MIT/LCS/TM-52

COMPUTING IN LOGARITHMIC SPACE

John C. Lind

September 1974

TM52

COMPUTING IN LOGARITHMIC SPACE

John C. Lind

PROJECT MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Cambridge Massachusetts 02139

ACKNOWLEDGEMENT

I would like to express my appreciation to Professor Albert R. Meyer for introducing the question and making key suggestions which made this work possible. His generous encouragemnt and enthusiasm have been experienced by many and always stimulate appreciation and respect.

COMPUTING IN LOGARITHMIC SPACE, John C. Lind

ABSTRACT

The set logspace, of logarithmic space computable string functions is defined. It is easily seen that logspace $\subseteq$ polytime, the set of polynomial time computable functions. logspace is shown to equal $\mathcal{L}$, the smallest class of recursive string functions containing concatenation and the equality function, and closed under explicit transformations, substitution of a function for a variable and two restricted types of recursion on notation. The first is called recursion of concatenation and only allows top level concatenation of the value of the recursive call. The second, called log bounded recursion on notation, will only define string functions whose length is bounded by O(log n) on arguments of length n. Some additional closure properties of logspace are also described.

TABLE OF CONTENTS

INTRODUCTION

It is useful to be able to prove that a certain well defined class of recursive functions is identical to the set of functions computed by a particular type of abstract machine. Such a result provides a machine independent characterization of the functions computed by those machines. In addition if the means to define the type of machine includes some limitation to its computational resources, as is often the case, then an upper bound on the computational complexity of that class of functions has been achieved. For example, R. W. Ritchie established [R] that the class $\mathcal{E}^2$ in the Grzegorcyk hierarchy [G], which is the smallest class of number theoretic functions containing addition and multiplication and closed under limited recursion and certain substitution operations, is identical to those functions computed by a Turing machine which uses an amount of space bounded by $O(n)$ on inputs of length n.

In the case of functions computable by a Turing machine which uses an amount of space bounded by $O(\log n)$ on inputs of length n, there is additional motivation for such a result. It has been shown by Cook [C] that any recognition problem decidable by a polynomial time bounded non-deterministic Turing machine can be polynomial time reduced to that of recognizing satisfiable propositional fomulas, where polynomial time reduced means that the translation algorithm can be computed in deterministic polynomial time. In fact a large class of decision problems can play the role of satisfiability [K]. Such problems are called complete in non-deterministic polynomial time. Since it is easy to show that Turing machines using logarithmic space operate in polynomial time (Theorem 1), logarithmic space reducibility is a refinement of polynomial

time reducibility. Neil Jones has discovered two decision problems which are complete in non-deterministic logarithmic space. In addition he states that all the polynomial time reductions he has examined in [C] and [K] are also logarithmic space reductions [J1]. Thus a machine independent characterization of deterministic logarithmic space computable functions would provide a programming tool for the description of other logarithmic space reduction algorithms.

The main result of this paper is to show that the set of logarithmic space computable functions is identical to the smallest set of recursive string functions which (1) contain concatenation and the equality function, and (2) are closed under explicit transformations, substitution of a function for a variable and two restricted types of recursion on notation. Bennett [B] and Jones [J2] have used recursion on notation to establish results about rudimentary functions. By restricting the form of recursion on notation so that we can only either combine the values of recursive calls by concatenation, or define string functions of length bounded by $O(\log n)$ on arguments of length n, we can maintain logarithmic space computability and still be able to describe all logarithmic space computable functions.

TERMINOLOGY AND NOTATION

The following definitions make precise the notion of character string function which most of the functions discussed in this paper are. An alphabet is any finite non-empty set of elements called characters or symbols. A string x over some alphabet Σ is any finite sequence, $x = a_1a_2\cdots a_n$, of symbols in Σ. The string containing no symbols is called the empty string and is denoted λ. The length of a string $x = a_1a_2\cdots a_n$ over Σ is written $|x|$ and equals n the number of symbols in x (the notation $|A|$ will also be used to denote the cardinality of a set A, but the meaning should be clear from context). λ is the only string of length 0. The set of all finite strings over Σ including λ is denoted Σ^*.

For any $n \geq 1$ the set $(\Sigma^*)^n$ is the set of n-tuples of strings over Σ. We will write n-tuples of strings with their components separated by blanks to suggest the convention which will be used to code them as input to a Turing machine. The n-tuple $x_1\ x_2\ \cdots\ x_n$ will be abbreviated as $\overline{x_n}$ throughout this paper and by definition has length $|\overline{x_n}| = |x_1| + |x_2| + \cdots + |x_n| + n - 1$. We will also use the abbreviation $\overline{{}_jx_n}$ to represent $x_j\ x_{j+1}\ \cdots\ x_n$ the last n-j+1 components of $\overline{x_n}$.

An n-place predicate over Σ is a subset P of $(\Sigma^*)^n$. We will say $P(\overline{x_n})$ is true if $\overline{x_n} \in P$ and false if $\overline{x_n} \in (\Sigma^*)^n - P$. The following 2-place predicates over any alphabet Σ will be used extensively throughout this paper. Let $x = a_1a_2\cdots a_n$ and $y = b_1b_2\cdots b_m$ be arbitrary strings over Σ. The predicate xBy (read x begins y) will be true iff $m \geq n$ and $a_i = b_i$ for all $1 \leq i \leq n$, i.e. x is composed of exactly the same symbols which begin y. Similarly xEy (read x ends y) is true iff $m \geq n$

and $a_i = b_{i+m-n}$ for all $1 \leq i \leq n$, i.e. x is composed of exactly the same symbols with which y ends. In general xPy (read x is-a-part-of y) is true iff $m \geq n$ and there is some $1 \leq j \leq m$ such that $a_i = b_{i+j-1}$ for all $1 \leq i \leq n$. This means that x is composed of exactly the same symbols which occur in y starting with b_j and ending with b_{j+n-1}. If xBy and $m = n$ then x is the same string as y and we write x=y, which is the equality predicate over any alphabet Σ.

An n-variable function over Σ is a subset f of $(\Sigma^*)^{n+1}$ such that if $\overline{x_n}\ y \in f$ then for any other $\overline{x_n}\ z \in f$ with the same first n components, y=z. We say that y is the value of f at $\overline{x_n}$ which is written $f(\overline{x_n}) = y$. That f is an n-variable function over Σ will be denoted $f:(\Sigma^*)^n \to \Sigma^*$. A characteristic function for an n-place predicate P is an n-variable function p over the same alphabet Σ as P such that $p(\overline{x_n}) = \lambda$ iff $P(\overline{x_n})$ is false. If $P(\overline{x_n})$ is true then the value of p at $\overline{x_n}$ is any string over Σ not equal to λ. Notice that every n-variable function is a characteristic function for some n-place predicate and that every predicate has many characteristic functions. Whenever it is necessary we will associate an n-place predicate P with its standard characteristic function $c_P:(\Sigma^*)^n \to \Sigma^*$ such that given an enumeration of Σ, whenever P is true c_P has the value σ_1, the first symbol in Σ.

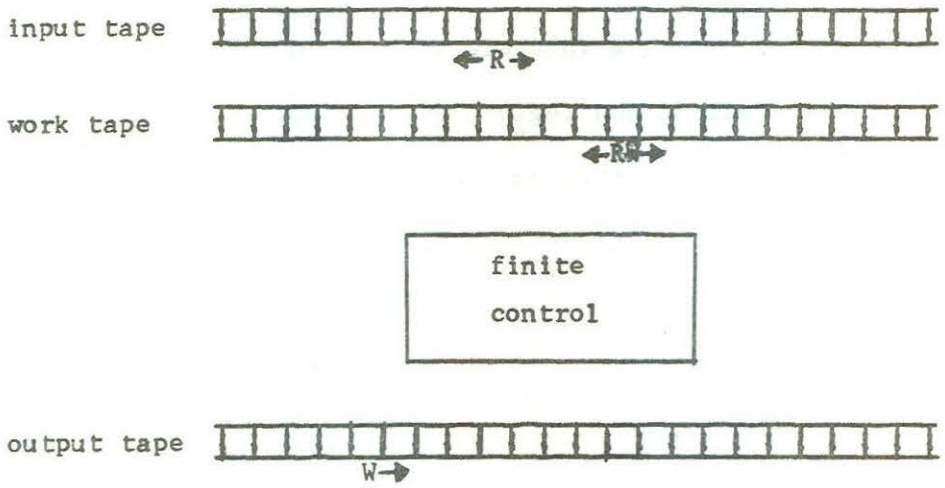
The concatenation function over any alphabet Σ will be the 2-variable function con over Σ such that if $x = a_1a_2\cdots a_n$ and $y = b_1b_2\cdots b_m$ are any strings in Σ^* then $con(x\ y) = a_1a_2\cdots a_nb_1b_2\cdots b_m$. We will almost always abbreviate con(x y) as xy (in order to avoid confusion, multiplication of numbers will always be written x · y). Note that $x\lambda = \lambda x = x$ and that concatenation is obviously associative which makes Σ^* into a monoid.

Some of the character string functions in this paper interpret one or more of their arguments as the representation of a non-negative integer. We will agree that such functions will be over alphabets containing the symbols 0 and 1, and that non-negative integers will be represented in reverse binary notation where the least significant bit appears first. If K is some non-negative integer then $\overleftarrow{K}$ denotes the reverse binary coding of K. In addition we will use the notation $\langle s \rangle$ to be the non-negative integer for which the string s is a reverse binary coding.

Some miscellaneous conventions follow. We will be using the numeric function $\log_2 n$ on non-negative integers with the understanding that if $n = 0$ then $\log_2 n$ is defined and $\log_2 0 = 0$. The notation $\lceil x \rceil$ will mean the smallest integer greater than or equal to x.

DEFINITION OF logspace

In order to formally discuss a Turing machine computation which may only require a number of temporary tape squares less than the length of the input, it is useful to define a special class of Turing machines which have a separate work tape. One such configuration, which we will call a work tape Turing machine, has the input placed on a two-way read only input tape, intermediate results written on a two-way read and write work tape, and the output appearing on a one-way write only output tape.



We will assume that by convention the finite control for a work tape Turing machine is designed to never move the work tape head to the left of the initially scanned square. Such a restriction does not change the computations which can be performed by a Turing machine [H, p2-42].

Formally a work tape Turing machine (henceforth written WTTM or simply machine) M is an ordered 7-tuple of sets

$$M = (I, W, V, Q, q_0, F, \delta)$$

where I, W and V are the finite sets of input, work and output tape symbols respectively and β_i and β_w will be the input and work tape blank

symbols respectively; Q is the finite set of control states; $q_0 \in Q$ is the start state; $F \subseteq Q$ is the set of final states; and

$$\delta : (Q-F) \times I \times W \rightarrow Q \times \{-1,0,+1\} \times W \times \{-1,0,+1\} \times (V \cup \{\lambda\})$$

is the transition function. For convenience of notation

$$\delta(q,i,w) = (ST(q,i,w),\ IT(q,i,w),\ WS(q,i,w),\ WT(q,i,w),\ OS(q,i,w))$$

where $ST:(Q-F)\times I\times W \rightarrow Q$ is the state transition function; $IT:(Q-F)\times I\times W \rightarrow \{-1,0,+1\}$ is the input tape transition function; $WS:(Q-F)\times I\times W \rightarrow W$ is the symbol written on the work tape; $WT:(Q-F)\times I\times W \rightarrow \{-1,0,+1\}$ is work tape transition function; and $OS:(Q-F)\times I\times W \rightarrow (V \cup \{\lambda\})$ is the symbol written on the output tape where λ means no symbol is written and the write head is not advanced. It is agreed by convention that V does not contain the output tape blank symbol, i.e. M cannot write blanks on its output tape.

When a WTTM M is placed in an initial configuration, the input tape contains a finite number of non-blank tape squares, the input tape head is scanning a square somewhere to the left of the leftmost non-blank input square, the work tape and output tape are entirely blank with their tape heads placed in arbitrary positions and the finite control is in the start state of M. The <u>input</u> to a WTTM M is the string over the input tape alphabet composed of the symbols appearing on each square of the input tape beginning with the square to the right of the initially scanned square and ending with the square just before the n-th blank to the right of the initially scanned square where n is some constant depending only on M. We will interpret the input to M as an n-tuple of string over $I-\{\beta_i\}$ where the components of the n-tuple are separated by occurances of the input blank symbol β_i. It is agreed by convention that a WTTM never scans on its input tape to the left of the intitially scanned square (which

must be blank) or beyond the n-th blank to the right of the initially scanned square.

Beginning with an initial configuration a WTTM M makes a sequence of steps or moves determined by δ the transition function of M. Each move involves reading the current input and work tape symbols scanned and using this information together with the current control state to assume a new control state, write a new work tape symbol on the square scanned, move the input tape and work tape tape heads at most one square in either direction, and possibly write an output symbol on the output tape moving the tape head one square to the right if a symbol is written.

Proceeding formally, an instantaneous description (henceforth ID) of a WTTM M is the ordered pair (k, α) where k is the current position of the input head numbered from the left end of the input and $\alpha = rq\sigma s$ such that, if W is the work tape alphabet then $r,s \in W^*$ and $\sigma \in W$. $r\sigma s$ is the entire (necessarily finite) contents of the work tape scanned so far and q is the current control state of M and σ is the symbol on the square scanned by the work tape head. The ID of any initial configuration for any WTTM M is $(0, q_0\beta_w)$ where q_0 is the start state of M and β_w is the work tape blank symbol.

