On A Class of Uniform Algorithms

Sanat K. Basu

Tech Report: Dept. of C.S., Univ of Neb. Lincoln
Jan 1, 1985

Abstract

A nondeterministic Turing machine model called S-machine is introduced. This machine performs its computations during successive sweeps of the tape, using sets of states and sets of symbols. A deterministic simulator of this machine in the form of a while-do loop is constructed. A relation of equivalence between sets of states and a corresponding relation between sets of symbols is introduced. These relations are then extended to define equivalence classes of crossing functions. Information about the equivalence class of the crossing function of the scanned segment of the tape is contained in a context graph. A procedure for computing this graph is described. It is shown that successive context graphs can be computed in a constant number of steps. Using the context graph it is shown that the while-do simulator of the S-machine is uniform over a linear data domain. We then obtain a deterministic algorithm to decide whether an input of size n will be accepted by the S-machine. This algorithm uses no more than $O(S(n)^2)$ steps where S(n) is the spacebound of the S-machine. Corresponding results in terms of the standard Turing machine model are then derived.

Keywords: non-determins-m, uniform programs, time and space bound.

The problems of specification, development and verification of programs have constituted a very active area of research over the past decade. The central problem in this area can be stated in the following way.

Suppose we are given the formal specifications of a program, in the form of a mapping of input data objects to output data objects. How should a program be constructed so that its total correctness with respect to the given specifications can be systematically demonstrated. It became clear very early that the process of verification of the program and its development were closely interrelated. A very important step towards a solution of this problem was made possible by the strategies of top down program development and the notion of structured programming. [4] Suppose now a structured program is given as a candidate, and the problem is to demonstrate the total correctness of this program relative to given specifications. Using the method of inductive assertions [5] requires that an assertion be provided at each cutpoint of a loop. These loop assertions are induction hypotheses, required for proof by induction. If we wish to automate the proof mechanism, we should be able to generate these loop assertions from the given program and its specifications.

Investigations into this problem have shown [1,2,8] that it is possible to obtain this information, provided the input-output behavior of the program has certain properties.

Consider the domain of (input) data objects that are transformed into (output) data objects. We impose a criterion of unique decomposability on the data objects. Thus we assume that each

(non-basic) data object is uniquely decomposable into a linear sequence of basic data objects. We regard the program behavior as uniform, if the output data object can be systematically obtained from the composition of the input data object and the program behavior on the basic objects. This property in particular implies that the program behavior on a basic object be independent of the context in which it may appear within an arbitrary data object. A number of examples of programs uniform in this sense have been studies in [1], [2].

The question now arises as to whether a uniform program can be effectively obtained from an arbitrary recursive decision procedure. In general, we can expect such a uniform decision algorithm to be more efficient, provided the decomposition of the input data object can be achieved easily.

In this paper we answer the above question affirmatively. We begin with a modified Turing machine acceptor model and from this obtain an equivalent uniform program. The resulting decision algorithm is highly efficient.

A fundamental relation between time and space complexity is revealed by this approach. We have shown that every language accepted by a deterministic Turing machine within space bound $S(n)$ can be accepted by a deterministic algorithm within $O(S(n)^2)$ steps. This solves a problem of Karp [7].

The notion of complexity of computations was introduced by Rabin [10]. A comprehensive review of developments since then can be found in [3].

The relation between time and space complexities have been

studied extensively. Hopcroft et al. [6] showed that every deterministic multitape Turing machine of time complexity T(n) can be simulated by a deterministic Turing machine of space complexity T(n)/log T(n). Similar time space trade off results have been obtained for several other classes of machines. [9]

The contents of this paper are organized into five sections. In section 1, we introduce the S-machine model. We then obtain a deterministic simulator for this machine in the form of a while-do loop, and show that it accepts the same language as the S-machine. The memory space used by this simulator is no more than a constant times that used by the S-machine.

In section 2 we introduce an equivalence relation between sets of states and a corresponding relation between sets of symbols. This equivalence relation is then extended to crossing functions.

In section 3 we define a context graph. The node labels of the context graph contain the equivalence class of the crossing function of the scanned segment of the tape at the end of a certain sequence of sweeps. An inductive procedure for computing the context graph is described. An upper bound on the number of steps required to compute the context graph at a stage of computation of the S-machine is then derived.

Section 4 shows that the context graph has the required property with respect to the scanned segment of the tape and sequences of sweeps over this segment.

In section 5 we define a configuration of the while-do simulator using the context graph. We define a weak and strong equivalence relation on the configurations. The input output behavior of the

while-do loop is shown to be uniform over a linear data domain. We then obtain an algorithm to decide whether an input of length n will be accepted by the S-machine. This deterministic algorithm uses no more than $(S(n)^2)$ steps where S(n) is the space bound of the S-machine.

Section 1: A nondeterministic sweep machine (S-machine):

In this section we introduce a nondeterministic Turing machine model called the sweep machine or S-machine. The S-machine performs its computations during successive sweeps of the tape. The ith sweep, $i > 1$ consists of i left moves followed by (i+1) right moves. The leftmost symbol on the tape in a special end marker symbol, which also signals the reversal of a sweep.

The S-machine operates with sets of tape symbols and sets of states. At any step of computation, the machine state would consist of a set of states and the scanned cell would contain a set of tape symbols. The set of next states of the machine and the set of tape symbols to be written in the scanned cell are determined by the use of a choice function. This choice function decides how the current states of the machine are to be paired with the symbols in the scanned cell. There are finitely many possible choice function values, each choice function is also a finite function. A computation of the S-machine consisting of k sweeps is completely specified by a choice function sequence of length $k^2 + 2k$. The S-machine is an acceptor model. An input is accepted by this machine if and only if there exists a choice function sequence that results in the machine entering an accepting state at the end of k sweeps, for some k. If the input is not accepted by the machine, then the sequence of sweeps could be continued indefinitely, with all possible choice function sequences, without M ever entering an accepting state. Formal definition of M follows.

An S-machine M is a sixtuple $M = \langle Q, \sum, \Gamma, \delta, \{q_0\}, H \rangle$ where Q is a

finite set of states

and $Q = Q^L \cup Q^R \cup P^R \cup \{q_0\}$.

Q^L is a set of leftmoving states, Q^R, P^R are sets of right moving states and $\{q_0\}$ a designated set of start states. All of these sets of states are mutually disjoint.

$\sum$ is a finite set of input symbols, excluding the symbols ɓ, ȼ, z. Γ is a finite set of tape symbols. Each element of Γ is an ordered pair $\langle x,y\rangle$ of symbols.

$$\Gamma = \{\langle x,y\rangle \mid x \in \sum \cup \{ɓ\} \text{ and } y \in \{ɓ,z\}\} \cup \{\langle ȼ,ȼ\rangle\};$$

$H \subseteq Q$ is a set of accepting states. Properties of H will be discussed later. The transition function δ is defined by the instruction formats listed below. The superscripts L and R are used to denote membership in Q^L and $Q^R \cup P^R$. L denotes a left move and R a right move.

(i) For each $q_0 \in \{q_0\}$, $\langle x,z\rangle \in \Gamma$, there exists $\langle x_1,z\rangle,\ldots,\langle x_k,z\rangle \in \Gamma$ and $q^L_{i_1},\ldots,q^L_{i_k} \in Q_L$ such that

$$(q_0,\langle x,z\rangle) \longrightarrow \{(q^L_{i_1},\langle x_1,z\rangle),\ldots,(q^L_{i_k},\langle x_k,z\rangle)\} \text{ is in } \delta.$$

(ii) For each $q^L \in Q^L$, $\langle x,ɓ\rangle \in \Gamma$, there exists $\langle x_1,ɓ\rangle,\ldots,\langle x_k,ɓ\rangle \in \Gamma$ and $q^L_{i_1},\ldots,q^L_{i_k} \in Q^L$ such that

$$(q^L,\langle x,ɓ\rangle) \longrightarrow \{(q^L_{i_1},\langle x_1,ɓ\rangle),\ldots,(q^L_{i_k},\langle x_k,ɓ\rangle)\} \text{ is in } \delta.$$

(iii) For each $q^L \in Q^L$, there exists $q^R \in Q^R$ such that

$$(q^L,\langle ȼ,ȼ\rangle) \longrightarrow (q^R,\langle ȼ,ȼ\rangle) \text{ is in } \delta.$$

(iv) For each $q^R \in Q^R$, $\langle x,ɓ\rangle \in \Gamma$, there exists $q^R_{i_1},\ldots,q^R_{i_k} \in Q^R$, $\langle x_1,ɓ\rangle,\ldots,\langle x_k,ɓ\rangle \in \Gamma$ such that

$$(q^R,\langle x,ɓ\rangle) \longrightarrow \{(q^R_{i_1},\langle x_1,ɓ\rangle)\ldots.(q^R_{i_k},\langle x_k,ɓ\rangle)\} \text{ is in } \delta.$$

(v) For each $q^R \in Q^R$, $\langle x,z\rangle \in \Gamma$, there exists $p^R_{i_1},\ldots,p^R_{i_k} \in P^R$ and

$\langle x_1, \not b\rangle, \ldots, \langle x_k, \not b\rangle \in \Gamma$ such that

$(q^R, \langle x, z\rangle) \longrightarrow \{(p^R_{i_1}, \langle x_1, \not b\rangle), \ldots, (p^R_{i_k}, \langle x_k, \not b\rangle)\}$ is in δ.

(vi) For each $p^R \in P^R$ and $\langle x, \not b\rangle \in \Gamma$, there exists $q^L_{i_1}, \ldots, q^L_{i_k} \in Q^L$ and $\langle x_1, z\rangle, \ldots, \langle x_k, z\rangle \in \Gamma$ such that

$(p^R_i, \langle x, \not b\rangle) \longrightarrow \{(q^L_{i_1}, \langle x_1, z\rangle), \ldots, (q^L_{i_k}, \langle x_k, z\rangle)\}$ is in δ.

We shall consider δ to be the smallest set of instructions closed under rules (i) to (vi) above. We shall assume that once the machine M is in a halting state, all subsequent states it enters into are halting states. The symbol $\not\in$ is used as a left end marker. The symbol z which only appears as the second component of a tape symbol is called a sweep marker. It marks the cell at which the next sweep will begin. The symbol $\not b$ plays a special role. We assume that M does not enter a halt state on scanning a $\langle \not b, \not b\rangle$ or $\langle \not b, z\rangle$ tape symbol. The machine M performs all its computations in terms of sets. At any stage of computation, each cell of the tape of M contains a (finite) set of tape symbols. The state of the machine M is a subset of left moving or right moving states. Each step of computation of M is assumed to consist of the following sequence of operations. a) scan the symbols in the cell, b) write new symbols, c) change states and d) move one cell to left or right. According to the description above, the direction of move in (d) is known. At any step i of computation, sc(i) will denote the index of the cell scanned in (a) and Ac(i) the set of states at the end of (c) above. It follows that at step i, $i \geq 1$, the machine scans the symbols of the cell sc(i) in states Ac(i-1). In computing Ac(i) from the symbols in cell sc(i) and the

states Ac(i-1), we use a choice function α. A particular choice function α maps each state of M to a subset of Γ. Before defining this computation, we introduce the notion of a product.

Let A be a set of states and B a set of tape symbols of M. We introduce the following operator *, called 'product'.

$$A*B = \{q \mid \exists x \varepsilon \Gamma, q_1 \varepsilon A, a_1 \varepsilon B \text{ such that } (q,x) \varepsilon \delta(q_1,a_1)\}$$

$$B*A = \{x \mid \exists q \varepsilon Q, q_1 \varepsilon A, a_1 \varepsilon B \text{ such that } (q_1,x) \varepsilon \delta(q_1,a_1)\}.$$

Thus A*B is the set of all possible next states of M and B*A the set of all possible symbols written in the scanned cell, when the machine state is a member of A and the symbol in the scanned cell a member of B.

