k+1}, q_{k+1,0}) \cdot \Sigma^*).$$

Clearly $L(\sim F_{k+1}) = SF_{k+1}$.

The length and depth of E_{k+1} and F_{k+1} are given by:

$$(2\ell) \quad |E_{k+1}| < |F_{k+1}| = 2|E_k| + |F_k| + O(s);$$

$$(2d) \quad \text{depth}(E_{k+1}) = \text{depth}(F_{k+1}) = \max(\text{depth}(E_k), \text{depth}(F_k), 1) + 1.$$

The relations (1ℓ), (1d), (2ℓ), (2d) imply:

$$\begin{aligned} |E_k| &< |F_k| = O(3^k z^2 s) \\ \text{and} \qquad \qquad \qquad & \text{for } 1 \leq k \leq m. \\ \text{depth}(E_k) &= \text{depth}(F_k) = k + 1 \end{aligned}$$

Final stage m+1.

E_{m+1} is now constructed such that

$$L(E_{m+1}) \neq \Sigma^* \text{ iff } \text{Comp}_M(q_0 x b^{d(m,z)-n-2}) \neq \emptyset.$$

Recall that M accepts A within space $S(n)$, and $S(|x|) \leq g(m, z)$ by assumption. Also, $g(m, z) \leq d(m, z) - 4$ by Lemma 4.26.2.

Therefore $L(E_{m+1}) \neq \Sigma^*$ iff $x \in A$.

$$\text{We write } E_{m+1} = \left(\bigcup_{i=0}^5 e_{m+1,i} \right).$$

Construct $e_{m+1,0}$ and $e_{m+1,1}$ exactly like $e_{k+1,0}$ and $e_{k+1,1}$ above where $k = m$. Then $\omega \notin L(e_{m+1,0} \cup e_{m+1,1})$ iff ω is a word of the form $(***)$ where $k = m$.

$$|e_{m+1,0}| = |E_m| + O(s); \text{depth}(e_{m+1,0}) = \max(\text{depth}(E_m), 2).$$

$$|e_{m+1,1}| = O(s); \text{depth}(e_{m+1,1}) = 2.$$

For the remainder, of the construction of E_{m+1} , assume

$\omega \notin L(e_{m+1,0} \cup e_{m+1,1})$ and therefore that ω is a word of the form $(***)$ where $k = m$. Also let $r_1, r_2, r_3, \dots, r_\ell$ be as in $(***)$.

$e_{m+1,2}$ is constructed so that

$$\omega \notin L(e_{m+1,2}) \text{ iff } r_1 = \rho(q_0 x b^{d(m,z)-n-2}).$$

The construction is similar to and somewhat simpler than the construction of $e_{k+1,2}$ above.

Let $x = x_1 x_2 x_3 \cdots x_n$. Let $y_1, y_2, y_3, \dots, y_{n+4} \in \Sigma_{m+1}$ be such that

$$\rho(q_0 x \emptyset^{d(m,z)-n-2}) \in y_1 y_2 y_3 \cdots y_{n+2} y_{n+3}^* y_{n+4}.$$

That is, $y_1 = (\$, q_0, x_1)$, $y_2 = (q_0, x_1, x_2)$, $y_i = (x_{i-2}, x_{i-1}, x_i)$ for $3 \leq i \leq n$, $y_{n+1} = (x_{n-1}, x_n, \emptyset)$, $y_{n+2} = (x_n, \emptyset, \emptyset)$, $y_{n+3} = (\emptyset, \emptyset, \emptyset)$, $y_{n+4} = (\emptyset, \emptyset, \$)$.

For $1 \leq j \leq n+4$, let \bar{y}_j denote $(\Sigma_{m+1} - \{y_j\})$.

$$\begin{aligned} \underline{e_{m+1,2}} &= \Sigma_{m+2} \cdot (\Sigma_{\leq m})^* \cdot (\bar{y}_1 \cup y_1 \cdot (\Sigma_{\leq m})^* \cdot (\bar{y}_2 \cup y_2 \cdot (\Sigma_{\leq m})^* \cdot (\bar{y}_3 \cup \dots \\ &\quad \dots \cup y_{n+1} \cdot (\Sigma_{\leq m})^* \cdot (\bar{y}_{n+2} \cup y_{n+2} \cdot (\Sigma_{\leq m+1})^* \cdot \bar{y}_{n+3} \cdot (\Sigma_{\leq m+1})^* \\ &\quad \cdot \Sigma_{m+1} \cdot (\Sigma_{\leq m})^* \cdot \Sigma_{m+2})) \dots) \cdot \Sigma^* \\ &\cup \Sigma_{m+2} \cdot (\Sigma_{\leq m+1})^* \cdot \bar{y}_{n+4} \cdot (\Sigma_{\leq m})^* \cdot \Sigma_{m+2} \cdot \Sigma^*. \end{aligned}$$

The argument that $\omega \notin L(e_{m+1,2})$ iff $r_1 \in y_1 y_2 y_3 \cdots y_{n+2} y_{n+3}^* y_{n+4}$ is analogous to the one given above for $e_{k+1,2}$.

To bound the length of $e_{m+1,2}$, recall that " $(\Sigma_{\leq m})^*$ " abbreviates $\sim(\Sigma^* \cdot (\Sigma_{\geq m+1}) \cdot \Sigma)$, " $(\Sigma_{\leq m+1})^*$ " abbreviates $\sim(\Sigma \cdot (\Sigma_{\geq m+2}) \cdot \Sigma)$, " Σ^* " abbreviates $(\sim\# \cup \#)$, and $\Sigma_{m+2} = \{\#\}$.

Let $s' = \text{card}(\Sigma_{\geq m+1})$.

Then $|e_{m+1,2}| = O(s'n)$; $\text{depth}(e_{m+1,2}) = 2$.

Also note that only alphabet symbols from $\Sigma_{\geq m+1}$ appear within $e_{m+1,2}$. This fact is used below to obtain an improved bound on the length of $e_{m+1,2}$ after the alphabet symbols have been coded into

binary. In particular, we wish to bound the length of the coded version of $e_{m+1,2}$ by cn , where c depends on M but not on x , m , or z .

Construct $e_{m+1,3}$ and $e_{m+1,4}$ exactly like $e_{k+1,3}$ and $e_{k+1,4}$ above, where $k = m$ and $J_M(R_M)$ replaces $J_{G(k+1)}(R_{G(k+1)})$. By the discussion concerning $e_{k+1,4}$, it then follows that $\omega \notin L(e_{m+1,3} \cup e_{m+1,4})$ iff $r_{i+1} \in \text{Next}_M(r_i)$ for all i , $1 \leq i \leq \ell-1$, such that r_i is an r.i.d. of M . Of course, since $\omega \notin L(e_{m+1,2})$ ensures that r_1 is an r.i.d. of M , we conclude that $r_{i+1} \in \text{Next}_M(r_i)$ for all i , $1 \leq i \leq \ell-1$.

$$|e_{m+1,3}| = O(s); \text{ depth}(e_{m+1,3}) = 2.$$

$$|e_{m+1,4}| = |F_m| + O(s); \text{ depth}(e_{m+1,4}) = \max(\text{depth}(F_m), 1) + 1.$$

Finally $\omega \notin L(e_{m+1,5})$ iff ω contains the symbol $(\$, q_a, \emptyset)$.

(Recall the acceptance convention for STM's.)

$$e_{m+1,5} = (\Sigma - \{(\$, q_a, \emptyset)\})^*$$

$$|e_{m+1,5}| = O(s); \text{ depth}(e_{m+1,5}) = 2.$$

$$\text{Let } E_{m+1} = \left(\bigcup_{i=0}^5 e_{m+1,i} \right).$$

Assume $\omega \in \Sigma^*$ is now arbitrary. Now

$\omega \notin L(E_{m+1})$ iff ω is of the form $(***)$ where $k = m$
 and $r_1 = \rho(q_0 x \emptyset^{d(m,z)-n-2})$
 and $r_{i+1} \in \text{Next}_M(r_i)$ for $1 \leq i \leq \ell-1$
 and $(\$, q_a, \emptyset)$ appears in ω

iff M accepts x within space $d(m, z) - 2$

iff $x \in A$.

Therefore $L(E_{m+1}) \neq \Sigma^*$ iff $x \in A$.

The next Lemma 4.26.4 describes a coding of many alphabet symbols into binary in the case where \sim appears in expressions. There are of course several alternative methods of coding, some of which are simpler to prove correct than the one given here. This particular method of coding is chosen to obtain a better bound on the length of the coded $e_{m+1,2}$ as described above, and thus a better bound on the length of $E_M(x, m, z)$.

Lemma 4.26.4. There is a constant $c > 0$ such that the following holds.

Let $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_s\}$ be a finite alphabet. Let φ be one of the sets $\{U, \cdot, \sim\}$ or $\{U, \cdot, \sim, *\}$.

Define the map $h: \Sigma \rightarrow \{0, 1\}^+$ by

$$h(\sigma_i) = 10^i \quad \text{for } 1 \leq i \leq s.$$

Extend h , $h: \Sigma^* \rightarrow 2^{\{0, 1\}^*}$, in the obvious way (cf. proof of Lemma 4.10).

Let G be the $\{0, 1\}$ - $\{U, \cdot, \sim\}$ -expression

$$G = (0 \cdot (\sim 0 \cup 0) \cup (\sim 0 \cup 0) \cdot 1).$$

If E is a Σ - φ -expression, define the $\{0, 1\}$ - φ -expression $\beta(E)$ inductively by the rules:

$$\begin{aligned} \beta((\sigma)) &= (h(\sigma)) \quad \text{if } \sigma \in \Sigma \cup \{\lambda\} \\ \beta((E_1 @ E_2)) &= (\beta(E_1) @ \beta(E_2)) \\ \beta((E_1 @)) &= (\beta(E_1) @) \\ \beta((\sim E_1)) &= (\sim(\beta(E_1) \cup G)) \end{aligned} \quad \left. \vphantom{\begin{aligned} \beta((E_1 @ E_2)) &= (\beta(E_1) @ \beta(E_2)) \\ \beta((E_1 @)) &= (\beta(E_1) @) \end{aligned}} \right\} \text{ where } @ \neq \sim$$

Let $C = (10^i \mid 1 \leq i \leq s)$ be the set of code words.

Let E be an arbitrary Σ - φ -expression. (All occurrences of \sim in E denote complementation relative to Σ .)

- Then:
- (1). $L(\beta(E)) \cap C^* = h(L(E))$;
 - (2). $\text{depth}(\beta(E)) \leq \text{depth}(E) + 1$;
 - (3). $|\beta(E)| \leq cs|E|$.

Proof. (3) should be obvious by inspection. (2) is easily proved by induction on $\text{depth}(E)$.

To prove (1), let $L_0 = L(\sim G) = (A) \cup 1 \cdot \{0,1\} \cdot 0$. Note that $C \subset L_0$ and that C is a uniquely decipherable code, that is, h is one-to-one as a map from Σ^* to $\{0,1\}^*$.

Now by induction on the length of E , one can show that

- (i). $L(\beta(E)) \subseteq L_0$
- and
- (ii). $L(\beta(E)) \cap C^* = h(L(E))$.

We prove the induction step for one case. Assume (i) and (ii) hold for an expression E_1 . Let $E = (\sim E_1)$ so $\beta(E) = (\sim(\beta(E_1) \cup G))$. Assume $\omega \in \{0,1\}^*$.

$$\begin{aligned} \text{(i). } \omega \in L(\beta(E)) &\Rightarrow \omega \in L(\sim(\beta(E_1) \cup G)) \\ &\Rightarrow \omega \in L(\sim G) = L_0. \end{aligned}$$

$$\begin{aligned} \text{(ii). } \omega &\in L(\beta(E)) \cap C^* \\ \text{iff } \omega &\in C^* - L(\beta(E_1)) \quad \text{because } C^* \subset L_0 \\ \text{iff } \omega &\in C^* - h(L(E_1)) \quad \text{by induction} \\ \text{iff } \omega &\in h(L(E)) \quad \text{because } h \text{ is one-to-one on } \Sigma^*, \\ &\quad \text{and } h(R) \subseteq C^* \text{ for } R \subseteq \Sigma^*. \end{aligned}$$

The remaining cases, $E = E_1 \cup E_2$, $E = E_1 \cdot E_2$, and $E = E_1^*$, all follow in a straightforward way from the facts that C is uniquely decipherable, and that if $\omega_1, \omega_2 \in L_0$ and $\omega_1 \cdot \omega_2 \in C^*$ then $\omega_1, \omega_2 \in C^*$. \square

Returning to the proof of the main Lemma 4.26, let h and C be as in Lemma 4.26.4 for the alphabet Σ used to construct E_{m+1} . By Lemma 4.26.4(1), $L(E_{m+1}) \neq \Sigma^*$ iff $L(\beta(E_{m+1})) \cap C^* \neq C^*$.

Let $H = 0 \cdot (\sim 0 \cup 0) \cup (\sim 0 \cup 0) \cdot 1 \cup (\sim 0 \cup 0) \cdot (0^{s+1} \cup 11) \cdot (\sim 0 \cup 0)$, and note $L(H) = \{0, 1\}^* = C^*$.

Therefore

$$L(\beta(E_{m+1}) \cup H) \neq \{0, 1\}^* \text{ iff } L(E_{m+1}) \neq \Sigma^* \text{ iff } x \in A.$$

Let $E_M(x, m, z) = \beta(E_{m+1}) \cup H$.

We must now bound the depth and length of $E_M(x, m, z)$.

$$\text{depth}(E_{m+1}) = \max(\text{depth}(E_m), \text{depth}(F_m) + 1, 2) = m + 2,$$

$$\text{depth}(\beta(E_{m+1})) \leq \text{depth}(E_{m+1}) + 1 = m + 3 \quad \text{by Lemma 4.26.4(2).}$$

and thus $\text{depth}(E_M(x, m, z)) \leq m + 3$.

To bound the length of $E_M(x, m, z)$, note that:

- (i). $|E_M(x, m, z)| = |\beta(E_{m+1})| + |H| + O(1)$;
- (ii). $|H| = O(s)$;
- (iii). $|\beta(E_{m+1})| = \sum_{i=0}^5 |\beta(e_{m+1,i})| + O(1)$;
- (iv). $|e_{m+1,i}| \leq |F_m| + O(s) \quad \text{for } i \neq 2.$

Now (iv), Lemma 4.26.4(3), and the bound $|F_m| = O(3^m z^2 s)$ derived above gives

$$(v). \quad |\beta(e_{m+1,i})| = O(3^m z^2 s^2) \quad \text{for } i \neq 2.$$

It remains only to bound the length of $\beta(e_{m+1,2})$. To achieve the desired bound, assume Σ is enumerated so that

$\Sigma_{\geq m+1} = \{\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_{s'}\}$, where $s' = \text{card}(\Sigma_{\geq m+1})$ depends only on M (not on x, m , or z). Therefore $\sigma \in \Sigma_{\geq m+1}$ implies $|h(\sigma)| \leq s'+1$.

By our remarks above that $|e_{m+1,2}| = O(s'n)$ and that $e_{m+1,2}$ contains only alphabet symbols in $\Sigma_{\geq m+1}$, it is clear that

$$(vi). \quad |\beta(e_{m+1,2})| = O(s'^2 n).$$

(That is, the application of β to $e_{m+1,2}$ "expands" each alphabet symbol by at most a factor of $O(s')$, and "expands" each operation symbol by at most the fixed factor $|G|$, cf. Lemma 4.26.4.)

Combining (i), (ii), (iii), (v), and (vi) gives

$$|E_M(x, m, z)| = O(3^m z^2 s^2 + s'^2 n).$$

Finally, note that $s = s' + O(m)$ because the alphabets $\Sigma_1, \Sigma_2, \dots, \Sigma_m$ are each of fixed size.

We conclude that there is a constant a (depending only on M) such that

$$|E_M(x, m, z)| \leq a(3^m z^2 m^2 + |x|) \quad \text{for all } x, m, \text{ and } z.$$

We let the reader supply his own argument that $E_M(x, m, z)$ can be computed uniformly from x, m , and z , within time polynomial in and space linear in $|E_M(x, m, z)|$. The basic argument is by induction on k , noting that, given $\beta(E_k)$ and $\beta(F_k)$, $\beta(E_{k+1})$ and $\beta(F_{k+1})$ can be

constructed within time polynomial in and space linear in their lengths.

This completes part (1) of the proof.

(2). Operation $*$ is now available. We describe modifications to the construction just given. For all k , construct E_k' and F_k' exactly like E_k and F_k except:

(M1). If $\Theta \subseteq \Sigma$, write Θ^* as $(\theta_1 \cup \theta_2 \cup \dots \cup \theta_j)^*$ where $\Theta = \{\theta_1, \theta_2, \dots, \theta_j\}$. Now Θ is an expression of depth 0.

(M2). In the construction of E_1' , write subexpression e_{15}' as:

$$e_{15}' = \Sigma \cdot \Sigma_1^* \cdot \left(\bigcup_{a \in \Sigma_1} (\Sigma_1 - R_{G(1)}(a)) \cdot \Sigma^{z+3} \cdot (\Sigma^{z+4})^* \cdot \sigma \right) \cdot \Sigma_1^* \cdot \Sigma.$$

Note that $|e_{15}'| = 0(zs)$; $\text{depth}(e_{15}') = 0$.

We claim that if $\omega \in (\Sigma_{\geq 2} \cdot \Sigma_1^{z+3})^d \cdot \Sigma_{\geq 2}$ for some $d \geq 2$ (that is, if ω is of the form $(*)$, cf. the construction of e_{11}), then

$$\omega \in L(e_{15}') \text{ iff } \omega \in L(e_{15}).$$

To see this, assume ω is of the form $(*)$ and write

$$\omega = t_0 w_{11} w_{12} w_{13} \dots w_{1,z+3} t_1 w_2 t_2 w_3 t_3 \dots t_{\ell-1} w_{\ell 1} w_{\ell 2} w_{\ell 3} \dots w_{\ell,z+3} t_\ell$$

where $t_i \in \Sigma_{\geq 2}$ for $0 \leq i \leq \ell$, $w_i \in \Sigma_1^{z+3}$ for $2 \leq i \leq \ell-1$,

and $w_{1j}, w_{\ell j} \in \Sigma_1$ for $1 \leq j \leq z+3$.