A computation of a WTTM M on input $x \in I^*$ is the finite sequence of ID's $id_0, id_1, \ldots, id_N$ such that:

1. $id_0 = (0, q_0\beta_w)$ the ID of the initial configuration and
2. Letting x_j denote the j-th symbol of x numbered from the left end of x with $x_0 = x_{|x|+1} = \beta_i$ the input blank symbol, if $id_k = (j, r\sigma_1 q\sigma_2 s)$ where $r,s \in W^*$, $\sigma_1 \in (W \cup \{\lambda\})$ and $\sigma_2 \in W$ then

if $WT(q,x_j,\sigma_2) = 0$

then $id_{k+1} = (j+IT(q,x_j,\sigma_2),\ r\sigma_1 ST(q,x_j,\sigma_2)WS(q,x_j,\sigma_2)s)$

if $WT(q,x_j,\sigma_2) = -1$

then $id_{k+1} = (j+IT(q,x_j,\sigma_2),\ rST(q,x_j,\sigma_2)\sigma_1 WS(q,x_j,\sigma_2)s)$

if $WT(q,x_j,\sigma_2) = +1$

then $id_{k+1} = (j+IT(q,x_j,\sigma_2),\ r\sigma_1 WS(q,x_j,\sigma_2)ST(q,x_j,\sigma_2)un\lambda(s))$

where

$$un\lambda(s) = \begin{cases} s & \text{if } s \neq \lambda \\ \beta_w & \text{if } s = \lambda \end{cases}$$

extends the worktape representation with a blank if necessary.

A <u>halting computation</u> of a WTTM M on input x is a computation $id_0, id_1, \ldots, id_N$ such that:

1. $id_N = (j, rq_f s)$ where $q_f \in F$ the set of final states of M and
2. For all $k < N$, if $id_k = (j, rqs)$ then $q \notin F$.

The <u>time</u> used by M on input x is denoted $t_M(x)$ and equals N, the length of the halting computation of M on input x. It is undefined if M does not halt on input x, i.e., if there exists no halting computation of M on input x. The <u>space</u> used by M on input x is denoted $s_M(x)$ and equals the maximum number of worktape square scanned in the halting computation of M on input x. It is undefined if M does not halt on input x. If $id_0, id_1, \ldots, id_N$ is the halting computation of M on input x then

$$s_M(x) = \max_{0 \leq k \leq N} \{|rs| \text{ such that } id_k = (j, rqs),\ r,s \in W^* \text{ and } q \in Q\}$$

The <u>output</u> of the halting computation of WTTM M on input x is the string over the output tape alphabet V of M composed of the non-blank contents of the output tape when M halts. If $id_0, id_1, \ldots, id_N$ is a halting computation of M on input x then define

$\text{Output}_M(x) = \text{out}(id_0)\text{out}(id_1)\cdots\text{out}(id_{N-1})$

where if $id_k = (j, rq\sigma s)$ then $\text{out}(id_k) = OS(q, x_j, \sigma)$ the value of OS, the output symbol function, at this step in the computation of M on input x.

Since by convention a WTM M will scan only as far as the n-th blank to the right of the initially scanned input square for some n fixed by M, we can view M as computing an n-variable function over $I-\{\beta_i\}$, its input alphabet without the input blank symbol β_i. On input $x = x_1\beta_i x_2\beta_i\cdots\beta_i x_n$ (which henceforth will be written $\overline{x_n} = x_1\ x_2\ \cdots\ x_n$ to conform to the string notation for n-tuples) where each $x_j \in (I-\{\beta_i\})^*$, M <u>computes the n-variable function</u> $f_M:((I-\{\beta_i\})^*)^n \rightarrow V^*$ such that for all $\overline{x_n}$ over $I-\{\beta_i\}$

$$f_M(\overline{x_n}) = \begin{cases} \text{Output}_M(\overline{x_n}) & \text{if there is a halting computation of M on input } \overline{x_n} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that any of the x_j's in $\overline{x_n}$ may be λ, in which case there will be two adjacent blanks in the input to M. If M halts on any input $\overline{x_n}$ over $I-\{\beta_i\}$ then M computes a <u>total function</u>. The restriction that a WTM M interprets any input it starts on as an n-tuple over $I-\{\beta_i\}$ for some fixed n determined by M is not critical since none of the functions we will consider in this paper have an unbounded number of arguments. The addition of endmarkers to the input of M would eliminate this restriction but that technique seems unnecessary.

An n-variable function f over some alphabet Σ is <u>WTM computable</u> iff there is some WTM M with input alphabet $I \subseteq \Sigma \cup \{\beta_i\}$ and output alphabet $V \subseteq \Sigma$ such that M computes $f:(\Sigma^*)^n \rightarrow \Sigma^*$.

We now define logspace as the set of total WTM computable functions which use an amount of space bounded by O(log m) on inputs of length m. For definiteness, an n-variable function f over some alphabet Σ is a member of logspace (written $f \in$ logspace) iff there exists some WTM M which computes f and some constants K_1 and K_2 fixed by M such that

$$s_M(\overline{x_n}) \leq K_1 \cdot \log_2|\overline{x_n}| + K_2 \quad \text{for all } \overline{x_n} \text{ over } \Sigma.$$

Note that $\overline{x_n}$ interpreted as a character string input to a WTM has length $|\overline{x_n}| = |x_1| + |x_2| + \cdots + |x_n| + n - 1$. Similarly polytime is the set of total WTM computable functions which use an amount of time bounded by a polynomial in the length of the input. The n-variable function f over some alphabet Σ is a member of polytime iff there is some WTM M which computes f and some polynomial p fixed by M such that $t_M(\overline{x_n}) \leq p(|\overline{x_n}|)$ for all $\overline{x_n}$ over Σ.

Theorem 1: logspace $\subseteq$ polytime

proof: If $f \in$ logspace is an n-variable function over some alphabet Σ then there is some WTM M which computes f and

$$s_M(\overline{x_n}) \leq K_1 \cdot \log_2|\overline{x_n}| + K_2$$

for all $\overline{x_n}$ over Σ and constants K_1 and K_2 fixed by M.

Let $id_0, \ldots, id_N$ be the halting computation of M on some input $\overline{x_n}$ over Σ. The ID's $id_0, \ldots, id_N$ must be unique, for if $id_i = id_j$ for some $i < j$, then because ID id_{k+1} depends only on id_k and the fixed input $\overline{x_n}$ according to the transition function of M, an inductive argument shows $id_i = id_j \rightarrow id_{i+m} = id_{j+m}$ for all $m \leq N-j$. Since $j \leq N$, $j+k = N$ for some k therefore

$$id_{i+k} = id_{j+k} = id_N = (\ell, rqs) \quad \text{some } q \in F.$$

But $i+k < j+k = N$ which contradicts the condition in the definition of a halting computation that id_N be the first ID which contains a final state. Therefore since i and j were arbitrary above, $id_i \neq id_j$ for all $i \neq j$.

Let A be the set of all possible ID's of M on input $\overline{x_n}$.

$$A = \{(\ell, rqs) \text{ such that } (0 \leq \ell \leq |\overline{x_n}|+1) \wedge (q \in Q) \wedge (|rs| \leq s_M(\overline{x_n}))\}$$

$$|A| \leq (|\overline{x_n}| + 2) \cdot |Q| \cdot s_M(\overline{x_n}) \cdot |W|^{s_M(\overline{x_n})}$$

$$\leq (|\overline{x_n}| + 2) \cdot |Q| \cdot (K_1 \cdot \log_2|\overline{x_n}| + K_2) \cdot (2^{\log_2|W|})^{(K_1 \cdot \log_2|\overline{x_n}| + K_2)}$$

$$\leq (|\overline{x_n}| + 2) \cdot |Q| \cdot (K_1 \cdot |\overline{x_n}| + K_2) \cdot |\overline{x_n}|^{K_1 \cdot \log_2|W|} \cdot |W|^{K_2}$$

$$\leq p(|\overline{x_n}|)$$

for some polynomial p of degree $\lceil K_1 \cdot \log_2|W| \rceil + 2$. Note that $\{id_0, \ldots, id_N\} \subseteq A$ and that $id_i \neq id_j$ for all $i \neq j$ implies that $|\{id_0, \ldots, id_N\}| = N + 1$. Therefore

$$t_M(\overline{x_n}) = N \leq |\{id_0, \ldots, id_N\}| \leq |A| \leq p(|\overline{x_n}|)$$

and since $\overline{x_n}$ was arbitrary we conclude that $f \in$ <u>polytime</u> for the same WTM M.

<u>Corollary 2</u>: If $f \in$ <u>logspace</u> is an n-variable function over some alphabet Σ then there exist constants K_1 and K_2 such that $|f(\overline{x_n})| \leq |\overline{x_n}|^{K_1} + K_2$ for all $\overline{x_n}$ over Σ.

proof: By Theorem 1, $f \in$ <u>polytime</u> hence there is a WTM M which computes f and a polynomial p such that

$$t_M(\overline{x_n}) \leq p(|\overline{x_n}|) \qquad \text{for all } \overline{x_n} \text{ over } \Sigma.$$

For any polynomial p there exist constants K_1 and K_2 such that

$p(z) \leq z^{K_1} + K_2$ for all z (let K_1 be greater than the degree of p and let K_2 equal the maximum value of p(z) before z^{K_1} dominates p).

Therefore

$$|f(\overline{x_n})| = |\text{Output}_M(\overline{x_n})| \leq t_M(\overline{x_n}) \leq p(|\overline{x_n}|) \leq |\overline{x_n}|^{K_1} + K_2.$$

DEFINITION OF $\mathcal{L}$

We now turn to the task of defining a class of recursive string functions $\mathcal{L}$ which will be shown later to be equal to logspace, the class of logarithmic space computable function. A standard inductive definition of $\mathcal{L}$ from the base functions con (string concatenation) and $c_=$ (the standard characteristic function of the equality predicate) for any alphabet Σ, and the following functional operations will be used.

A function $f:(\Sigma^*)^n \to \Sigma^*$ is an explicit transformation of the function $g:(\Sigma^*)^n \to \Sigma^*$ if

$$f(\overline{x_n}) = g(\overline{\xi_m})$$

where each of the ξ_i is either some x_j in $\overline{x_n}$ or a constant string over Σ. It can be shown that f is an explicit transformation of g iff f can be obtained from g by a finite sequence of substitutions of a variable for a constant, interchanging two variables, identifying two variables and adding a redundant variable [W, p16].

A function $f:(\Sigma^*)^{n+m-1} \to \Sigma^*$ is defined by substitution of a function for a variable (henceforth simply substitution) from functions $g:(\Sigma^*)^n \to \Sigma^*$ and $h:(\Sigma^*)^m \to \Sigma^*$ if

$$f(\overline{x_{i-1}\ y_m\ x_{i+1}^n}) = g(\overline{x_{i-1}\ h(y_m)\ x_{i+1}^n}).$$

Explicit transformation and substitution can also be extended to operate on a predicate by applying the above functional operations to a characteristic function of the defining predicate to arrive at a characteristic function of the defined predicate.

A function $f:(\Sigma^*)^{n+1} \to \Sigma^*$ is defined by recursion on notation from functions $g:(\Sigma^*)^n \to \Sigma^*$ and $h:(\Sigma^*)^{n+3} \to \Sigma^*$ if f satisfies

$$f(\overline{x_n}\ \lambda) = g(\overline{x_n})$$

$$f(\overline{x_n}\ y\sigma) = h(\overline{x_n}\ y\ \sigma\ f(\overline{x_n}\ y))$$

where σ ranges over symbols in Σ. Note that for any arguments $\overline{x_n}\ y$ in Σ^*, $f(\overline{x_n}\ y)$ can be effectively determined given $y = \sigma_1\sigma_2\cdots\sigma_m$ in $m+1$ steps by evaluating $f(\overline{x_n}\ \lambda)$, $f(\overline{x_n}\ \sigma_1)$, $f(\overline{x_n}\ \sigma_1\sigma_2)$, $\cdots$, and $f(\overline{x_n}\ \sigma_1\sigma_2\cdots\sigma_m) = f(\overline{x_n}\ y)$.

We will actually use the following two restricted forms of recursion on notation. The first will be called <u>recursion of concatenation</u> where h takes the special form $h(\overline{x_n}\ y\ \sigma\ z) = (z)h'(\overline{x_n}\ y\ \sigma)$. In other words $f(\Sigma^*)^{n+1} \to \Sigma^*$ is defined by recursion of concatenation from functions $g:(\Sigma^*)^n \to \Sigma^*$ and $h':(\Sigma^*)^{n+2} \to \Sigma^*$ if f satisfies

$$f(\overline{x_n}\ \lambda) = g(\overline{x_n})$$

$$f(\overline{x_n}\ y\sigma) = f(\overline{x_n}\ y)h'(\overline{x_n}\ y\ \sigma)$$

where σ ranges over symbols in Σ.

The second special form of recursion on notation will be called <u>log bounded recursion on notation</u> (henceforth <u>bounded recursion</u>) where the length of the defined function $f(\overline{x_n}\ y)$ is bounded by $O(\log|\overline{x_n}\ y|)$. For definiteness, $f:(\Sigma^*)^{n+1} \to \Sigma^*$ is defined by bounded recursion from the functions $g:(\Sigma^*)^n \to \Sigma^*$ and $h:(\Sigma^*)^{n+3} \to \Sigma^*$ if f satisfies

$$f(\overline{x_n}\ \lambda) = g(\overline{x_n})$$

$$f(\overline{x_n}\ y\sigma) = h(\overline{x_n}\ y\ \sigma\ f(\overline{x_n}\ y))$$

$$|f(\overline{x_n}\ y)| \leq K_1 \cdot \log_2|\overline{x_n}\ y| + K_2$$

where again σ ranges over symbols in Σ and K_1 and K_2 are some fixed constants.

$\mathcal{L}$ is defined to be the smallest set of functions containing concatenation and the equality function, which is closed under explicit

transformations, substitutions, recursion of concatenation and bounded recursion. Formally $f \in \mathcal{L}$ iff there is some alphabet Σ and a finite sequence $f_1, f_2, \ldots, f_n$ of functions over Σ such that $f = f_n$ and every f_i is either con, the concatenation function over Σ; or $c_=$, the standard characteristic function of the equality predicate over Σ; or f_i is defined from some previous function in the sequence by an explicit transformation, substitution, recursion of concatenation or bounded recursion.