Henceforth, we shall refer to a set of states of M as a <u>stateset</u> and a set of tape symbols of M as a <u>symbolset</u> respectively. Statesets and symbolsets are closed under union. Thus, if q_1, q_2 are statesets, so is q_1+q_2 and if a_1, a_2 are symbol subsets so is a_1+a_2. The following lemma shows that distributive properties hold. We use lower case symbols for statesets and symbolsets.

<u>Lemma 1.1</u> For any statesets q, q_1, q_2 and symbolsets a, a_1, a_2

$$q*(a_1+a_2) = q*a_1+q*a_2$$

$$(q_1+q_2)*a = q_1*a+q_2*a$$

$$(a_1+a_2)*q = a_1*q+a_2*q$$

$$a*(q_1+q_2) = a*q_1+a*q_2$$

<u>Proof</u>: From the definition above,

$p \varepsilon q * (a_1+a_2) \Leftrightarrow \exists p' \varepsilon q$, $x \varepsilon \Gamma$ and $b \varepsilon a_1 + a_2$ such that $(p,x) \varepsilon \delta(p',b)$

$\Leftrightarrow \exists p' \varepsilon q$, $x \varepsilon \Gamma$, and $b \varepsilon a_1$ such that

$(p,x) \varepsilon \delta (p',b)$ or $\exists p' \varepsilon q$, $x \varepsilon \Gamma$ and $b \varepsilon a_2$ such that

$(p,x) \varepsilon \delta (p',b)$

$\Leftrightarrow p \varepsilon q * a_1$ or $p \varepsilon q * a_2$

$\Leftrightarrow p \varepsilon q * a_1 + q * a_2$

Therefore, $q * (a_1+a_2) = q * a_1 + q * a_2$.

Other properties follow similarly.

A stateset q is <u>left moving</u> if $q \subseteq Q^L$, <u>right moving</u> if $q \subseteq Q^R$ and <u>halting</u> if $q \cap H \neq \emptyset$. A symbolset a is <u>marked</u> if every tape symbol x of a contains the sweep marker. The definitions of the instructions of M show that a stateset of M during any computaiton is a left moving, right moving or a halting stateset. The symbolset is either marked or unmarked.

Consider now the set of halting statesets Q_H. If M has been constructed to accept a recursive set L, then for every input w (encoded as defined subsequently), M must halt in an accepting or rejecting stateset. Accordingly, we assume that $H = H_A \cup H_R$ where H_A is the set of accepting states and H_R the set of rejecting states. $q \varepsilon Q_H$ is <u>accepting</u> if $q \cap H_A \neq \emptyset$ and <u>rejecting</u> if $q \cap H_A = \emptyset$.

We shall assume that there exist symbols $a_1, a_2 \varepsilon \Gamma$ such that $q \varepsilon H_A \Rightarrow \forall x \varepsilon \Gamma, \exists q' \varepsilon H_A$ such that $\delta(q,x) \overset{\text{contains}}{\wedge} (q', a_1)$ and $q \varepsilon H_R \Rightarrow \forall x \varepsilon \Gamma, \exists q' \varepsilon H_R$ such that $\delta(q,x) \overset{\text{contains}}{\wedge} (q', a_2)$. a_1 is therefore an accepting tapesymbol, and a_2 a rejecting tapesymbol. H_A and H_R would contain left moving and right moving states.

Let $\sum_A = \{a \subseteq \Gamma \mid a_1 \varepsilon a\}$, $\sum_R = \{a \subseteq \Gamma \mid a_2 \varepsilon a\}$,

$Q_A = \{q \subseteq Q \mid q \cap H_A \neq \emptyset\}$, $Q_R = \{q \subseteq Q \mid q \cap H \neq \emptyset$ and $q \cap H_A = \emptyset\}$

and $\{Q\} = \{q \mid q \subseteq Q^L \text{ or } q \subseteq Q^R\}$. Then we have $Q_H = Q_A \cup Q_R$, $Q_A \cap Q_R = \emptyset$. $\sum_A (\sum_R)$ is the set of accepting (rejecting) symbolsets, $Q_A (Q_R)$ the set of accepting (rejecting) halt statesets and $\{Q\}$ the set of statesets of M. We extend the product to sets of statesets. Let A,B be sets of statesets and symbolsets respectively. Then,

$A*B = \{q*a \mid q\varepsilon A, a\varepsilon B\}$, and $B*A = \{a*q \mid q\varepsilon A, a\varepsilon B\}$.

Then, $\{Q\}*\sum_A \subseteq Q_A$, $\sum_A *\{Q\} \subseteq \sum_A$, $\{Q\}*\sum_R \subseteq Q_R$ and $\sum_R *\{Q\} \subseteq \sum_R$.

Each Q_A and Q_R can be partitioned into leftmoving and rightmoving parts, so that $Q_A = Q_A^L \cup Q_R^R$ and $Q_R = Q_R^L \cup Q_R^R$.

Suppose $w = a_1 a_2 \ldots a_n \varepsilon \sum^*$ be an input string of length n. The initial tape configuraiton of M is the string of symbolsets

$\{\langle \not\vdash, \not\vdash \rangle\}\{\langle a_1, z\rangle\}\{\langle a_2, \not\dashv\rangle\}\ldots\{\langle a_n, \not\dashv\rangle\}$.

The leftmost symbolset is assumed to be in the cell with index 0. Cells with index > n are assumed to contain $\{\langle \not\dashv, \not\dashv\rangle\}$. The initial stateset is $\{q_0\}$, and the cell scanned is 1. From rule 1 of M, the machine would rewrite the symbolset in cell 1 and move left. Step 1 of the computation of M begins with M scanning cell 0 = sc(1) in stateset Ac(0) where,

$Ac(0) = \{q_0\} * \{\langle a_1, z\rangle\}$

and $cell'(1) = \{\langle a_1 z\rangle\} * \{q_0\}$

cell'(1) denotes the contents of cell(1) at the end of initial step. From rules (i) of M, $Ac(0) \subseteq Q^L$ and sc(1) = 0. The function sc(i) defining the index of the scanned cell at step i can be obtained from the description of M. In general,

sc(1) = 0,

and for i > 1, sc(i+1) = sc(i)+1 if $Ac(i) \subseteq Q^R$ or sc(i) = 0

$$= sc(i)-1 \text{ if } Ac(i) \subseteq Q^L \text{ and } sc(i) > 1$$
$$\text{or } Ac(i) \subseteq P^R.$$

If the scanned cell at step i is to the right of cell 0 and the machine moves left at the end of step i then the index of the scanned cell at step i+1 is one less than that at step i. A left move at the end of step i happens if either $Ac(i) \subseteq Q^L$ or by rule (vi) of M, $Ac(i) \subseteq P^R$. If $Ac(i) \subseteq Q^R$ or $sc(i) = 0$, the machine moves right at the end of the ith step, so that the index of the scanned cell at step i+1 is obtained by adding one to that at step i. A <u>choice function</u> α: $Q \dashrightarrow 2^{\Gamma}$ maps each state of M to a subset of tape symbols. Since there are only finitely many states and tape symbols, there are only finitely many possible choice functions.

For a given initial configuration, (Ac(0),sc(1)), a computation of length n of the S-machine M is uniquely described by a sequence $\alpha_1, \alpha_2, \ldots \alpha_n$ of choice functions. Symbolically,

$$(Ac(i), sc(i+1)) \xrightarrow[\alpha_{i+1}]{} (Ac(i+1), sc(i+2))$$

for $0 \leq i < n$.

We have defined Ac(1), sc(2) above. Given Ac(i), sc(i+1) and a choice function α_{i+1} we define below Ac(i+1) and cell (sc(i+1)) after the completion of the (i+1)st step. Since sc(i+1) and Ac(i+1) are now known, sc(i+2) can be computed. We shall use the notation cell'(sc(i+1)) to denote the contents of cell (sc(i+1)) after the completion of the (i+1)st step. cell(sc(i+1)) denotes the contents of cell(sc(i+1)) at the beginning of the (i+1)st step.

The set of active states at step (i+1), Ac(i+1) is defined as follows for $i > 0$:

(i) $sc(i+1) = 0$; then $cell(sc(i+1)) = \{\langle ¢, ¢\rangle\}$

and $Ac(i+1) = Ac(i) * \{\langle ¢, ¢\rangle\}$

$cell'(sc(i+1)) = \{\langle ¢, ¢\rangle\} * Ac(i) = \{\langle ¢, ¢\rangle\}$ by rule (iii) of M.

(ii) $sc(i+1) > 0$; for each $q \varepsilon Ac(i)$, let

$$B_q = \alpha_{i+1}(q) \cap cell(sc(i+1))$$

then,

$$Ac(i+1) = \bigcup_{q \varepsilon Ac(i)} \{q\} * B_q$$

$$cell'(sc(i+1)) = \bigcup_{q \varepsilon Ac(i)} B_q * \{q\}$$

Given the set of active states at stepi, Ac(i), and the symbols to be scanned in step i+1, cell(sc(i+1)), the choice function for (i+1)st step α_{i+1} assigns a (possibly empty) subset B_q of symbols in cell(sc(i+1) to each state q of Ac(i).

The new set of active states Ac(i+1) is the set of all possible next states of M resulting from the assignment above. i.e. Ac(i+1) is the set of all states that appear in the right hand side of an instruction $\delta(q,x)$, $q \varepsilon Ac(i)$, $x \varepsilon B_q$. The new set of symbols at cell (sc(i+1)) are all symbols that appear in the right hand side of an instruction $\delta(q,x)$, $q \varepsilon Ac(i)$, $x \varepsilon B_q$. Having computed Ac(i+1), sc(i+2) can be computed form Ac(i+1) and sc(i+1), thus completing the (i+1)st step of M.

Consider now a sequence $\alpha = \alpha_1\alpha_2\ldots\alpha_n$ of choice functions. α uniquely defines n steps of M's computation, since the initial step is defined without the use of any choice function. If however for some i, $1 \leq i \leq n$, $Ac(i) = \emptyset$, then for all $j \geq i$, $Ac(j) = \emptyset$. For,

from the definition of product, for any symbol set a, $\emptyset * a = \emptyset$.

Let α be a sequence of choice functions that does not lead to a null computation. Consider now the location of the marked symbolset. From the initial configuration, we find that the symbol set at cell 1 is marked. Consider the first sweep consisting of the moves LRR.

From rule (i) of M and the definition of cell'(1), all symbols in cell'(1) are marked, at the end of step 0. Further $Ac(0) \subseteq Q^L$ and $sc(1) = 0$. Since $cell(sc(1)) = \{\langle \not\phi, \not\phi \rangle\}$, by rule (iii) of M, $Ac(1) \subseteq Q^R$ for any choice funcction α_1, provided α_1 does not result in $Ac(1) = \emptyset$. Then $sc(2) = 1$ and the symbolset in cell 1 is marked. Using α_2 and rule (v) of M yields $Ac(2) \subseteq P^R$ and cell'(sc(2)) is unmarked. Since $Ac(2) \subseteq P^R$, and $sc(3) = 2$, from rule (vi) of M, each symbol in cell'(sc(3)) at end of step 3 is marked. Also from (vi) of M, $Ac(3) \subseteq Q^L$. Therefore at the end of the first sweep, the cell containing the marked symbolset has shifted one cell to right. The location of the sweep marker therefore defines both the number of sweeps performed as well as the cell from which the next sweep begins. Further, this behavior of the sweep marker holds for any sequence of choice functions that does not lead to a null computation. We note that the sweep marker is moved only at the last right moving step of a sweep. If therefore, the choice function sequence defines only part of a sweep, the location of the marked symbol-set defines the last complete sweep performed.