In particular, note that $t_1, t_{\ell-1} \in \Sigma_{\geq 2}$ and recall $\Sigma_{\geq 2} \cap \Sigma_1 = \emptyset$.

It is now easy to see that

$$\omega \notin L(e_{15}') \text{ iff } w_{1j} \in R_{G(1)}(w_{\ell j}) \text{ for } 1 \leq j \leq z+3 \text{ iff } \omega \notin L(e_{15}).$$

Clearly modification (M1) does not alter the language described by an expression. In particular, since $\omega \notin L(e_{11}') \text{ iff } \omega \notin L(e_{11})$ iff ω is of the form $(*)$, it follows that $L(\sim E_1') = L(\sim E_1)$.

In general, it then follows that $L(\sim E_k') = L(\sim E_k)$ for all k .

One further modification concerns the method of coding expressions over alphabet Σ to expressions over alphabet $\{0,1\}$.

Let $h: \Sigma \rightarrow \{0,1\}$ and C be as in Lemma 4.26.4. If E is a Σ - $\{U, \cdot, \sim, *\}$ -expression, define the $\{0,1\}$ - $\{U, \cdot, \sim, *\}$ -expression $\beta'(E)$ by the rules given for $\beta(E)$ in Lemma 4.26.4 where β' replaces β and G' replaces G , where

$$G' = (0 \cdot (0 \cup 1)^* \cup (0 \cup 1)^* \cdot 1) \cdot$$

Since $L(G) = L(G')$, the proof of Lemma 4.26.4 shows that
(1'). $L(\beta'(E)) \cap C^* = h(L(E))$ for all E .

Since $\text{depth}(G') = 0$, we also have

(2). $\text{depth}(\beta'(E)) = \text{depth}(E)$ for all E .

Now let $E_M'(x, m, z) = \beta'(E_{m+1}') \cup H$.

By the argument above that $L(E_{m+1}') = L(E_{m+1})$ and by (1') we conclude that $L(E_M'(x, m, z)) = L(E_M(x, m, z))$.

The new bounds on the depth and length of $E_M'(x, m, z)$ are as follows.

From (M1) and (2'):

$$\text{depth}(E_1') = \text{depth}(F_1') = 0 ;$$

$$\text{depth}(E_{k+1}') = \text{depth}(F_{k+1}') = \max(\text{depth}(E_k'), \text{depth}(F_k')) + 1 = k$$

for $1 \leq k \leq m$;

and $\text{depth}(E_M'(x, m, z)) = \text{depth}(E_{m+1}') = m$ (provided $m \geq 1$).

Modification (M2) gives $|E_1'| < |F_1'| = O(s z)$. The relation (28) derived earlier remains the same, and therefore

$$|E_m'| < |F_m'| = O(3^m s z).$$

It can now be checked that $|E_{m+1}'| = O(3^{m+1} s z + s n)$ and therefore

$$\begin{aligned} |E_m'(x, m, z)| &= O(s(3^m s z + s n)) \\ &\leq a(3^m z^2 + m^2 |x|) \end{aligned}$$

for some constant a depending only on M

This completes the proof of Lemma 4.26. □

Section 4.2 closes with two open questions **concerning** the complexity of inequivalence problems with \sim . The first concerns the gap between known lower and upper bounds for $\text{NEC}(\{0,1\}, \{U, \cdot, \sim\})$.

open Question 4.33. Precisely where between $g(\lceil \log_b n \rceil, 0)$ and $g(n, 0)$ does the space requirement for $\text{NEC}(\{0,1\}, \{U, \cdot, \sim\})$ lie?

In particular, is $\text{NSPACE}(g(n, 0)) \leq \text{NEC}(\{0,1\}, \{U, \cdot, \sim\})$?

In the proof of Lemma 4.26 we essentially use three occurrences of expressions for the $g(k, 0)$ "ruler" to construct **expressions** for the

$g(k+1,0)$ "ruler". Thus the size of the expression for the $g(k,0)$ "ruler" grows exponentially in k and we obtain only a $g(\log_b n, 0)$ lower bound on the complexity of $NEC(\{0,1\}, \{U, \cdot, \sim\})$. The lower bound could be raised to $g(cn, 0)$ for some constant c , thereby settling Open Question 4.33, if one could construct an appropriate $g(k+1,0)$ "ruler" using only one copy of a $g(k,0)$ "ruler". Some of the logical theories mentioned in Chapter 5 contain enough notational power that only one occurrence of the formula corresponding to a $g(k,0)$ ruler is required to obtain a $g(k+1,0)$ ruler and so one can obtain $g(cn, 0)$ lower bounds on their complexity. However, for the case of regular-like expressions using U , \cdot , and \sim , or even allowing $*$ as well, we are unable to settle Open Question 4.33.

For technical completeness, we would like to show that no two out of three of the operations U, \cdot, \sim yield a nonelementary inequivalence problem. We know by Theorem 4.27 that $INEQ(\{0,1\}, \{U, \cdot, \sim\})$ is nonelementary. The complexity of $INEQ(\Sigma, \{U, \cdot\})$ (where $\text{card}(\Sigma) \geq 2$) is characterized by Theorem 4.19 as being precisely NP which is certainly elementary. Also, it is easy to see that $INEQ(\Sigma, \{U, \sim\}) \in P$. If E is a $\Sigma - \{U, \sim\}$ -expression then either $L(E) = \emptyset$ or $L(E) = \Sigma^* - \emptyset$ for some $\emptyset \subseteq \Sigma$. Moreover,

such a description of $L(E)$ can be obtained deterministically within time polynomial in the length of E . The case $\{\cdot, \sim\}$ is open.

Open Question. Characterize the complexity of $INEQ(\Sigma, \{\cdot, \sim\})$ for $\text{card}(\Sigma) \geq 2$. In particular, is it elementary-recursive?²

4.4 Expressions Over a One-Letter Alphabet.

We have seen in previous sections that the complexity of $\text{INEQ}(\Sigma, \varphi)$ or $\text{NEC}(\Sigma, \varphi)$ for a particular φ does not depend significantly on Σ provided $\text{card}(\Sigma) \geq 2$ (cf. Lemma 4.10). This section shows that the complexity of a problem can be affected, sometimes drastically, by the restriction to a one-letter alphabet.

This is best illustrated by the case $\varphi = \{U, \cdot, \sim\}$. The results of section 4.2 show that $\text{INEQ}(\Sigma, \{U, \cdot, \sim\})$ is not elementary-recursive if $\Sigma = \{0, 1\}$. However, this problem becomes relatively trivial if $\Sigma = \{0\}$.

Theorem 4.37.

$$\text{INEQ}(\{0\}, \{U, \cdot, \sim\}) \in P.$$

Proof. The proof rests on the fact that a $\{0\}$ - $\{U, \cdot, \sim\}$ -expression E describes either a finite or cofinite set of words, and moreover that all words $\omega \in \{0\}^*$ of length exceeding $|E|$ are either all not in $L(E)$ or all in $L(E)$. That is:

Lemma 4.37.1. Let E be a $\{0\}$ - $\{U, \cdot, \sim\}$ -expression. Then either

- (i). (finite) For all $\omega \in \{0\}^*$, $\omega \in L(E) \Rightarrow |\omega| \leq |E|$,
 or
 (ii). (cofinite) For all $\omega \in \{0\}^*$, $\omega \notin L(E) \Rightarrow |\omega| \leq |E|$.

Proof. By induction on the length of E .

If $E = (0)$ or $E = (\lambda)$, the lemma is certainly true.

Suppose the lemma is true of expressions E_1 and E_2 .

If $E = (\sim E_1)$, then $\omega \in L(E)$ iff $\omega \notin L(E_1)$, $|E_1| < |E|$, and thus the lemma is true of E .

Suppose $E = (E_1 \cdot E_2)$. First suppose $L(E_1)$ and $L(E_2)$ are both finite. Then $L(E)$ is finite, and $\omega \in L(E)$ implies $\omega = \omega_1 \omega_2$ for some $\omega_1 \in L(E_1)$ and $\omega_2 \in L(E_2)$. Since $|\omega_1| \leq |E_1|$ and $|\omega_2| \leq |E_2|$ by induction, $|\omega| \leq |E_1| + |E_2| < |E|$. Now suppose $L(E_1)$ is cofinite and $L(E_2)$ is finite. If $L(E_2) = \emptyset$, then $L(E) = \emptyset$ and the lemma is true of E . If $L(E_2) \neq \emptyset$, then $0^k \in L(E_2)$ for some $k \leq |E_2|$ by induction. Also by induction, $z > |E_1|$ implies $0^z \in L(E_1)$ for all integers z . Therefore $z > |E_1| + k$ implies $0^z \in L(E_1) \cdot L(E_2) = L(E)$. But $|E_1| + k \leq |E_1| + |E_2| < |E|$, and thus the lemma is true of E . The case in which both $L(E_1)$ and $L(E_2)$ are cofinite is handled similarly.

The reader can check the case $E = (E_1 \cup E_2)$ in a similar fashion. This completes the proof of Lemma 4.37.1. \square

Thus if E is a $\{0\}$ - $\{\cup, \cdot, \neg\}$ -expression, $L(E)$ has a finite representation of the form $[F, t]$, where $F \subset \mathbb{N}$, F is finite, $t \in \{0, 1\}$,

$$[F, t] \text{ represents } \begin{cases} \{0^z \mid z \in F\} & \text{if } t = 0 \\ \{0^z \mid z \notin F\} & \text{if } t = 1, \end{cases}$$

and either $\max(F) \leq |E|$ or $F = \emptyset$.

Also it is not hard to see that, given finite representations for $L(E_1)$ and $L(E_2)$, a deterministic algorithm can find a finite representation for $L(E_1 \cup E_2)$, $L(E_1 \cdot E_2)$, or $L(\sim E_1)$ within time bounded by

a fixed polynomial in $|E_1| + |E_2|$. Therefore, using this algorithm recursively, the time required to find a finite representation of $L(E)$ is bounded above by $T(|E|)$ where

$$T(n) = \max\{ T(n_1) + T(n_2) \mid n_1, n_2 > 0 \text{ and } n_1 + n_2 < n \} + p(n)$$

where $p(n)$ is a polynomial.

Therefore $T(n) = O(n \cdot p(n))$ assuming (without loss of generality)

that $p(n_1) + p(n_2) \leq p(n)$ for all $n_1, n_2 > 0$ with $n_1 + n_2 < n$.

Also, a deterministic algorithm can check that two finite representations describe different sets of words within polynomial time.

The first step of the main algorithm, checking that x is of the form (E_1, E_2) where E_1 and E_2 are $\{0\}$ - $\{U, \cdot, \sim\}$ -expressions, can be done deterministically within time $O(|x|^3)$ [cf, You67].

The various pieces can be put together to give a deterministic polynomial time acceptance algorithm for $INEQ(\{0\}, \{U, \cdot, \sim\})$. \square

For another φ , $NEC(\{0\}, \varphi)$ is complete in a class which may lie strictly above P . The inequivalence problem for regular expressions ($\varphi = \{U, \cdot, *\}$) over a one-letter alphabet is \leq -complete in NP. (Recall that in the two-letter case, $NEC(\{0, 1\}, \{U, \cdot, *\})$ is \leq_{\log} -complete in POLYSPACE (cf. Remark 4.14(3)).)

Theorem 4.38. $NEC(\{0\}, \{U, \cdot, *\})$ is \leq -complete in NP.

We **omit** the proof of Theorem 4.38. A proof can be found in [SM73].

Chapter 5. Nonelementary Logical Theories

By using efficient reducibility techniques, several workers [Mey73], [FR74], [Rob73] have obtained lower bounds on the complexities of decision problems for certain decidable logical theories. In fact, the first example of a **provably** difficult natural decision problem was provided by Meyer [Mey73] who showed that the decision problem for the weak monadic second order theory of successor (WS1S) is not elementary-recursive. Subsequently, Robertson [Rob73] showed that the satisfiability problem for sentences in the first order language of the nonnegative integers with \leq and a single uninterpreted monadic predicate is not elementary-recursive. The purpose of this chapter is to show that these two results and others follow as simple corollaries of the result that the emptiness problem for star-free expressions is not elementary-recursive (cf. §4.2).

To simplify notation in this chapter:

A star-free expression is a $\{0,1\}$ - $\{U,\cdot,\sim\}$ -expression;

$$\begin{aligned} \underline{NE(\text{star-free})} = \{ E \mid E \text{ is a star-free expression} \\ \text{and } L(E) \neq \emptyset \} . \end{aligned}$$

Note that $E \in NEC(\{0,1\},\{U,\cdot,\sim\})$ iff $(\sim E) \in NE(\text{star-free})$.

The next fact is now **immediate** from Theorem 4.27.

Fact 5.1. For all rational $b > 3$, $\text{NE}(\text{star-free}) \notin \text{NSPACE}(g(\lceil \log_b n \rceil, 0))$.

In particular, $\text{NE}(\text{star-free})$ is not elementary-recursive.

In this chapter we consider several decision problems concerning restricted forms of symbolic logic such as the two mentioned in the opening paragraph. In each case we show that $\text{NE}(\text{star-free})$ is efficiently (in particular $\leq_{\text{p}\ell}$) reducible to the particular decision problem, and thus that these decision problems are not elementary-recursive.

The main advantage of obtaining such results as corollaries of Fact 5.1 (rather than by a direct arithmetization of Turing machines) is simplicity. In the cases we consider, there is a simple, easily described transformation from $\text{NE}(\text{star-free})$ to the particular decision problem, and so we may avoid repeating for each decision problem the arithmetization of Turing machines which we have already carried out in terms of star-free expressions.

WSIS can also play the role of $\text{NE}(\text{star-free})$ as a starting point for further reductions. However, for several particular theories T , we know of no direct transformation from WSIS to the decision problem for T^\dagger , even though there is a simple transformation from $\text{NE}(\text{star-free})$ to T . Intuitively, $\text{NE}(\text{star-free})$ succeeds where WSIS fails because WSIS is a considerably richer language than the language of star-free

[†]In certain cases, the only known efficient transformation from WSIS to T involves first taking a decision procedure (Turing machine) M for WSIS and then arithmetizing M in the language of T .

expressions; in the language of star-free expressions there is no direct analogue of logical quantifiers or variables.

• A disadvantage of obtaining such results as corollaries of the star-free result is that (in **the cases** we consider) the implied lower complexity bound is somewhat weaker than the bound which can be obtained by a direct arithmetization. Since space $g(\lceil \log_b n \rceil, 0)$ is the best known lower bound on the complexity of $NE(\text{star-free})$, space $g(\lceil \log_b n \rceil, 0)$ is the best lower bound one can obtain on a set B by a transformation f from $NE(\text{star-free})$ to B , assuming $|f(x)| \geq |x|$ for all x . However, as was first pointed out by Rabin[†] for $WS1S$, and then by Meyer for the satisfiability problem for sentences in the first order theory of linear order, one can show that these problems require space $g(\lceil cn \rceil, 0)$ for some $c > 0$. This lower bound is closer to known upper bounds of $g(\lceil dn \rceil, 0)$ for some constant d , [Buc60a], [Elg61], [Rab69]. Of course, if one wants only to show that a certain decision problem is not elementary-recursive, then an efficient transformation from $NE(\text{star-free})$ to the problem is sufficient.

We **assume** the reader is familiar **with** the basic notions of the predicate calculus, (see for example [Sho67]).

Let $L(<, P)$ be the set of formulas written in first order predicate calculus using only the binary relational symbol $<$ and the monadic predicate symbol P , together with the usual logical connectives $\wedge, \vee, \sim, \Rightarrow$, etc., quantifiers \exists and \forall , variables, and parentheses.

[†]Personal communication.

We shall use other relational symbols such as \leq and $=$ in writing formulas since these can be expressed in terms of $<$ by formulas of fixed size; for example $(x = y)$ iff $\sim((x < y) \vee (y < x))$. Lower case Roman letters are used to denote first order variables. (Variables may in general be subscripted by a binary number, although the particular formulas we shall write require only a fixed (approximately 8) number of variables.)

A formula F is a sentence if F contains no free variables.

Let S be a set and let $<_S$ be a linear (i.e. total) order on S . Let φ be a sentence in $L(<, \underline{P})$. φ is satisfiable with respect to $(S, <_S)$ iff there is an interpretation $\underline{P}S \rightarrow \{0,1\}^t$ of \underline{P} such that φ is true under the interpretation $(S, <_S, \underline{P})$. Let SAT $(S, <_S)$ be the set of all such satisfiable sentences.

The main result is that if S is an infinite set with linear order $<_S$, then $NE(\text{star-free}) \leq_{p\ell} SAT(S, <_S)$ and hence $SAT(S, <_S)$ is not elementary-recursive.

Remark: The first order theories of $(\mathbb{N}, <)$, $(\mathbb{Q}, <)$, and various other orders without a predicate \underline{P} are all elementary-recursive [cf. Fer74].

Before proving the general result, it is instructive to prove a somewhat simpler special case, namely $S = \mathbb{N}$ (the nonnegative integers) and $<_S = <$ (the usual relation "less than" on integers). Decidability of $SAT(\mathbb{N}, <)$ follows from [Buc60b]. The result that $SAT(\mathbb{N}, <)$ is not

^tView 1 as "true" and 0 as "false".

elementary-recursive was obtained independently by Robertson [Rob73] using a direct arithmetization.

Theorem 5.2.