We will say that an n-place predicate P over some alphabet Σ is in $\mathcal{L}$ if some n-variable function $\mathfrak{p}$ over Σ which is a member of $\mathcal{L}$ is a characteristic function for P. If there is no ambiguity we will often write the predicate P where some characteristic function of P should appear.

PROOF THAT $\mathcal{L} \subseteq$ logspace

The following lemmas establish that logspace contains the concatenation function and the equality function and is closed under explicit transformations, substitutions, recursion of concatenation and bounded recursion. Throughout this section we will only give sketches of how a WTTM might operate and assume that the reader is familiar enough with the construction of Turing machines to fill in the laborious details of how to actually construct a set of states and the octuples of a transition function which yield a WTTM which operates as described. Appendix 1 gives more detailed descriptions of some particular submachines used.

Lemma 3: con $\in$ logspace, where con is the concatenation function over some alphabet Σ.

proof: Actually we will show that concatenation can be computed in unit space.

We must exhibit a WTTM M which has output con(x y) on input x y for all $x,y \in \Sigma^*$ and $s_M(x\ y) = 1$.

Let M have input alphabet $\Sigma \cup \{\beta_i\}$ (where β_i is the input blank symbol) and output alphabet Σ. It can be easily seen that M can be constructed to operate as follows on input x y:

1. M copies x onto its output tape symbol by symbol as it scans x on its input tape using no work space.
2. M ignores the β_i between x and y in the input, writing nothing on its output tape or work tape.
3. M now copies y from its input tape to its output tape symbol by symbol using no work space. When the first blank after y is reached, M halts.

M clearly produces con(x y) = xy on input x y never writing on its work tape. Therefore M computes the concatenation function over Σ and

$$s_M(x\ y) = 1 \leq K_1 \cdot \log_2|x\ y| + K_2 \qquad \text{for all } x,y \in \Sigma^*$$

where $K_1 = K_2 \neq 1$ and we therefore conclude that con $\in$ logspace.

Lemma 4: $c_= \in$ logspace, where $c_=$ is the standard characteristic function of the equality predicate for some alphabet Σ.

proof: We must exhibit a WTM M which has output $c_=(x\ y)$ on input x y and $s_M(x\ y) \leq K_1 \cdot \log_2|x\ y| + K_2$ for all $x,y \in \Sigma^*$ and some fixed constants K_1 and K_2.

Such an M with input alphabet $\Sigma \cup \{\beta_i\}$ and output alphabet Σ and work tape alphabet $\{0, 1, \underline{0}, \underline{1}, A, B, C\}$ (the symbols $\underline{0}$ and $\underline{1}$ are used for marking positions while copying or comparing for equality, see Appendix 1) can be constructed to operate as follows on input x y:

1. M places an A marker on its worktape and then writes the reverse binary representation of $|x|$ on its work tape (see Appendix 1) using $\lceil \log_2|x| \rceil + 1$ squares.
2. M writes a B marker and then copies what is between the A and B markers on the other side of the B marker (see Appendix 1) placing a C marker at the end.
3. M now returns the input head to the first blank before x and writes 0's between the A, B and C markers. (M will use the space between the A and B markers to store the reverse binary representation of the current position of the symbols in x and y being compared for equality and the space between the B and C markers as a counter. The string currently written on the work

tape between the A and B markers will be denoted AB and similarly for BC.)

4. M now does the following:

   a. M advances the input head and if the symbol scanned is a blank M goes to step 5 below. Otherwise M remembers the symbol being scanned internally (see Appendix 1) and adds 1 to ⟨AB⟩ (see Appendix 1).

   b. M then scans to the blank between x and y and advances symbol by symbol through y adding 1 to ⟨BC⟩ after each advance until ⟨AB⟩ equals ⟨BC⟩ (see Appendix 1).

   c. If the symbol being scanned in y on the input tape is not the same as the symbol remembered by M in the corresponding position in x then M halts having never written on its output tape.

   d. M zeros BC and returns the input head to the first blank before x.

   e. M now advances symbol by symbol through x adding 1 to ⟨BC⟩ until ⟨AB⟩ equals ⟨BC⟩.

   f. M then zeros BC and continues with step 4a above.

5. If M has not halted during step 4 then xBy. In order to see if y has the same length as x, M advances through y symbol by symbol adding 1 to ⟨BC⟩until ⟨BC⟩ equals ⟨AB⟩ (⟨AB⟩ = $|\overleftarrow{x}|$ after step 4). If the next input square is not a blank then M halts without writing on its output tape. Otherwise, M writes σ_1, the first symbol in an enumeration of Σ, on the output tape and halts.

M clearly computes $c_{-}(x\ y)$ and since no square to the left of the

A marker or to the right of the C marker is ever scanned and x y was arbitrary above

$$s_M(x\ y) \leq 2 \cdot \lceil \log_2|x| \rceil + 5$$
$$\leq K_1 \cdot \log_2|x\ y| + K_2 \quad \text{for all } x,y \in \Sigma^*$$

where $K_1 = 2$ and $K_2 = 7$, therefore $c_{=} \in$ logspace.

The following four lemmas serve to prove that logspace is closed under explicit transformations, substitutions, recursion of concatenation and bounded recursion. Each of the proofs is very similar in nature and we will be detailed only about the new aspects of each successive construction. The convention the XY denotes the string contained between the worktape markers X and Y will be continued.

Lemma 5: logspace is closed under explicit transformations.

proof: Let $g \in$ logspace be an m-variable function over some alphabet Σ computed by WTTM $M_g = (\Sigma \cup \{\beta_i\}, W, \Sigma, Q, q_0, F, \delta)$ such that $s_{M_g}(\overline{y_m}) \leq C_1 \cdot \log_2|\overline{y_m}| + C_2$ for all $\overline{y_m}$ over Σ.

Define $f:(\Sigma^*)^n \rightarrow \Sigma^*$ as an explicit transformation of g by $f(\overline{x_n}) = g(\overline{y_m})$ where each of the y_i in $\overline{y_m}$ is either some x_j in $\overline{x_n}$ or a constant string over Σ. Note that this explicit transformation is fixed by the definition of f and for some constant C_3, $|\overline{y_m}| \leq m \cdot |\overline{x_n}| + C_3$.

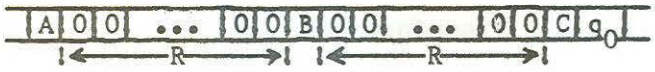
We must exhibit a WTTM M_f which has output $f(\overline{x_n})$ on input $\overline{x_n}$ and $s_{M_f}(\overline{x_n}) \leq K_1 \cdot \log_2|\overline{x_n}| + K_2$ for some fixed constants K_1 and K_2.

M_f will simulate M_g on the transformed input $\overline{y_m}$ generating the proper input symbol for each simulated step of M_g.

Such an M_f with input alphabet $\Sigma \cup \{\beta_i\}$, output alphabet Σ and

work tape alphabet $W \cup Q \cup \{0, 1, \underline{0}, \underline{1}, A, B, C\}$ can be constructed to operate as follows on input $\overline{x_n}$:

1. M_f writes an A marker and then m successive copies of $|\overline{x_n}|$ in reverse binary followed by $\overleftarrow{C_3 + 1}$ on its work tape.
2. M_f writes a B marker and then copies AB after it, followed by a C marker.
3. M_f now writes q_0 after the C and zeros BC and AB. The work tape now looks as follows:



where $R = m \cdot |\overleftarrow{|\overline{x_n}|}| + |\overleftarrow{C_3+1}| \leq m \cdot \log_2|\overline{x_n}| + C_4$

for some C_4.

(M_f will use the space between A and B for the reverse binary coded representation of the current read position in the simulated input to M_g, the space BC as a counter and the space to the right of C for the representation of the worktape and state of M_g with the state symbol occurring on the square just before the currently scanned work tape square in the simulation of M_g.)

4. M_f now simulates the computation of M_g on the transformed input $\overline{y_m}$ as follows:
   a. If the state symbol of M_g written on the work tape of M_f is in F then M_f halts. Otherwise M_f simulates the generation of the transformed input to M_g symbol by symbol counting each symbol generated on BC until $\langle AB \rangle$, the current read position of M_g, is reached on BC. This generation is done in the following manner. M_f generates each successive argument

remembering internally which one is being generated. If the argument y_i is x_j for some $j \leq n$ then M_f scans across the input $\overline{x_n}$ starting with the first blank to the left of x_1 until the j-th blank is scanned whereupon M_f produces x_j symbol by symbol. If y_i, the argument being generated, is a fixed constant string over Σ then M_f produces that string from an internal representation of it symbol by symbol. In either case since the input to M_g is a fixed explicit transformation of the input to M_f it can be generated one symbol at a time using the work space BC only to count symbols. By convention a WTTM never scans beyond the first blank right of its last argument, hence $\langle AB \rangle = \langle BC \rangle$ before $|BC| > R$.

b. When the proper input symbol for the next step of M_g is remembered internally, M_f scans the representation of the worktape of M_g remembering the current state and work tape symbol scanned in the representation of M_g. M_f uses this information to perform the next step of the simulation of M_g. This includes changing the state, writing a new work tape symbol, possibly changing the work tape scan position by moving the state symbol, possibly changing the value of $\langle AB \rangle$ (and hence the current simulated read position of M_g) by adding or subtracting 1, and possibly writing on output symbol on the actual output tape of M_f. (Since a WTTM never scans to the left of the initially scanned work tape

square, there is no need to shift the work tape representation of M_g to the right to accommodate adding symbols to its left end.)

c. M_f zeros BC and continues with 4a above.

M_f clearly computes $f(\overline{x_n}) = g(\overline{y_m})$ on input $\overline{x_n}$ and the space used is

$$s_{M_f}(\overline{x_n}) \leq 2 \cdot R + 3 + s_{M_g}(\overline{y_m}) + 1$$

$$\leq 2 \cdot (m \cdot \log_2|\overline{x_n}| + C_4) + 4 + C_1 \cdot \log_2|\overline{y_m}| + C_2$$

Since $|\overline{y_m}| \leq m \cdot |\overline{x_n}| + C_3$ implies that $\log_2|\overline{y_m}| \leq \log_2|\overline{x_n}| + C_5$ for some constant C_5 and since $\overline{x_n}$ was arbitrary above

$$s_{M_f}(\overline{x_n}) \leq K_1 \cdot \log_2|\overline{x_n}| + K_2 \quad \text{for all } \overline{x_n} \text{ over } \Sigma$$

where $K_1 = 2 \cdot m + C_1$ and $K_2 = 2 \cdot C_4 + C_2 + C_1 \cdot C_5 + 4$ are fixed constants, therefore $f \in$ logspace.

Lemma 6: logspace is closed under substitution.

proof: Let $g,h \in$ logspace be m and n-variable function respectively over some alphabet Σ computed by WTTM's

$$M_g = (\Sigma \cup \{\beta_i\}_3^7, W_g, \Sigma, Q_g, q_g, F_g, \delta_g)$$

$$M_h = (\Sigma \cup \{\beta_i\}_3^7, W_h, \Sigma, Q_h, q_h, F_h, \delta_h)$$

such that for all $\overline{y_m}$ and $\overline{x_n}$ over Σ

$$s_{M_g}(\overline{y_m}) \leq C_1 \cdot \log_2|\overline{y_m}| + C_2$$

$$s_{M_h}(\overline{x_n}) \leq C_3 \cdot \log_2|\overline{x_n}| + C_4$$

for some constants C_1, C_2, C_3 and C_4.

Define $f:(\Sigma^*)^{n+m-1} \rightarrow \Sigma^*$ by substitution of g into h where

$$f(\overline{x_{i-1}}\ \overline{y_m}\ \overline{x_{i+1}^n}) = h(\overline{x_{i-1}}\ g(\overline{y_m})\ \overline{x_{i+1}^n}).$$

Note that by Lemma 2 $|g(\overline{y_m})| \leq |\overline{y_m}|^{K_5} + K_6$ for some constants K_5 and K_6, hence for some constants K_7 and K_8

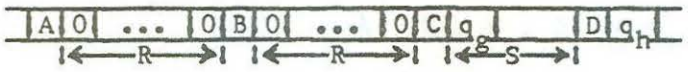
$$|\overline{x_{i-1}}\ g(\overline{y_m})\ \overline{i+1^{x_n}}| \leq |\overline{x_n}| + |g(\overline{y_m})| + 1$$
$$\leq |\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}|^{K_7} + K_8.$$

We must exhibit a WTTM M_f which computes f and uses space $s_{M_f}(\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}) \leq K_1 \cdot \log_2|\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}| + K_2$ for some fixed constants K_1 and K_2.

M_f will simulate M_h generating the proper input symbol at each simulated step of M_h. This will involve simulating M_g if some symbol in the j-th argument of M_h is needed for any $j \geq i$.

Such an M_f with input alphabet $\Sigma \cup \{\beta_i\}$, output alphabet Σ and work tape alphabet $\{1, 0, \underline{1}, \underline{0}, A, B, C, D\} \cup W_g \cup W_h \cup Q_g \cup Q_h$ can be constructed to operate as follows on input $\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}$:

1. M_f initializes the work tape to look as follows using methods similar to those described in previous lemmas:



where $R = \left|\overleftarrow{|\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}|^{K_7}}\right| + \left|\overleftarrow{K_8 + 1}\right|$

$$\leq K_7 \cdot \log_2|\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}| + K_9$$

for some constant K_9 and

$$S = \left|\overleftarrow{|\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}|^{C_1}}\right| + C_2 + 1$$
$$\geq C_1 \cdot \log_2|\overline{y_m}| + C_2 + 1$$
$$\geq s_{M_g}(\overline{y_m}) + 1$$

$$S \leq C_1 \cdot \log_2 \left| \overline{x_{i-1}}\ \overline{y_m}\ \overline{x_{i+1} x_n} \right| + C_1 + C_2 + 2$$

(The space AB will be used for the reverse binary representation of the current read position in the simulated input to M_h, BC is a counter, CD will be used for the work tape and state representation of any computation simulation of M_g and the space to the right of D will be used for the representation of the work tape and state of the simulation of M_h.)