Consider now the ith sweep of M. This consists of i left moves followed by i+1 right moves.

The total number of steps of M required for i sweeps is therefore

$\sum_{j=1}^{i} (2j+1) = i^2+2i$, the last step of sweep i being the use of a choice function at cell (i+1).

Consider now the (i+1)st sweep. This sweep starts with states $Ac(i^2+2i) \subseteq Q^L$ and $sc(i^2+2i+1) = i$. Since the (i+1)st sweep would move left until cell 0 is scanned and then move right to cell (i+2), for $i^2 + 2i < j \leq i^2 + 3i + 1$, $sc(j) = (i^2+3i+1) - j$, and for $i^2+3i+1 < j \leq i^2+4i+3$, $sc(j) = j-(i^2+3i+1)$.

At step $i^2 + 2i + k$, $1 \leq k \leq i + 1$, the set of active states is $Ac(i^2+2i+k-1)$ and the scanned cell is $sc(i^2+2i+k) = i^2 + 3i + 1 - (i^2+2i+k) = (i-k+1)$. The cell $sc(i^2+2i+k)$ was last scanned by M during the right moving half of the ith sweep. At the end of the ith sweep, $sc(i^2+2i) = i + 1$ and therefore $sc(i^2+2i-k) = i + 1 - k = i - (k - 1)$. Therefore for $1 \leq k \leq i+1$, at step $i^2 + 2i + k$ the active states $Ac(i^2+2i+k-1)$ scan the symbols in cell $i - (k - 1)$ which were written in step $i^2 + 2i - k$. We thus have the following:

Lemma 1.2 For $i \geq 1$, $1 \leq k \leq i+1$, $cell(sc(i^2+2i+k)) = cell'(sc(i^2+2i-k))$.

Consider now the right moving half of the (i+1)st sweep. The machine moves from cell 0 to cell(i+2). At step $i^2 + 3i + k$, $1 < k \leq i+3$, the scanned cell is $sc(i^2+3i+k) = k - 1$ since $sc(i^2+3i+1) = 0$. For $1 < k < i+3$, the cell (k-1) is last scanned by the left half of the (i+1)st sweep. Since during this part of the (i+1)st sweep, $sc(j) = (i^2+3i+1) - j$, $i^2 + 2i < j \leq i^2 + 3i + 1$, it follows that $sc(i^2+3i+2-k) = k - 1$. We thus have the following:

Lemma 1.3: For $i \geq 1$, $1 \leq k \leq i+1$, $cell(sc(i^2+3i+1+k)) = cell'(i^2+3i+1-k)$.

For $k = i + 2$, $sc(i^2+4i+3) = i + 2$. This cell is scanned for the first time during sweep(i+1) and thus $cell(sc(i^2+4i+3))$ is the initial contents of cell(i+2).

Consider now a computation of M defined by a sequence of choice function $\alpha = \alpha_1, \alpha_2 \ldots \alpha_i$. Let $|\alpha| = i$ be the length of α. Then $(Ac(i-1), Sc(i)) \xrightarrow[\alpha_i]{} (Ac(i), Sc(i+1))$. Let $\alpha^*_{(i)}$ denote all sequences of choice functions of length = i. We have the following:

Lemma 1.4 $\bigcup_{\alpha \varepsilon \alpha^*_{(i)}} Ac(i) * \bigcup_{\alpha \varepsilon \alpha^*_{(i)}} cell(sc(i+1)) = \bigcup_{\alpha \varepsilon \alpha^*_{(i+1)}} Ac(i+1)$

and

$$\bigcup_{\alpha \varepsilon \alpha^*_{(i)}} cell(sc(i+1)) * \bigcup_{\alpha \varepsilon \alpha^*_{(i)}} Ac(i) = \bigcup_{\alpha \varepsilon \alpha^*_{(i+1)}} cell'(sc(i+1))$$

for $i > 0$.

Proof: From definitions, for $\alpha \varepsilon \alpha^*_{(i+1)}$, $\alpha = \beta\alpha_{i+1}$, $\beta \varepsilon \alpha^*_{(i)}$

$$Ac(i+1) = \{ q' \mid \exists\ q \varepsilon Ac(i),\ x \varepsilon B_q,\ x' \varepsilon \Gamma \text{ such that } (q',x') \varepsilon \delta(q,x) \}$$

and $cell'(sc(i+1)) = \{ x' \mid \exists\ q' \varepsilon Q,\ q \varepsilon Ac(i),\ x \varepsilon B_q$ such that $(q',x') \varepsilon \delta(q,x) \}$

where (Ac(i),sc(i+1)) is defined by the choice sequence β of length i, and $B_q = \alpha_{i+1}(q) \cap cell(sc(i+1))$ for each $q \varepsilon Ac(i)$. If α_{i+1} takes all possible values, $\alpha_{i+1}(q)$ is an arbitrary subset of Γ and hence each x in B_q could be an arbitrary element of cell(sc(i+1)). If now, β takes all possible values, then we have:

$$\bigcup_{\alpha \varepsilon \alpha^*_{(i+1)}} Ac(i+1) = \{ q' \mid \exists\ q \varepsilon \bigcup_{\alpha \varepsilon \alpha^*_{(i)}} Ac(i),\ x \varepsilon \bigcup_{\alpha \varepsilon \alpha^*_{(i)}} cell(sc(i+1)),\ x' \varepsilon \Gamma \text{ such that } (q',x') \varepsilon \delta(q,x) \}$$

$$= \bigcup_{\alpha \varepsilon \alpha^*_{(i)}} Ac(i) * \bigcup_{\alpha \varepsilon \alpha^*_{(i)}} cell(sc(i+1)) \text{ by definition.}$$

The second equation follows similarly.

In the following discussion, we shall omit the quantification of

the union over Ac(i) and cell(sc(i+1)) etc. This would not result in any ambiguity since defining Ac(i) (cell(sc(i+1))) requires a sequence of choice functions of length equal to i.

We now construct a program that simulates a S-machine on an arbitrary input of length n: We use a pascal like notation. T(n) is a function of n.

```
type cell = record
                st : set of states;
                sym : set of symbols;
            end;
var A : array[1..T(n)] of cell.
```

Initially, $symA(1) = cell'(1)$, $stA(1) = Ac(0)$,

$symA(j) = \{\langle a_j, \not b\rangle\}$ for $2 \leq j \leq n$,

$= \{\langle \not b, \not b\rangle\}$ for $n < j \leq T(n)$.

$stA(j) = \emptyset$ for $1 < j \leq T(n)$.

We shall permit the array index 0 for A, and let $symA(0) = \{\langle \not\phi, \not\phi\rangle\}$, $stA(0) = \emptyset$ initially. This notation preserves the correspondence between the indices of A and the cell indices of the tape of the S-machine M. We define the following sweep function;

```
function sweep(j:integer) : set of states;
    var ℓ : integer;
begin L1 : for ℓ = j downto 1 do begin
               stA(ℓ-1) := stA(ℓ) * symA(ℓ-1)
               symA(ℓ-1) := symA(ℓ-1) * stA(ℓ) end
           L2 : for ℓ = 1 to j+1 do begin
               stA(ℓ) := stA(ℓ-1) * symA(ℓ)
               symA(ℓ) := symA(ℓ) * stA(ℓ-1)
```

```
            end
            sweep := stA(j);
        end.
```

In the definition of function sweep, we have used the parameter j to locate the sweep marker. Since the location of the sweepmarker at the end of a sweep is predetermined, sweep could be considered as a parameterless function.

The following procedure sim simulates the S-machine M. Array A defined above is considered a global variable.

```
    procedure sim:
P̄1: var: i:integer; R;set of states;
    begin i := 1; R := nil;
    L: while R ∩ H <> nil do
        begin R := sweep (i);
            i := i+1
        end
    end
```

The following theorem establishes the contents of array A after ith iteration of the while-do loop L in procedure sim.

Theorem 1.1: After the ith iteration of the loop L, $i \geq 1$, for all j, $0 \leq j \leq i + 1$,

$$stA(j) = \bigcup_{\alpha} Ac(i^2+i-1+j)$$

$$symA(j) = \bigcup_{\alpha} cell'(sc(i^2+i-1+j))$$

Proof: By induction on i.

For i=1 consider the first iteration of L. The initial contents of array A have been defined above.

For the call sweep(1), in the first for loop, j=1=1 and thus

stA(0) = Ac(0) * {⟨¢,¢⟩};

symA(0) = {⟨¢,¢⟩} * Ac(0);

Since stA(1) = Ac(0), symA(0) = {⟨¢,¢⟩} at the beginning of the first for loop L_1 of function sweep. By definition, stA(0) = $\bigcup_\alpha$ Ac(1) and symA(0) = $\bigcup_\alpha$ cell'(sc(1)); Since cell(sc(1)) = cell(0) = {⟨¢,¢⟩} and by (iii) of M, (q,x) ε δ(q,⟨¢,¢⟩) implies x = ⟨¢,¢⟩, it follows that symA(0) = {⟨¢,¢⟩}, stA(0) = $\bigcup_\alpha$ Ac(1) after the execution of the first for loop in sweep. Consider now the second for loop L_2 in sweep for i=1. For ℓ=1, stA(1) = stA(0) * symA(1) = Ac(1) * cell(sc(2)) since sc(2) = sc(1) + 1 = 1. The contents of cell (1) have been defined as {⟨a_1,z⟩} * {q_0}, independent of a choice function and thus cell(sc(2)) = $\bigcup_\alpha$ cell(sc(2)).

Using lemma 1.4, stA(1) = $\bigcup_\alpha$ Ac(2) and similarly

symA(1) = $\bigcup_\alpha$ cell'(sc(2)).

For ℓ=2, in L_2, stA(2) = $\bigcup_\alpha$ Ac(2) * {⟨a_2,¢⟩}

= $\bigcup_\alpha$ Ac(2) * cell(sc(3)) = $\bigcup_\alpha$ Ac(3)

and symA(2) = $\bigcup_\alpha$ cell'(sc(3)).

Assume as induction hypothesis (H1) that the statement of the theorem holds after the ith iteration of the loop L in procedure sim. Consider the (i+1)st iteration of L with the call sweep (i+1). The range of the first for loop index ℓ is i+1 to 1. Let k, $1 \leq k \leq i+1$ be the number of iterations of the loop L_1 in function sweep. We show by induction on k that for $1 \leq k \leq i+1$,

(H2) stA(i+1-k) = $\bigcup_\alpha$ Ac(i^2+2i+k) (1.1)

symA(i+1-k) = $\bigcup_\alpha$ cell'(sc(i^2+2i+k))

For k=1, ℓ = i+1

$$stA(i) = stA(i+1) * symA(i)$$
$$= \bigcup_{\alpha} Ac(i^2+2i) * \bigcup_{\alpha} cell'(sc(i^2+2i-1))$$

using the induction hypothesis H1 and the fact that stA(i+1), symA(i) values are those at the end of ith iteration of loop L in procedure sim. From lemma 1.2, $cell'(sc(i^2+2i-1)) = cell(sc(i^2+2i+1))$.

Then,

$$stA(i) = \bigcup_{\alpha} Ac(i^2+2i) * \bigcup_{\alpha} cell(sc(i^2+2i+1))$$
$$= \bigcup_{\alpha} Ac(i^2+2i+1) \text{ by lemma 1.4}$$

Similarly $symA(i) = \bigcup_{\alpha} cell'(sc(i^2+2i+1))$ and thus the basis. Assume now that (H2) holds for k and consider the (k+1) st iteration of the loop L_1; $k + 1 \leq i + 1$. Then $\ell = i - k + 1$ and

$$stA(i-k) = stA(i-k+1) * symA(i-k)$$

From H2, $stA(i-k+1) = \bigcup_{\alpha} Ac(i^2+2i+k)$.