- (1). $NE(\text{star-free}) \leq_{p\ell} SAT(N, <).$
- (2). Therefore $SAT(N, <)$ is not elementary-recursive, and in fact $SAT(N, <) \not\leq NSPACE(g(\lceil \log_b n \rceil, 0))$ for all $b > 3$.

Proof. (1). Given a star-free expression E , we construct a formula with two free variables $F_E(x, y) \in L(<, \underline{P})$ such that:

(*) If $P: N \rightarrow \{0, 1\}$ and $i, j \in N$, then $F_E(i, j)$ is true under the interpretation $(N, <, P)$ iff

- (i). $i = j$ and $P(i)P(i+1)P(i+2) \cdots P(j-1) \in L(E)$
- or
- (ii). $i < j$ and $\lambda \in L(E).$

$F_E(x, y)$ is constructed inductively on the structure of E :

$$F_{(\lambda)}(x, y) \text{ is } (x = y) ;$$

$$F_{(0)}(x, y) \text{ is } ("y = x + 1" \wedge \sim \underline{P}(x)) ;$$

$$F_{(1)}(x, y) \text{ is } ("y = x + 1" \wedge \underline{P}(x)) ;$$

where $("y = x + 1")$ abbreviates $((x < y) \wedge \sim (\exists z)(x < z < y))$.

Inductively, if E and E' are star-free expressions then:

$$F_{(E \cup E')}(x, y) \text{ is } (F_E(x, y) \vee F_{E'}(x, y)) ;$$

$$F_{(E \cdot E')}(x, y) \text{ is } (\exists z)(F_E(x, z) \wedge F_{E'}(z, y)) ;$$

$$F_{(\sim E)}(x, y) \text{ is } ((x \leq y) \wedge \sim F_E(x, y)).$$

By renaming variables appropriately, note that $F_E(x, y)$ can be written **using** exactly three variables. It is also easy to prove by induction that $F_E(x, y)$ has the property (*) above for all star-free expressions E .

Now let φ_E be the sentence

$$\varphi_E = (\exists x)(\exists y)(F_E(x, y)).$$

Then clearly $E \in \text{NE}(\text{star-free})$ iff $\varphi_E \in \text{SAT}(\mathbb{N}, <)$.

Let f be the function mapping E to φ_E for all E . Clearly f can be computed within polynomial time and linear space, and f is linear bounded. (To be completely precise, $f(x)$ must also be defined if x is not a well-formed star-free expression. However an IOTM computing f can first check within space $\log n$ that x is well-formed, and output some ill-formed or false sentence if not.)

(2). This is now immediate by Fact 5.1 and Lemma 3.7. □

A transformation similar to that of Theorem 5.2 can be used to embed $\text{NE}(\text{star-free})$ in the language of certain weak monadic second order theories of \mathbb{N} . For **example**, let WS1S be the set of true sentences written in weak monadic second order logic using only the predicates $y = x+1$ (y is the successor of x) and $x \in X$. [Mey73]

shows that WS1S is not elementary-recursive. This also follows easily from Fact 5.1.

Theorem 5.3. $\text{NE}(\text{star-free}) \leq_{\text{pl}} \text{WS1S}.$

Therefore WS1S is not elementary-recursive.

Proof. A formula $F_E(x, y, \underline{p})$ is constructed to satisfy property (*) of Theorem 5.2, where \underline{p} is now viewed as a finite set variable.

$$\begin{aligned} F_{(\lambda)}(x, y, \underline{p}) & \text{ is } (x = y) ; \\ F_{(0)}(x, y, \underline{p}) & \text{ is } ((y = x+1) \wedge \sim(x \in \underline{p})) ; \\ F_{(1)}(x, y, \underline{p}) & \text{ is } ((y = x+1) \wedge (x \in \underline{p})). \end{aligned}$$

$F_{(E \cup E')}$, $F_{(E \cdot E')}$, and $F_{(\sim E)}$ are written as in Theorem 5.2 where " $(x \leq y)$ " is expressed by the formula

$$(\exists A)((x \in A) \wedge \sim(y+1 \in A) \wedge (\forall z)(z+1 \in A \Rightarrow z \in A)).$$

As before, $F_E(x, y, \underline{p})$ can be written using a fixed number of variables.

Finally, if $E = (\exists \underline{p})(\exists x)(\exists y)(F_E(x, y, \underline{p}))$ then

$$E \in NE(\text{star-free}) \text{ iff } \varphi_E \in \text{WS1S}. \quad \square$$

$k \geq 1$,

Remark. If $\text{depth}(E) = k$, then φ_E of Theorem 5.3 is transformable within polynomial time to a sentence φ'_E in prenex normal form with $k-1$ alternations of set quantifiers. Also, from Theorem 4.29 it follows that, for any $k \geq 1$, $\text{NSPACE}(g(k, n)) \leq NE(\text{star-free}) \cap \{ E \mid \text{depth}(E) \leq k+4 \}$.

Therefore, for $k \geq 1$, $\text{NSPACE}(g(k, n))$ is transformable within polynomial time to WS1S restricted to prenex sentences with at most $k+3$ alternations of set quantifiers. By a direct proof, Robertson [Rob73] has obtained the stronger result that, for $k \geq 2$, $\text{NSPACE}(g(k, n))$ is transformable within polynomial time to prenex sentences with at most $k-1$ alternations of set quantifiers.

Similarly in Theorem 5.2 one can relate the complexity of deciding $\text{SAT}(\mathbb{N}, <)$ to the number of alternations of first order quantifiers.

We now turn to the main result of this section, that $\text{NE}(\text{star-free})$ is efficiently reducible to $\text{SAT}(S, <)$ for an arbitrary infinite set S with linear order $<$. Of course $\text{SAT}(S, <)$ may not be decidable for certain choices of S and $<$. However, whatever the upper complexity bound, $\text{SAT}(S, <)$ is never elementary-recursive.

It should first be pointed out that the simple transformation of Theorem 5.2 does not work for general S . This is illustrated by choosing $(S, <) = (Z^*, <_*)$ where $Z^* = Q \times Z$ and

$$(q_1, z_1) <_* (q_2, z_2) \text{ iff either } (q_1 < q_2) \text{ or } (q_1 = q_2 \text{ and } z_1 < z_2)$$

for $q_1, q_2 \in Q, z_1, z_2 \in Z$.

Now let E be a particular star-free expression which describes the set of words which "start with 0" and "end with 1" and "do not contain 01 as a subword". That is

$$E = \sim(\sim(0 \cdot (\sim 0 \cup 0)) \cup \sim((\sim 0 \cup 0) \cdot 1) \cup (\sim 0 \cup 0) \cdot 01 \cdot (\sim 0 \cup 0)) \cdot$$

(Recall $L((\sim 0 \cup 0)) = \{0, 1\}$.)

Certainly $L(E) = \emptyset$ and therefore $E \notin \text{NE}(\text{star-free})$. However letting $F_E(x, y)$ and $\varphi_E = (\exists x)(\exists y)(F_E(x, y))$ be as in Theorem 5.2, we claim that $\varphi_E \in \text{SAT}(Z^*, <_*)$. To see this, choose (for example)

$$P(q, z) = \begin{cases} 0 & \text{if } q \leq 0 \\ 1 & \text{if } q > 0 \end{cases} \quad \text{for all } q \in Q, z \in Z.$$

[†] Q denotes the rational numbers. Z denotes the integers.

It is now straightforward to verify that $F_E((0,0), (1,0))$ is true under the interpretation (Z, \leq_*, P) and therefore $\varphi_E \in SAT(Z, \leq_*)$. (Informally, the infinite word $P(0,0)P(0,1)P(0,2)\dots\dots P(1,-1)P(1,0) = 000\dots\dots 111$ correctly starts with 0 and ends with 1 and yet doesn't contain 01 as a subword.)

The proof that $L(E) \neq \emptyset$ iff $\varphi_E \in SAT(N, \leq)$ implicitly uses the property of N that for all $i, j \in N$ there are at most finitely many $k \in N$ such that $i < k < j$. This property does not hold for other sets such as Z causing the difficulty illustrated above. However this difficulty can be overcome by a modification to the transformation of Theorem 5.2.

Fix a particular infinite set S with linear order $<$. The first step utilizes the predicate P to pick out a set of discrete "points" from the (possibly dense) set S . The formula $point(x)$ is satisfied by an interpretation of x and \underline{P} iff \underline{P} is identically false on some open interval below x and is identically true on some interval above x . The truth value of $\underline{P}(x)$ under the interpretation is not constrained by $point(x)$.

$$point(x) \text{ is } (3s)(\exists t)(\forall w) ((s < x < t) \wedge ((s < w < x) \Rightarrow \sim \underline{P}(w)) \wedge ((x < w < t) \Rightarrow \underline{P}(w))).$$

Let $nextpt(x, y)$ be the following formula which is satisfied by an interpretation of x, y , and \underline{P} iff x and y are "points" and y is the next point after x .

$\text{nextpt}(x,y)$ is $(\text{point}(x) \wedge \text{point}(y) \wedge (\forall z)((x < z < y) \Rightarrow \neg \text{point}(z)))$.

Let $P:S \rightarrow \{0,1\}$ be a given interpretation of \underline{P} .

Define $\text{Points}(P) = \{ x \in S \mid \text{point}(x) \}$.

If $p_1, p_2 \in \text{Points}(P)$, we say that p_1 and p_2 are finitely far apart iff $\text{card}\{ w \mid p_1 < w < p_2 \text{ and } w \in \text{Points}(P) \}$ is finite.

If $x, y \in \text{Points}(P)$, $x < y$, and x and y are finitely far apart, define $\text{word}_P(x,y) = P(x_0)P(x_1)P(x_2) \cdots P(x_\ell)$ where $x_0 = x$, $\text{nextpt}(x_\ell, y)$, and $\text{nextpt}(x_{i-1}, x_i)$ for $1 \leq i \leq \ell$.

Define $\text{word}_P(x,x) = \lambda$ for all $x \in \text{Points}(P)$.

Lemma 5.4. For any star-free expression E there is a formula $F_E(x,y,u,v)$ in $L(<, \underline{P})$ with the following properties.

Let $\mathcal{I}:S \rightarrow \{0,1\}$ be any interpretation of \underline{P} .

(i). For all $s_1, s_2, s_3, s_4 \in S$, $FE(s_1, s_2, s_3, s_4)$ is true (under the interpretation $(S, <, \mathcal{I})$) only if $s_1, s_2, s_3, s_4 \in \text{Points}(P)$ and $s_1 \leq s_2 \leq s_3 \leq s_4$.

(ii). If $p_1, p_2, p_3, p_4 \in \text{Points}(P)$, $p_1 \leq p_2 \leq p_3 \leq p_4$, p_1 and p_2 are finitely far apart, and p_3 and p_4 are finitely far apart, then

$F_E(p_1, p_2, p_3, p_4)$ is true iff $\text{word}_P(p_1, p_2) \cdot \text{word}_P(p_3, p_4) \in L(E)$.

(iii). If $p_1, p_2, p_4 \in \text{Points}(P)$ and $p_1 \leq p_2 \leq p_4$, then

$F_E(p_1, p_2, p_2, p_4)$ is true iff $F_E(p_1, p_4, p_4, p_4)$ is true.

(iv). Moreover there is a linear bounded function $f \in \text{polylin}$ such that $f(E) = F_E$ for all star-free expressions E .

Proof. F_E is defined inductively.

Let $\text{point}(x, y, u, v) = (\text{point}(x) \wedge \text{point}(y) \wedge \text{point}(u) \wedge \text{point}(v))$

$F_{(\lambda)}(x, y, u, v)$ is $(\text{point}(x, y, u, v) \wedge (x = y) \wedge (u = v) \wedge (x \leq u))$;

$F_{(0)}(x, y, u, v)$ is $(\text{point}(x, y, u, v) \wedge (y \leq u) \wedge ((\text{nextpt}(x, y) \wedge (u = v) \wedge \sim \underline{P}(x)) \vee (\text{nextpt}(u, v) \wedge (x = y) \wedge \sim \underline{P}(u)))$);

$F_{(1)}(x, y, u, v)$ is similar to $F_{(0)}(x, y, u, v)$;

$F_{(E \cup E')}(x, y, u, v)$ is $(F_E(x, y, u, v) \vee F_{E'}(x, y, u, v))$;

$F_{(E \cdot E')}(x, y, u, v)$ is $((\exists z)(F_E(x, z, z, z) \wedge F_{E'}(z, y, u, v)) \vee (\exists z)(F_E(x, y, u, z) \wedge F_{E'}(z, v, v, v)))$;

$F_{(\sim E)}(x, y, u, v)$ is $(\text{point}(x, y, u, v) \wedge (x \leq y \leq u \leq v) \wedge \sim F_E(x, y, u, v))$.

The assertions (i), (ii), and (iii) all follow by straightforward inductive proofs.

For example, one part of the inductive step for (iii) is as follows. Assume (iii) is true for expressions E and E' .

Assume $p_1, p_2, p_4 \in \text{Points}(P)$ and $p_1 \leq p_2 \leq p_4$. Then

$$\begin{aligned}
 F_{(E \cdot E')}(p_1, p_2, p_2, p_4) & \text{ iff } ((\exists z)(F_E(p_1, z, z, z) \wedge F_{E'}(z, p_2, p_2, p_4)) \\
 & \quad \vee (\exists z)(F_E(p_1, p_2, p_2, z) \wedge F_{E'}(z, p_4, p_4, p_4))) \\
 & \text{ iff } (\exists z)(F_E(p_1, z, z, z) \wedge F_{E'}(z, p_4, p_4, p_4)) \\
 & \quad \text{by induction} \\
 & \text{ iff } ((\exists z)(F_E(p_1, z, z, z) \wedge F_{E'}(z, p_4, p_4, p_4)) \\
 & \quad \vee (\exists z)(F_E(p_1, p_4, p_4, z) \wedge F_{E'}(z, p_4, p_4, p_4))) \\
 & \quad \text{(because by part (i), the second disjunct} \\
 & \quad \text{implies the first)} \\
 & \text{ iff } F_{(E \cdot E')}(p_1, p_4, p_4, p_4) \quad \text{by definition.}
 \end{aligned}$$

The remaining cases are easier and are left to the reader. \square

Let $P:S \rightarrow \{0,1\}$, let E be a star-free expression, and let $p_1, p_2, p'_1, p'_2 \in \text{Points}(P)$ with $p_1 \leq p_2$ and $p'_1 \leq p'_2$.

Define $(p_1, p_2) \equiv_{P,E} (p'_1, p'_2)$ iff $(F_E(p_1, p_2, u, v) \Leftrightarrow F_E(p'_1, p'_2, u, v))$ is true for all $u, v \in S$.

Note that $\equiv_{P,E}$ is an equivalence relation on

$$\hat{S} = \{ (p_1, p_2) \in S \times S \mid p_1, p_2 \in \text{Points}(P) \text{ and } p_1 \leq p_2 \}.$$

Let $\text{index}(\equiv_{P,E})$ be the index (number of equivalence classes) of $\equiv_{P,E}$.

Lemma 5.5. For all $P:S \rightarrow \{0,1\}$ and all star-free expressions E , $\text{index}(\equiv_{P,E})$ is finite,

Proof. Fix some $P:S \rightarrow \{0,1\}$, and abbreviate $\equiv_{P,E}$ as \equiv_E . We prove by induction that $\text{index}(\equiv_E)$ is finite. The proof is **similar** to

Brzozowski's proof that any extended regular expression has a finite number of types of derivatives [Brz64].

If $E = (\lambda)$ or (0) or (1) it is trivial to check that $\text{index}(\equiv_E)$ is finite.

Let E and E' be star-free expressions with $\text{index}(\equiv_E) = n$ and $\text{index}(\equiv_{E'}) = n'$. From the inductive definition of $F_E(x, y, u, v)$ we have for all $(p_1, p_2), (p'_1, p'_2) \in \hat{S}$:

(1). If $(p_1, p_2) \equiv_E (p'_1, p'_2)$ then $(p_1, p_2) \equiv_{\sim E} (p'_1, p'_2)$.

Therefore $\text{index}(\equiv_{\sim E}) \leq n$.

(2). If $(p_1, p_2) \equiv_E (p'_1, p'_2)$ and $(p_1, p_2) \equiv_{E'} (p'_1, p'_2)$, then $(p_1, p_2) \equiv_{E \cup E'} (p'_1, p'_2)$.

Therefore $\text{index}(\equiv_{E \cup E'}) \leq nn'$.

(3). Let $C_1, C_2, C_3, \dots, C_n \subseteq \hat{S}$ be the equivalence classes of \equiv_E .

If $(x, y) \in \hat{S}$ define

$$\text{Classes}(x, y) = \{ i \mid (\exists z) [x \leq z \leq y \text{ and } F_E(x, z, z, z) \text{ and } (z, y) \in C_i] \}.$$

Now if $(p_1, p_2) \equiv_E (p'_1, p'_2)$ and $\text{Classes}(p_1, p_2) = \text{Classes}(p'_1, p'_2)$, then $(p_1, p_2) \equiv_{E \cdot E'} (p'_1, p'_2)$. Therefore $\text{index}(\equiv_{E \cdot E'}) \leq n2^{n'}$.

(1), (2), and (3) are easy to verify from the definition of F_E .

We sketch the verification of (3). Let $u, v \in S$.

$$F_{E \cdot E'}(p_1, p_2, u, v) \text{ is true iff } ((\exists z)(F_E(p_1, z, z, z) \wedge F_{E'}(z, p_2, u, v)) \vee (\exists z)(F_E(p_1, p_2, u, z) \wedge F_{E'}(z, v, v, v))).$$

But $F_E(p_1, p_2, u, z)$ is true iff $F_E(p_1^i, p_2^i, u, z)$ is true because

$$(p_1, p_2) \equiv_E (p_1^i, p_2^i).$$

Also, $(\exists z)(F_E(p_1, z, z, z) \wedge F_{E'}(z, p_2, u, v))$ is true
iff $(\exists z)(F_E(p_1', z, z, z) \wedge F_{E'}(z, p_2', u, v))$ is true
because $\text{Classes}(p_1, p_2) = \text{Classes}(p_1', p_2')$.