2. M_f now simulates the computation of M_h on input $\overline{x_{i-1}}\ g(\overline{y_m})\ \overline{x_{i+1} x_n}$ as follows:

   a. If the state symbol of M_h written on the work tape of M_f is in F_h then M_f halts.

   b. Otherwise M_f begins to generate the input to M_h symbol by symbol by scanning through the input tape of M_f starting with the first blank before x_1 and counting each symbol on BC. If $\langle BC \rangle$ equals $\langle AB \rangle$ then M_f remembers the input symbol scanned and continues with step 2e below. If the blank before the i-th argument is reached then M_f continues with step 2c below.

   c. M_f now simulates the computation of M_g on input $\overline{y_m}$ as follows:

      i. M_f scans the work tape and state representation of M_g in CD remembering the current state and work tape symbol scanned. If the currently simulated state of M_g is in F_g then M_f continues at step 2d below.

      ii. Otherwise M_f uses the state and symbol scanned information together with the currently scanned input tape symbol of M_f (the simulation of M_g can read the input

$\overline{y_m}$ in place) to update the representation of the work tape and state of M_g, possibly move the read head one square and remember any output symbol produced at this simulated step of M_g.

iii. If there was an output symbol produced at this simulated step of M_g then M_f remembers it and adds 1 to $\langle BC \rangle$. If $\langle BC \rangle$ equals $\langle AB \rangle$ then M_f continues with step 2e below.

iv. Otherwise M_f continues with step 2ci above.

d. M_f continues to generate the input to M_h by scanning the input tape beginning with the blank before the i+m-th argument, which is x_{i+1}, counting each symbol on BC. When $\langle BC \rangle$ equals $\langle AB \rangle$ then M_f remembers the symbol scanned and continues with step 2e below.

e. When the proper input symbol for M_h has been remembered internally, M_f scans the representation of the work tape and state of M_h on its work tape and remembers the current state and work tape symbol scanned. M_f uses this information to perform the next simulated step of M_h by updating the work tape and state representation of M_h, updating the current read position in AB and possibly writing an output symbol into the output tape of M_f.

f. M_f now zeros BC, erases CD, places q_g after C and continues with 2a above.

M_f clearly computes $f(\overline{x_{i-1}}\ \overline{y_m}\ \overline{x_{i+1}x_n})$ and the space used is

$$s_{M_f}(\overline{x_{i-1}}\ \overline{y_m}\ \overline{x_{i+1}x_n}) \leq 2 \cdot R + S + 5 + s_{M_h}(\overline{x_{i-1}}\ g(\overline{y_m})\ \overline{x_{i+1}x_n}).$$

Since

$$s_{M_h}(\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}) \leq C_3 \cdot \log_2|\overline{x_{i-1}}\ g(\overline{y_m})\ \overline{i+1^{x_n}}| + C_4$$

$$\leq C_3 \cdot \log_2(|\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}|^{K_7} + K_8) + C_4$$

$$\leq K_{10} \cdot \log_2|\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}| + K_{11}$$

for some constants K_{10} and K_{11}, therefore

$$s_{M_f}(\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}) \leq K_1 \cdot \log_2|\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}| + K_2$$

where $K_1 = 2 \cdot K_7 + C_1 + K_{10}$ and $K_2 = 2 \cdot K_9 + C_1 + C_2 + K_{11} + 7$ are fixed constants. We conclude therefore that $f \in$ logspace since $\overline{x_{i-1}}\ \overline{y_m}\ \overline{i+1^{x_n}}$ was arbitrary above.

Lemma 7: logspace is closed under recursion of concatenation.

proof: Let $g,h \in$ logspace be n and n+2-variable functions respectively over some alphabet Σ computed by WTTM's

$$M_g = (\Sigma \cup \{\beta_i\}, W_g, \Sigma, Q_g, q_g, F_g, \delta_g)$$

$$M_h = (\Sigma \cup \{\beta_i\}, W_h, \Sigma, Q_h, q_h, F_h, \delta_h)$$

such that for all $\overline{x_n}$ and $\overline{x_{n+2}}$ over Σ

$$s_{M_g}(\overline{x_n}) \leq C_1 \cdot \log_2|\overline{x_n}| + C_2$$

$$s_{M_h}(\overline{x_{n+2}}) \leq C_3 \cdot \log_2|\overline{x_{n+2}}| + C_4.$$

Define $f:(\Sigma^*)^{n+1} \rightarrow \Sigma^*$ by recursion of concatenation where f satisfies

$$f(\overline{x_n}\ \lambda) = g(\overline{x_n})$$

$$f(\overline{x_n}\ y\sigma) = f(\overline{x_n}\ y)h(\overline{x_n}\ y\ \sigma)$$

where σ ranges over symbols in Σ.

We must exhibit a WTTM M_f which computes f and uses space

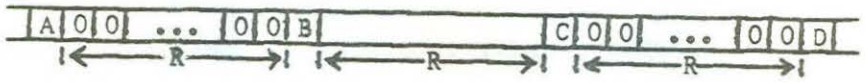
$$s_{M_f}(\overline{x_n}\ y) \leq K_1 \cdot \log_2|\overline{x_n}\ y| + K_2$$

for some fixed constants K_1 and K_2.

On input $\overline{x_n}\ y$, where $y = \sigma_1\sigma_2\cdots\sigma_{m-1}\sigma_m$, M_f will simulate M_g on input $\overline{x_n}$ and then simulate M_h m times on inputs $\overline{x_n}\ \lambda\ \sigma_1$, $\overline{x_n}\ \sigma_1\ \sigma_2$, $\overline{x_n}\ \sigma_1\sigma_2\ \sigma_3$, $\ldots$, $\overline{x_n}\ \sigma_1\sigma_2\cdots\sigma_{m-1}\ \sigma_m$ writing any output produced onto the output tape of M_f.

Such an M_f with input tape alphabet $\Sigma \cup \{\beta_i\}$, output tape alphabet Σ and work tape alphabet $W_g \cup W_h \cup Q_h \cup \{0, 1, \underline{0}, \underline{1}, A, B, C, D\}$ can be constructed to operate as follows on input $\overline{x_n}\ y$:

1. M_f initializes its work tape to look as follows using methods similar to those described previously:



where $R = \left|\overleftarrow{\left|\overline{x_n}\ y\right| + 1}\right| \leq \log_2\left|\overline{x_n}\ y\right| + 2$

(The space AB will be used for the reverse binary representation of the current read position in the simulated input to M_h; BC will be the reverse binary representation of the current recursive position in y numbered from the left end of the whole input; CD will be used as a counter and the space to the right of D will be used as a work space for M_g and as the representation of the work tape and state of the simulation of M_h.)

2. M_f now performs the computation of M_g in place on input $\overline{x_n}$ using the space to the right of D as its workspace. Any output symbols produced by M_g are written on the output tape of M_f. When M_g halts M_f erases all worktape symbols written by M_g to the right of D and places q_h on the square to the right of D.

M_f also puts the reverse binary representation of $|\overline{x_n}| + 1$ in BC which is the zeroth recursive position in y.

3. M_f now simulates the computation of M_h m times on the inputs described above as follows:

   a. M_f increments BC and counts up on CD as it scans the symbols of the input tape from the blank to the left of x_1 until the value ⟨BC⟩ is reached on CD. If M_f is scanning a blank then M_f halts. Otherwise M_f zeros CD.

   b. M_f now simulates the computation of M_h on the input $\overline{x_n}\,\mathcal{T}_1\mathcal{T}_2\cdots\mathcal{T}_{k-1}\,\mathcal{T}_k$ where $k = ⟨BC⟩ - |\overline{x_n}| - 1$ as follows:

      i. If the current simulated state of M_h is in F_h then M_f erases the symbols of the representation of the work tape and the state of M_h to the right of D, replaces q_h to the right of D, zeros CD and places the value 0 in AB. M_f is now ready to start the next simulation of M_h and hence continues with 3a above.

      ii. M_f now scans across the input counting up on CD until ⟨CD⟩ equals ⟨AB⟩. If during this scan M_f reaches ⟨BC⟩ on CD and ⟨AB⟩ equals ⟨BC⟩ at this point then M_f remembers a blank. Otherwise M_f increments CD without moving the the input head and if ⟨AB⟩ does not equal ⟨CD⟩ at this point then M_f remembers a blank. In any other case M_f remembers whatever symbol is being scanned. This serves to simulate the separation of the symbol at the current recursive position in y from the beginning part of y by a blank and to allow the simulation

of the blank following σ_k which by convention is the furthest M_h will scan on its simulated input at this recursive level in y.

iii. M_f now scans across the representation of the work tape and state of M_h remembering the current state and work tape symbol scanned, using this information with the remembered input symbol to update the representation of the work tape and state of M_h, update the current read position in AB and possibly write an output symbol of M_h on the output tape of M_f.

iv. M_f zeros CD and continues with step 3bi above.

M_f clearly computes $f(\overline{x_n}\ y)$ on input $\overline{x_n}\ y$ and the space used is

$$s_{M_f}(\overline{x_n}\ y) \leq 3 \cdot R + 5 + s_{M_g}(\overline{x_n}) + \max_{1 \leq k \leq m} \{s_{M_h}(\overline{x_n}\ \sigma_1\sigma_2 \cdots \sigma_{k-1}\ \sigma_k)\}.$$

Since $\log_2$ is a non-decreasing function

$$\max_{1 \leq k \leq m} \{s_{M_h}(\overline{x_n}\ \sigma_1\sigma_2 \cdots \sigma_{k-1}\ \sigma_k)\} \leq C_3 \cdot \log_2 |\overline{x_n}\ \sigma_1\sigma_2 \cdots \sigma_{m-1}\ \sigma_m| + C_4$$

$$\leq C_3 \cdot \log_2 |\overline{x_n}\ y| + C_3 + C_4.$$

Therefore since $\overline{x_n}\ y$ was arbitrary above

$$s_{M_f}(\overline{x_n}\ y) \leq K_1 \cdot \log_2 |\overline{x_n}\ y| + K_2 \quad \text{for all } \overline{x_n}\ y \text{ over } \Sigma$$

where $K_1 = 3 + C_1 + C_3$ and $K_2 = 3 + C_2 + C_3 + C_4 + 5$ are fixed constants and we conclude that $f \in$ logspace.

Lemma 8: logspace is closed under log bounded recursion on notation.

proof: Let $g,h \in$ logspace be n and n+3-variable functions respectively over some alphabet Σ computed by WTM's

$$M_g = (\Sigma \cup \{\beta_i\}, W_g, \Sigma, Q_g, q_g, F_g, \delta_g)$$

$$M_h = (\Sigma \cup \{\beta_i\}, W_h, \Sigma, Q_h, q_h, F_h, \delta_h)$$

such that for all $\overline{x_n}$ and $\overline{x_{n+3}}$ over Σ

$$s_{M_g}(\overline{x_n}) \leq C_1 \cdot \log_2|\overline{x_n}| + C_2$$

$$s_{M_h}(\overline{x_{n+3}}) \leq C_3 \cdot \log_2|\overline{x_{n+3}}| + C_4.$$

Define $f:(\Sigma^*)^{n+1} \rightarrow \Sigma^*$ by bounded recursion where f satisfies

$$f(\overline{x_n}\ \lambda) = g(\overline{x_n})$$

$$f(\overline{x_n}\ y\sigma) = h(\overline{x_n}\ y\ \sigma\ f(\overline{x_n}\ y))$$

$$|f(\overline{x_n}\ y)| \leq C_5 \cdot \log_2|\overline{x_n}\ y| + C_6$$

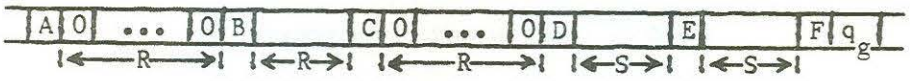
where σ ranges over symbols in Σ.

We must exhibit a WTTM M_f which computes f and uses space $s_{M_f}(\overline{x_n}\ y) \leq K_1 \cdot \log_2|\overline{x_n}\ y| + K_2$ for some fixed constants K_1 and K_2.

On input $\overline{x_n}\ y$, where $y = \sigma_1\sigma_2\cdots\sigma_m$, M_f will simulate M_g on input $\overline{x_n}$ and then simulate M_h m times on inputs $\overline{x_n}\ \lambda\ \sigma_1\ f(\overline{x_n}\ \lambda)$, $\overline{x_n}\ \sigma_1\ \sigma_2\ f(\overline{x_n}\ \sigma_1)$, $\overline{x_n}\ \sigma_1\sigma_2\ \sigma_3\ f(\overline{x_n}\ \sigma_1\sigma_2)$, $\ldots$, $\overline{x_n}\ \sigma_1\sigma_2\cdots\sigma_{m-1}\ \sigma_m\ f(\overline{x_n}\ \sigma_1\cdots\sigma_{m-1})$ writing the output of each successive simulation on the work tape of M_f and finally copying the output of the last simulation onto the output tape of M_f.