Since symA(i-k) has the value at the end of sweep (i), from H1, $symA(i-k) = \bigcup_{\alpha} cell'(sc(i^2+2i-1-k))$. Since $k + 1 \leq i + 1$, from lemma 1.2, $cell'(sc(i^2+2i-(k+1))) = cell(sc(i^2+2i+k+1))$.

Thus $stA(i-k) = \bigcup_{\alpha} Ac(i^2+2i+k) * \bigcup_{\alpha} cell(sc(i^2+2i+k+1))$

$$= \bigcup_{\alpha} Ac(i^2+2i+k+1) \text{ by lemma 1.4,}$$

and similarly, $symA(i-k) = \bigcup_{\alpha} cell'(sc(i^2+2i+k+1))$ as was to be shown. In Equation (1.1) as k ranges from 1 to i+1, i+1-k ranges from i to 0. Let j = i+1-k. Then at the termination of the for loop L_1 with the call sweep (i+1), we have for all j, $0 \leq j \leq i$,

$$stA(j) = \bigcup_{\alpha} Ac(i^2+3i+1-j)$$

and $$symA(j) = \bigcup_{\alpha} cell'(sc(i^2+3i+1-j)) \qquad (1.2)$$

Consider now the second for loop L_2. The range of ℓ is from 1 to i + 2. We show by induction that after ℓth iteration of the second loop, $1 \leq \ell \leq i + 2$,

$$stA(\ell) = \bigcup_{\alpha} Ac(i^2+3i+1+\ell)$$

$$symA(\ell) = \bigcup_{\alpha} cell'(sc(i^2+3i+1+\ell) \qquad (1.3)$$

For $\ell=1$,

$$stA(1) = stA(0) * symA(1)$$

$$= \bigcup_{\alpha} Ac(i^2+3i+1) * \bigcup_{\alpha} cell'(sc(i^2+3i)) \text{ from equation 1.2 above.}$$

From lemma 1.3,

$$cell'(sc(i^2+3i)) = cell(sc(i^2+3i+2))$$

and hence

$$stA(1) = \bigcup_{\alpha} Ac(i^2+3i+1) * \bigcup_{\alpha} cell(sc(i^2+3i+2))$$

$$= \bigcup_{\alpha} Ac(i^2+3i+2) \text{ by lemma 1.4.}$$

Similarly $symA(1) = \bigcup_{\alpha} cell'(sc(i^2+3i+2))$. Assume as induction hypothesis (H3) that equation 1.3 holds for ℓ. Consider $(\ell+1)$ st iteration of the second loop, $\ell+1 \leq i+2$. Then $stA(\ell+1) = stA(\ell) * symA(\ell+1)$. By H3, $stA(\ell) = \bigcup_{\alpha} Ac(i^2+3i+1+\ell)$ and from equation 1.2, and H2 $symA(\ell+1) = \bigcup_{\alpha} cell'(sc(i^2+3i-\ell))$ since $symA(\ell+1)$ value is that at end of L_1, assuming $\ell+1 \leq i+1$.‡ We shall consider the case $\ell+1 = i+2$ separately. By lemma 1.3, $cell'(sc(i^2+3i-\ell)) = cell(sc(i^2+3i+2+\ell))$. Then,

$$stA(\ell+1) = \bigcup_{\alpha} Ac(i^2+3i+1+\ell) * \bigcup_{\alpha} cell(sc(i^2+3i+2+\ell))$$

$$= \bigcup_{\alpha} Ac(i^2+3i+\ell+2))$$

Similarly $symA(\ell+1) = \bigcup_{\alpha} cell'(sc(i^2+3i+\ell+2))$. Consider the case now, $\ell+1 = i+2$.

Then $stA(\ell+1) = stA(\ell) * symA(i+2)$. From H3, $stA(\ell) = \bigcup_{\alpha} Ac((i^2+3i+1+\ell) = \bigcup_{\alpha} Ac(i^2+4i+2)$. From lemma 1.3, $sc(i^2+4i+3) = i+2$ and $symA(i+2) = cell(sc(i^2+4i+3))$ = the initial contents of cell $(i+2)$.

---

‡ For $1+1 \leq i$, we use equation 1.2. For $1+1=i+1$, we use H2(equation 1.1), since contents of cell $(i+1)$ are those written at the end of sweep i.

Then,

$$stA(\ell+1) = \bigcup_{\alpha} Ac(i^2+4i+2) * \bigcup_{\alpha} cell(sc(i^2+4i+3))$$

$$= \bigcup_{\alpha} Ac(i^2+4i+3) = \bigcup_{\alpha} Ac(i^2+3i+\ell+2)$$

and similarly $symA(\ell+1) = \bigcup_{\alpha} cell'(sc(i^2+3i+\ell+2))$. This completes the inductive step for H3 and hence equation 1.3 holds for all ℓ, $1 \leq \ell \leq i+2$. Therfore from equation 1.3, for $1 \leq j \leq i+2$, at the end of the second loop, L_2,

$$stA(j) = \bigcup_{\alpha} Ac(i^2+3i+1+j)$$

$$symA(j) = \bigcup_{\alpha} cell'(sc(i^2+3i+1+j))$$

and from equation 1.2 for j=0

$$stA(0) = \bigcup_{\alpha} Ac(i^2+3i+1)$$

$$symA(0) = \bigcup_{\alpha} cell'(sc(i^2+3i+1)$$

These equations together are H1 with i replaced by i+1. Thus the theorem.

The statement that M terminates after i sweeps is an abbreviation of the statement that there exists a sequence α of choice functions of length i^2+2i-1 such that the active states at step i^2+2i contain a member of H. We now have the following.

Corollary 1.1 For each input $\omega \in \sum^*$, procedure sim terminates after i iterations of the loop L if and only if M on input ω terminates after i sweeps.

Proof: From the theorem above, on exit from the call sweep(i) in L,

$$R = stA(i) = \bigcup_{\alpha} Ac(i^2+2i-1).$$

Since R is the union of active states of M at the end of ith sweep, for all possible sequences of choice functions of length i^2+2i-1, $R \cap H \neq 0$ if and only if M terminates after i sweeps.

The S-machine M has been defined as a non-deterministic acceptor model. The language accepted by M is defined as $L(M) = \{w \in \sum^* |$ for some i and choice sequence α, $Ac(i^2+2i-1) \in Q_A\}$.

From the assumption made earlier about M, it follows that procedure sim terminates on every input array configurations that correspond to an input ω. We define S-machine M to be T(n) time bounded if for input ω, $|\omega| = n$ there exists a halting computation of M on input w of length $\leq T(n)$. M is S(n) space bounded if for input ω, $|\omega| = n$ implies that for every computation of M on input ω, cell(j) $\in \{\langle \not b, \not b\rangle, \langle \not b, z\rangle\}$ for all $j \geq S(n)$. Thus, M is S(n) space bounded if for any input ω of length n, no computation of M on input ω, writes a symbol other that $\langle \not b, \not b\rangle, \langle \not b, z\rangle$ on any cell of index greater than S(n). The time and space bounds of M are directly related to those of the procedure sim. From corollary 1.1 it follows that if M is T(n) time bounded then so is procedure sim. Also from theorem 1.1, at the end of ith iterations of L, for each j, $0 \leq j \leq i + 1$, symA(j) is exactly equal to the set of symbols that appear in cell(j) at the end of a computation sequence of M of length $(i^2+i+j-1)$, at which step cell(j) is visited by M for the last time in sweep i. Since all possible computation sequences of this length are included, it follows that for an input $\omega \in L(M)$, $|\omega| = n$, symA(j) = cell(j) for $j \geq s(n)$. We shall therefore consider procedure sim to be S(n) space bounded in the same sense as the machine M above.

Consider now a language L accepted by a deterministic Turing machine M within space bound S(n). Since M does not visit any cell to the right of the S(n)+1th cell on the tape of M , M can be simulated by a S-machine with space bound S(n). Since each stateset

and symbolset of the S-machine will consist of a single element or be empty, the use of choice functions will result in the only possible computation.

If L is accepted by a nondeterministic Turing machine within space bound S(n), then L can be accepted by a deterministic Turing machine within space bound $\theta S^2(n)$ and hence by a S-machine within space bound $\theta S^2(n)$ [11].

Section 2: Equivalence of statesets and symbol sets:

In this section we characterize the computation of the S-machine in terms of the successive sweeps on array A performed by the procedure sym. Using this characterization, we introduce the notion of equivalence of statesets and symbolsets of the S-machine. We then extend this equivalence to define equivalence of crossing functions and show several properties.

As defined in section 1, the array A consists of a sequence of array cells indexed 0,1,2,..., etc. In this section, instead of indexing the cells, we shall index the cell boundaries, so that the left boundary of cell i is indexed i, $i \geq 0$. Each cell contains a symbolset of M.

The ith sweep will be assumed to begin at boundary i moving left and end at boundary (i+1) moving right. Such a sweep will be designated a (i,i+1) sweep. This definition of a sweep corresponds exactly to the computation performed by the function sweep, with the call sweep (i). The procedure sim therefore performs the sweep sequence (1,2), (2,3)...(i,i+1)... .

Consider now a call sweep (i) corresponding to a (i,i+1) sweep. The content of stA(i) on entry to the function sweep is called the entry stateset for this sweep. The content of stA(i) on exit from the function is called the exit stateset. We could therefore assume that an (i,i+1) sweep crosses the ith boundary to the left with the given entry stateset and the (i+1)th boundary moving to right with the exit stateset. The sequence of sweeps performed by the procedure sim will be called the primary sweep sequence. At the conclusion of (i,i+1) primary sweep, it is possible to perform additional (i,i+1)

sweeps. Such sweeps will be designated (i, i+1) secondary sweeps. We use the symbol $p_i(Q_{i+1})$ to denote the entry stateset (exit stateset) of the (i, i+1) primary sweep. We note that p_i is obtained from Q_i and the symbolset at cell i at the end of (i-1, i) primary sweep.

Suppose now (i, i+1) primary sweep has been performed. To perform a (i, i+1) secondary sweep with entry stateset q we execute the following:

```
stA(i) := q;
   sweep(i);
```

In general, let $\alpha = q_{i_1}, q_{i_2}, \ldots q_{i_m}$, $m \geq 0$ be a (possibly empty) sequence of entry statesets. The (i, i+1) secondary sweep sequence defined by α is executed by the following program segment $P(\alpha)$:

```
P(α):  for k := 1 to m do
          begin stA(i) := q_{i_k};
                sweep(i)
          end;
```

The array A on entry to $P(\alpha)$ is that at the end of the (i, i+1) primary sweep. We now introduce a notation for the contents of array A at different stages of the sweep sequence

σ_i, $i \geq 0$ represents the initial contents of cell i. Each σ_i is a symbol set.

$\sigma_{i,\Delta}$ represents the contents of symA(i) at the end of the (i, i+1) primary sweep.

$\sigma_{i,\alpha}$ represents the contents of symA(i) at the end of (i, i+1) secondary sweep sequence α. $\sigma_{i,\alpha}$ is also defined by $P(\alpha)$.

σ_i' represents the contents of symA(i) at the end of the (i-1, i) primary sweep.

$\omega_{i,\alpha}$ represents the string of symbolsets in A(0)...A(i-1) at the end of the (i,i+1) secondary sweep sequence α.

We now define a crossing function f_w associated with a string w of symbolsets. f_w maps statesets to statesets.

Let $q_1, q_2, \ldots, q_n$ be all the possible state sets, and let $w = w_0 w_1 w_2 \ldots w_i$ be a string of symbolsets. Then, $f_w(q_j)$ is the exit stateset of an (i,i+1) sweep on w with entry stateset q_j, $1 \leq j \leq n$. The (finite) function f_w is computed by the following program segment P(w).