It follows that $F_{E \cdot E'}(p_1, p_2, u, v)$ is true
iff $F_{E \cdot E'}(p_1', p_2', u, v)$ is true. \square

Theorem 5.6. Let S be an infinite set with linear order $<$,

$$\text{NE}(\text{star-free}) \leq \text{SAT}(S, <).$$

Therefore $\text{SAT}(S, <)$ is not elementary-recursive, and in fact

$$\text{SAT}(S, <) \notin \text{NSPACE}(g(\lceil \log_b n \rceil, 0)) \text{ for all } b > 3.$$

Proof. Let φ_E be the sentence

$$\begin{aligned} \varphi_E = (\exists x)(\exists y)(& F_E(x, y, y, y) \\ & \wedge (\forall z)(\forall z')((x \leq z < z' \leq y) \Rightarrow \sim F_E(x, z, z', y)) \quad). \end{aligned}$$

We claim that $E \in \text{NE}(\text{star-free})$ iff $\varphi_E \in \text{SAT}(S, <)$.

(only if). Let $\omega \in L(E)$ be a shortest word in $L(E)$; that is,
for all $\omega' \in \{0, 1\}^*$, $|\omega'| < |\omega|$ implies $\omega' \notin L(E)$.

Since S is infinite, we can choose $P: S \rightarrow \{0, 1\}$ and $x, y \in \text{Points}(P)$
such that x and y are finitely far apart and $\text{word}_P(x, y) = \omega$. Therefore
 $F_E(x, y, y, y)$ is true by Lemma 5.4(ii).

Choose any $z, z' \in \text{Points}(P)$ with $x \leq z < z' \leq y$. Since $z < z'$,
 $|\text{word}_P(x, z) \cdot \text{word}_P(z', y)| < |\text{word}_P(x, y)| = |\omega|$. Again by Lemma 5.4(ii),
and since ω is a shortest word in $L(E)$, we have that $F_E(x, z, z', y)$ is
false. By Lemma 5.4(i), $F_E(x, z, z', y)$ is also false if $z \notin \text{Points}(P)$
or $z' \notin \text{Points}(P)$.

Thus φ_E is true under the interpretation $(S, <, P)$.

(if). Let $P: S \rightarrow \{0, 1\}$ be such that φ_E is true under $(S, <, P)$.

Therefore there are points x and y such that $F_E(x, y, y, y)$ and $(\forall z)(\forall z')((x \leq z < z' \leq y) \Rightarrow \sim F_E(x, z, z', y))$.

Suppose x and y are not finitely far apart. Then since $\text{index}(\equiv_{P, E})$ is finite, there must be $z, z' \in \text{Points}(P)$ such that $x \leq z < z' \leq y$ and $(x, z) \sqsubset_{P, E} (x, z')$. Now

$F_E(x, z, z', y)$ is true

iff $F_E(x, z', z', y)$ is true (by definition of $\equiv_{P, E}$)

iff $F_E(x, y, y, y)$ is true (by Lemma 5.4(iii)).

Therefore $F_E(x, z, z', y)$ is true contrary to assumption.

It follows that x and y are finitely far apart and thus $\text{word}_P(x, y) \in L(E)$. □

*

For example, $\text{SAT}(\mathbb{Z}, <)$, $\text{SAT}(\mathbb{Q}, <)$, and $\text{SAT}(\mathbb{Z}, <_*)$ are not elementary-recursive.

A related decision problem is the satisfiability problem for sentences in the first order theory of linear order. Let $L(<)$ be the set of formulas written in first order predicate calculus using only the binary relational symbol $<$. Let $\text{SAT}<$ be the set of satisfiable sentences in $L(<)$; that is, if $\varphi \in L(<)$ is a sentence, then $\varphi \in \text{SAT}<$ iff there is a set S and a linear order $<_S$ on S such that φ is true under the interpretation $(S, <_S)$.

By a direct arithmetization, Meyer has shown that $\text{SAT}< \notin \text{NSPACE}(g(\lceil cn \rceil, 0))$ for some constant $c > 0$.

Also, $SAT< \in DSPACE(g(\lceil dn \rceil, 0))$ for some $d > 0$ by [Rab69].

A nonelementary lower bound on $SAT<$ also follows by a transformation very similar to the one just given.

Theorem 5.7. $NE(\text{star-free}) \leq_{pl} SAT<.$

Thus $SAT<$ is not elementary-recursive.

Proof. Given a star-free expression E , a sentence φ_E in $L(<)$ is constructed such that $E \in NE(\text{star-free})$ iff $\varphi_E \in SAT<.$ The construction is very similar to that of Theorem 5.6 and Lemma 5.4. The main difference is that the linear order is used to pick out a set of discrete "points" and also to "simulate" the monadic predicate \underline{p} .

If S is a set with linear order $<$, $x \in S$ is a "point" iff x is isolated below. " $\underline{p}(x)$ is true" iff x is also isolated above (so " $\underline{p}(x)$ is false" if x is isolated below but not above).

Construct $F_E(x, y, u, v)$ and φ_E exactly as in Lemma 5.4 and Theorem 5.6 except:

(i). Write $\text{point}(x)$ as

$$(\exists s) ((s < x) \wedge \sim(\exists w) (s < w < x));$$

(ii). Replace each occurrence of $\underline{p}(x)$ by

$$(\exists t) ((x < t) \wedge \sim(\exists w) (x < w < t)).$$

Exactly as in the proofs of Lemmas 5.4 and 5.5 and Theorem 5.6, it follows that $E \in NE(\text{star-free})$ iff $\varphi_E \in SAT<.$ \square

As a final example, we consider the first order theory of two successors and prefix. Formulas in the language of this theory^{*} contain first order variables interpreted as ranging over $\{0,1\}$, **atomic** predicates $S_0(x,y)$ and $S_1(x,y)$ interpreted as $y = x \cdot 0$ and $y = x \cdot 1$ respectively, and the atomic predicate $x \leq y$ interpreted as $(\exists w \in \{0,1\}) [x \cdot w = y]$.

This theory, with the additional predicate of equal length, $E(x,y)$ interpreted as $|x| = |y|$, is $\equiv_{p\ell}$ to WS1S [ER66], which implies a fortiori an upper bound of space $g(dn,0)$ for the theory without the equal length predicate. The following theorem implies a lower bound of space $g(\lceil \log_b n \rceil, 0)$ for $b > 3$.

Theorem 5.8. $NE(\text{star-free}) \leq_{p\ell}$ The first order theory of two successors and prefix.

Proof. Given a star-free expression E , we construct a formula with two free variables $G_E(x,y)$ such that for all $a,b \in \{0,1\}$

$$G_E(a,b) \text{ iff } (\exists w \in L(E)) [a \cdot w = b].$$

$G_E(x,y)$ is constructed inductively on the structure of E :

$$\begin{aligned} G_{(\lambda)}(x,y) & \text{ is } (x = y) ; \\ G_{(0)}(x,y) & \text{ is } S_0(x,y) ; \quad G_{(1)}(x,y) \text{ is } S_1(x,y) ; \\ G_{(E \cdot E')}(x,y) & \text{ is } (\exists z) (G_E(x,z) \wedge G_{E'}(z,y)) ; \\ G_{(E \cup E')}(x,y) & \text{ is } (G_E(x,y) \vee G_{E'}(x,y)) ; \\ G_{(\sim E)}(x,y) & \text{ is } ((x \leq y) \wedge \sim G_E(x,y)) . \end{aligned}$$

The remainder of the proof is essentially the same as for Theorem 5.2. \square

Remark. (Length of proofs).■

In the study of logical theories, it is natural to consider the length of proofs of true sentences, as well as the time and space required by procedures which recognize the true sentences. Of course, given any complete consistent system of axioms AX for a theory T, an upper bound on the length of proofs from the axioms AX implies a corresponding upper bound on the space required to decide T, assuming that membership of words in AX can be decided efficiently (say, within polynomial time). In particular, for the decision problems considered in this chapter there is no upper bound on the length of proofs elementary-recursive in the length of sentences, provided the axioms are "**efficiently** recognizable" as above. See [FR74] for further discussion on the relation between length of proofs and computational complexity.

Chapter 6. Complexity of Finite Problems

The previous two chapters have shown that efficient reducibility techniques can yield non-trivial lower bounds on the complexities of certain decision problems. For reasons of technical simplicity, lower bounds have been stated in a **form** which implies that, no matter which algorithm is used to solve the particular problem, the time or space used by the algorithm must exceed the lower bound on some input of length n for infinitely many n . The fact that any algorithm must use excessively large amounts of time or space infinitely often might be viewed as plausible evidence that any algorithm will also perform badly on inputs of reasonable size which actually arise in practice.

Indeed, in order to draw meaningful conclusions about computational complexity, it is essential to know at what finite point the asymptotic lower **bounds** we have derived begin to take effect. Such information is implicit in our earlier proofs (cf. §3.3B).

Our purpose in this chapter is to demonstrate that our methods yield astronomical lower bounds (in the most literal sense, cf. Theorem 6.1 below) on the complexity of decision problems for expressions with only a few hundred characters.

We first consider the decision problem for the weak monadic second order theory of the natural numbers and successor. Let $WS1S$ be the set of true sentences written in weak monadic second order logic using only the relations $y = x+1$ and $x \in A$; that is, the second order sentences which are true under the standard interpretation $(\mathbb{N}, \text{successor})$

with set variables ranging over finite subsets of \mathbb{N} . Büchi [Buc60a] and Elgot [Elg61] have shown that $WS1S$ is decidable.[†]

For the purposes of this chapter, logical formulas are written in a language $\mathcal{L}^{††}$ enriched by certain notational abbreviations. In particular we may use decimal constants within formulas, writing 5 for $0+1+1+1+1+1$, $x+4$ for $x+1+1+1+1$, etc. Also, the binary relational symbols $\leq, <, =, \neq, >, \geq$ on integers may be used.

Let $EWSIS$ be the set of true sentences in \mathcal{L} . Note that the additional predicates of $EWSIS$ are all expressible in $WS1S$, so $EWSIS$ has no more expressive power than $WS1S$, and $EWSIS$ is also decidable. Let Σ be the alphabet of \mathcal{L} . For fixed integers n , we seek lower bounds on the complexity of recognizing the finite set $EWSIS \cap \Sigma^n$.^{†††}

Turing machine time and space are not sufficient to measure the complexity of finite sets. Any finite set is accepted by a finite state automaton within real time (time $T(n) = n$) and within space zero. This is accomplished by coding a finite table of the elements of a set into the states of the automaton.

Thus, for assessing the complexity of finite sets, account must be

[†]On the other hand, see [Mey73] or Theorem 5.3 of this paper for a lower bound on the i.o. time and space complexity of $WS1S$.

^{††} \mathcal{L} is defined precisely below.

^{†††}We shall include a blank symbol in Σ , so that $EWSIS \cap \Sigma^n$ essentially contains the true sentences of length less than or equal to n .

taken of the size or complexity of the device performing an algorithm as well as the time and space required by the algorithm. One quite general way to do this is to measure the number of basic operations on bits or the amount of logical circuitry required to decide membership in finite sets. We assume the basic operations on bits are binary operations performed by "gates" with two inputs and one output which may itself be fanned out to serve as input to other gates in a circuit. This circuit model yields a basic measure of complexity for Boolean functions as well as finite sets (via appropriate encoding into Boolean vectors) called combinational complexity [cf. Sav72]. Precise definitions appear below.

It will turn out that the alphabet Σ used for EWS1S contains 63 characters, each of which can therefore be coded into six binary digits. In particular, sentences of length 616 correspond to binary words or Boolean vectors of $6 \cdot 616 = 3696$ bits and this will be the number of inputs to a circuit which "accepts" the true sentences. The circuit is to have a single output line which gives the value one if and only if the input vector is the code of a true sentence of length 616.

One main result can now be informally stated.

Theorem 6.1. If C is a Boolean circuit which accepts $\text{EWS1S} \cap \Sigma^{616}$, then C contains more than 10^{123} gates.

Thus if a circuit C accepts EWS1S restricted to sentences of length not exceeding 616, and if each gate is the size of a proton,

then to accommodate C the entire known universe would be packed with gates.[†]

The first lower bound on the combinational complexity of sentences of logic was obtained by Ehrenfeucht [Ehr72; originally written in 1967] who showed that the size of circuits which accept true sentences of length n about integer arithmetic with all quantifiers bounded by constants described using exponential notation (e.g., 3^{2^5}) must exceed c^n for some $c > 1$ and all sufficiently large n . More generally, Meyer [Mey74] has observed that if $\text{MPSPACE} \leq_{\text{pl}} A$ for some language A , then the combinational complexity of A must grow exponentially. This observation implies Ehrenfeucht's original result (indeed $\text{SPACE}(g(\epsilon n, 0))$ is \leq_{pl} to Ehrenfeucht's formulation of bounded arithmetic), and also implies that the combinational complexity of most of the decision problems studied in this thesis grows exponentially.

bounds on size

However,

in order to obtain significant lower bounds for as small sentences as possible, it seems better to carry out a more direct arithmetization based on this efficient transformation result instead of appealing explicitly to the result.

We now define more precisely the notion of combinational complexity.

[†]We take 10^{-13} cm. to be the radius of a proton, and 11×10^9 light years $\approx 10^{28}$ cm. to be the radius of the universe.

A circuit is best defined as a straight-line algorithm. Straight-line algorithms are defined in [Sav72] for general domains and functional bases. We repeat the definition, restricting it to the Boolean case.

Definition 6.2. Let $\Omega_{16} = \{ g \mid g: \{0,1\}^2 \rightarrow \{0,1\} \}$ be the set of Boolean functions of two arguments.

Let $\Omega \subseteq \Omega_{16}$, $m \in \mathbb{N}^+$, and $t \in \mathbb{N}$. An Ω -straight-line algorithm or Ω -circuit of size t with m inputs is a sequence

$$C = \beta_m, \beta_{m+1}, \beta_{m+2}, \dots, \beta_{m+t-1}$$

such that for $m \leq k \leq m+t-1$, $\beta_k = (i, j, g)$ where i and j are integers with $0 \leq i, j < k$ and $g \in \Omega$.

With each step β_k for $k \geq m$ we identify an associated function $\xi_k: \{0,1\}^m \rightarrow \{0,1\}$ by induction. First, if $0 \leq k \leq m-1$, define ξ_k to be the k^{th} projection,

$$\xi_k(b_0 b_1 b_2 \dots b_{m-1}) = b_k \quad \text{for all } b_0 b_1 b_2 \dots b_{m-1} \in \{0,1\}^m.$$

If $m \leq k \leq m+t-1$ and $\beta_k = (i, j, g)$ then define

$$\xi_k(x) = g(\xi_i(x), \xi_j(x)) \quad \text{for } x \in \{0,1\}^m.$$

If f is a function, $f: \{0,1\}^m \rightarrow \{0,1\}^p$ for positive integers m and p , then the circuit C computes f iff C has m inputs and there are integers $0 \leq i_1, i_2, \dots, i_p \leq m+t-1$ such that

$$f(x) = \xi_{i_1}(x) \xi_{i_2}(x) \dots \xi_{i_p}(x) \quad \text{for all } x \in \{0,1\}^m.$$

The combinational complexity of a function $f: \{0,1\}^m \rightarrow \{0,1\}^p$ is the smallest t such that there is a Ω_{16} -circuit of size t which

computes f .^t

Let S be a finite alphabet. An encoding for S is a one-to-one function $h: S \rightarrow \{0,1\}^s$ where $s = \lceil \log(\text{card}(S)) \rceil$.^{tt}

Let $\hat{h}: S^* \rightarrow \{0,1\}^*$ be the extension of h .

Let $A \subseteq S^n$ for some $n \in \mathbb{N}^+$. Define $f_{A,h}: \{0,1\}^{sn} \rightarrow \{0,1\}$ by

$$f_{A,h}(w) = 1 \quad \text{iff} \quad w \in \{ \hat{h}(x) \mid x \in A \}.$$

The combinational complexity of the finite set A is the minimum over all encodings h of the combinational complexity of $f_{A,h}$.

If $L \subseteq S^+$, then the combinational complexity of L is a function $C_\infty(L): \mathbb{N}^+ \rightarrow \mathbb{N}$ such that for each n ,

$$C_\infty(L)(n) = \text{the combinational complexity of } L \cap S^n.$$

(Note: The subscript ∞ denotes unbounded fan-out [cf. Sav74].)

Remark 6.3. The notion of combinational complexity is in a sense incomparable with time or space complexity on Turing machines.

For example, define $L_A \subseteq \{0,1\}^+$ by $x \in L_A$ iff $|x| \in A$ where A is some non-recursive set of integers. Then L_A is non-recursive

^tOf course there is no loss of generality in not allowing basic functions of one argument. For example, an inversion gate $\sim b$ can be computed as $g_{NA}(b,b)$ where $g_{NA}(v_1, v_2) = \sim(v_1 \wedge v_2)$.

^{tt}**Logarithms** with no specified base are taken to the base 2.

By considering only block encodings, the exposition is somewhat simplified and there is essentially no loss of generality.

and its time and space complexity are not even defined. But $C_{\infty}(L)(n) = 1$ for all n because, for each fixed n , $L \cap \{0,1\}^n$ is either \emptyset or $\{0,1\}^n$. Thus, non-recursive and arbitrarily complex recursive sets can have a trivially small combinational complexity.

Another contrast is that time or space complexity of recursive languages can be as large as any recursive function, whereas any language L has combinational complexity $C_{\infty}(L)(n) \leq c^n$ for some $c > 1$ [cf. **Lup50**]. Moreover, there are elementary-recursive languages, in fact languages in EXPSPACE, whose combinational complexity is maximum for all values of n (over any given alphabet S), so that relatively "easy" recursive languages can have maximally large combinational complexity.