Such an M_f with input tape alphabet $\Sigma \cup \{\beta_i\}$, output tape alphabet Σ and work tape alphabet $W_g \cup W_h \cup Q_g \cup Q_h \cup \{0, 1, \underline{0}, \underline{1}, A, B, C, D, E, F\}$ can be constructed to operate as follows on input $\overline{x_n}\ y$:

1. M_f initializes its work tape to look as follows using methods similar to those described in previous lemmas:



where $S = \left|\overleftarrow{\left|\overline{x_n}\ y\right|^{C_5}}\right| + C_6 + 1$

$$\geq C_5 \cdot \log_2\left|\overline{x_n}\ y\right| + C_6 + 1$$

$$\geq \max_{0\leq k\leq m}\left\{\left|f(\overline{x_n}\ \sigma_1\sigma_2\cdots\sigma_k)\right| + 1\right\}$$

since $\log_2$ is a non-decreasing function. Also

$$S \leq C_5 \cdot \log_2\left|\overline{x_n}\ y\right| + C_5 + C_6 + 2.$$

In addition R was constructed such that

$$R = \left|\overleftarrow{\left|\overline{x_n}\ y\right|}\right| + 4 + \left|\overleftarrow{S}\right|$$

$$\geq \left|\overleftarrow{\left|\overline{x_n}\ y\right| \cdot 4 \cdot S}\right|$$

$$\geq \left|\overleftarrow{\left|\overline{x_n}\ y\right| + 4 + S}\right|$$

$$\geq \left|\overleftarrow{\left|\overline{x_n}\ y\right| + 4 + \max_{0\leq k\leq m}\left\{\left|f(\overline{x_n}\ \sigma_1\sigma_2\cdots\sigma_k)\right|\right\}}\right|$$

$$\geq \left|\overleftarrow{\max_{0\leq k\leq m}\left\{\left|\overline{x_n}\ \sigma_1\sigma_2\cdots\sigma_{k-1}\ \sigma_k\ f(\overline{x_n}\ \sigma_1\sigma_2\cdots\sigma_{k-1})\right|\right\} + 2}\right|$$

since $\log_2$ is a non-decreasing function. Also

$$R \leq \log_2\left|\overline{x_n}\ y\right| + 5 + S$$

$$\leq (C_5+1) \cdot \log_2\left|\overline{x_n}\ y\right| + C_5 + C_6 + 7$$

(The space AB will be used for the reverse binary representation of the current read position in the simulated input to M_h. BC will be the reverse binary coding of the current recursive position in y. CD will be used as a counter. DE will contain the output generated at the last recursive level. EF will contain the output being generated at the current recursive level and the space beyond F will be used for the representation of the work tape and state of the simulations of M_g and M_h.)

2. M_f now simulates the computation of M_g on input $\overline{x_n}$ as follows:

   a. M_f scans the representation of the work tape and state of M_g written to the right of F remembering the current state and work tape symbol scanned. If the currently simulated state of M_g is a member of F_g then M_f continues with step 3 below. Otherwise M_f uses the state and symbol scanned information together with the currently scanned input tape symbol (the simulation of M_g can read the input $\overline{x_n}$ in place) to update the representation of the work tape and state of M_g, possibly move the read head one square and remember any output symbol of M_g on this step in the simulation.

   b. M_f then scans across EF locating the first blank square to the right of E and writes the remembered output symbol for this step of M_g where if no output symbol was produced then M_f leaves the blank.

   c. M_f continues with step 2a above.

3. M_f puts the reverse binary representation of $|\overline{x_n}| + 1$ in BC.

4. M_h now performs m levels of recursion on M_h as follows:

   a. M_f increments BC, the recursive level, and counts up from 0 on CD as it scans across the input tape until the value $\langle BC \rangle$ is reached on CD. If M_f is scanning a blank then M_f continues with step 5 below.

   b. Otherwise M_f sets $\langle AB \rangle$ to 0, copies the output of the last level of recursion from EF to DE, erases EF and the space beyond F and places q_h on the square after F.

   c. M_f now simulates the computation of M_h on the input

$\overline{x_n}\ \sigma_1\sigma_2\ldots\sigma_{k-1}\ \sigma_k\ f(\overline{x_n}\ \sigma_1\sigma_2\ldots\sigma_{k-1})$, where again $y = \sigma_1\sigma_2\ldots\sigma_m$ and $k = \langle BC\rangle - |\overline{x_n}| - 1$, as follows:

i. If the current state of M_h in the work tape representation beyond F is in F_h then M_f continues with step 4a above.

ii. Otherwise M_f begins scanning through the input from the first blank square to the left of $\overline{x_n}$ counting up from 0 on CD. If $\langle BC\rangle$ is reached on CD then M_f continues with step iii below. If $\langle AB\rangle$ is reached on CD then M_f remembers the symbol being scanned in the input and continues with step v below.

iii. If $\langle BC\rangle = \langle AB\rangle$ then M_f remembers a blank and continues with step v below. Otherwise M_f increments CD without moving the input head and if $\langle CD\rangle = \langle AB\rangle$ then M_f remembers the symbol being scanned and continues with v below. Otherwise M_f increments CD again and if $\langle CD\rangle = \langle AB\rangle$ then M_f remembers a blank and continues at step v below. This all serves to separate $\sigma_1\sigma_2\ldots\sigma_{k-1}$, σ_k and $f(\overline{x_n}\ \sigma_1\sigma_2\ldots\sigma_{k-1})$ with blanks as is required for the input to M_h.

iv. M_f now proceeds to scan through DE which contains $f(\overline{x_n}\ \sigma_1\sigma_2\ldots\sigma_{k-1})$ the output of the last recursive level continuing to count up on CD again until $\langle AB\rangle$ is reached, whereupon M_f remembers the symbol being scanned in DE on the work tape. By convention M_h never scans beyond the first blank to the right of its last argument, hence $\langle CD\rangle = \langle AB\rangle$ before DE is exhausted.

v. M_f now scans the representation of the work tape and state of M_h remembering the current state and work tape symbol scanned, using this information with the remembered input symbol to update the representation of the work tape and state of M_h, to update the current read position in AB and to remember the output symbol, if any, which was generated at this step of M_h.

vi. M_f then scans across EF locating the first blank square to the right of E and writes the remembered output symbol for this step of M_h if one was produced. M_f now continues with step 4ci above.

5. Finally M_f writes EF onto its output tape and halts.

Clearly M_f computes $f(\overline{x_n}\ y)$ on input $\overline{x_n}\ y$ and the space used (letting σ range over symbols in Σ) is

$$s_{M_f}(\overline{x_n}\ y) \leq 3 \cdot R + 2 \cdot S + 7 + s_{M_g}(\overline{x_n}) + \max_{z \sigma By} \{ s_{M_h}(\overline{x_n}\ z\ \sigma\ f(\overline{x_n}\ z)) \}.$$

Since $\log_2$ is a non-decreasing function

$$\max_{z \sigma By} \{ s_{M_h}(\overline{x_n}\ z\ \sigma\ f(\overline{x_n}\ z)) \} \leq C_3 \cdot \log_2 |\overline{x_n}\ y| + C_4.$$

Therefore

$$s_{M_f}(\overline{x_n}\ y) \leq 3 \cdot R + 2 \cdot S + 7 + C_1 \cdot \log_2 |\overline{x_n}\ y| + C_2 + C_3 \cdot \log_2 |\overline{x_n}\ y| + C_4$$

$$\leq K_1 \cdot \log_2 |\overline{x_n}\ y| + K_2$$

where $K_1 = 5 \cdot C_5 + C_1 + C_3 + 3$ and $K_2 = 5 \cdot C_6 + 5 \cdot C_5 + C_2 + C_4 + 26$ are fixed constants. We conclude that $f \in$ logspace.

Theorem 9: $\mathcal{L} \subseteq$ logspace

proof: If $f \in \mathcal{L}$ then there is some alphabet Σ and a finite sequence $f_1, f_2, \ldots, f_n$ of functions over Σ such that $f = f_n$ and every f_i is either the concatenation function over Σ or the standard characteristic function of the equality predicate over Σ, or f_i is defined from some previous functions in the sequence by an explicit transformation, substitution of a function for a variable, recursion of concatenation or bounded recursion.

By Lemmas 3 and 4 logspace contains the concatenation function and the equality function over Σ, and by Lemmas 5, 6, 7 and 8 logspace is closed under explicit transformations, substitutions, recursion of concatenation and bounded recursion. Induction over the f_i yields that $f \in$ logspace.

ADDITIONAL FUNCTIONS AND FUNCTIONAL OPERATIONS FOR $\mathcal{L}$

Our next major goal is to show that every logarithmic space computable function is definable in terms of the base functions and functional operations which define $\mathcal{L}$. This result will establish the equality of logspace and $\mathcal{L}$. Before proceeding, it is useful to prove that $\mathcal{L}$ contains the following auxiliary functions and predicates, and is closed under certain additional functional operations.

Lemma 10: $\mathcal{L}$ contains the following auxiliary functions over an arbitrary alphabet Σ, where we will let σ_1 denote the first symbol in an enumeration of Σ:

(1) The identity function $e(x) = x$,

constant functions $c_y^n(\overline{x_n}) = y$ for any $y \in \Sigma^*$, and

projection functions $u_i^n(\overline{x_n}) = x_i$.

(2) Certain symbolic functions:

$$\text{not}\lambda(x) = \begin{cases} \sigma_1 & \text{if } x = \lambda \\ \lambda & \text{if } x \neq \lambda \end{cases}$$

$$\text{lastchar}(x) = \begin{cases} \lambda & \text{if } x = \lambda \\ \sigma & \text{if } x = y\sigma \text{ for some } y \in \Sigma^* \text{ and } \sigma \in \Sigma \end{cases}$$

$$s(x\ y\ z) = \begin{cases} x & \text{if } z \neq \lambda \\ y & \text{if } z = \lambda \end{cases}$$

(3) Certain numeric functions over alphabets which contain the symbols 0 and 1:

$$\text{ones}(x) = \begin{cases} \lambda & \text{if } x = \lambda \\ \underbrace{1\ldots1}_{|x|} & \text{if } x \neq \lambda \end{cases}$$

ones(x) produces a string which is the same length as x and is composed entirely of the symbol 1. In general we will define the n-variable function $ones^n(\overline{x_n})$:

$$ones^n(\overline{x_n}) = ones(x_1)(1)ones(x_2)(1)\ldots(1)ones(x_n) = \underbrace{1\ldots1}_{|\overline{x_n}|}$$

$$plus1(x) = \overleftarrow{\langle x\rangle + 1}$$

plus1(x) produces a string which is the reverse binary representation of m+1 if x is the reverse binary representation of m. Similarly, monus1 is self explanatory

$$monus1(x) = \overleftarrow{\langle x\rangle \dot{-} 1}$$

(where $\dot{-}$ indicates monus, proper subtraction) and

$$len(x) = \overleftarrow{|x|}$$

the reverse binary representation of the length of x.

(4) Certain functions which interpret some of their arguments as numbers equal to the length of that argument:

$$shorter(x\ y) = \begin{cases} \lambda & \text{if } |x| \geq |y| \\ \sigma_1 & \text{if } |x| < |y| \end{cases}$$

skip(x y) = the substring of y remaining after the first |x| characters have been deleted.

proof: Throughout this proof σ will range over symbols in Σ.

(1) $e(x) = x\lambda$ an explicit transformation of concatenation. $c_y^n(\overline{x_n}) = e(y)$ and $u_i^n(\overline{x_n}) = e(x_i)$ are all explicit transformations of e.

(2) Define notλ by bounded recursion from $c_{\sigma_1}^0$, and c_λ^3:

$$\text{not}\lambda(\lambda) = c^0_{\sigma_1} = \sigma_1$$

$$\text{not}\lambda(y\sigma) = c^3_\lambda(y\ \sigma\ \text{not}\lambda(y)) = \lambda$$

$$|\text{not}\lambda(y)| \leq |\sigma_1| = 1$$

Define lastchar by bounded recursion from c^0_λ and u^3_2:

$$\text{lastchar}(\lambda) = c^0_\lambda = \lambda$$

$$\text{lastchar}(y\sigma) = u^3_2(y\ \sigma\ \text{lastchar}(y)) = \sigma$$

$$|\text{lastchar}(y)| \leq |\sigma| = 1$$

The switching function s is defined in three stages. First s_1 is defined from c^1_λ, u^4_1 and lastchar using bounded recursion and substitution such that

$$s_1(x\ y) = \begin{cases} \lambda & \text{if } y = \lambda \\ \text{lastchar}(x) & \text{if } y \neq \lambda \end{cases}$$

$$s_1(x\ \lambda) = c^1_\lambda(x) = \lambda$$

$$s_1(x\ y\sigma) = \text{lastchar}(u^4_1(x\ y\ \sigma\ s_1(x\ y))) = \text{lastchar}(x)$$

$$|s_1(x\ y)| \leq |\text{lastchar}(x)| \leq 1$$

Then s_2 is defined from c^1_λ, u^3_1, u^3_3 and s_1 using recursion of concatenation and substitution such that

$$s_2(x\ y) = \begin{cases} \lambda & \text{if } x = \lambda \\ y & \text{if } x \neq \lambda \end{cases}$$

$$s_2(x\ \lambda) = c^1_\lambda(x) = \lambda$$

$$s_2(x\ y\sigma) = s_2(x\ y)s_1(u^3_3(x\ y\ \sigma)\ u^3_1(x\ y\ \sigma))$$

$$= s_2(x\ y)s_1(\sigma\ x)$$

Finally since one of z or $\text{not}\lambda(z)$ must not be equal to λ, s can be defined from concatenation, $\text{not}\lambda$ and s_2 using

substitution and explicit transformations:

$s(x\ y\ z) = s_2(z\ x)s_2(not\lambda(z)\ y)$

(3) Assume for the rest of this proof that Σ contains the symbols 0 and 1.

ones is defined from c^0_λ and c^2_1 using recursion of concatenation:

$ones(\lambda) = c^0_\lambda = \lambda$

$ones(y\sigma) = ones(y)c^2_1(y\ \tau) = ones(y)(1)$

Define plus1 in four stages. First plussym is defined such that

$$plussym(x\ y) = \begin{cases} 1 & \text{if } x \neq y \ (xy = 01 \text{ or } xy = 10) \\ 0 & \text{if } x = y \ (x = y = 0 \text{ or } x = y = 1) \end{cases}$$

from s and $c_=$ using substitution and explicit transformations:

$plussym(x\ y) = s(0\ 1\ c_=(x\ y))$

Then carry is defined such that

$$carry(y) = \begin{cases} 1 & \text{if } y \text{ contains no 0's} \\ 0 & \text{otherwise} \end{cases}$$

from concatenation, c^0_1, s and $c_=$ using substitution, explicit transformations and bounded recursion:

$carry(\lambda) = c^0_1 = 1$

$carry(y\sigma) = s(0\ 1\ c_=(\tau\ 0)c_=(carry(y)\ 0))$

$|carry(y)| \leq |c^0_1| = 1$

We now define shortplus1, which produces all of the characters of plus1 except the last 1 whenever the carry propagates all the way through the input, from c^0, plussym and carry using recursion of concatenation, explicit transformation and substitution:

shortplus1(λ) = $c^0_\lambda = \lambda$

shortplus1(yσ) = shortplus1(y)plussym(σ carry(y))

Finally plus1 is defined from shortplus1, concatenation, s, $c_=$ and carry using explicit transformations and substitution so that an additional 1 is appended to shortplus1 if the carry propagated all the way through the input:

plus1(y) = shortplus1(y)s(1 λ $c_=$(carry(y) 1))

If y is not a proper reverse binary representation of some number then plus1(y) is some arbitrary string of 0's and 1's. Note that plus1(y) is the same length as y unless y = 11...11 (a string of 1's) in which case

$|\text{plus1}(y)| = |y| + 1$.