P(w):

```
for j := 1 to n do
begin for k=1 to i+1 do symA(k-1) := w_{k-1};
      stA(i) := q_j;
   f_w(q_j) := sweep(i);
end.
```

For each value q_j of entry stateset, P(w) initializes array cells 0 through i to w, calls the sweep function and assigns the exit stateset to f_w. Since there are finitely many statesets, the set of all crossing functions is finite.

We define a sequence of partitions of the set of statesets {Q} and the set of symbolsets. These partitions define an equivalence relation on the statesets and the symbolsets. st_i denotes the ith partition of the set of statesets. {Q}, and sym_i denotes the ith partition of the set of symbolsets, $0 < i$.

Let $st_i = \{S_{i,0}, S_{i,1}, \ldots, S_{i,m_i}\}$ and

$$sym_i = \{\sum_{i,0}, \sum_{i,1} \ldots \sum_{i,n_i}\}$$

where $S_{i,0} \ldots S_{i,m_i}$ are the m_i blocks of the st_i partition and

$\sum_{i,0} \ldots \sum_{i,n_i}$ are the n_i blocks of the sym_i partition.

For i=0, $n_i = 1$ and $m_i = 3$; let

$$S_{0,0} = Q_A^L,\ S_{0,1} = \{Q\}^L - Q_A^L,\ S_{0,2} = Q_A^R \text{ and } S_{0,3} = \{Q\}^R - Q_A^R.$$

$$\sum_{0,0} = \sum_A,\ \sum_{0,1} = \{\Gamma\} - \sum_A.$$

We have used $\{Q\}^L$ ($\{Q\}^R$) to denote the leftmoving (rightmoving) statesets of {Q}. $\{\Gamma\}$ denotes the set of all symbolsets.

Assume st_i and sym_i are defined.

Let $q_1, q_2 \in S_{i,j}$. Then $q_1, q_2 \in S_{i+1,j'}$ if and only if for all $a_1, a_2 \in \sum_{i,k}$, $0 \le k \le n_i$, $\exists\ k_1, j_1$, $0 \le k_1 \le n_i$ $0 \le j_1 \le m_i$ such that

$$q_1 * a_1,\ q_2 * a_2 \in S_{i,j_1};\ a_1 * q_1,\ a_2 * q_2 \in \sum_{i,k_1}.$$

$S_{i+1,j'}$ is the largest block with the property above. Let $a_1, a_2 \in \sum_{i,j}$. Then $a_1, a_2 \in \sum_{i+1,j'}$ iff for all $q_1, q_2 \in S_{i,k}$, $0 \le k \le m_i$, $\exists\ k_1, j_1$, $0 \le k_1 \le n_i$ $0 \le j_1 \le m_i$ such that $q_1 * a_1,\ q_2 * a_2 \in S_{i,j_1}$; $a_1 * q_1$, $a_2 * q_2 \in \sum_{i,k_1}$. $\sum_{i+1,j'}$ is the largest block with the property above.

Let q_1, q_2 belong to same block of st_i. Then q_1, q_2 belong to same block of st_{i+1} if and only if for every pair of symbol-sets a_1, a_2 that belong to same block of sym_i, $q_1 * a_1$, $q_2 * a_2$ belong to same block of st_i and $a_1 * q_1$, $a_2 * q_2$ belong to same block of sym_i. Similarly, let a_1, a_2 belong to same block of sym_i. Then a_1, a_2 belong to same block of sym_{i+1} if and only if for every pair of statesets q_1, q_2 that belong to the same block of st_i, $q_1 * a_1$, $q_2 * a_2$ belong to the same block of st_i and $a_1 * q_1$, $a_2 * q_2$

belongs to same block of sym_i.

It follows that $q_1, q_2 \in S_{i+1,j}$ implies that for some j', $0 \leq j' \leq m_i$, $q_1, q_2 \in S_{i,j'}$ and similarly for sym_{i+1}. Therefore $st_{i+1}(sym_{i+1})$ is a refinement of the partition $st_i(sym_i)$. Suppose further for some i_0, $st_{i_0} = st_{i_0+1}$ and $sym_{i_0} = sym_{i_0+1}$. Then $st_{i_0+2} = st_{i_0+1}$ and $sym_{i_0+2} = sym_{i_0+1}$. For let $q_1, q_2 \in S_{i_0+1,j} = S_{i_0,j}$. Then for all ℓ, $1 \leq \ell \leq n_{i_0}$, $a_1, a_2 \in \sum_{i_0,\ell} = \sum_{i_0+1,\ell}$ $\Rightarrow \exists\ \ell', j'$ such that $q_1 * a_1$, $q_2 * a_2 \in S_{i_0,j'} = S_{i_0+1,j'}$, and $a_1 * q_1$, $a_2 * q_2 \in \sum_{i_0,\ell'} = \sum_{i_0+1,\ell'}$, and thus $q_1, q_2 \in S_{i_0+2,j}$. Similarly for sym_{i_0+2}. Thus $st_{i_0+2} = st_{i_0+1}$ and $sym_{i_0+2} = sym_{i_0+1}$.

Definition 2.1: $q_1 \doteq q_2$ if and only if q_1 and q_2 belong to same block of st_i for each i and

$a_i \doteq a_2$ if and only if a_1 and a_2 belong to same block of sym_i for each i.

Equivalently,

$q_1 \doteq q_2$ if $\forall_i, \exists\ k_i$, $0 \leq k_i \leq m_i$ such that $q_1, q_2 \in S_{i,k_i}$; $a_1 \doteq a_2$ is defined similarly.

Lemma 2.1: $q_1 \doteq q_2$ and $a_1 \doteq a_2 \Rightarrow q_1 * a_1 \doteq q_2 * a_2$ and $a_1 * q_1 \doteq a_2 * q_2$.

Proof: $q_1 \doteq q_2$ implies that for each i, $q_1, q_2 \in S_{i,k_i}$, $0 \leq k_i \leq m_i$ and $a_1 \doteq a_2$ implies that for each i $a_1, a_2 \in \sum_{i,\ell_i}$ for $0 \leq \ell_i \leq n_i$. Further $q_1, q_2 \in S_{i+1,k_{i+1}}$ for some k_{i+1}, $0 \leq k_{i+1} \leq m_{i+1}$. Thus,

$\exists\ p_i, t_i$ such that $q_1 * a_1$, $q_2 * a_2 \in S_{i,p_i}$,

$$a_1*q_1, a_2*q_2 \in \sum_{i,t_i}$$

Since this holds for all i, $q_1*a_1 \doteq q_2*a_2$ and $a_1*q_1 \doteq a_2*q_2$.

We note from the discussion above that if $q_1 \doteq q_2$ then the statesets q_1 and q_2 must both be leftmoving or both be right moving; both be accepting or both be nonaccepting (reject or non halting). If $a_1 \doteq a_2$ then the symbolsets a_1, a_2 must both be accepting or both nonaccepting. From lemma 2.1, if $q_1 \doteq q_2$ and $a_1 \ldots a_k$ are symbol sets, then

$$(\ldots(q_1*a_1)*a_2)\ldots*a_k) \doteq (\ldots(q_2*a_1)*q_2)\ldots*a_k).$$

We could thus conclude that if $q_1 \doteq q_2$ then a sequence of symbol sets takes q_1 to acceptance if and only if it takes q_2 to acceptance. Parallel statement holds for $a_1 \doteq a_2$. In view of lemma 2.1, we could replace each a_i in the sequence $a_1 \ldots a_k$ by any member of the equivalence class containing a_i.

Lemma 2.2: Let q_1, q_2, p_1, p_2 be statesets. If $q_1 \doteq q_2$ and $p_1 \doteq p_2$ then $p_1 + q_1 \doteq p_2 + q_2$.

Proof: We shall assume that the statesets q_1, q_2, p_1, p_2 are all left moving (or all right moving). Consider the 0th partition $\{S_{0,0}, S_{0,1}\}$. Let $q_1, q_2 \in S_{0,i}$ and $p_1, p_2 \in S_{0,j}$, $0 \leq i \leq 1$, $0 \leq j \leq 1$. If $i = j = 0$ or $i = j = 1$ then p_1+q_1, $p_2+q_2 \in S_{0,i}$ by definition. If $i = 0$ and $j = 1$, then p_1+q_1, $p_2+q_2 \in S_{0,0}$ and similarly if $i = 1$ and $j = 0$. Let $q_1 = {}_k q_2$ denote that q_1, q_2 belong to same block of st_k. $a_1 = {}_k a_2$ if a_1, a_2 belong to same block of sym_k. We then have

$$q_1 = {}_0\, q_2 \text{ and } p_1 = {}_0\, p_2 \text{ implies } p_1 + q_1 = {}_0\, p_2 + q_2.$$

Assume as induction hypothesis that

$$q_1 =_{k-1} q_2 \text{ and } p_1 =_{k-1} p_2 \text{ implies } p_1 + q_1 =_{k-1} p_2 + q_2.$$

Let $q_1 =_k q_2$ and $p_1 =_k p_2$. Let $a_1 \doteq a_2$ and hence $a_1 =_{k-1} a_2$. Since $q_1 =_{k-1} q_2$ and $p_1 =_{k-1} p_2$, from the proof of lemma 2.1, $q_1{*}a_1 =_{k-1} q_2{*}a_2$ and $p_1{*}a_1 =_{k-1} p_2{*}a_2$. By induction hypothesis

$$q_1{*}a_1 + p_1{*}a_1 =_{k-1} q_2{*}a_2 + p_2{*}a_2$$

By lemma 1.1,

$$(q_1+p_1){*}a_1 =_{k-1} (q_2+p_2){*}a_2$$

Since by induction hypothesis $q_1+p_1 =_{k-1} q_2+p_2$ and a_1, a_2 are arbitrary, we have

$$q_1+p_1 =_k q_2+p_2$$

Then for all $k \geq 0$,

$$q_1 =_k q_2 \text{ and } p_1 =_k p_2 \text{ implies } q_1 + p_1 =_k q_2 + p_2$$

i.e.

$$q_1 \doteq q_2 \text{ and } p_1 \doteq p_2 \text{ implies } q_1 + p_1 \doteq q_2 + p_2.$$

Definition 2.2: f_1, f_2 be two crossing functons. f_1, f_2 are equivalent ($f_1 \doteq f_2$) if and only if for all q_1, q_2,

$$q_1 \doteq q_2 \Rightarrow f_1(q_1) \doteq f_2(q_2)$$

The relation of equivalence of crossing functions is clearly symmetric and transitive. Following lemma shows that it is also reflexive.

Lemma 2.3: For any crossing function f,

$$q_1 \doteq q_2 \Rightarrow f(q_1) \doteq f(q_2).$$

Proof: We use induction on the lngth n of a symbolset string w with crossing function f_w. For n=1, w = $\{\langle \phi, \phi \rangle\}$. If q_1, q_2 are any two left moving statesets crossing 1-boudnary to the left, then the exit statesets at 1-boundary are $q_1' = q_1 {*}\{\langle \phi, \phi \rangle\}$ and $q_2' = q_2 {*}\{\langle \phi, \phi \rangle\}$. By

lemma 2.1, $q_1 \doteq q_2 \Rightarrow q_1' \doteq q_2' \Rightarrow f_w(q_1) \doteq f_w(q_2)$.

Assume as induction hypothesis that for a symbolset string w of length n, $q_1 \doteq q_2 \Rightarrow f_w(q_1) \doteq f_w(q_2)$.