However, there is a basic relation in one direction between these two notions of computational complexity. Combinational complexity in effect always provides a lower bound on time complexity.

M. Fischer and N. Pippenger [FP74] have shown that

$$L \in \text{DTIME}(T(n)) \text{ implies } C_{\infty}(L)(n) \leq O(T(n) \cdot \log T(n)) .$$

So in particular, an exponential lower bound on $C_{\infty}(L)(n)$ implies an exponential lower bound on time complexity.

6.1 Second Order Theory of Successor.

Since our numerical results depend on the language \mathcal{L} used to write sentences, we give a BNF grammar for \mathcal{L} .

$\langle \text{member of } \mathcal{L} \rangle ::= \langle \text{formula} \rangle \mid \langle \text{member of } \mathcal{L} \rangle \vee$

$\langle \text{formula} \rangle ::= \exists \langle \text{variable} \rangle \langle \text{formula} \rangle \mid \forall \langle \text{variable} \rangle \langle \text{formula} \rangle \mid$
 $\sim \langle \text{formula} \rangle \mid \langle \text{formula} \rangle \langle \text{logical cnctv} \rangle \langle \text{formula} \rangle \mid$
 $(\langle \text{formula} \rangle) \mid \langle \text{atom} \rangle$

$\langle \text{atom} \rangle ::= \langle \text{term} \rangle \langle \text{order relation} \rangle \langle \text{term} \rangle \mid$
 $\langle \text{term} \rangle \in \langle \text{set variable} \rangle \mid \langle \text{term} \rangle \notin \langle \text{set variable} \rangle$

$\langle \text{term} \rangle ::= \langle \text{integer variable} \rangle \mid \langle \text{constant} \rangle \mid$
 $\langle \text{integer variable} \rangle + \langle \text{constant} \rangle$

$\langle \text{logical cnctv} \rangle ::= \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow$

$\langle \text{order relation} \rangle ::= < \mid \leq \mid = \mid \neq \mid \geq \mid >$

$\langle \text{variable} \rangle ::= \langle \text{integer variable} \rangle \mid \langle \text{set variable} \rangle$

$\langle \text{integer variable} \rangle ::= \langle \text{integer variable} \rangle \langle \text{lower case} \rangle \mid \langle \text{lower case} \rangle$

$\langle \text{set variable} \rangle ::= \langle \text{set variable} \rangle \langle \text{upper case} \rangle \mid \langle \text{upper case} \rangle$

$\langle \text{lower case} \rangle ::= a \mid b \mid c \mid \dots \mid p \mid q$

$\langle \text{upper case} \rangle ::= A \mid B \mid C \mid \dots \mid P \mid Q$

$$\langle \text{constant} \rangle ::= \langle \text{constant} \rangle \langle \text{digit} \rangle \mid \langle \text{digit} \rangle$$

$$\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid \dots \mid 8 \mid 9 .$$

Let Σ be the alphabet of \mathcal{L} , that is, the set of terminal symbols above. Note that $\text{card}(\Sigma) = 63$.

If $\Phi \in \mathcal{L}$, then $|\Phi|$ denotes the length of Φ viewed as a word in Σ . *

In the absence of parentheses, the precedence order for logical connectives is $\sim, \wedge, \vee, \Rightarrow, \Leftrightarrow$ (decreasing). Binding of quantifiers to formulas takes precedence over all logical connectives. To improve readability, redundant parentheses are sometimes used in the text in writing formulas; these are underlined, $\underline{(\quad)}$ and $\underline{)\quad}$, and are not counted in the length of formulas.

$\varphi \in \mathcal{L}$ is a sentence if φ contains no free variables. Let EWS1S be the set of sentences in \mathcal{L} which are true under the standard interpretation of the **integers**, with set variables ranging over finite subsets of \mathbb{N} . (Leading zeroes are ignored in interpreting constants.) The symbol \emptyset denotes a blank "padding" character which is ignored in determining the truth value of a sentence. Since sentences can be padded with blanks, $C_{\infty}(\text{EWS1S})(n)$ serves to measure the **combinational** complexity of deciding sentences of length $\leq n$.

Theorem 6.4. Let k , m , and n be positive integers such that:

- (1). $2^m > 2^{k+1} \cdot \log(2^k + m)$, and
- (2). $k - 24 \geq 3 \log m$, and
- (3). $n \geq 466 + \lfloor (\log_{10} 2) m \rfloor + 11 \lfloor \log_{10} m \rfloor$.

Then $C_{\infty}(\text{EWS1S})(n) > 2^{k-4}$.

Theorem 6.4 is proved below. For a fixed numerical value of n , a lower bound on $C_{\infty}(\text{EWS1S})(n)$ is obtained by choosing k and m to satisfy the above constraints. For example, we can now obtain the precise formulation of

Theorem 6.1. $C_{\infty}(\text{EWS1S})(616) > 10^{123}$.

Proof. Choose $k = 414$, $m = 424$, $n = 616$, and note that $2^{410} > 10^{123}$. \square

The proof of Theorem 6.4 is similar to the proofs of Chapter 4 which utilize efficient transformations between sets to obtain lower **complexity** bounds. The basic argument is as follows. We first prove **Lemma 6.5** which states that if k , m , and n satisfy certain constraints then there is a function $f_0: \{0,1\}^m \rightarrow \{0,1\}$ of "large" ($> 2^{k-3}$) combinational complexity such that questions about the value of f_0 on words of length m can be transformed to questions about membership of sentences of length n in **EWS1S**; moreover, the combinational complexity of the transformation τ is relatively "small". It then follows that the **combinational** complexity of **EWS1S** must be almost as large as that of f_0 . For assume that the **combinational** complexity of

EWS1S is small. Then by placing a circuit which **computes** τ in series with a circuit which accepts **EWS1S**, we obtain a "**small**" circuit which computes f_0 contrary to assumption.

One preliminary is required before proving **Lemma 6.5**. We shall use a special case of an "abbreviation trick" due to M. Fischer and A. Meyer [FM74]. If Φ is a logical formula involving several occurrences of a subformula, the trick allows one to write Φ equivalently as a formula involving only one occurrence of the subformula.

In the proof of **Lemma 6.5**, we shall always apply the **trick** to formulas **I** of the form

$$\Phi(u_1, \dots, u_n) = Q_1 z_1 Q_2 z_2 \dots Q_m z_m A(u_1, \dots, u_n, z_1, \dots, z_m)$$

where Q_1, \dots, Q_m are quantifiers, u_1, \dots, u_n denote variables which occur free in **I**, and z_1, \dots, z_m denote variables. A denotes a **formula** (with free variables $u_1, \dots, u_n, z_1, \dots, z_m$) of the form

$$A = (\dots G(v_{11}, \dots, v_{1p}) \dots G(v_{21}, \dots, v_{2p}) \dots G(v_{\ell 1}, \dots, v_{\ell p}) \dots)$$

where $G(v_1, \dots, v_p)$ denotes a formula of p free variables v_1, \dots, v_p , and for $1 \leq i \leq \ell$ the i^{th} occurrence, $G(v_{i1}, \dots, v_{ip})$, of G in A denotes a substitution instance of $G(v_1, \dots, v_p)$ with v_1 replaced by v_{i1} , v_2 replaced by v_{i2} , and soon. Each v_{ij} , $1 \leq i \leq \ell$, $1 \leq j \leq p$, denotes either a variable or a constant. In the cases we consider, each v_{ij} which is a variable is either free in Φ or is bound by one of the quantifiers Q_1, Q_2, \dots, Q_m .

Under these conditions, **I** can be written equivalently as a

formula Φ' involving one occurrence of G as follows. First let A' be the formula obtained from A by replacing the i^{th} occurrence, $G(v_{i1}, \dots, v_{ip})$, of G by the atomic formula $y_i = 1$ for $i = 1, 2, 3, \dots, \ell$, where y_1, y_2, \dots, y_ℓ denote new variables. Now we use "dummy variables" y, d_1, \dots, d_p , and write a separate formula to ensure that if $y = y_i$ and $d_j = v_{ij}$ for some i and all $j = 1, 2, 3, \dots, p$, then $y = 1$ iff $G(d_1, \dots, d_p)$ is true. That is:

$$\begin{aligned} \Phi'(u_1, \dots, u_n) = & Q_1 z_1 \dots Q_m z_m \exists y_1 \dots \exists y_\ell (A' \\ & \wedge \forall d_1 \dots \forall d_p \forall y (\bigvee_{i=1}^{\ell} (d_1 = v_{i1} \wedge \dots \wedge d_p = v_{ip} \wedge y = y_i)) \\ & \Rightarrow (y = 1 \Leftrightarrow G(d_1, \dots, d_p)))). \end{aligned}$$

In the cases we consider, Φ uses sufficiently few variables that the additional variables $y_1, \dots, y_\ell, y, d_1, \dots, d_p$ can each be written as a single letter. Also, each of the v_{ij} is either a single letter or a single digit.

Under these conditions, the length of Φ' is related to the lengths of I and G by:

Length relation for the abbreviation trick:

$$|\Phi'| = |\Phi| + (1 - \ell)|G| + (4\ell p + 9\ell + 2p + 13).$$

In particular, the symbols $Q_1 z_1 \dots Q_m z_m$ plus those symbols in A' contribute $(|\Phi| + 3\ell - \ell|G|)$ to $|\Phi'|$.

Lemma 6.5. Let k , m , and n be positive integers which satisfy (1) and (3) of Theorem 6.4. Then there is a function $f_0: \{0,1\}^m \rightarrow \{0,1\}$ such that:

- (i). The combinational complexity of f_0 is greater than 2^{k-3} ;
- and
- (ii). For each $x \in \{0,1\}^m$ there is a sentence $\varphi_x \in \mathcal{L}$ such that

$$|\varphi_x| = n, \text{ and } \varphi_x \in \text{EWS1S} \text{ iff } f_0(x) = 1.$$

Moreover, if $h: \Sigma \rightarrow \{0,1\}^6$ is any encoding, and if τ is the function which maps x to $\hat{h}(\varphi_x)$ for all $x \in \{0,1\}^m$, then the combinational complexity of τ is less than $2^{20} 3^m$.

Proof. Let k , m , and n be fixed integers which satisfy constraints (1) and (3) of Theorem 6.4.

We first describe the formula $\text{Easy}'(F)$ (of one free set variable) which is used within φ_x . $\text{Easy}'(F)$ is constructed in Lemma 6.5.1 which comprises the major technical portion of the proof of Lemma 6.5. Some definitions are required to state this sublemma.

Let **NAND** be the singleton set consisting of the Boolean function g_{NA} of two arguments defined by $g_{\text{NA}}(v_1, v_2) = \sim(v_1 \wedge v_2)$.

If $x \in \{0,1\}^m$, $\text{int}(x)$ is the nonnegative integer z such that x is a reverse binary representation (possibly with following zeroes) of z .

For example, $\text{int}(111000) = 7$ and $\text{int}(101100) = 13$ (if $m = 6$).

Let $F \subset \mathbb{N}$. $\text{fct}(F)$ is the function mapping $\{0,1\}^m$ to $\{0,1\}$ defined by $\text{fct}(F)(x) = 1$ iff $m(\text{int}(x) + 1) \in F$.

$\text{fct}(F)$ is the means by which functions from $\{0,1\}^m$ to $\{0,1\}$ are represented as sets of integers in our arithmetization of circuits.

Lemma 6.5.1. Let k and m satisfy (1) of Theorem 6.4. There is a formula $\text{Easy}'(F)$ in \mathfrak{L} such that:

- (i). For all finite $F \subset \mathbb{N}$, $\text{Easy}'(F)$ is true iff there is a NAND-circuit of size 2^k with m inputs which computes $\text{fct}(F)$;
 and
 (ii). $|\text{Easy}'(F)| = 380 + 10 \lfloor \log_{10} m \rfloor$.

Proof. We first write a formula $\text{Easy}(F)$ involving several occurrences of a subformula, and then obtain $\text{Easy}'(F)$ from $\text{Easy}(F)$ via the abbreviation trick described above.

Some notation is helpful. If $S \subset \mathbb{N}$, let $\text{seq}(S)$ denote the (infinite) binary sequence $b_0 b_1 b_2 b_3 \dots$, where $b_i = 1$ if $i \in S$ and $b_i = 0$ if $i \notin S$. Let $\text{m-word}(S, j)$ denote the finite binary word $b_j b_{j+1} b_{j+2} \dots b_{j+m-1}$ of $\text{seq}(S)$.

Let $\text{dec}(m)$ denote the decimal representation of m . Let $\text{dec}(k)$ be a decimal representation of k with leading zeroes if necessary to make $|\text{dec}(k)| = |\text{dec}(m)|$. (Constraint (1) implies $k < m$)

$\text{Easy}(F)$ is a conjunction of five terms. The first four terms $\psi_1, \psi_2, \psi_3, \psi_4$ place constraints on the variables B, P, d , and q . The last term ψ_5 expresses the fact that $\text{fct}(F)$ is computable by a NAND-circuit of size $\leq 2^k$ (which is the same as being computable by a NAND-circuit of size exactly 2^k) .

(ψ_1). $\forall a (\psi_1(B, d, a))$ is true iff $d \in B$ and $B = B_0$ where

$$B_0 = \{ z \mid m \leq z \leq d \text{ and } z \equiv 0 \pmod{m} \}.$$

$$\begin{aligned} \psi_1 \text{ is } & \underline{d \in B \wedge \text{dec}(m) \in B} \\ & \wedge (\underline{a < \text{dec}(m) \vee a > d} \Rightarrow a \notin B) \\ & \wedge (\underline{a < d \wedge a \neq 0} \Rightarrow (a \in B \Leftrightarrow a + \text{dec}(m) \in B)) \underline{).} \end{aligned}$$

(\$2). Assuming $B = B_0$ and $d \in B$, then $\forall a(\psi_2(B, P, d, a))$ is true iff for all integers i with $0 \leq mi \leq d$, $\mathbf{m-word}(P, mi)$ is a reverse binary representation of the integer z where $z \equiv (i-1)(\text{mod } 2^m)$ and $0 \leq z < 2^m$. That is,

$$\text{seq}(P) = \overbrace{111\dots 11}^m \overbrace{000\dots 00}^m \overbrace{100\dots 00}^m \overbrace{010\dots 00}^m \overbrace{110\dots 00}^m \overbrace{00010\dots 00}^m \dots ,$$

and where, if $\text{seq}(P) = p_0 p_1 p_2 \dots$, then this pattern continues at least to bit p_{d+m-1} of $\text{seq}(P)$. The bits of $\text{seq}(P)$ beyond the $(d+m-1)^{\text{th}}$ are not constrained by ψ_2 . (The formula ψ_2 is similar to one used by Robertson [Rob73].)

$$\begin{aligned} \psi_2 \text{ is } & \underline{(a < \text{dec}(m) \Rightarrow a \in P)} \\ & \wedge (a < d \Rightarrow \\ & \quad ((a \in P \Leftrightarrow a + \text{dec}(m) \notin P) \\ & \quad \Leftrightarrow \\ & \quad \exists b((b \in B \vee b = 0) \wedge b \leq a \\ & \quad \wedge \forall i((b \leq i \wedge i < a) \Rightarrow i \in P)))) \underline{).} \end{aligned}$$

(\$3). Assuming that $B = B_0$, $d \in B$, and that $\text{seq}(P)$ is as above, then $\forall a(\psi_3(P, d, a))$ is true iff $d \equiv 0 \pmod{m2^m}$.

ψ_3 states simply that $\mathbf{m-word}(P, d) = 1^m$.

ψ_3 is $(\lfloor d \leq a \wedge a < d + \text{dec}(m) \rfloor \Rightarrow a \in P)$.

Recall $d \in B$ and $0 \notin B$ by (\$I), and thus $d > 0$. Now the truth of $\forall a(\psi_i)$ for $i = 1, 2, 3$ together imply that $\text{seq}(P)$ cycles at least once through the 2^m binary words of length m (See Figure 6.1. Upward arrows point to those positions of $\text{seq}(P)$ which belong to B)

$\text{seq}(P) =$
 $\overbrace{111 \dots 11}^m \overbrace{000 \dots 00}^m \overbrace{100 \dots 000}^m \overbrace{10 \dots 00}^m \dots \overbrace{011 \dots 11}^m \overbrace{111 \dots 11}^m \text{ don't care } \dots$
 $\uparrow \quad \quad \quad \uparrow \quad \quad \quad \uparrow \quad \quad \quad \uparrow \quad \quad \quad \uparrow \quad \quad \quad \uparrow$
 $\quad \quad \quad t \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad d$

Figure 6.1. P , B , and d .

(\$4). If B and P are as in Figure 6.1, then $\forall a(\psi_4(B, P, q, a))$ is true iff $q \in B$ and $q \leq m 2^k$.

ψ_4 is $(q \in B \wedge (\lfloor a \in B \wedge a \leq q \rfloor \Rightarrow a + \text{dec}(k) \notin P))$.

To summarize (\$1) through (ψ_4), if

$\forall a(\psi_1(B, d, a) \wedge \psi_2(B, P, d, a) \wedge \psi_3(P, d, a) \wedge \psi_4(B, P, q, a))$ is true then:

- (*) $\left\{ \begin{array}{l} (1). \quad B = \{ z \mid m \leq z \leq d \text{ and } z \equiv 0 \pmod{m} \}, \\ (2). \quad \text{seq}(P) \text{ is as in Figure 6.1,} \\ (3). \quad d \equiv 0 \pmod{m 2^m} \text{ and } d > 0, \\ (4). \quad q \in B \text{ and } q \leq m 2^k. \end{array} \right.$

(ψ_5). We first describe the formula Match which is used as a subformula within ψ_5 .