Similarly monus1 is defined in four stages from concatenation, c^0_1, c^0_λ, $c_=$ and s using explicit transformations, substitution, recursion of concatenation and bounded recursion:

minussym(x y) = s(0 1 $c_=$(x y))

decr(λ) = $c^0_\lambda = \lambda$

decr(yσ) = s(0 1 $c_=$(σ 1)$c_=$(decr(y) 0))

$|\text{decr}(y)| \leq 1$

minus1(λ) = $c^0_\lambda = \lambda$

minus1(yσ) = minus1(y)minussym(σ decr(y))

monus1(y) = s(0 minus1(y) $c_=$(y 0))

If y is not a proper reverse binary representation of some

number then monus1(y) is some arbitrary string of 0's and 1's. Note that $|\text{monus1}(y)| = |y|$.

The function len is now defined from plus1 and c_0^0 by bounded recursion and an explicit transformation:

$\text{len}(\lambda) = c_0^0 = 0$

$\text{len}(y\sigma) = \text{plus1}(\text{len}(y))$

$|\text{len}(y)| \leq \log_2 |y| + 1$

where the bound on $|\text{len}(y)|$ is arrived at by noting that plus1 increases the length of len(y) by one symbol iff len(y) is the reverse binary representation of $2^k - 1$ for some k.

Define shorter from c_λ^1, $c_=$ and ones using recursion of concatenation, substitution and explicit transformations:

$\text{shorter}(x\ \lambda) = c_\lambda^1(x) = \lambda$

$\text{shorter}(x\ y\sigma) = \text{shorter}(x\ y)c_=(\text{ones}(x)\ \text{ones}(y))$

Finally skip is defined from c_λ^1, s, concatenation and shorter using substitution, explicit transformations and recursion of concatenation:

$\text{skip}(x\ \lambda) = c_\lambda^1(x) = \lambda$

$\text{skip}(x\ y\sigma) = \text{skip}(x\ y)s(\sigma\ \lambda\ \text{shorter}(x\ y\sigma))$

Since all of the functions defined in this proof were either derived from the base functions of $\mathcal{L}$ using the functional operations for which $\mathcal{L}$ is closed by definition, or derived in $\mathcal{L}$ using the defining closure operations of $\mathcal{L}$ from functions with earlier such derivations in $\mathcal{L}$, by induction we can conclude that all of the functions defined in this proof

are members of $\mathcal{L}$.

Throughout the remainder of this paper we will be less explicit about the exact nature of the definition of a function in $\mathcal{L}$. For example the projection functions $u_i^n(\overline{x_n})$ will be written x_i for all n and the constant functions $c_y^n(\overline{x_n})$ will be written simply as y for all n. Also many occurrences of definition by explicit transformation, substitution, recursion of concatenation and bounded recursion will be left unanotated. The reader should be able to fill in the details of any definition claimed to be carried out in $\mathcal{L}$.

Lemma 11: $\mathcal{L}$ is closed under the following additional functional operations:

(1) The Boolean operations $\wedge$, $\vee$, $\neg$, $\rightarrow$, and $\Leftrightarrow$; and the substring quantifiers $\exists xBy$, $\exists xPy$, $\exists xEy$, $\forall xBy$, $\forall xPy$, and $\forall xEy$ on predicates which have a characteristic function in $\mathcal{L}$.

(2) Substring minimization, min xBy, on predicates which have a characteristic function in $\mathcal{L}$.

(3) Definition by cases.

proof: Let Σ be some arbitrary alphabet and let τ range over symbols in Σ.

(1) If $r_1:(\Sigma^*)^n \rightarrow \Sigma^*$ is a characteristic function of the predicate R_1 over Σ then not$\lambda(r_1(\overline{x_n}))$ is a characteristic function of its negation $\neg R_1$.

If in addition $r_2:(\Sigma^*)^k \rightarrow \Sigma^*$ is a characteristic function of the predicate R_2 over Σ then $r_3(\overline{x_n}\ \overline{y_k}) = r_1(\overline{x_n})r_2(\overline{y_k})$ is a characteristic function of the predicate $R_1 \vee R_2$, the disjunction of R_1 and R_2.

Since negation and disjunction form a complete set of Boolean operations, $\mathcal{L}$ is closed under all Boolean operations. $R_1 \wedge R_2$, $R_1 \rightarrow R_2$, and $R_1 \Leftrightarrow R_2$ will be abbreviations for $\neg(\neg R_1 \vee \neg R_2)$, $R_1 \vee \neg R_2$ and $(R_1 \rightarrow R_2) \wedge (R_2 \rightarrow R_1)$ respectively.

Suppose R is some n-place predicate over Σ contained in $\mathcal{L}$ and define Q as the n-place predicate over Σ such that

$$Q(\overline{x_{n-1}}\ y) \Leftrightarrow \exists_{zBy}\left[R(\overline{x_{n-1}}\ z)\right].$$

Let $r:(\Sigma^*)^n \rightarrow \Sigma^*$ contained in $\mathcal{L}$ be a characteristic function of R and define $q:(\Sigma^*)^n \rightarrow \Sigma^*$ by recursion of concatenation

$$q(\overline{x_{n-1}}\ \lambda) = r(\overline{x_{n-1}}\ \lambda)$$

$$q(\overline{x_{n-1}}\ y\sigma) = q(\overline{x_{n-1}}\ y)r(\overline{x_{n-1}}\ y\sigma).$$

If $R(\overline{x_{n-1}}\ z)$ is true for some zBy then $r(\overline{x_{n-1}}\ z) \neq \lambda$ for some zBy and hence $q(\overline{x_{n-1}}\ y) \neq \lambda$. Therefore q is a characteristic function of Q and $\mathcal{L}$ is closed under $\exists_{xBy}$ quantification.

The following formulas demonstrate how $\exists_{zEy}$, and $\exists_{zPy}$ quantifications can be defined from $\exists_{zBy}$ quantification for any n-place predicate R over Σ with a characteristic function in $\mathcal{L}$:

$$\exists_{zEy}\left[R(\overline{x_{n-1}}\ z)\right] \Leftrightarrow \exists_{zBy}\left[R(\overline{x_{n-1}}\ \mathrm{skip}(z\ y))\right]$$

$$\exists_{zPy}\left[R(\overline{x_{n-1}}\ z)\right] \Leftrightarrow \exists_{zBy}\left[\exists_{wEz}\left[R(\overline{x_{n-1}}\ w)\right]\right]$$

Using the usual reduction of universal quantification to existential quantification and negation we define $\forall_{zBy}$, $\forall_{zEy}$ and $\forall_{zPy}$ from $\exists_{zBy}$, $\exists_{qEy}$ and $\exists_{qPy}$ as follows:

$$\forall_{zBy}\left[R(\overline{x_{n-1}}\ z)\right] \Leftrightarrow \neg\exists_{zBy}\left[\neg R(\overline{x_{n-1}}\ z)\right]$$

$$\forall_{zEy}\left[R(\overline{x_{n-1}}\ z)\right] \Leftrightarrow \neg\exists_{zEy}\left[\neg R(\overline{x_{n-1}}\ z)\right]$$

$$\forall_{zPy}\left[R(\overline{x_{n-1}}\ z)\right] \Leftrightarrow \neg\exists_{zPy}\left[\neg R(\overline{x_{n-1}}\ z)\right]$$

Therefore we conclude that $\mathcal{L}$ is closed under all of the substring

quantifiers.

(2) Let R be some n-place predicate over Σ contained in $\mathcal{L}$ and define $m_R:(\Sigma^*)^n \to \Sigma^*$ as follows:

First define $prem_R$ as an n-variable function over Σ such that $prem_R(\overline{x_{n-1}}\ y)$ is the shortest substring z which begins y such that $R(\overline{x_{n-1}}\ z)$ is true and is y if no such z exists.

$$prem_R(\overline{x_{n-1}}\ \lambda) = \lambda$$

$$prem_R(\overline{x_{n-1}}\ y\sigma) = prem_R(\overline{x_{n-1}}\ y)s(\lambda\ \sigma\ \exists z By[R(\overline{x_{n-1}}\ z)]).$$

Now define m_R to equal $prem_R$ unless $prem_R(\overline{x_{n-1}}\ y) = y$ and $R(\overline{x_{n-1}}\ y)$ is false, in which case $m_R(\overline{x_{n-1}}\ y) = \lambda$.

$$m_R(\overline{x_{n-1}}\ y) = s(\lambda\ prem_R(\overline{x_{n-1}}\ y)\ (prem_R(\overline{x_{n-1}}\ y)=y \wedge \neg R(\overline{x_{n-1}}\ y)))$$

Note that $m_R \in \mathcal{L}$ and $m_R(\overline{x_{n-1}}\ y) = \min z By[R(\overline{x_{n-1}}\ z)]$. Therefore $\mathcal{L}$ is closed under substring minimization.

(3) Suppose $f:(\Sigma^*)^n \to \Sigma^*$ is defined by cases

$$f(\overline{x_n}) = \begin{cases} f_1(\overline{x_n}) & \text{if } R_1(\overline{x_n}) \\ \vdots & \vdots \\ f_k(\overline{x_n}) & \text{if } R_k(\overline{x_n}) \end{cases}$$

where each $f_i \in \mathcal{L}$, each $R_i \in \mathcal{L}$ and for every $\overline{x_n}$ over Σ exactly one of the predicates $R_1, \ldots, R_k$ is true. Then we can define f in $\mathcal{L}$ by

$$f(\overline{x_n}) = s(f_1(\overline{x_n})\ \lambda\ R_1(\overline{x_n}))s(f_2(\overline{x_n})\ \lambda\ R_2(\overline{x_n}))\ldots s(f_k(\overline{x_n})\ \lambda\ R_k(\overline{x_n}))$$

since concatenation, s, each f_i and each R_i are members of $\mathcal{L}$. Therefore $\mathcal{L}$ is closed under definition by cases.

Lemma 12: $\mathcal{L}$ contains the predicates xBy, xPy and xEy.

proof: The following formulas show how these predicates can be defined from the equality predicate and the substring quantifiers $\exists_z$By, $\exists_z$Py and $\exists_z$Ey:

$$xBy \Leftrightarrow \exists_z By[x=z]$$

$$xPy \Leftrightarrow \exists_z Py[x=z]$$

$$xEy \Leftrightarrow \exists_z Ey[x=z]$$

Since by Lemma 11 $\mathcal{L}$ is closed under the above substring quantifiers and equality is in $\mathcal{L}$ by definition, $\mathcal{L}$ contains the predicates xBy, xPy and xEy.

PROOF THAT logspace $\subseteq \mathcal{L}$

We are now prepared to carry out a symbolization of the computation of a WTM which computes some function contained in logspace using functions contained in $\mathcal{L}$ and functional operations under which $\mathcal{L}$ is closed. Normally a proof that some class of functions computed by a type of abstract machine is contained within a particular set of mathematically defined functions is called an arithmetization. However, in this case $\mathcal{L}$ is a set of recursive string functions rather than numeric functions, hence the description of the following proof as a symbolization seems more appropriate.

Theorem 13: logspace $\subseteq \mathcal{L}$

proof: Let $f \in$ logspace be an n-variable function over some alphabet Γ computed by WTM

$$M = (\Gamma \cup \{\beta_i\}, W, \Gamma, Q, q_0, F, \delta)$$

such that for all $\overline{x_n}$ over Γ

$$s_M(\overline{x_n}) \leq K_1 \cdot \log_2|\overline{x_n}| + K_2.$$

Consider the alphabet $\Sigma = \Gamma \cup \{\beta_i\} \cup W \cup Q \cup \{1, 0, \$\}$ over which many of the functions in this proof are defined. Note that $\Gamma \subseteq \Sigma$.

We begin by defining a bounding function $b_M(\overline{x_n})$ whose length is greater than the maximum number of steps in a computation of M on input $\overline{x_n}$ over Γ. Since M computes $f \in$ logspace, by Corollary 2 for all $\overline{x_n}$ over Γ

$$t_M(\overline{x_n}) \leq |\overline{x_n}|^{K_3} + K_4$$

for some constants K_3 and K_4. We can define b_M within $\mathcal{L}$ as follows using $K_3 - 1$ levels of recursion of concatenation.

First a function m over Σ is defined such that m(x y) is $|y|$

copies of x concatenated together.

$$m(x\ \lambda) = \lambda$$

$$m(x\ y\sigma) = m(x\ y)e(x) \qquad \sigma \in \Sigma$$

We now define a function B using a constant number of occurrences of m nested together by $K_3 - 2$ applications of substitution of a function for a variable.

$$B(x) = \underbrace{m(x\ m(x \ldots m(x\ x)\ldots))}_{K_3-1 \text{ nested occurrences of } m}$$

m can be thought of as computing unary multiplication which means that $|m(x\ x)| = |x|^2$, $|m(x\ m(x\ x))| = |x|^3$, and in general

$$\left|\underbrace{m(x\ m(x \ldots m(x\ x)\ldots))}_{k-1 \text{ nested occurrences of } m}\right| = |x|^k$$

Therefore letting

$$b_M(\overline{x_n}) = B(ones^n(\overline{x_n}))\underbrace{11\ldots11}_{K_4}$$

we have defined b_M within $\mathcal{L}$ and for all $\overline{x_n}$ over $\Gamma \subseteq \Sigma$

$$\left|b_M(\overline{x_n})\right| = \left|\overline{x_n}\right|^{K_3} + K_4 \geq t_M(\overline{x_n}).$$

Let $Q = \{q_0, q_1, \ldots, q_C\}$ and $F = \{h_1, h_2, \ldots, h_D\}$ and define the predicates inQ and inF over Σ as

$$inQ(w) \Leftrightarrow w{=}q_0 \vee w{=}q_1 \vee \ldots \vee w{=}q_C$$

$$inF(w) \Leftrightarrow w{=}h_1 \vee w{=}h_2 \vee \ldots \vee w{=}h_D$$

We will need to define the n-variable function over Σ

$$ins_{\beta_i}(\overline{x_n}) = x_1\beta_i x_2\beta_i \cdots \beta_i x_n$$

which produces the input string to M complete with input blanks.