Let wa be a symbolset string of length n+1. Consider two copies of wa, one with entry stateset q_1 at (n+1) boundary and the other with entry stateset q_2 at (n+1) boundary. Then the entry statesets at n boundary are q_1*a and q_2*a respectively. By lemma 2.1 $q_1*a \doteq q_2*a$. By induction hypothesis, the exit statesets at n boundary is $f_w(q_1*a) \doteq f_w(q_2*a)$. Since the symbolset at cell n are $a*q_1$, $a*q_2$ respectively and $a*q_1 \doteq a*q_2$, the exit statesets at (n+1) boundary are $f_w(q_1*a) * (a*q_1) \doteq f_w(q_1*a) * (a*q_2)$ by lemma 2.1.

Thus $f_{wa}(q_1) \doteq f_{wa}(q_2)$ which completes the inductive step.

Definition 2.3: f be a crossing function and a be a symbolset. We define the product crossing function f*a by

$$f*a(q) = f(q*a)*(a*q)$$

From the proof of lemma 2.3 we have

Corollary 2.3: Let w be any symbolset string with the left end marker and a be a symbolset. Then $f_{wa} \doteq f_w*a$

Lemma 2.4: $f_1 \doteq f_2$ and $a_1 \doteq a_2$ implies $f_1 * a_1 \doteq f_2 * a_2$

Proof: Let $q_1 \doteq q_2$. Since $a_1 \doteq a_2$, by lemma 2.1,

$$q_1*a_1 \doteq q_2*a_2 \text{ and } a_1*q_1 \doteq a_2*q_2$$

Since $f_1 \doteq f_2$, $f_1(q_1*a_1) \doteq f_2(q_2*a_2)$

and by lemma 2.1,

$$f_1(q_1*a_1) * (a_1*q_1) \doteq f_2(q_2*a_2) * (a_2*q_2)$$

i.e.

$$f_1*a_1 \doteq f_2*a_2$$

Definition 2.4: f,g be two crossing functions. Their sum f+g is defined by

$$f+g(q) = f(q) + g(q).$$

Lemma 2.5: $f_1 \doteq f_2$ and $g_1 \doteq g_2$ implies $f_1 + g_1 \doteq f_2 + g_2$ for any crossing functions f_1, f_2, g_1, g_2.

Proof: Let q_1, q_2 be statesets such that $q_1 \doteq q_2$. Then

$$f_1(q_1) \doteq f_2(q_2) \text{ and } g_1(q_1) \doteq g_2(q_2).$$

By lemma 2.2,

$$f_1(q_1) + g_1(q_1) \doteq f_2(q_2) + g_2(q_2)$$

or

$$f_1+g_1(q_1) \doteq f_2 + g_2(q_2) \text{ by definition.}$$

Thus, $f_1 + g_1 \doteq f_2 + g_2$.

Corollary 2.5.1: For any crossing function f, $f+f \doteq f$

Corollary 2.5.2: For any crossing functions f,h $f \doteq h \Rightarrow f \doteq f+h$

Corollary 2.5.3: $f \doteq g$ and $f \doteq h \Rightarrow f \doteq g+h$.

Let [q], [a], [f] denote the equivalence classes containing q, a and f respectively.

We extend the product to these classes.

Definition 2.5: $[q] * [a] = \{q*a \mid q \in [q], a \in [a]\}$

$$[a] * [q] = \{a*q \mid q \in [q], a \in [a]\}$$

Lemma 2.6: $[q] * [a] = [q*a]$; $[a] * [q] = [a*q]$

Proof: Let $q_1 \doteq q_2 \doteq q$ and $a_1 \doteq a_2 \doteq a$. Then by lemma 2.1, $q_1*a_1 \doteq q_2*a_2 \doteq q*a$ and $a_1*q_1 \doteq a_2*q_2 \doteq a*q$

Definition 2.6: $[f]([q]) = [q'] \Leftrightarrow f(q) = q'$ where $f \in [f]$, $q \in [q]$, $q' \in [q']$.

The definition is sound since $f_1 \doteq f_2$, $q_1 \doteq q_2$ implies

$f_1(q_1) \doteq f_2(q_2)$.

Lemma 2.7: $[f]([q]) = [f(q)]$

Proof: Let $x \varepsilon [f]([q]) \Rightarrow x = f(q)$ for $f \varepsilon [f]$ $q \varepsilon [q] \Rightarrow x \varepsilon [f(q)]$.

Conversely, if $x \varepsilon [f(q)]$ then $x \doteq f(q)$

or $x \doteq x_0$ and $x_0 = f(q)$

or $x \doteq x_0$ and $[x_0] = [f]([q]) \Rightarrow x \varepsilon [f]([q])$ by definition.

Definition 2.7:

$[f] * [a]$ is defined by

$[f] * [a]([q]) = [f]([q]*[a]) * ([a] * [q])$

Lemma 2.8: $[f] * [a] = [f*a]$

Proof: $[f] * [a]([q]) = [f]([q] * [a]) * ([a] * [q])$ by definition

$= [f]([q*a]) * ([a*q])$ by lemma 2.6

$= [f(q*a)] * [a*q]$ by lemma 2.7

$= [f(q*a) * (a*q)]$ by lemma 2.6

$= [f*a(q)]$ by definition 2.3

$= [f*a]([q])$ by lemma 2.7

Hence the lemma.

Corollary 2.8:

$[f]([q]) * [a] = [f(q)*a]$

Proof: $[f]([q]) * [a] = [f(q)] * [a]$ by lemma 2.7

$= [f(q) * a]$ by lemma 2.6

Let $w_1 = a_1a_2\ldots a_k$ and $w_2 = b_1b_2\ldots b_k$ be two string of symbolsets with end markers. We define $[w_1] = [w_2]$ if for all i, $1 \leq i \leq k$, $[a_i] = [b_i]$.

Lemma 2.9: $[w_1] = [w_2] \Rightarrow [f_{w_1}] = [f_{w_2}]$.

Proof: We use induction on the length of w_1. For n=1, $w_1 = a$, $w_2 = b$, and $[a] = [b] = [\{\langle \phi, \phi \rangle\}]$, and the lemma holds. Assume as induction hypothesis that the lemma holds for w_1, w_2 of length n. Let $w_1 = w_1'a$ and $w_2 = w_2'b$. Then $[w_1'] = [w_2']$ and $[a] = [b]$. By corollary 2.3, and lemma 2.8, $[f_{w_1}] = [f_{w_1'}] * [a] = [f_{w_2'}] * [b]$ by hypothesis

$$= [f_{w_2}].$$

Suppose now w_1, w_1 are as above. Let $w_1^{q_1}$, $(w_2^{q_2})$ be the strings of symbol sets obtained if we perform a (k,k+1) sweep on $w_1(w_2)$ with entry stateset $q_1(q_2)$.

Lemma 2.10: $[w_1] = [w_2]$ and $q_1 \doteq q_2 \Rightarrow [w_1^{q_1}] = [w_2^{q_2}]$.

Proof: By induction on length of w and lemma 2.1.

From the definition of halting statesets and symbolsets given earlier, it follows that if w contains a halt symbolset, then $f_w(q) \varepsilon Q_A$ or $f_w(q) \varepsilon Q_\Gamma$ for all stateset q. Thus, for all q_1, q_2, w^{q_1}, w^{q_2} are both halting symbol sets.

Section 3: The left context graph:

In this section we define a (class) of directed labeled graphs called the (left) context graphs. A context graph G_i is associated with each boundary i at the end of the (i, i+1) primary sweep. The graphs G_i, $i \geq 1$ are constructed inductively. We define the graph G_1 ab-initio and then assuming G_{i-1} has been constructed define the procedure for construction of G_i. The node lables of a context graph G_i contain two components, an equivalence class of crossing functions F and an equivalence class of symbolsets D. The edges of G_i are labeled with equivalence classes of statesets. The graph G_i is constructed so that the equivalence class of the crossing function of the symbolset string to the left of the i-boundary and the equivalence class of the symbolset at the ith cell, at the end of the (i, i+1) primary sweep, are equal to the F and D components of the label of the root of G_i.

Let $\alpha = q_{i_1}, q_{i_2}, \ldots q_{i_m}$ be an entry state-set sequence. Suppose now, we perform m (i, i+1) secondary sweeps following the (i, i+1) primary sweep, with the entry statesets defined by α. Then the equivalence class of the crossing function of the symbolset string to the left of the i-boundary (cells 0 through i-1), and the equivalence class of the symbol set at cell i at the end of the secondary sweep sequence will be equal to the F and D components of the α-successor fo the root of G_i.

The procedures for the construction of the context graphs are described using a pascal-like notation. This facilitates the estimation of an upper bound on the number of steps required for the

computation. We now describe a procedure for computing the graph G_1. The label information in the vertices of a graph are assumed to be contained in records of type node and referenced to by pointers.

```
type node = record
              F : set of crossing function;
              D : set of symbol set
            end
ptr = ↑node;
```

We use a datatype queue together with insert, delete operators and the empty-que predicate. The queue elements are pointers. The type edge denotes a triple of the form (a,b,[q]) where a,b are pointers, and [q] a set of statesets. The identifier table is a set of pointer and the identifier edges a set of edge.

The graph G_1 is constructed using a breadth first technique, starting with a designated root. The procedure is dynamic in that new nodes are created as needed. To construct G_1, we first define the root node and then call the procedure graph, which calls the procedure S. The procedure call S(s,t,[q]) computes t, the [q]-successor of s. If t is a pointer to an existing node x of G_1, at this stage of computation as determined by the searchtable function, then the edge (s,x,[q]) is added to edges. Otherwise, t is added to table and (s,t,[q]) added to edges.

The initial contents of array A prior to the beginning of the first sweep was defined in section 1. For a given input string $a_1a_2\ldots a_n \in \sum^*$, we have

symA(0) = {<¢,¢>}, stA(0) = ∅

symA(1) = cell(1), stA(1) = Ac(0) = p_1,

symA(j) = {<a_j,∅>} for $2 \leq j \leq n$ and

= {<∅,∅>} for $n < j \leq T(n)$

stA(j) = ∅ for $1 < j \leq T(n)$,

where cell(1) = {<a_1,z>} * {q_0} and Ac(0) = {q_0} * {<a_1,z>}. Since initially symA(1) = {<a_1,z>}, using notation introduced in section 2, σ_1' = cell(1) and $\sigma_{1,\Lambda}$ is the set of symbols written in cell 1 at the end of 2nd step of computation of the call sweep(1). Then

$\sigma_{1,\Lambda} = \bigcup_{\alpha}$ cell'(sc(2))

From the proof of theorem 1.1,

$\bigcup_{\alpha}$ cell'(sc(2)) = cell(1) * $\bigcup_{\alpha}$ Ac(1)

where $\bigcup_{\alpha}$ Ac(1) = stA(0) at the end of the execution of loop L1 in the call sweep(1). Therefore, $\sigma_{1,\Lambda} = \sigma_1'$ * stA(0). Let $f_{<¢,¢>}$ denote the crossing function of the symbolset string {<¢,¢>}, and (F_0,D_0) the labels of the root of G_1. Then $F_0 = [f_{<¢,¢>}]$, $D_0 = [\sigma_{1,\Lambda}]$. We assign a pointer r_0 to the root of G_1 as follows:

new(r_0); r_0↑.F := F_0; r_0↑.D := D_0;

To compute G_1, we initialize que to r, edges to null, table to {r} and then call the procedure graph (r). We first define procedure S. The notation [q] ([a],[f]) will denote equivalence classes of statesets (symbolsets, crossing functions). We abbreviate these to eq stateset (eq symbolset, eq crossing function).