Match(X_1, w_1, X_2, w_2) is

$$\begin{aligned} \exists K \forall b (& w_1 < w_2 \wedge (w_1 \in B \vee w_1 < \text{dec}(m))) \\ & \wedge ((w_1 \leq b \wedge b < w_2) \Rightarrow (b \in K \Leftrightarrow b + \text{dec}(m) \in K)) \\ & \wedge (b < w_1 + \text{dec}(m) \Rightarrow (b \in K \Leftrightarrow b \in X_1)) \\ & \wedge (w_2 \leq b \Rightarrow (b \in K \Leftrightarrow b \in X_2))) . \end{aligned}$$

The following lemma describes certain properties of Match.

Lemma 6.5.2. Assume B, P, d, and q are as in (*). Let $S, S_1, S_2 \subset \mathbb{N}$.

- (i). Let $z_1, z_2 \in B \cup \{0\}$. Match(S_1, z_1, S_2, z_2) is true iff
 $z_1 < z_2$ and $\text{m-word}(S_1, z_1) = \text{m-word}(S_2, z_2)$.
- (ii). Let $a \in B$. Match(P, i, S, a) is true iff $i < a$ and
either ($i \in B$ and $\text{m-word}(P, i) = \text{m-word}(S, a)$)
or ($0 \leq i < m$ and $\text{m-word}(S, a) = 0^i 1^{m-i}$).
- (iii). Let $a \in B$ with $a \leq q$. Then there is at most one $i \in \mathbb{N}$
--
such that Match(P, i, S, a) is true.

Proof. (i) and (ii) are left as exercises. See Figure 6.2 which shows how K can be chosen in two particular cases. In Figure 6.2, $m = 6$ and words are divided into blocks of length six for readability.

To verify (iii), let $a \in B$ with $a \leq q$ be fixed. Constraint (1) of Theorem 6.4 implies $k \leq m - 1$. Now $a \leq q \leq m2^k \leq m2^{m-1}$ implies that for all $i_1, i_2 \in B$ with $i_1, i_2 < a$:

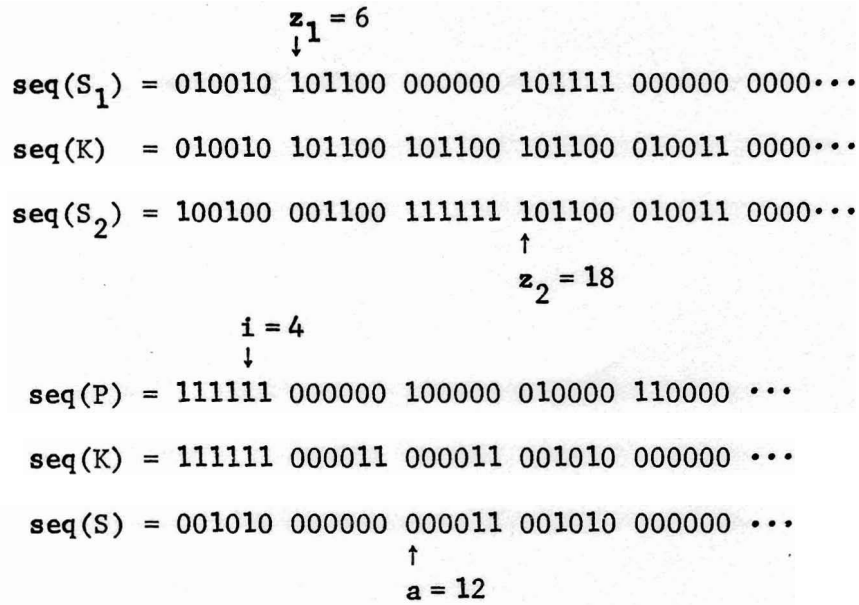


Figure 6.2. Illustrating the proof of Lemma 6.5.2 (i) and (ii).

.....

(**) $\text{m-word}(P, i_1) = \text{m-word}(P, i_2)$ iff $i_1 = i_2$; and

(***) $\text{m-word}(P, i_1) = b_0 b_1 b_2 \dots b_{m-2} 0$ for some $b_0, b_1, \dots, b_{m-2} \in \{0, 1\}$.

Now suppose that $\text{Match}(P, i_1, S, a)$ and $\text{Match}(P, i_2, S, a)$ are both true.

Part (ii) of the lemma implies $i_1, i_2 < a$ and one of four cases:

First, if $i_1, i_2 \in B$ then part (i) of the lemma together with

(**) implies $i_1 = i_2$;

Second, if $i_1, i_2 < m$ then part(ii) of the lemma implies

$i_1 = i_2 = i$ where $\text{m-word}(S, a) = 0^i 1^{m-i}$,

The other two cases, namely where one of i_1, i_2 belongs to B and the other is less than m , cannot occur because of (***) together with parts (i) and (ii). For example, if $i_1 \in B$ and $i_2 < m$, then

$$\begin{aligned} \mathbf{m}\text{-word}(\mathbf{P}, \mathbf{i}_1) &= \mathbf{m}\text{-word}(\mathbf{S}, \mathbf{a}) \quad \text{because } \text{Match}(\mathbf{P}, \mathbf{i}_1, \mathbf{S}, \mathbf{a}) \text{ is true} \\ &= 0^{\mathbf{i}_2} 2^{\mathbf{m}-\mathbf{i}_2} \quad \text{because } \text{Match}(\mathbf{P}, \mathbf{i}_2, \mathbf{S}, \mathbf{a}) \text{ is true.} \end{aligned}$$

However this now contradicts (***) which states that $\mathbf{m}\text{-word}(\mathbf{P}, \mathbf{i}_1)$ must end with 0. \square

We now describe how sets of integers are viewed as representing circuits and "computations" of circuits.

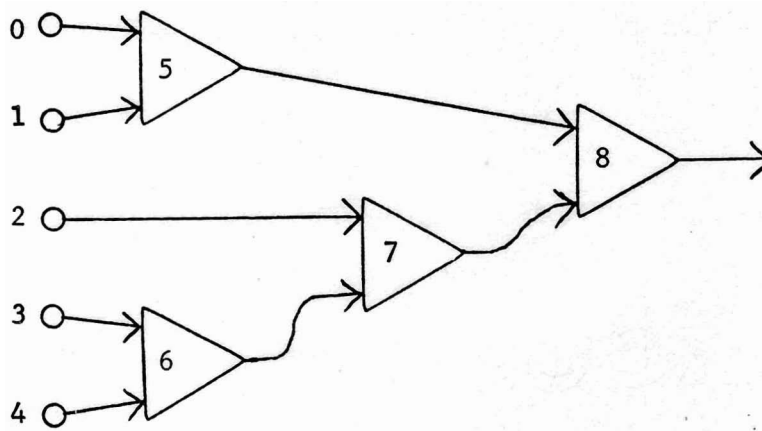
Let B, P, d, q be as in (*), and let $I, J \subset \mathbb{N}$. Then $\mathbf{q}\text{-circuit}(I, J)$ is defined and $\mathbf{q}\text{-circuit}(I, J)$ is the NAND-circuit C of size $t = q/m$ with m inputs where $C = \beta_m, \beta_{m+1}, \beta_{m+2}, \dots, \beta_{m+t-1}$ iff for each $a \in B$ with $m \leq a \leq q$ there exist i, j such that

- (i). $\text{Match}(\mathbf{P}, \mathbf{i}, \mathbf{I}, \mathbf{a})$ and $\text{Match}(\mathbf{P}, \mathbf{j}, \mathbf{J}, \mathbf{a})$ are both true, and
- (ii). $\beta_{\alpha(a)} = (\alpha(i), \alpha(j), g_{\text{NA}})$, where α is given by

$$\alpha(z) = \begin{cases} z & \text{if } z < m \\ z/m + m - 1 & \text{if } z \in B. \end{cases}$$

It is important to note by Lemma 6.5.2(iii) that $\mathbf{q}\text{-circuit}(I, J)$ is uniquely defined when it is defined.

Figure 6.3 illustrates how a particular pair $I, J \subset \mathbb{N}$ codes a circuit in the case $q = 20, m = 5$, (so $t = 4$). In Figure 6.3: $\text{seq}(\mathbf{P})$ is shown for reference; \times is a "don't care" symbol; words are divided into blocks of length five for readability.



$$\begin{aligned} \mathbf{q-circuit}(I,J) &= (0,1,g_{NA}), (3,4,g_{NA}), (2,6,g_{NA}), (5,7,g_{NA}) \\ &= \beta_5, \beta_6, \beta_7, \beta_8 \end{aligned}$$

$\mathbf{seq}(P) = 11111 \ 00000 \ 10000 \ 01000 \ 11000 \ \dots$

$\mathbf{seq}(I) = \text{XXXXX} \ 11111 \ 00011 \ 00111 \ 00000 \ \dots$

$\mathbf{seq}(J) = \text{XXXXX} \ 01111 \ 00001 \ 10000 \ 01000 \ \dots$

$\mathbf{seq}(D) = 11101 \ 0\text{XXXX} \ 1\text{XXXX} \ 0\text{XXXX} \ 1\text{XXXX} \ \dots$

a	01234	5	10	15	20
$\alpha(a)$	01234	5	6	7	8

Figure 6.3. I and J "code" a circuit.

For arbitrary $I, J \subset N$, if $C = \mathbf{q-circuit}(I,J) = \beta_m, \beta_{m+1}, \dots, \beta_{m+t-1}$ is a circuit as on the preceding page, if $x \in \{0,1\}^m$, and $D \subset N$, then D represents the computation of C on x iff for all a with $a \in \{z \mid 0 \leq z < m\} \cup \{z \in B \mid m \leq z \leq q\}$

$$a \in D \text{ iff } \xi_{\alpha(a)}(x) = 1$$

where the $\{\xi_i\}$ are the associated functions of C (cf. Definition 6.2).

Note in particular that if D represents the computation of C on x , then $\mathbf{m-word}(D, 0) = x$.

Figure 6.3 also shows a set D which represents the computation of $q\text{-circuit}(I, J)$ on input **11101**.

We note one fact and then write ψ_5 . Fact 6.5.3 is immediate from the definition of $\mathbf{int}(x)$ and $\mathbf{fct}(F)$, and the fact that P is constrained as in Figure 6.1.

Fact 6.5.3. Let $x \in \{0, 1\}^m$, $F \subset N$, and $e = m(\mathbf{int}(x) + 1)$. Then

$$\mathbf{m-word}(P, e) = x, \text{ and } e \in F \text{ iff } \mathbf{fct}(F)(x) = 1.$$

Now assuming that B , P , d , and q are as in (*) above, $\psi_5(F, B, P, q)$ is true iff there is a NAND-circuit of size q/m ($\leq 2^k$) which computes $\mathbf{fct}(F)$.

ψ_5 is $\exists I \exists J \forall e \exists D \forall a \exists i \exists j \psi'_5$, where ψ'_5 is

$$(5.1) \quad (e \in B \Rightarrow$$

$$(5.2) \quad \bigwedge \text{Match}(D, 0, P, e)$$

$$(5.3) \quad \wedge (a \in B \Rightarrow$$

$$" \quad \bigwedge \text{Match}(P, i, I, a) \wedge \text{Match}(P, j, J, a)$$

$$" \quad \wedge (a \in D \Leftrightarrow \sim(i \in D \wedge j \in D)) \bigwedge)$$

$$(5.4) \quad \wedge (q \in D \Leftrightarrow e \in F) \bigwedge) .$$

Informally, ψ_5 expresses the following.

There exists a circuit, **q-circuit**(I,J), of size $t = q/m$ such that:

(5.1) For all inputs $x \in \{0,1\}^m$ (where $e = m(\text{int}(x) + 1)$), there exists a computation D such that:

(5.2) $m\text{-word}(D,0) = m\text{-word}(P,e) = x$ by Lemma 6.5.2(i) and Fact 6.5.3; and

(5.3) for all gates $\beta_{\alpha(a)}$ with $a \in B$ there exist i and j such that the output $\xi_{\alpha(a)}(x)$ of $\beta_{\alpha(a)}$ is computed correctly as $\sim(\xi_{\alpha(i)}(x) \wedge \xi_{\alpha(j)}(x))$; and

(5.4) gate $\beta_{\alpha(q)}$ produces output 1 iff $e \in F$ (iff $\text{fct}(F)(x) = 1$, cf. Fact 6.5.3).

Finally let **Easy**(F) be

$$\exists B \exists P \exists d \exists q \exists i \exists j \forall e \exists D \forall a \exists i \exists j (\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge \psi_4 \wedge \psi_5)$$

so that by standard manipulation of quantifiers **Easy**(F) is equivalent to $\exists B \exists P \exists d \exists q (\forall a (\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge \psi_4) \wedge \psi_5)$.

We let the reader supply any additional argument required to convince himself that **Easy**(F) is true iff there is a NAND-circuit of size 2^k which computes $\text{fct}(F)$. (In the "if" direction, always choose $p = m2^m$, $q = m2^k$, and choose I,J such that (gate $\beta_{\alpha(q)}$ of) **q-circuit**(I,J) computes $\text{fct}(F)$ and moreover that

$\exists i \exists j (\text{Match}(P,i,I,a) \wedge \text{Match}(P,j,J,a))$ is true also for those $a \in B$ with $a > q$.)

We now count the length of **Easy**(F).

Let $\mu = \lfloor \log_{10} m \rfloor + 1$. Note that $|\text{dec}(k)| = |\text{dec}(m)| = \mu$.

First, $|Match| = 72 + 3\mu$.

The lengths of $\psi_1, \psi_2, \psi_3, \psi_4, \psi_5$ are respectively $40 + 3\mu$, $61 + 2\mu$, $14 + \mu$, $18 + \mu$, and $41 + 3|Match|$. The length of Easy is the sum of these plus 28 additional symbols, so

$$(Easy) = 202 + 7\mu + 3|Match|.$$

Using the Fischer-Meyer abbreviation trick with $\ell = 3$ and $p = 4$ to reduce the three occurrences of Match to one, Easy can be written equivalently as Easy' where

$$\begin{aligned} |Easy'| &= |Easy| - 2|Match| + 96 \\ &= 380 + 10 \lfloor \log_{10} m \rfloor. \end{aligned}$$

Note that the additional variables $d_1, d_2, d_3, d_4, y_1, y_2, y_3, y$ used in the abbreviation trick can be named $E, c, L, f, g, h, \ell, o$ respectively.

This completes the proof of Lemma 6.5.1. \square

We now return to the proof of Lemma 6.5 and the construction of φ_x . Let φ_x'' be the following sentence, where $\omega(x)$ and $Lessthan(G, F)$ are defined below.

$$\begin{aligned} \varphi_x'' \text{ is } & \exists F \forall G (\omega(x) \in F \wedge \sim Easy'(F) \\ & \wedge (Lessthan(G, F) \Rightarrow Easy'(G))). \end{aligned}$$

The formula $Lessthan(G, F)$ is

$$\exists a(a \in F \wedge a \notin G \wedge \forall b(b > a \Rightarrow (b \in G \Leftrightarrow b \in F))).$$

$Lessthan(G, F)$ is easily seen to define a linear order on finite subsets of N .

$\omega(x)$ is a decimal representation of $m(\text{int}(x) + 1)$; leading zeroes are appended so that

$$|\omega(x)| = \lfloor (\log_{10} 2)^m \rfloor + \lfloor \log_{10} m \rfloor + 2.$$

(Note that $x \in \{0, 1\}^m$ implies $\text{int}(x) \leq 2^m - 1$. It follows that the decimal representation of $m(\text{int}(x) + 1)$ need never be longer than

$$\lfloor \log_{10}(m2^m) \rfloor + 1 \leq \lfloor (\log_{10} 2)^m \rfloor + \lfloor \log_{10} m \rfloor + 2.)$$

It is easy to see that there are at most $(2^k + m)^{2^{k+1}}$ NAND-circuits of size 2^k with m inputs. (That is, each of the total 2^{k+1} possible inputs to gates is filled with a number between 0 and 2^{k+m-1} .)

However there are 2^{2^m} functions from $\{0, 1\}^m$ to $\{0, 1\}$. Constraint (1)

of Theorem 6.4 ensures $2^{2^m} > (2^k + m)^{2^{k+1}}$ and therefore that there is a finite $F \subset \mathbb{N}$ such that $\text{Easy}'(F)$ is false.

Since **Lessthan** defines a linear order, there is exactly one finite $F_0 \subset \mathbb{N}$ such that $\forall G (\neg \text{Easy}'(F_0) \wedge (\text{Lessthan}(G, F_0) \Rightarrow \text{Easy}'(G)))$ is true.

We take $f_0 = \text{fct}(F_0)$.

Since any Boolean function of two arguments can be synthesized using at most five "NAND-gates" [cf. Har65], and since $\text{Easy}'(F_0)$ is false, it follows that the combinational complexity of $f_0 = \text{fct}(F_0)$ must exceed $(1/5) \cdot 2^k > 2^{k-3}$.

Also by the definition of $\text{fct}(F)$, $\omega(x) \in F_0$ iff $f_0(x) = 1$, so φ_x'' is true iff $f_0(x) = 1$.

$$|\text{Lessthan}| = 29.$$

$$\begin{aligned} |\varphi_x''| &= 14 + |\omega(x)| + |\text{Lessthan}| + 2|\text{Easy}'| \\ &= 45 + 2|\text{Easy}'| + \lfloor (\log_{10} 2)^m \rfloor + \lfloor \log_{10} m \rfloor. \end{aligned}$$

The abbreviation trick with $\ell=2$ and $p=1$ applied to φ_x'' and **Easy'** gives φ_x' equivalent to φ_x'' and

$$\begin{aligned} |\varphi_x'| &= |\varphi_x''| - |\text{Easy}'| + 41 \\ &= 466 + \lfloor (\log_{10} 2) m \rfloor + 11 \lfloor \log_{10} m \rfloor. \end{aligned}$$

The additional variables d_1, y_1, y_2, y can be named M, k, m, n respectively.

By constraint (3) of Theorem 6.4, $j \geq 0$ can be chosen so that

$$\varphi_x = \varphi_x' y^j \quad \text{and} \quad |\varphi_x| = n.$$

φ_x and f_0 satisfy the requirements of Lemma 6.5.