ID's in a computation of M will be symbolized as strings over Σ. This is done straight forwardly by symbolizing the ID (k, α) as $\overleftarrow{k}\$\alpha$ where, as before, $\overleftarrow{k}$ is the reverse binary representation of the number k.

The following auxiliary functions defined within $\mathcal{L}$ extract various parts of a symbolized ID y which is part of a computation of WTTM M on input $\overline{x_n}$ over Γ.

$$\text{Ipos}(y) = \min w\text{By}[\$\text{B}(\text{skip}(w\ y))]$$

Ipos(y) is the substring of y which is the reverse binary coding of the symbolized position of the input head of M.

Let $\text{eq}(x\ y) \Leftrightarrow x\text{By} \wedge \neg\exists v\text{P}(\text{skip}(x\ y))[v=1]$ and define

$$\text{Iscan}\lambda(\overline{x_n}\ y) = \text{lastchar}(\min w\text{B}(\text{ins}_{\beta_i}(\overline{x_n}))[\text{eq}(\text{len}(w)\ \text{Ipos}(y))])$$

$\text{Iscan}\lambda(\overline{x_n}\ y)$ is the actual character of $\overline{x_n}$ over Γ at the symbolized input head position Ipos(y), unless $\langle\text{Ipos}(y)\rangle = 0$ or $\langle\text{Ipos}(y)\rangle > |\overline{x_n}|$ in which case $\text{Iscan}\lambda(\overline{x_n}\ y) = \lambda$. Since we want to simulate reading blanks at the ends of the input $\overline{x_n}$, we define

$$\text{Iscan}(\overline{x_n}\ y) = s(\text{Iscan}\lambda(\overline{x_n}\ y)\ \beta_i\ \text{Iscan}\lambda(\overline{x_n}\ y))$$

where β_i is the input blank symbol.

$$\text{State}(y) = \text{lastchar}(\min w\text{By}[\text{inQ}(\text{lastchar}(w))])$$

State(y) is the current state symbol contained in y.

$$\text{Wscan}(y) = \text{lastchar}(\min w\text{By}[\text{State}(y)\text{lastchar}(w)\text{P}y])$$

Wscan(y) is the symbol following State(y) in y and hence is the currently scanned symbol on the work tape symbolized in y. We shall agree to always append a β_w if there is no other character to the right of the state symbol in y after the state symbol has been moved, hence $\text{Wscan}(y) \neq \lambda$ for all valid y.

Wprescan(y) = lastchar(min wBy[lastchar(w)State(y)Py])

Wbegin(y) = skip(Ipos(y)$ (min wBy[(w)Wprescan(y)State(y)Py]))

Wend(y) = skip((min wBy[State(y)Wscan(y)Ew]) y)

The above three functions are designed to extract the remaining portions of the symbolized work tape in y so that

y = Ipos(y)$Wbegin(y)Wprescan(y)State(y)Wscan(y)Wend(y).

We now define three functions which depend in detail on the structure of M, in particular the octuples which compose δ. Since $\Gamma \cup \{\beta_i\}$, W and Q are finite and the transition function δ is defined over the domain $(\Gamma \cup \{\beta_i\}) \times W \times Q$ it must be finite and we can use definition by cases to define NextIpos_M, NextWT_M and Outsym_M, as will be described shortly.

Let $\delta = \{\text{oct}_1, \text{oct}_2, \ldots, \text{oct}_N\}$ and define the set $\{R_1, R_2, \ldots, R_N\}$ of predicates over Σ such that if

$$\text{oct}_j = (q_j, i_j, w_j, q'_j, i\tau_j, w'_j, w\tau_j, \theta_j)$$

then R_j is defined in $\mathcal{L}$ by

$$R_j(\overline{x_n}\ y) \Leftrightarrow \text{State}(y)=q_j \wedge \text{Iscan}(\overline{x_n}\ y)=i_j \wedge \text{Wscan}(y)=w_j.$$

If the WTTM M is well-defined there is at most one octuple in the transition function δ for each combination of state, input symbol and work tape symbol. Therefore at most one R_j is true for any $\overline{x_n}$ y assuming that y is a valid symbolization of an ID in the computation of M on input $\overline{x_n}$ over Γ. Since we will be symbolizing computations of a WTTM M that computes a total function, exactly one R_j will be true at each step of the symbolization.

Let oct_j be as above throughout the following definitions.

$$\text{NextIpos}_M(\overline{x_n}\ y) = \begin{cases} f_1(\overline{x_n}\ y) & \text{if } R_1(\overline{x_n}\ y) \\ \vdots & \vdots \\ f_N(\overline{x_n}\ y) & \text{if } R_N(\overline{x_n}\ y) \end{cases}$$

$\text{NextIpos}_M(\overline{x_n}\ y)$ is the reverse binary representation of the next input position of M computing on input $\overline{x_n}$ over Γ' after the ID symbolized by y, where each

$$f_j(\overline{x_n}\ y) = \begin{cases} \text{monus1(Ipos(y))} & \text{if } i\tau_j = -1 \\ \text{Ipos(y)} & \text{if } i\tau_j = 0 \\ \text{plus1(Ipos(y))} & \text{if } i\tau_j = +1 \end{cases}$$

is one of the above three functions in $\mathcal{L}$ depending on the structure of δ as shown.

$$\text{NextWT}_M(\overline{x_n}\ y) = \begin{cases} g_1(\overline{x_n}\ y) & \text{if } R_1(\overline{x_n}\ y) \\ \vdots & \vdots \\ g_N(\overline{x_n}\ y) & \text{if } R_N(\overline{x_n}\ y) \end{cases}$$

$\text{NextWT}_M(\overline{x_n}\ y)$ is the symbolization of the next work tape and state of M computing on input $\overline{x_n}$ after the ID symbolized by y, where each

$$g_j(\overline{x_n}\ y) = \begin{cases} \text{Wbegin(y)}(q_j')\text{Wprescan(y)}(w_j')\text{Wend(y)} & \text{if } w\tau_j = -1 \\ \text{Wbegin(y)Wprescan(y)}(q_j')(w_j')\text{Wend(y)} & \text{if } w\tau_j = 0 \\ \text{Wbegin(y)Wprescan(y)}(w_j')(q_j')\text{add}\beta(\text{Wend(y)}) & \text{if } w\tau_j = +1 \end{cases}$$

is one of the above three functions in $\mathcal{L}$ depending on the structure of δ as shown and $\text{add}\beta(v) = s(v\ \beta_w\ v)$ adds a work tape blank to the symbolization of the next work tape if the state symbol is moved to the right end of the symbolized work tape. This assures that there is always some character following the state symbol in a symbolized ID y in order to insure that $\text{Wscan}(y) \neq \lambda$ for all y which occur in our symbolized computation of M. Since all WTM's are designed by convention not to attempt to move left of the initially scanned work tape square, no such

mechanism is necessary for left transitions on the work tape.

$$\text{Outsym}_M(\overline{x_n}\ y) = \begin{cases} \sigma_1 & \text{if } R_1(\overline{x_n}\ y) \\ \vdots & \vdots \\ \sigma_N & \text{if } R_N(\overline{x_n}\ y) \end{cases}$$

$\text{Outsym}_M(\overline{x_n}\ y)$ is the output symbol written on the output tape by M computing on input $\overline{x_n}$ over Γ for the move after the ID symbolized by y, where $\sigma_j = \theta_j \in \Gamma \cup \{\lambda\}$. Outsym_M has the value λ whenever some transition of M does not write an output symbol.

We can now define the function NextID_M such that $\text{NextID}_M(\overline{x_n}\ y)$ is the symbolization of the ID which follows y in the symbolized computation of M on input $\overline{x_n}$ over Γ.

$$\text{NextID}_M(\overline{x_n}\ y) = \text{NextIpos}_M(\overline{x_n}\ y)\$\text{NextWT}_M(\overline{x_n}\ y)$$

NextID_M serves to define a trace function $\text{ID}_M(\overline{x_n}\ y)$ which equals the symbolization of the $|y|$-th ID in the computation of M on input $\overline{x_n}$ over Γ.

$$\text{ID}_M(\overline{x_n}\ \lambda) = 0\$q_0\beta_w$$

$$\text{ID}_M(\overline{x_n}\ y\sigma) = \text{NextID}_M(\overline{x_n}\ \text{ID}_M(\overline{x_n}\ y)) \qquad \sigma \in \Sigma$$

Both NextIpos_M and NextWT_M were designed to lengthen the symbolization of an ID only if a new input or work tape square is scanned. Thus the length of the symbolization of the work tape and state of M is bounded by $s_M(\overline{x_n}) + 1$. Also since WTTM's do not scan beyond the blanks at the ends of their input, the length of the reverse binary representation of the current input position of M is bounded by $\left|\overleftarrow{|\overline{x_n}|} + 1\right|$. Therefore

$$\begin{aligned} \left|\text{ID}_M(\overline{x_n})\right| &\leq \left|\overleftarrow{|\overline{x_n}|}\right| + s_M(\overline{x_n}) + 2 \\ &\leq (K_1 + 1) \cdot \log_2|\overline{x_n}| + (K_3 + 3) \\ &\leq (K_1 + 1) \cdot \log_2|\overline{x_n}\ y| + (K_3 + 3) \end{aligned}$$

and $ID_M(\overline{x_n}\ y)$ is properly definable by log bounded recursion on notation.

The trace function ID_M is now used to define $Halts_M$ and Out_M. $Halts_M(\overline{x_n}\ y)$ is the predicate over Σ which is true iff the $|y|$-th ID of the computation of WTTM M on input $\overline{x_n}$ over Γ contains a halting state.

$$Halts_M(\overline{x_n}\ y) \Leftrightarrow \exists q P(ID_M(\overline{x_n}\ y))[inF(q)]$$

$Out_M(\overline{x_n}\ y)$ is the output of the symbolized computation of WTTM M on input $\overline{x_n}$ over Γ after $|y|$ steps.

$$Out_M(\overline{x_n}\ \lambda) = \lambda$$

$$Out_M(\overline{x_n}\ y\sigma) = Out_M(\overline{x_n}\ y)Outsym_M(\overline{x_n}\ ID_M(\overline{x_n}\ y)) \qquad \sigma \in \Sigma$$

Since $Outsym_M(\overline{x_n}\ y) \in \Gamma \cup \{\lambda\}$, $Out_M(\overline{x_n}\ y) \in \Gamma^*$ for any arguments over Γ. An examination of the definitions of b_M, $Halts_M$ and Out_M reveals that for any $\overline{x_n}$ over Γ

$$f(\overline{x_n}) = Out_M(\overline{x_n}\ \min\ yB(b_M(\overline{x_n}))[Halts_M(\overline{x_n}\ y)])$$

In addition each stage in the construction of b_M, $Halts_M$ and Out_M used only functions and predicates proven in previous sections to be in $\mathcal{L}$ and functional operations for which $\mathcal{L}$ has been proven to be closed. Therefore b_M, $Halts_M$ and Out_M are all contained in $\mathcal{L}$ and we conclude that $f \in \mathcal{L}$. Since f was an arbitrary function in logspace, logspace $\subseteq \mathcal{L}$ has been established.

Corollary 14: logspace = $\mathcal{L}$

proof: Immediate from Theorem 9 and Theorem 13.

FURTHER RESULTS

The following section contains additional closure properties of $\mathcal{L}$ = logspace.

We will need to define the following auxiliary functions in $\mathcal{L}$ over some arbitrary alphabet Σ.

$\text{firstchar}(\lambda) = \lambda$

$\text{firstchar}(y\sigma) = s(\text{firstchar}(y)\ \sigma\ y) \qquad \sigma \in \Sigma$

$|\text{firstchar}(y)| \leq 1$

firstchar(y) is the symbol in Σ which begins y.

$\text{back}(x\ y) = \text{firstchar}(\text{skip}(\text{skip}(x\ y)\ y))$

back(x y) is the $|x|$-th symbol back from the right end of y.

$\text{revl}(x\ \lambda) = \lambda$

$\text{revl}(x\ y\sigma) = \text{revl}(x\ y)\,\text{back}(x\ y\sigma) \qquad \sigma \in \Sigma$

revl(x y) is the string which contains the last $|y|$ symbols of x in reverse order.

$\text{rev}(x) = \text{revl}(x\ x)$

rev(x) is the string composed of the symbols of x in reverse order. Notice that rev $\in \mathcal{L}$.