We first define a procedure S':

```
procedure S'(s:ptr;var t:ptr;[q]:eq stateset;var Q2':eq stateset);
   var d: eq symbolset; q': eq stateset;
```

```
begin q' := [q]*[[<ϕ,ϕ>]];
      d := s↑.D*q';
      Q2' := q'*s↑.D;
      new(t);
      t↑.F := F0 ; t↑.D := d;
end
```

```
procedure S(s:ptr;var t:ptr;[q]:eq state set);
   var Q: eq state set;
      begin S'(s,t,[q],Q) end;
```

In the procedure S', s is a node of G_1 already constructed. Consider a(1,2) secondary sweep with entry stateset q. Suppose s↑.F(s↑.D) is the eq crossing function of symbolset in cell 0[1] (eq symbolset at cell 1). Then q' is the eq stateset when the sweep exits at boundary 1, d the eq symbolset at cell 1 and Q_2' the eq stateset when the sweep exits at boundary 2. Procedure S ignores the last parameter of procedure S'.

```
function search (aset:set of ptr;var t:ptr):boolean;
   var x:ptr;
      begin search := false;
         for x in aset do if t↑. = x↑. then begin search := true;
                                                  t := x
                                             end
      end.
```

function search for given values of aset and t, modifies t to the corresponding entry in table if t is found in table.

---

[1] Symbol-set in cell 0 is always $\{\langle \varphi, \varphi \rangle\}$.

```
var que:queue; edges:set of edge; table:set of ptr;
procedure graph (r:ptr);
   var t,r':ptr; [q]:eq stateset;
   begin
   L:while not emptyque(que) do
     begin r' := delete(que);
        for [q] in Q̄ do begin S(r',t,[q]);
        if search (table,t) then
             edges := edges + {(r',t,[q])}
             else begin insert (que,t);
                    table := table + {t};
                    edges := edges + {(r',t,[q])}
                  end;
             end;
     end;
  end graph;
```

To obtain G_1, we execute the following:

```
begin que := nil; edges := nil; table := {r0};
      que := insert (que, r0);
      graph (r0);
end
```

In the procedure graph, $\bar{Q}$ denotes the set of all eq statesets.

If $[q_1] \ldots [q_m]$ are all the eq statesets, then

$\bar{Q} = \{[q_1] \ldots [q_n]\}$

The vertices of the graph G_1 is the set of nodes contained in table.

The labeled edges are defined by the variable edges on exit. In order

to claim that G_1 is well defined we have to show that the procedure graph terminates. Following lemmas are addressed to that point.

Let $table_k$, que_k, r'_k denote the values of the variables table, que and r' at the beginning of the kth iteration of the while-do loop L in the procedure graph. Since no element is ever deleted from table it follows that $table_k \subseteq table_{k+1}$ for all $k \geq 1$. For the purpose of the following lemmas, we shall consider a queue to also define a set, so that membership in queue has the usual meaning.

Lemma 3.1: If for some k, $x \in table_k$ and $x \notin que_k$ then for all $j>k$, $x \notin que_j$.

Proof: Suppose that for some k, $x \in table_k$ and $x \notin que_k$. Suppose further that for some $j > k$, $r'_j = y$ in the procedure graph and for some q in $\bar{Q}$, the procedure S(y,t,q) returns node t such that $t\uparrow. = x\uparrow.$. Then, since $x \in table_k$, function search (table,t) returns value true with t = x; this implies that x is not inserted in the que. Since $table_{k+1} \supseteq table_k$, for all $j > k$, search ($table_j$,t) will return true. Hence t(=x) is never inserted in que_j for $j > k$. Thus $x \notin que_j$ for $j > k$.

Lemma 3.2: If for some k_0, $x \in table_{k_0}$ then there exists $k \leq k_0$ such that $x \in que_k$.

Proof: Let $k \geq 1$ be the smallest integer such that $x \in table_{k+1}$. Then $x \in table_{k+1} - table_k$ and x must have been added to table in the kth iteration of the while-do loop L of the procedure graph. This implies that for some q in $\bar{Q}$, the procedure S(r',t,q) returns x as the value of t. Since $x \notin table_{k-1}$, search (table,t) must return false. Thus x is inserted in the que and in table for the first time. Since

$table_{k+1} \supseteq table_k$, it follows that if $x \in table_{k_0}$ then $x \in que_k$ for some $k \leq k_0$.

Let n be the number of statesets, 2^p the number of symbolsets and 2^m the number of crossing functions. n,m and p are defined by the machine M. We then have,

Lemma 3.3: The procedure graph terminates in at most $cn2^{2(m+p)}$ steps for some constant c.

Proof: Since there are at most 2^m equivalence classes of crossing functions and 2^p eq symbolsets, the number of possible $\langle F,D \rangle$ pairs is at most $2^m.2^p$. Let x be a pointer with $\langle F,D \rangle$ as the corresponding node components. By lemma 3.2, if $x \in table_k$ for some k, then $x \in que_{k'}$ for some $k' \leq k$. Since the procedure graph deletes an element from que in each iteration and $table_{k+1} \supseteq table_k$ it follows that for some $k_2 > k'$, $x \in table_{k_2}$ and $x \notin que_{k_2}$. By lemma 3.1, $x \notin que_k$ for any $k > k_2$. Thus que must be empty after no more than 2^{m+p} deletions. Further at most n elements are added to table in each iteration of the procedure graph. Thus, the function search uses no more than ckn steps in the kth iteration for some constant c. Procedure S on any call requires only constant number of steps. Thus the total number of steps of algorithm graph is

$$\leq n \sum_{i=1}^{2^{m+p}} c_1.i \leq c_2.2^{m+p}(1+2^{m+p}).n$$

$$\leq cn2^{2(m+p)} \text{ as was to be shown.}$$

The graph G_1 obtained from the procedure graph above is a finite deterministic graph. i.e. for each eq stateset [q] and each vertex r of G , the [q] successor of r is defined and is unique. Let α =

$q_{i_1}, \ldots q_{i_j}$ be a sequence of (entry) statesets. We let $[\alpha] = [q_{i_1}] \ldots [q_{i_j}]$ denote the corresponding sequence of eq statesets. Then, $[\alpha q] = [\alpha][q]$.

The $[\alpha]$ successor of a vertex r of G is defined as follows.

$[\Delta]\ succ(r) = r$

$[q_{i_1}] \ldots [q_{i_j}]\ succ(r) = [q_{i_j}] succ([q_{i_1}] \ldots [q_{i_{j-1}}] succ(r))$

For every α, $[\alpha]$ successor(r) is a unique vertex of G_1. We now show the following property of graph G_1. Suppose following the (1,2) primary sweep in entry stateset p_1, we perform j (1,2) secondary sweeps with entry statesets $q_{i_1} \ldots q_{i_j}$ respectively. Then the equivalence class of the crossing function of the string of symbolsets to the left of 1-boundary will be equal to the F component of the $[\alpha]$ successor of the root in G_1. The equivalence class of the symbolset in cell 1 will be equal to the D component of the $[\alpha]$ successor of the root of G_1. This is the content of the following

Lemma 3.4: Let $\alpha = q_{i_1}, q_{i_2} \ldots q_{i_j}$, $j \geq 0$ be a (possibly empty) sequence of the statesets. If v is the $[\alpha]$-successor of the root in G_1 then

$$[f_{w_{1,\alpha}}] = v\uparrow.F \text{ and } [\sigma_{1,\alpha}] = v\uparrow.D$$

Proof: by induction on j. r_0 be the root of G_1.

For j=0, $\alpha=\Delta$ and $[\alpha]succ(r_0) = r_0$

By construction $r_0\uparrow.F = [f_{\langle \phi, \phi \rangle}] = [f_{w_{1,\Lambda}}]$ asnd $r_0\uparrow.D = [\sigma_{1,\Delta}]$

Hence the basis. Assume as induction hypothesis that the statement of the lemma holds for $j < k$. If $\alpha = q_{i_1} \ldots q_{i_j}$ then a total of (j+1) (1,2) sweeps have been performed with entry statesets $p_1, q_{i_1} \ldots q_{i_j}$

respectively. Let j=k and $\alpha = \alpha_1 q_{i_k}$ where $\alpha_1 = q_{i_1} \ldots q_{i_{k-1}}$. By induction hypothesis

$$[f_{w_{1,\alpha_1}}] = v_1\uparrow.F \text{ and } [\sigma_{1,\alpha_1}] = v_1\uparrow.D$$

where $v_1 = [\alpha_1]succ(r_0)$. If $v = [\alpha]succ(r_0)$ then by definition

$$v = [q_{i_k}]succ(v_1)$$

From procedure graph, we show that the procedure call $S(v_1, t, [q_{i_k}])$ on exit would yield t = v. From procedure body of S, $q' = [q_{i_k}]*[[\{\langle \phi, \phi \rangle\}]]$; since q_{i_k} is the entry stateset at 1-boundary for the kth (1,2) secondary sweep, the exit stateset at 1 boundary is $q_1 = q_{i_k} *\{\langle \phi, \phi \rangle\}$ ε $[q_{i_k} *\{\langle \phi, \phi \rangle\}] = [q_{i_k}]*[[\{\langle \phi, \phi \rangle\}]]$ by lemma 2.6.

Also the symbolset at cell1, when the sweep crosses 2-boundary is

$$\sigma_{1,\alpha} = \sigma_{1,\alpha_1} *q_1$$

By induction hypothesis, $[\sigma_{1,\alpha_1}] = v_1\uparrow.D$

Then $[\sigma_{1,\alpha}] = [\sigma_{1,\alpha_1} *q_1]$

$= [\sigma_{1,\alpha_1}]*[q_1]$ by lemma 2.6

$= [\sigma_{1,\alpha_1}]*q'$ since q_1 ε q'

$= v_1\uparrow.D*q'$

$= v\uparrow.D$ from procedure S.

Since cell(0) = $\{\langle \phi, \phi \rangle\}$,

$$[f_{w_{1,\alpha}}] = [f_{\langle \phi, \phi \rangle}] = v\uparrow.F$$

Which completes the inductive step.

The graph $\bar{G}_i$:

We have defined the graph G_1 associated with the 1-boundary at the end of the (1,2) primary sweep. We now define a graph $\bar{G}_i$

associated with the i-boundary, after the completion of the (i, i+1) primary sweep, assuming inductively that the graph G_{i-1} has been computed. We then obtain G_i from $\bar{G}_i$, thus completing the inductive step. We shall assume as part of inductive hypothesis that G_{i-1} is deterministic and show that so is G_i.

The information contained in a node of $\bar{G}_i$ has four components, F, D, R and E. F is an eq crossing function, D and E are eq symbolsets and R a pointer to a node of G_{i-1}. Let $\alpha = q_{i_1} \ldots q_{i_j}$ be a sequence of statesets, $j \geq 0$. Suppose after the (i,i+1) primary sweep, we perform j (i,i+1) secondary sweeps with entry statesets $q_{i_1} \ldots q_{i_j}$. The edges of $\bar{G}_i$ are labeled with eq statesets. Consider the labels (F,D,R,E) for the (unique) [α]-successor of the root of $\bar{G}_i$. As discussed earlier, the crossing function of the symbolset string in cells 0 through (i-1) (left of i boundary) is contained in the eq crossing function F and the symbolset at cell i is a member of eq symbolset D. The equivalence class of the crossing function of the symbolset string in cells 0 through (i-2) (left of (i-1) boundary) will be equal to the F component of a node of G_{i-1}, a pointer to which will be equal to R. E will be equal to the equivalence class of the symbolset at cell (i-1) at the end of the j (i,i+1) secondary sweeps.

The vertices of $\bar{G}_i$ are defined using records of the type newnode. We define the type newnode as follows:

```
type newnode = record
                    F : set of crossing function;
                  D,E : set of symbolset;
                    R : ptr
                end;
nptr = ↑ newnode;
```

The vertices of G_{i-1} consist of records of type node defined earlier, with ptr a pointer to a record of type node. Following the definitions used in the construction of G_1, we define the identifiers nedge, ntable and nque. nedge is a set of triples of the form (a,b,q) where a, b are nptrs and q an eq stateset. ntable is a set of nptr. nque is a que with insert, delete operators and que-empty predicate. The elements of nque are of type nptr.