It remains only to bound the combinational complexity of the **trans-**formation τ mapping x to $\hat{h}(\varphi_x)$. For fixed k, m , and n , $\hat{h}(\omega(x))$ is the only part of $\hat{h}(\varphi_x)$ which depends on x . (Recall that the length of $\omega(x)$ is independent of x .) Thus all bits of $\hat{h}(\varphi_x)$ **excluding** $\hat{h}(\omega(x))$ can be computed using exactly two gates, namely the two gates with constant output. Now $2^{20} m^3$ is a gross upper bound on the combinational complexity of the transformation mapping x to $\hat{h}(\omega(x))$, using straightforward classical algorithms for binary addition, binary multiplication, and **binary-to-decimal** conversion [cf. Knu69].

This completes the proof of Lemma 6.5. □

Proof of Theorem 6.4. Let k, m , and n satisfy the constraints (1), (2), and (3) of the theorem. Assume the conclusion is false, that is

$$C_{\infty}(\text{EWS1S})(n) \leq 2^{k-4}.$$

Therefore there is an encoding $h: \Sigma \rightarrow \{0, 1\}^6$ and a Ω_{16} -circuit C of size 2^{k-4} with $6n$ inputs which computes a function f where in

particular for all sentences $\varphi \in \mathcal{L} \cap \Sigma^n$,

$$f(\hat{h}(\varphi)) = 1 \text{ iff } \varphi \in \text{EWS1S}.$$

Let f_0 and τ be as in Lemma 6.5 for this k , m , n , and encoding h .
Let T be an Ω_{16} -circuit of size $< 2^{20}m^3$ which computes τ .

Now let C_0 be the circuit shown in Figure 6.4. [†]

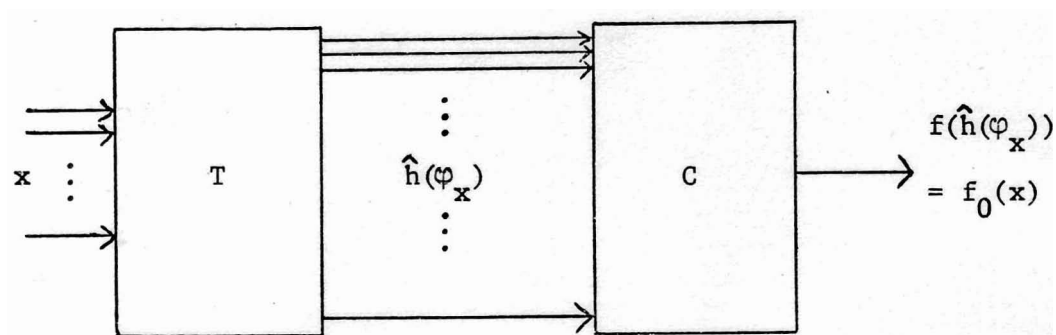


Figure 6.4. The circuit C_0

Since $f(\hat{h}(\varphi_x)) = 1$ iff $\varphi_x \in \text{EWS1S}$ iff $f_0(x) = 1$, C_0 computes f_0 .

$$\begin{aligned} \text{But "size of } C_0" &= \text{"size of } T" + \text{"size of } C" \\ &< 2^{20}m^3 + 2^{k-4} \leq 2^{k-3}, \end{aligned}$$

because constraint (2) implies $2^{20}m^3 \leq 2^{k-4}$. This contradicts the fact that the combinational complexity of f_0 is greater than 2^{k-3} .

Therefore we must have $C_\infty(\text{EWS1S})(n) > 2^{k-4}$. \square

[†]It is clear how to define C_0 from C and T within the formalism of straight-line algorithms.

6.2 First Order Integer Arithmetic.

. In this section we obtain even stronger lower bounds on the combinational complexity of a logical decision problem. Consider the first order theory of the nonnegative integers with primitives addition, multiplication, and exponentiation to the base 2. Sentences are again written in a language \mathcal{L}' allowing decimal constants and the relations $\leq, <, =, \neq, >, \geq$. Terms are any arithmetic expressions involving constants, variables, addition; multiplication; and base 2 exponentiation. For example, $x + 300 \cdot y$ and $z \cdot 2^{(i+1)}$ are terms, and $x \cdot u + 6 < 2^z$ is an atomic formula.

\mathcal{L}' is defined by the following BNF grammar, where $\langle \text{formula} \rangle$, $\langle \text{order relation} \rangle$, and $\langle \text{constant} \rangle$ are defined as in the grammar given for \mathcal{L} in §6.1.

$$\langle \text{member of } \mathcal{L}' \rangle ::= \langle \text{member of } \mathcal{L} \rangle \vee \mid \langle \text{formula} \rangle$$
$$\langle \text{atom} \rangle ::= \langle \text{term} \rangle \langle \text{order relation} \rangle \langle \text{term} \rangle$$
$$\begin{aligned} \langle \text{term} \rangle ::= & \langle \text{term} \rangle + \langle \text{term} \rangle \mid \langle \text{term} \rangle \cdot \langle \text{term} \rangle \mid \uparrow \langle \text{term} \rangle \mid \\ & (\langle \text{term} \rangle) \mid \langle \text{variable} \rangle \mid \langle \text{constant} \rangle \end{aligned}$$
$$\langle \text{variable} \rangle ::= \langle \text{variable} \rangle \langle \text{lower case} \rangle \mid \langle \text{lower case} \rangle$$
$$\langle \text{lower case} \rangle ::= a \mid b \mid c \mid \dots \mid y \mid z .$$

$\uparrow \langle \text{term} \rangle$ denotes $2^{\langle \text{term} \rangle}$, and the latter notation is used in the text in writing formulas. The precedence order for arithmetic

operations is $\uparrow, \cdot, +$ (decreasing). As before, redundant parentheses, $($ and $)$, are sometimes used. Let Σ' be the alphabet of \mathfrak{L}' ; note that $\text{card}(\Sigma') = 55$.

Let FIA be the set of sentences in \mathfrak{L}' which are true under the standard interpretation for $+, \cdot, \uparrow$, etc. with variables ranging over \mathbb{N} .

Theorem 6.6. Let k, m , and n be positive integers such that:

- (1). $2^{m-2/m} > 2k(2^k - m + 1)$, and
- (2). $2^{k-7}/k \geq 2^{20}m^3$, and
- (3). $n \geq 242 + \lfloor (\log_{10} 2) m \rfloor + 6 \lfloor \log_{10} m \rfloor$.

Then $C_{\infty}(\text{FIA})(n) > 2^{k-7}/k$.

For example, with $k = 426$, $m = 447$, and $n = 388$:

Corollary 6.6.1. $C_{\infty}(\text{FIA})(388) > 2^{410} > 10^{123}$.

If we seek a more modest bound, say a trillion gates, then choosing $k = 53$, $m = 69$, and $n = 268$ gives:

Corollary 6.6.2. $C_{\infty}(\text{FIA})(268) > 2^{40} > 10^{12}$.

Note: In Corollaries 6.6.1 and 6.6.2, the lengths of sentences in "bits" are respectively $6.388 = 2328$ and $6 \cdot 268 = 1608$.

Proof of Theorem 6.6. There are a number of similarities between this proof and that of Theorem 6.4 and Lemma 6.5. We sketch only the essential details.

Fix k, m , and n to satisfy the constraints (1), (2), and (3).

Let $\pi(z)$ denote the number of prime positive integers that do not exceed z .

Fact 6.6.1 [cf. NZ66]. For $z \geq 2$,

$$(1/4)z/\log z < \pi(z) < 9z/\log z.$$

For $d, i \in \mathbb{N}$, $\text{Bit}(d, i)$ is true iff the coefficient of 2^i in the binary expansion of d is 1.

For $u, i \in \mathbb{N}$ and $a \in \mathbb{N}^+$, $\text{Res}(u, a, i)$ is true iff $u \equiv i \pmod{a}$
and $i < a$.

. We now describe how integers are viewed as representing functions, circuits, and computations of circuits. As in §6.1, let $\text{int}(x)$ be the integer i such that x is a reverse binary representation of i .

If $z \in \mathbb{N}$, $\text{fct}(z)$ is the function mapping $\{0, 1\}^m$ to $\{0, 1\}$ defined by $\text{fct}(z)(x) = 1$ iff $\text{int}(x)$ divides z .
Of course there are functions from $\{0, 1\}^m$ to $\{0, 1\}$ which do not equal $\text{fct}(z)$ for all z . However the following is true.

Lemma 6.6.2. Let $F = \{ f \mid f: \{0, 1\}^m \rightarrow \{0, 1\} \text{ and } (\exists z)[f = \text{fct}(z)] \}$.

Then $\text{card}(F) > 2^{m-2}/m$.

Proof. Let $X = \{ x \in \{0, 1\}^m \mid \text{int}(x) \text{ is prime} \}$.

Given any choices of $b_x \in \{0, 1\}$ for $x \in X$, there is a z such that $\text{fct}(z)(x) = b_x$ for all $x \in X$. Namely $z = \prod_{b_x=1} \text{int}(x)$.

Therefore $\text{card}(F) \geq 2^{\text{card}(X)}$.

But $\text{card}(X) = \pi(2^m - 1) = \pi(2^m) > 2^{m-2}/m$ (since (1) implies $m \geq 2$). \square

Let $u, v \in \mathbb{N}$ and $t \in \mathbb{N}^+$. Then $\underline{t\text{-circuit}(u, v)}$ is the NAND-circuit of size $t - m$ with m inputs, $\beta_m, \beta_{m+1}, \beta_{m+2}, \dots, \beta_{t-1}$, where for each a with $m \leq a \leq t-1$,

$$\beta_a = (i_a, j_a, g_{NA}) \quad \text{where} \quad \text{Res}(u, a, i_a) \quad \text{and} \quad \text{Res}(v, a, j_a).$$

Not all circuits can be represented exactly in this way because the residues of u and v (mod a) cannot be chosen independently for $a = m, m+1, m+2, \dots$. However:

Lemma 6.6.3. Let C be a NAND-circuit of size t with m inputs, and assume $t \leq \pi(t' - 1) - \pi(m - 1)$ for some t' .

Then there are $u, v \in \mathbb{N}$ such that $\underline{t'\text{-circuit}(u, v)}$ computes the same function as C .

Proof. Consider $A = \{ a \mid m \leq a \leq t' - 1 \text{ and } a \text{ is prime} \}$.

For each $a \in A$, let i_a and j_a with $0 \leq i_a, j_a < a$ be arbitrary.

By the Chinese Remainder Theorem [cf. NZ66] there are $u, v \in \mathbb{N}$ such that $\text{Res}(u, a, i_a)$ and $\text{Res}(v, a, j_a)$ for all $a \in A$.

$$\text{card}(A) = \pi(t' - 1) - \pi(m - 1) \geq t.$$

Therefore, i_a and j_a for $a \in A$ can be chosen so that the steps β_a for $a \in A$ of $\underline{t'\text{-circuit}(u, v)}$ mimic the circuit C . The steps β_k with $k \notin A$ and $k \geq m$ are irrelevant. \square

If C is a circuit of size $t - m$ with m inputs, if $x \in \{0, 1\}^m$, and $d \in \mathbb{N}$, then d represents the computation of C on x iff

$$\text{Bit}(d, i) = \xi_i(x) \quad \text{for} \quad 0 \leq i \leq t-1$$

where the $\{\xi_i\}$ are the associated functions of C (cf. Definition 6.2).

(E4) the output of $\beta_{2^k}, \xi_{2^k}(x)$, is 1 iff e divides z
 (iff $\text{fct}(z)(x) = 1$).

The next lemma describes properties of **Easy**(z).

Lemma 6.6.4. Let $z \in \mathbb{N}$.

- (i). **Easy**(z) is true iff there exist $u, v \in \mathbb{N}$ such that $(2^k + 1)$ -circuit(u, v) computes $\text{fct}(z)$.
- (ii). If **Easy**(z) is true, there is a NAND-circuit of size $2^k - m + 1$ which computes $\text{fct}(z)$.
- (iii). If **Easy**(z) is false, there is no NAND-circuit of size $2^{k-3}/k$ which computes $\text{fct}(z)$.

Proof. We let the reader check (i) by following the informal description of **Easy**(z) above. (ii) is immediate from (i).

To prove (iii), assume **Easy**(z) is false and suppose C is a NAND-circuit of size $t = 2^{k-3}/k$ which computes $\text{fct}(z)$.

Let $t' = 2^k + 1$. Now

$$t = 2^{k-3}/k \leq 2^{k-2}/k = 9m/\log m \quad (\text{because (2) of Theorem 6.6})$$

$$\text{implies } 2^{k-3}/k \geq 9m/\log m$$

$$(t - 1) = (m - 1) \quad (\text{by Fact 6.6.1}).$$

Therefore by Lemma 6.6.3, there are $u, v \in \mathbb{N}$ such that $(2^k + 1)$ -circuit(u, v) computes $\text{fct}(z)$. Part (i) of this lemma now contradicts the fact that **Easy**(z) is false. \square

After replacing the occurrences of Bit and Res by their definitions, we find

$$|\text{Easy}| = 179 + 6 \lfloor \log_{10} m \rfloor.$$

(Note: Using the abbreviation trick to eliminate multiple occurrences of Bit or Res does not yield a shorter formula In this case.)

Let $\varphi''_{\mathbf{x}}$ be the sentence

$$\begin{aligned} \exists \mathbf{z} \forall \mathbf{y} (\exists \mathbf{b} (\mathbf{z} = \mathbf{b} \cdot \omega(\mathbf{x})) \wedge \sim \text{Easy}(\mathbf{z}) \\ \wedge (\mathbf{y} < \mathbf{z} \Rightarrow \text{Easy}(\mathbf{y}))) . \end{aligned}$$

$\omega(\mathbf{x})$ is a decimal representation of $\text{int}(\mathbf{x})$, with leading zeroes appended if necessary to make the length of $\omega(\mathbf{x})$ be exactly $\lfloor (\log_{10} 2) m \rfloor + 1$.

There are at most $(2^k)^{2(2^k - m + 1)}$ NAND-circuits of size $2^k - m + 1$ with m inputs.

Constraint (1) of Theorem 6.6 implies $2^{2^{m-2}/m} > (2^k)^{2(2^k - m + 1)}$. Then Lemma 6.6.2 and Lemma 6.6.4(ii) together imply that there is some $\mathbf{z} \in \mathbb{N}$ such that $\text{Easy}(\mathbf{z})$ is false.

Thus there is precisely one $\mathbf{z}_0 \in \mathbb{N}$ such that

$$\forall \mathbf{y} (\sim \text{Easy}(\mathbf{z}_0) \wedge (\mathbf{y} < \mathbf{z}_0 \Rightarrow \text{Easy}(\mathbf{y}))) \text{ is true.}$$

Let $\mathbf{f}_0 = \text{fct}(\mathbf{z}_0)$.

By Lemma 6.6.4(iii) and by our remarks in §6.1 concerning the synthesis of Ω_{16} -circuits by NAND-circuits, it follows that the combinational complexity of \mathbf{f}_0 exceeds $(1/5) \cdot 2^{k-3}/k > 2^{k-6}/k$.

Also, $\varphi''_{\mathbf{x}}$ is true iff $\omega(\mathbf{x})$ divides \mathbf{z}_0 iff $\mathbf{f}_0(\mathbf{x}) = 1$.

Using the abbreviation trick to replace the two occurrences of Easy by one, we find $\varphi'_{\mathbf{x}}$ equivalent to $\varphi''_{\mathbf{x}}$ and

$$|\varphi'_{\mathbf{x}}| = 242 + \lfloor (\log_{10} 2) m \rfloor + 6 \lfloor \log_{10} m \rfloor .$$

Now $\varphi_{\mathbf{x}}$ is $\varphi'_{\mathbf{x}}$ padded with blanks if necessary to be of length exactly n .

As before, if τ is the transformation mapping x to $\hat{h}(\varphi_x)$ for an encoding h of Σ' , then the combinational complexity of τ is certainly bounded above by $2^{20}_m 3$.

The reader can now complete the proof that $C_{\infty}(\text{FIA})(n) > 2^{k-7}/k$ by following the proof of Theorem 6.4. The necessary facts are:

- (i). the combinational complexity of f_0 exceeds $2^{k-6}/k$;
- (ii). $f_0(\mathbf{x}) = 1$ iff $\varphi_{\mathbf{x}} \in \text{FIA}$; and (iii). by constraint (2) of the theorem, the combinational complexity of τ is $\leq 2^{k-7}/k$. \square

Chapter 7. Conclusion

We have demonstrated that efficient reducibility techniques can yield interesting lower bounds on the inherent computational complexity of a variety of decision problems from automata theory and logic. For several of these problems, such as the equivalence problem for star-free expressions (cf. §4.2) and the decision problems for the various logical theories discussed in Chapter 5, our results imply that any attempt to find an **efficient algorithm** for the problem is foredoomed.

Recent studies by coworkers (cf. [Fer74], [FR74], [Mey73], [Rac74], [Rob73]) of decision procedures for logical theories show that these reducibility methods are applicable to nearly all the classical decidable theories. Moreover ~~that~~ ^{with} the exception of the propositional calculus and certain theories resembling the first order theory of equality, all these decidable theories can be proved to require exponential or greater time.

Hopefully, both the general method of efficient reducibility and some of the particular techniques of efficiently **arithmetizing** Turing machines will extend to algebra, topology, number theory, and other areas where decision procedures arise, and will curtail wasted effort in searching for efficient procedures when none exist. The exhibition of provably difficult problems in these areas is one direction for further research.

Acknowledgement. Michael J. Fischer contributed to several discussions concerning this work, and his interest is appreciated.