An n+1-variable function f over Σ is defined from functions g, h_1 and h_2 by <u>two sided recursion of concatenation</u> if f satisfies

$f(\overline{x_n}\ \lambda) = g(\overline{x_n})$

$f(\overline{x_n}\ y\sigma) = h_1(\overline{x_n}\ y\ \sigma)f(\overline{x_n}\ y)h_2(\overline{x_n}\ y\ \sigma) \qquad \sigma \in \Sigma$

<u>Theorem 15</u>: $\mathcal{L}$ and hence <u>logspace</u> is closed under two sided recursion of concatenation.

proof: Let f be defined by two sided recursion of concatenation from the functions g, h_1 and h_2 contained in $\mathcal{L}$ over some arbitrary alphabet

such that

$$f(\overline{x_n}\ \lambda) = g(\overline{x_n})$$

$$f(\overline{x_n}\ y\sigma) = h_1(\overline{x_n}\ y\ \sigma)f(\overline{x_n}\ y)h_2(\overline{x_n}\ y\ \sigma) \qquad \sigma \in \Sigma$$

Define the functions f_1 and f_2 from rev, g, h_1 and h_2 using recursion of concatenation such that $f_1, f_2 \in \mathcal{L}$

$$f_1(\overline{x_n}\ \lambda) = \lambda$$

$$f_1(\overline{x_n}\ y\sigma) = f_1(\overline{x_n}\ y)\mathrm{rev}(h_1(\overline{x_n}\ y\ \sigma)) \qquad \sigma \in \Sigma$$

$$f_2(\overline{x_n}\ \lambda) = g(\overline{x_n})$$

$$f_2(\overline{x_n}\ y\sigma) = f_2(\overline{x_n}\ y)h_2(\overline{x_n}\ y\ \sigma) \qquad \sigma \in \Sigma$$

It is clear that $f(\overline{x_n}\ y) = \mathrm{rev}(f_1(\overline{x_n}\ y))f_2(\overline{x_n}\ y)$. Therefore since $\mathrm{rev} \in \mathcal{L}$ we conclude that $f \in \mathcal{L}$.

An n+1-variable function f over Σ is defined from functions g and h by <u>backwards recursion on notation</u> if f satisfies

$$f(\overline{x_n}\ \lambda) = g(\overline{x_n})$$

$$f(\overline{x_n}\ \sigma y) = h(\overline{x_n}\ \sigma\ y\ f(\overline{x_n}\ y)) \qquad \sigma \in \Sigma$$

We can define a function f' by (forward) recursion on notation from g and h such that

$$f'(\overline{x_n}\ \lambda) = g(\overline{x_n})$$

$$f'(\overline{x_n}\ y\sigma) = h(\overline{x_n}\ \sigma\ \mathrm{rev}(y)\ f(\overline{x_n}\ \mathrm{rev}(y))$$

Notice that $f(\overline{x_n}\ y) = f'(\overline{x_n}\ \mathrm{rev}(y))$.

If $g, h \in \mathcal{L}$ and the form of definition by backwards recursion on notation of f from g and h is restricted by either

$$h(\overline{x_n}\ w\ y\ z) = (z)h'(\overline{x_n}\ w\ y) \qquad h' \in \mathcal{L}, \text{ or}$$

$$|f(\overline{x_n}\ y)| \leq K_1 \cdot \log_2|\overline{x_n}\ y| + K_2 \text{ for constants } K_1 \text{ and } K_2,$$

then, since $\mathrm{rev} \in \mathcal{L}$, the above argument has informally proved that $\mathcal{L}$ is

closed under both backwards recursion of concatenation and backwards log bounded recursion on notation, which we will simply state as:

Claim 16: $\mathcal{L}$ is closed under backwards recursion of concatenation where if $g,h' \in \mathcal{L}$ then f satisfies

$$f(\overline{x_n}\ \lambda) = g(\overline{x_n})$$
$$f(\overline{x_n}\ \sigma y) = f(\overline{x_n}\ y)h'(\overline{x_n}\ \sigma\ y)$$

and $f \in \mathcal{L}$.

Claim 17: $\mathcal{L}$ is closed under backwards log bounded recursion on notation where if $g,h \in \mathcal{L}$ then f satisfies

$$f(\overline{x_n}\ \lambda) = g(\overline{x_n})$$
$$f(\overline{x_n}\ \sigma y) = h(\overline{x_n}\ \sigma\ y\ f(\overline{x_n}\ y))$$
$$|f(\overline{x_n}\ y)| \leq K_1 \cdot \log_2|\overline{x_n}\ y| + K_2$$

for some constants K_1 and K_2, and $f \in \mathcal{L}$.

The n+1-varibale functions $f_1, f_2, \ldots, f_k$ over some alphabet Σ are defined by simultaneous bounded recursion from the functions $g_1, g_2, \ldots, g_k$ and $h_1, h_2, \ldots, h_k$ if they satisfy

$$f_1(\overline{x_n}\ \lambda) = g_1(\overline{x_n})$$
$$\vdots$$
$$f_k(\overline{x_n}\ \lambda) = g_k(\overline{x_n})$$
$$f_1(\overline{x_n}\ y\sigma) = h_1(\overline{x_n}\ y\ \sigma\ f_1(\overline{x_n}\ y) \cdots f_k(\overline{x_n}\ y))$$
$$\vdots$$
$$f_k(\overline{x_n}\ y\sigma) = h_k(\overline{x_n}\ y\ \sigma\ f_1(\overline{x_n}\ y) \cdots f_k(\overline{x_n}\ y))$$
$$|f_1(\overline{x_n}\ y)| \leq K_{11} \cdot \log_2|\overline{x_n}\ y| + K_{21}$$
$$\vdots$$
$$|f_k(\overline{x_n}\ y)| \leq K_{1k} \cdot \log_2|\overline{x_n}\ y| + K_{2k}$$

where σ ranges over Σ.

Theorem 18: $\mathcal{L}$ and hence logspace is closed under simultaneous bounded recursion.

proof: Let the functions $f_1, f_2, \ldots, f_k$ over some alphabet Σ be defined by simultaneous bounded recursion from the functions $g_1, g_2, \ldots, g_k$ and $h_1, h_2, \ldots, h_k$ contained in $\mathcal{L}$ according to the form in the above definition.

We will define a function F over $\Sigma \cup \{\#\}$ such that

$$F(\overline{x_n}\ y) = f_1(\overline{x_n}\ y)\#f_2(\overline{x_n}\ y)\#\ldots\#f_k(\overline{x_n}\ y)\#.$$

This requires that we define the extraction functions $p_1, p_2, \ldots, p_k$ in $\mathcal{L}$ which obtain each of the k substrings before the #'s in a string containing k #'s.

$$p_1(z) = \min\ wBz\left[w\#Bz\right]$$

$$p_{j+1}(z) = \min\ wB(\mathrm{skip}(p_1(z)\ldots p_j(z))\ z)\left[w\#B(\mathrm{skip}(p_1(z)\ldots p_j(z))\ z)\right]$$

Define G and H in $\mathcal{L}$ such that

$$G(\overline{x_n}) = g_1(\overline{x_n})\#g_2(\overline{x_n})\#\ldots\#g_k(\overline{x_n})\#$$

$$H(\overline{x_n}\ y\ w\ z) = h_1(\overline{x_n}\ y\ w\ p_1(z)\ p_2(z)\ \ldots\ p_k(z)).$$

F is now defined in $\mathcal{L}$ by

$$F(\overline{x_n}\ \lambda) = G(\overline{x_n})$$

$$F(\overline{x_n}\ y\sigma) = H(\overline{x_n}\ y\ \sigma\ F(\overline{x_n}\ y)) \qquad \sigma \in \Sigma$$

Clearly $F(\overline{x_n}\ y) = f_1(\overline{x_n}\ y)\#f_2(\overline{x_n}\ y)\#\ldots\#f_k(\overline{x_n}\ y)\#$. Hence

$$\left|F(\overline{x_n}\ y)\right| \leq (K_{11} + \ldots + K_{1k}) \cdot \log_2\left|\overline{x_n}\ y\right| + (K_{21} + \ldots + K_{2k}) + k$$

Therefore $F \in \mathcal{L}$. For each $1 \leq j \leq k$

$$f_j(\overline{x_n}\ y) = p_j(F(\overline{x_n}\ y))$$

Hence $f_j \in \mathcal{L}$. Therefore $\mathcal{L}$ and hence logspace is closed under simultaneous bounded recursion.

Theorem 19: Addition is a member of $\mathcal{L}$ and hence is log space computable.

proof: The following construction of add(x y) which has the value of the reverse binary representation of $\langle x \rangle + \langle y \rangle$ is carried out in $\mathcal{L}$.

$$\text{carrysym}(x\ y\ z) = \begin{cases} 1 & \text{if } xyz{=}011 \vee xyz{=}101 \vee xyz{=}110 \vee xyz{=}111 \\ 0 & \text{if } xyz{=}000 \vee xyz{=}001 \vee xyz{=}010 \vee xyz{=}100 \end{cases}$$

carrysym(x y z) is the symbol representing the carry when the digits x, y and z are added.

$$\text{addsym}(x\ y\ z) = \begin{cases} 1 & \text{if } xyz{=}001 \vee xyz{=}010 \vee xyz{=}100 \vee xyz{=}111 \\ 0 & \text{if } xyz{=}011 \vee xyz{=}101 \vee xyz{=}110 \vee xyz{=}000 \end{cases}$$

addsym(x y z) is the symbol representing the least significant digit when the digits x, y and z are added.

carry(x λ) = 0

carry(x yτ) = carrysym(firstchar(skip(y x)) τ carry(x y))

carry(x y) is the carry digit when x and y are added. Note that

$|\text{carry}(x\ y)| \leq 1$.

Hence carry $\in \mathcal{L}$. Similarly

shortadd(x λ) = λ

shortadd(x yτ) = shortadd(x y)addsym(firstchar(skip(y x)) τ carry(x y))

shortadd(x y) is the sum of x and y up to the length of y. In order to add the remaining digits if $|y| \leq |x|$

add(x y) = shortadd(x y)s(plus1(skip(y x)) skip(y x) carry(x y)=1)

add(x y) has the value $\overleftarrow{\langle x \rangle + \langle y \rangle}$ and has been constructed in $\mathcal{L}$. Therefore add $\in \mathcal{L}$.

Theorem 20: Multiplication is contained in $\mathcal{L}$ and hence is log space computable.

proof: The following construction of mult(x y), which has the value of the reverse binary representation of $\langle x \rangle \cdot \langle y \rangle$, is carried out in $\mathcal{L}$.

bitmult(x u w z) = (w=1 $\wedge$ lastchar(min vBx[ones(v)=skip(u ones(z))])=1)

bitmult(x u w z) is not equal to λ iff the $|u|+1$-st product in the sum of the $|z|$-th column (numbered from the least significant bit) in the standard multiplication algorithm for computing $\langle x \rangle \cdot \langle y \rangle$ is a 1, where w is assumed to be the symbol in y such that uwBy.

addcol(x λ z) = 0

addcol(x uσ z) = s(plus1(addcol(x u z)) addcol(x u z) bitmult(x u σ z))

addcol(x u z) is the reverse binary representation of the $|u|$-th subtotal in the $|z|$-th column in the standard multiplication algorithm for computing $\langle x \rangle \cdot \langle y \rangle$, where it is assumed that uBy. Note that

$$|\text{addcol}(x\ u\ z)| \leq |\underbrace{\text{plus1}(\ldots\text{plus1}(0)\ldots)}_{|u| \text{ nested occurrences of plus1}}|$$

$$\leq |\overleftarrow{|u|}|$$

$$\leq \log_2|u| + 1$$

$$\leq \log_2|x\ u\ z| + 1$$

and hence addcol is defined properly within $\mathcal{L}$.

carry(x y λ) = 0

carry(x y zσ) = skip(1 add(addcol(x y zσ) carry(x y z))

carry(x y z) is the reverse binary representation of the carry for the $|z|$-th column in the standard multiplication algorithm for computing $\langle x \rangle \cdot \langle y \rangle$. Note that

$$|\text{carry}(x\ y\ z)| \leq |\underbrace{\text{add}(\text{len}(y) \ldots \text{add}(\text{len}(y)\ 0)\ldots)}_{|z| \text{ nested occurrences of } \text{add}(\text{len}(y)\ *)}|$$

$$\leq |\overleftarrow{|z| \cdot |y|}|$$

$$\leq |\overleftarrow{|z|}| + |\overleftarrow{|y|}|$$

$$\leq 2 \cdot \log_2 |x\ y\ z| + 2$$

and hence carry is defined properly within $\mathcal{L}$.

multl(x y λ) = λ

multl(x y z$\mathfrak{T}$) = multl(x y z)firstchar(add(addcol(x y z$\mathfrak{T}$) carry(x y z)))

multl(x y z) is the first |z| symbols in the reverse binary representation of $\langle x \rangle \cdot \langle y \rangle$.

mult(x y) = min zB(multl(x y xy))[$\neg\exists$wP(skip(z multl(x y xy))[w=1])]

mult(x y) has the value of the reverse binary representation of $\langle x \rangle \cdot \langle y \rangle$ and has been constructed within $\mathcal{L}$. Therefore mult $\in \mathcal{L}$.

APPENDIX 1

Appendix 1 was not written. For a description of Turing machines which copy, add 1, obtain the binary representation of a number in unary and other simple operations, see any text containing an introduction to Turing machines such as [H].

BIBLIOGRAPHY

[B] Bennett, James H., On Spectra, Doctoral Dissertation, Princeton University, 1962.

[C] Cook, Stephen A., The Complexity of Theorem Proving Procedures, Conference Record of 3rd Annual ACM Symposium on Theory of Computing, pp 151-158, 1970.

[G] Grzegorczyk, A., Some Classes of Recursive Functions, Rozprawy Matematyczne, Vol 4, p 4, 1953.

[H] Hennie, F. C., Notes for 6.262--Computability, Formal Systems and Logic, Massachusetts Institute of Technology, 1971.

[J1] Jones, Neil D., Preliminary Report, Reducibility Among Combinatorial Problems in log n Space.

[J2] Jones, Neil D., Context Free Languages and Rudimentary Attributes, Mathematical Systems Theory, Vol 3, No 2, 1968.

[K] Karp, Richard M., Reducibility Among Combinatorial Problems, Complexity of Computer Computations, Miller and Thatcher, eds, 1972.

[R] Ritchie, R. W., Classes of Predictably Computable Functions, Transactions American Mathematical Society, Vol 106, p 139, 1963.

[W] Warkentin, John C., Small Classes of Recursive Functions and Relations, Doctoral Dissertation, University of Waterloo, 1971.