Bibliography

- [AHU74] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., The Design and Analysis Of Computer Algorithms, to appear.
- [AU70] Aho, A.V., and Ullman, J.D., "A characterization of two-way deterministic classes of **languages**," J. Comput. Syst. Sci. **4**, 6 (Dec 1970), 523-538.
- [AU72] Aho, A.V., and Ullman J.D., The Theory Of Parsing, Translation, and Compiling, Vol. I: Parsing, Prentice-Hall, Englewood Cliffs, New Jersey, 1972.
- [Bl66] Blum, M., "Recursive function theory and speed of **computation**," Canadian Math. Bull. **9** (1966), 745-750.
- [Bl67] Blum, M., "A machine-independent theory of the complexity of recursive functions," J. ACM **14**, 2 (April 1967), 322-336.
- [Bl71] Blum, M., "On effective procedures for speeding up algorithms," J. ACM **18**, 2 (April 1971), 290-305.
- [Bo72] Book, R.V., "On languages accepted in polynomial time," SIAM J. Comput. **1**, 4 (Dec 1972), 281-287.
- [Brz62] Brzozowski, J.A., "A survey of regular expressions and their **applications**," IRE Trans. EC-11 (June 1962), 324-335.
- [Brz64] Brzozowski, J.A., "Derivatives of regular **expressions**," J. ACM **11**, 4 (Oct 1964), 481-494.
- [Buc60a] Büchi, J.R., "Weak second order arithmetic and finite automata," Zeit. f. Math. Log. and Grund. der Math. **6** (1960), 66-92.
- [Buc60b] Büchi, J.R., "On a decision method in restricted second order arithmetic," Proc. Internat. Congr. Logic, Method. and Philos. Sci. (1960), Stanford Univ. Press, Stanford, Cal., 1962, 1-11.
- [BGW70] Book, R.V., Greibach, S.A., and Wegbreit, B., "Time and **tape** bounded Turing acceptors and **AFL's**," J. Comput. Syst. Sci. **4** (1970), 606-621.

- [Co169] Cole, S.N., "Real-time computation by n-dimensional iterative arrays of finite state **machines**," IEEE Trans. C-18 (April 1969), 349-365.
- [Co71a] Cook, S.A., "The complexity of theorem proving **procedures**," Proc. 3rd ACM Symp. on Theory Of Computing (1971), 151-158.
- [Co71b] Cook, S.A., "**Characterizations** of pushdown machines in terms of time-bounded computers," J. ACM 18, 1 (Jan. 1971), 4-18.
- [Co73] Cook, S.A., "A hierarchy for nondeterministic time **complexity**," J. Comput. Syst. Sci. 7, 4 (Aug. 1973), 343-353.
- [CEW58] Copi, I.M., **Elgot**, C.C., and Wright, J.B., "**Realization** of events by logical **nets**," J. ACM 5 (April 1958), 181-196.
- [CR72] Cook, S.A., and Reckhow, R.A., "Time-bounded random access **machines**," Proc. 4th ACM Symp. on Theory of Computing (1972), 73-80.
- [Edm65] Edmonds, J., "Paths, trees and **flowers**," Canadian Jour. Math. 17 (1965), 449-467.
- [Ehr72] Ehrenfeucht, A., "**Practical decidability**," Report CU-CS-008-72, Dept. of Computer Science, Univ. of Colorado (Dec. 1972).
- [Elg61] Elgot, C.C., "Decision problems of finite automata design and related **arithmetics**," Trans. AMS 98 (1961), 21-51.
- [ER66] Elgot, C.C., and Rabin, M.O., "Decidability and undecidability of extensions of second (first) order theory of (generalized) successor," Jour. Symb. Logic 31, 2 (June 1966), 169-181.
- [Fer74] Ferrante, J., "**Some** upper and lower bounds on decision procedures in logic," Doctoral Thesis, Dept. of Mathematics, M.I.T., to appear 1974.
- [FM74] Fischer, M.J., and Meyer, A.R., personal **communication**.
- [FMR72] Fischer, P.C., Meyer, A.R., and Rosenberg, A.L., "Real-time simulation of multi-head tape units," J. ACM 19, 4 (Oct. 1972), 590-607.

- [FP74] Fischer, M.J., and Pippenger, N., to appear.
- [FR74] Fischer, M.J., and Rabin, M.O., "Super-exponential complexity of Presburger arithmetic," Proc. AMS Symp. on Complexity Of Real Computational Processes (1974), to appear; also, MAC Tech. Memo. 43, M.I.T., Project MAC, Cambridge, Mass. (Feb. 1974).
- [Gin67] Ginzburg, A., "A procedure for checking equality of regular expressions," J. ACM 14, 2 (April 1967), 355-362.
- [Gri71] Gries, D., Compiler Construction for Digital Computers, Wiley, New York, 1971.
- [GJS74] Garey, M.R., Johnson, D.S., and Stockmeyer, L.J., "Some simplified NP-complete problems," Proc. 6th ACM Symp. on Theory of Computing (1974), 47-63.
- [Har65] Harrison, M.A., Introduction to Switching and Automata Theory, McGraw-Hill, New York, 1965.
- [Hun73a] Hunt, H.B. III, "On the time and tape complexity of languages I," Tech. Report TR73-156, Dept. of Computer Science, Cornell University, (Jan. 1973).
- [Hun73b] Hunt, H.B. III, "The equivalence problem for regular expressions with intersection is not polynomial in tape," Tech. Report TR73-161, Dept. of Computer Science, Cornell University, (March 1973).
- [Hun73c] Hunt, H.B. III, "On the time and tape complexity of languages I," Proc. 5th ACM Symp. on Theory of Computing (1973), 10-19.
- [HR74] Hunt, H.B. III, and Rosenkrantz, D.J., "Computational parallels between the regular and context-free languages," Proc. 6th ACM Symp. on Theory Of Computing (1974), 64-74.
- [HS65] Hartmanis, J., and Stearns, R.E., "On the computational complexity of algorithms," Trans. AMS 117 (1965), 285-306.
- [HU69] Hopcroft, J.E., and Ullman, J.D., Formal Languages and Their Relation to Automata, Addison-Wesley, Reading, Mass., 1969.

- [Ib72] Ibarra, O.H., "A note concerning nondeterministic tape complexities," J. ACM 19, 4 (Oct. 1972), 608-612.
- [Jer72] Jeroslow, R.C., "On the stopping problem for computing machines with a time bound," SIGACT News, No. 15 (April 1972), 9-11.
- [Jon73] Jones, N.D., "Reducibility among combinatorial problems in $\log n$ space," Proc. 7th Annual Princeton Conf. on Information Sciences and Systems (1973), 547-551.
- [Kar72] Karp, R.M., "Reducibility among combinatorial problems," in Complexity Of Computer Computations, R.E. Miller and J.W. Thatcher, ed., Plenum Press, New York, 1972, 85-104.
- [Kle56] Kleene, S.C., "Representation of events in nerve nets and finite automata," in Automata Studies, Princeton Univ. Press, Princeton, New Jersey, 1956, 3-41.
- [Knu69] Knuth, D.E., The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Addison-Wesley, Reading, Mass., 1969.
- [Knu74] Knuth, D.E., "Postscript about NP-hard problems," SIGACT News 6, 2 (April 1974), 15-16.
- [KW70] Kameda, T., and Weiner, P., "On the state minimization of nondeterministic finite automata," IEEE Trans. C-19, 7 (July 1970), 617-527.
- [Lin73] Lind, J., "Computing in logarithmic space," Bachelor's Thesis, Dept. of Electrical Engineering, M.I.T., 1973.
- [Lup50] Lupanov, O.B., "On the synthesis of contact networks," Dokl. Akad. Nauk SSSR 70 (1950), 421-423.
- [LLS74] Ladner, R., Lynch, N., and Selman, A., "Comparison of polynomial-time reducibilities," Proc. 6th ACM Symp. on Theory of Computing (1974), 110-121.
- [LM74] Lind, J., and Meyer, A.R., "A characterization of log-space computable functions," to appear as a Project MAC Technical Report, 1974.

- [LSH65] Lewis, PM II, Stearns, R.E., and Hartmanis, J., "Memory bounds for recognition of context-free and context-sensitive languages," 6th IEEE Symp. on Switching Circuit Theory and Logical Design (1965), 191-202.
- [Mey73] Meyer, A.R., "Weak monadic second order theory of successor is not elementary-recursive," Boston Univ. Logic Colloquium Proc., to appear 1974; also MAC Tech. Memo 38, M.I.T., Project MAC, (1973).
- [Mey74] Meyer, A.R., ~~personal communication~~; 6.853 Lecture Notes, Dept. of Electrical Engineering, M.I.T., (1974).
- [Min67] Minsky, M.L., Computation: Finite, and Infinite Machines, Prentice-Hall, Englewood Cliffs, New Jersey, 1967.
- [MM71] Meyer, A.R., and McCreight, E.M., "Computationally complex and pseudo-random zero-one valued functions," in Theory of Machines and Computations, Academic Press, New York, 1971, 19-42.
- [MP71] McNaughton, R., and Papert, S., Counter-Free Automata, M.I.T. Press, Cambridge, Mass. 1971.
- [MS72] Meyer, A.R., and Stockmeyer, L.J., "The equivalence problem for regular expressions with squaring requires exponential space," Proc. 13th IEEE Symp. on Switching and Automata Theory (1973), 125-129.
- [MY60] McNaughton, R., and Yamada, H., "Regular expressions and state graphs for automata," IRE Trans. EC-9 (March 1960), 39-47.
- [NZ66] Niven, I., and Zuckerman, H.S., An Introduction to the Theory of Numbers, Wiley, New York, 1966.
- [Pet67] Péter, R., Recursive Functions, Academic Press, New York, 1967.
- [Rab60] Rabin, M.O., "Degree of difficulty of computing a function and a partial ordering of recursive sets," Tech. Report 2, Hebrew Univ., Jerusalem, Israel, (1960).
- [Rab69] Rabin, M O., "Decidability of second-order theories and automata on infinite trees," Trans. AMS 141 (1969), 1-35.

- [Rac74] Rackoff, C., "Complexity of some logical theories," Doctoral Thesis, Dept. of Electrical Engineering, M.I.T., to appear 1974.
- [Rit63] Ritchie, R.W., "Classes of predictably computable functions," Trans. AMS ~~106~~ (1963), 139-173.
- [Rob73] Robertson, E.L., "Structure of complexity in the weakmonadic second-order theories of the natural numbers," Research Report CS-73-31, Dept. of Applied Analysis and Computer Science, Univ. of Waterloo, (Dec. 1973); also Proc. 6th ACM Symp. on Theory of Computing (1974), 161-171.
- [Rog67] Rogers, H. Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967.
- [RF65] Ruby, S., and Fischer, P.C., "Translational methods and computational complexity," 6th IEEE Symp. on Switching Circuit Theory and Logical Design (1965), 173-178.
- [RS59] Rabin, M.O., and Scott, D., "Finite automata and their decision problems," IBM J. Research and Development **3** (1959), 115-125; also in Sequential Machines: Selected Papers, E.F. Moore, ed., Addison-Wesley, Reading, Mass., 1964, 63-91.
- [Sah72] Sahni, S., "Some related problems from network flows, game theory, and integer programming," Proc. 13th IEEE Symp. on Switching and Automata Theory (1972), 130-138.
- [Sal69] Salomaa, A., Theory of Automata, Pergamon Press, New York, 1969.
- [Sav70] Savitch, W. J., "Relationships between nondeterministic and deterministic tape complexities," J. Comput. Syst. Sci. **4**, 2 (April 1970), 177-192.
- [Sav72] Savage, J.E., "Computational work and time on finite machines," J. ACM ~~19~~, 4 (Oct. 1972), 660-674.
- [Sav74] Savage, J.E., "The complexity of computing," JPL Tech. Report, draft, June, 1974, Chapter 2.
- [Set73] Sethi, R., "Complete register allocation problems," Proc. 5th ACM Symp. on Theory of Computing (1973), 182-195.

- [Sho67] Shoenfield, J.R., Mathematical Logic, Addison-Wesley, Reading, Mass., 1967.
- [SFM73] Seiferas, J.I., Fischer, M.J., and Meyer, A.R., "Refinements of the nondeterministic time and space **hierarchies**," Proc. 14th IEEE Symp. on Switching and Automata Theory (1973), 130-137.
- [SHL65] Stearns, R.E., Hartmanis, J., and Lewis, P.M. II, "**Hierarchies of memory limited computations**," 6th IEEE Symp. on Switching Circuit Theory and Logical Design (1965), 179-190.
- [SM73] Stockmeyer, L.J., and Meyer, A.R., "**Word problems requiring exponential time: preliminary report**," Proc. 5th ACM Symp. on Theory Of Computing (1973), 1-9.
- [SS63] Shepherdson, J.C., and Sturgis, H. E., "**Computability of recursive functions**," J. ACM 10, 2 (April 1963), 217-255.
- [Tra70] Trachtenbrot, B.A., "**On autoreducibility**," Soviet Math. Dokl. 11, 3 (1970), 814-817.
- [U1173] Ullman, J.D., "**Polynomial complete scheduling problems**," 4th Symp. on Operating System Principles (1973), 96-101.
- [Win65] Winograd, S., "**On the time required to perform addition**," J. ACM 12, 2 (April 1965), 277-285.
- [Yam62] Yamada, H., "Real-time computation and recursive functions not real-time **computable**," IRE Trans. EC-11 (1962), 753-760.
- [You~~] Younger, D.H., "**Recognition and parsing of context-free languages in time n^3** ," Information and Control 10 (1967), 189-208.
- [Sto74] Stockmeyer, L. J. , "**The complexity of decision problems in automata theory and logic**," Doctoral Thesis, Dept. of Electrical Engineering, M. I. T. (June, 1974).

Appendix I. Notation.

\emptyset	The empty set.
$A - B$	$\{ x \mid x \in A \text{ and } x \notin B \}$ (set difference).
$A \oplus B$	$(A - B) \cup (B - A)$ (symmetric difference).
2^A	The set of all subsets of the set A .
$\text{card}(A)$	The cardinality of the set A .
$A^{\times k}$	$A \times A \times A \times \dots \times A$ (k times).
λ	The empty word.
$ \omega $	The length of the word ω .
$\omega\tau$ or $\omega \cdot \tau$	Concatenation of words ω and τ .
Σ^*	The set of all words over the alphabet Σ including λ .
Σ^+	$\Sigma^* - \{\lambda\}$.
Σ^k	$\{ w \in \Sigma^* \mid w = k \}$, for positive integer k .
$\Sigma^{\leq k}$	$\{ w \in \Sigma^* \mid w \leq k \}$.
σ^k	The word $\sigma\sigma\sigma \dots \sigma$ of length k .
$\text{bin}(k)$	The binary representation of positive integer k .
\mathbb{N}	The nonnegative integers.
\mathbb{N}^+	The positive integers.

\mathbb{Z}	The integers.
\mathbb{Q}	The rational numbers.
\mathbb{Q}^+	The positive rational numbers.
$\lfloor r \rfloor$	The integer part of real r .
$\lceil r \rceil$	The least integer z such that $z \geq r$.
$\log r$	$\log_2 r$.
$g(k, r)$	$2^2 \cdot \dots \cdot 2^r \Big\}^k$

Appendix II. Some Properties of logspace.

. An **IOTM** M computes a function $f: (\Sigma^*)^{Xn} \rightarrow A^*$ of n variables if M computes a function $f': (\Sigma \cup \{\#\})^* \rightarrow A^*$ where $\# \notin \Sigma$ and

$$f'(x_1 \# x_2 \# x_3 \# \dots \# x_n) = f(x_1, x_2, x_3, \dots, x_n) \text{ for all } x_1, x_2, \dots, x_n \in \Sigma^*.$$

Definition. A function $f: (\Sigma^*)^{X(n+1)} \rightarrow A^*$ of **dl** variables is defined from functions $g: (\Sigma^*)^{Xn} \rightarrow A^*$ and $h_1, h_2: (\Sigma^*)^{X(n+2)} \rightarrow A^*$ by two sided recursion of concatenation if f satisfies

$$\begin{aligned} \text{and} \quad & f(\bar{x}_n, \lambda) = g(\bar{x}_n) \\ & f(\bar{x}_n, y\sigma) = h_1(\bar{x}_n, y, \sigma) \cdot f(\bar{x}_n, y) \cdot h_2(\bar{x}_n, y, \sigma) \\ & \text{for all } \bar{x}_n \in (\Sigma^*)^{Xn}, y \in \Sigma^*, a \in \Sigma. \end{aligned}$$

Fact AII.1 [Lin73], [LM74]. **logspace** is closed under explicit transformation (substituting constants and renaming or identifying variables), composition, and two sided recursion of concatenation.

Fact AII.2 [Lin73], [LM74].

- (1). The concatenation function belongs to **logspace**.
- (2). For any alphabet Σ , $f \in \mathbf{logspace}$ where

$$f(x) = \mathbf{bin}(|x|) \quad (\text{the binary representation of } |x|)_{*} \\ \text{for all } x \in \Sigma^*.$$

- (3). Binary addition, **monus**, and multiplication belong to **logspace**. That is, there are functions $f_+, f_-, f_\times \in \mathbf{logspace}$

such that $f_{@}(\mathbf{bin}(m_1), \mathbf{bin}(m_2)) = \mathbf{bin}(m_1 @ m_2)$

for $@ \in \{+, -, \times\}$ and all $m_1, m_2 \in \mathbb{N}$.

Minus is defined as $m_1 \dot{-} m_2 = \begin{cases} m_1 - m_2 & \text{if } m_1 \geq m_2 \\ 0 & \text{otherwise} \end{cases} .$

Lemma AII.3. Let $p(n)$ be a polynomial with integer coefficients, let Σ be a finite alphabet, and let $\$$ be a symbol,

Then $f \in \mathbf{logspace}$ where

$$f(x) = \$p(|x|) \dot{-} 0 \quad \text{for all } x \in \Sigma^* .$$

The reader may verify **Lemma AII.3**. Fact **AII.2** (2) and (3) may be useful.