The function edgesearch locates the [q]-successor of a vertex r in G_{i-1}. We have assumed, as in the case of G_1, that the vertices of G_{i-1} are contained in 'table' which is a set of ptr and the edges in 'edges' which is a set of triples of the form (a,b,q) a,b ptr, q an eq stateset.

```
function edgesearch (r:ptr; [q]:eq stateset):ptr;
    var (a,b,q'):edge
  begin for (a,b,q') in edges do
          if a=r and q'=[q] then edgesearch := b
  end;

procedure S(s:nptr;var t:nptr;[q]:eq stateset;var Q':eq stateset);
  var A,C:eq stateset; B:eq symbolset;
      Y,X:eq crossing function;

begin A := [q]*s↑.E; B := s↑.E*[q]; new(t);
    L1:t↑.R := edgesearch (s↑.R,A);
    L2:   X := (t↑.R)↑.F;
    L3:   Y := (s↑.R)↑.F;
    L4:   C := Y(A);
```

```
        t↑.D := s↑.D*(s↑.F([q]));
        t↑.E := B*C;
        t↑.F := X*t↑.E;
          Q' := (s↑.F([q]))*s↑.D;
    end(S).
```

In the procedure S, the computations using eq symbolset, eq stateset and eq crossing functions are defined by lemmas 2.5-2.9.

```
    procedure S̄(s:nptr;var t:nptr;[q]:eq stateset);
        var Q':eq stateset;
      begin S(s,t,[q],Q'); end
```

We have defined the graph G_1.

Let $Q_2 = [(p_1*\{\langle \phi, \phi \rangle\})*\sigma_1']$

$P_2 = Q_2*[\sigma_2]$

$\sum_2' = [\sigma_2]*Q_2$

Assume inductively that G_{i-1}, P_i, $\sum_i'$ are known. Let r_1 be the (ptr to) the root of G_{i-1}. Following program segment P defines the root (nptr) t of $\bar{G}_i$ and $\sum_{i+1}'$, P_{i+1}. σ_{i+1} is assumed known.

```
P: var s,t:nptr; Q':eq stateset;
   begin new(s);
         s↑.F := (r1↑.F)*(r1↑.D);
         s↑.D := Σi';
         s↑.R := r1;
         s↑.E := r1↑.D;
         S(s,t,Pi,Q');
         Pi+1 := Q'*[σi+1];
```

$\sum'_{i+1} := [\sigma_{i+1}]*Q';$

end

We redefine the function search on ntable.

```
function search (aset:set of nptr;var t:nptr):boolean
   var x:nptr;
begin search := false;
      for x in aset do begin
      if (t↑. = x↑.) then begin search := true;
                                          t := x
                              end;
                              end;
end.
```

The procedure graph is modified to procedure graphbar.

```
procedure graphbar(r:nptr);
      var t,r':nptr;
begin
   while nque <> empty do
      begin r' := delete (nque);
         for [q] in Q̄ do begin S̄(r',t,[q]);
         if search (ntable,t) then nedges:=nedges+ {(r',t,[q])}
               else begin insert (nque,t);
                    ntable := ntable + {t};
                    nedges := nedges + {(r',t,[q])}
               end;
               end;
         end;
```

After execution of P, suppose $t = r_0$. r_0 is the nptr identifying the

root of $\bar{G}_i$. To obtain the rest of $\bar{G}_i$, we execute the following

```
begin initialize nque to r0;
      nedges := null;
      ntable := {r0};
      graphbar (r0);
end;
```

The graph $\bar{G}_i$ obtained above is deterministic. The function search together with the procedure graphbar do not permit two vertices of $\bar{G}_i$ to have identical fields. Further for every vertex r, and every [q], the [q]-successor of r is unique and defined.

To obtain G_i from $\bar{G}_i$ we define an equivalence relation $\bar{\equiv}$ on the nodes of $\bar{G}_i$. We shall use $r_1, r_2 \ldots$ etc. to denote (nptr to) nodes of $\bar{G}_i$. The components of node r_1, are designated F_{r_1}, D_{r_1}, R_{r_1} and E_{r_1} respectively.

Definition 3.1: r_1, r_2 be two nodes of $\bar{G}_i$.

$r_1 \bar{\equiv}_0 r_2 \iff F_{R_{r_1}} * E_{r_1} = F_{R_{r_2}} * E_{r_2}$ and $D_{r_1} = D_{r_2}$, $E_{r_1} = E_{r_2}$.

$r_1 \bar{\equiv}_k r_2 \iff$ for all [q] $\varepsilon \bar{Q}$, [q] succ$(r_1) \bar{\equiv}_{k-1}$ [q]succ(r_2).

We define $r_1 \bar{\equiv} r_2 \iff$ for all k, $r_1 \bar{\equiv}_k r_2$.

We summarize several properties of this relation.

1. $\bar{\equiv}$ is an equivalence relation on the ndoes of $\bar{G}_i$.

$\bar{\equiv}_0$ is by definition reflexive, symmetric and transitive.

Assume as induction hypothesis that so is $\bar{\equiv}_{k-1}$.

Since [Δ] $\varepsilon \bar{Q}$ and [Δ] succ(r) = r for all r in $\bar{G}_i$, $\bar{\equiv}_k$ is reflexive. Since $\bar{\equiv}_{k-1}$ is symmetric, so is $\bar{\equiv}_k$, by definition. Further, $r_1 \bar{\equiv}_k r_2$ and $r_2 \bar{\equiv}_k r_3$ implies

$\forall$ [q] $\varepsilon \bar{Q}$, [q] succ$(r_1) \bar{\equiv}_{k-1}$ [q] succ(r_2) and

$\forall [q] \in \bar{Q}$, $[q]\,succ(r_2) \equiv_{k-1} [q]\,succ(r_3)$

and hence $\forall [q] \in \bar{Q}$, $[q]\,succ(r_1) \equiv_{k-1} [q]succ(r_3)$ by induction hypothesis. This implies $r_1 \equiv_k r_3$ and hence the property.

2. The partition P_k over the nodes of $\bar{G}_i$ defined by $\equiv_k$ is a refinement of P_{k-1} defined by $\equiv_{k-1}$, since

$$r_1 \equiv_k r_2 \Rightarrow [\Delta]succ(r_1) \equiv_{k-1} [\Delta]\,succ(r_2) \Rightarrow r_1 \equiv_{k-1} r_2.$$

3. $P_{k-1} = P_k$ implies $P_k = P_{k+1}$:

For, $r_1 \equiv_{k+1} r_2 \Leftrightarrow \forall [q] \in \bar{Q}$, $[q]succ(r_1) \equiv_k [q]succ(r_2)$

$\Leftrightarrow \forall [q] \in \bar{Q}\ [q]succ(r_1) \equiv_{k-1} [q]succ(r_2)$

$\Leftrightarrow r_1 \equiv_k r_2$.

4. $r_1 \equiv r_2 \Leftrightarrow r_1 \equiv_0 r_2$ and $\forall [q] \in \bar{Q}$, $[q]succ(r_1) \equiv [q]succ(r_2)$.

For

$r_1 \equiv r_2 \Leftrightarrow \forall k\ r_1 \equiv_k r_2$

$\Leftrightarrow r_1 \equiv_0 r_2$ and $\forall k \geq 1 \forall [q]\ [q]succ(r_1) \equiv_{k-1} [q]succ(r_2)$

$\Leftrightarrow r_1 \equiv_0 r_2$ and $\forall [q]\ \forall k \geq 0\ [q]succ(r_1) \equiv_k [q]succ(r_2)$

$\Leftrightarrow r_1 \equiv_0 r_2$ and $\forall [q]\ [q]succ(r_1) \equiv [q]succ(r_2)$

5. From 4 above it follows that if $[r]$ is an equivalence class under $\equiv$, then for each $[q]$, $[q]succ[r]$ is an uniquely defined equivalence class.

6. $[r]$ be an equivalence class. Then, $r_1, r_2 \in [r] \Rightarrow F_{r_1} = F_{r_2}$ and $D_{r_1} = D_{r_2}$. For $r_1 \equiv r_2 \Rightarrow r_1 \equiv_0 r_2 \Rightarrow D_{r_1} = D_{r_2}$ and $E_{r_1} = E_{r_2}$ and $F_{R_{r_1}} * E_{r_1} = F_{R_{r_2}} * F_{r_2}$. From procedure S, for any node t of $\bar{G}_i$,

$$F_t = F_{R_t} * E_t$$

Thus $r_1 = r_2 \Rightarrow D_{r_1} = D_{r_2}$ and $F_{r_1} = F_{r_2}$.

Consider the equivalence classes of the nodes of $\bar{G}_i$. By (6) all nodes in the equivalence class have same F,D value. By (5) [q] successor of an equivalence class is unique.

G_i is the graph obtained form $\bar{G}_i$ by associating one vertex with each equivalence class. Let $[r_0]$, $[r_1]$ ... $[r_k]$ be the equivalence classes of the vertices of $\bar{G}_i$. To define G_i, we need to define the variables table and edges. $t_0, t_1 \ldots t_k$ be ptrs. Let table = $\{t_0, t_1 \ldots t_k\}$ and for $0 \leq i \leq k$, $t_i\uparrow. := r_i\uparrow.$; also, $(t_i, t_j, [q])$ ε edges if and only if [q] succ$(r_i) = r_j$. $[r_0]$ be the equivalence class containing the root of $\bar{G}_i$. Then t_0 is the root of G_i. The graph G_i is deterministic since for each vertex t, and [q], [q]-successor of t is uniquely defined. We now have the following.

Lemma 3.6: Let α bc an entry stateset sequence. Let $\bar{v}[\alpha]$ ($v[\alpha]$) be the $[\alpha]$ successor of root in $\bar{G}_i(G_i)$. Then

$$F_{\bar{v}[\alpha]} = F_{v[\alpha]} \text{ and } D_{\bar{v}[\alpha]} = D_{v[\alpha]}.$$

Proof: By induction on length m of α. For m=0, $\bar{v}[\alpha](v[\alpha])$ is the root of $\bar{G}_i(G_i)$, and the lemma holds by definition. Assume as induction hypothesis that the statement of the lemma holds for m. Let $\alpha_1 = \alpha q$. Let $[r_i]$ be the equivalence class to which the vertex $\bar{v}[\alpha]$ belongs. If $v[\alpha] = t_j$, then $t_j\uparrow. = r_i\uparrow.$ by construction. Also, $\bar{v}[\alpha q]$ = [q] successor ($\bar{v}[\alpha]$) and hence $\bar{v}[\alpha q]$ is defined by [q] successor (r_i). If [q] successor $(r_i) = r_1$ then $t_1\uparrow. = r_1\uparrow.$ and $(t_i, t_1, [q])$ ε edges. Therefore t_1 is $[\alpha][q] = [\alpha q]$ successor in G_i. Hence the lemma.

Lemma 3.6 shows that for each entry stateset sequence α, the F

and D components of the [α] successor in $\bar{G}_i$ and G_i are identical. This correspondence however is not unique in terms of α. There could be a sequence α' such that $F_{\bar{v}[\alpha']} = F_{v[\alpha]}$ and $D_{\bar{v}[\alpha']} = F_{v[\alpha]}$. For instance if in the proof of lemma 3.6, $\bar{v}[\alpha q] = \bar{v}[\alpha]$, then $F_{v[\alpha q^i]} = F_{v[\alpha]}$ and similarly for the D components.[1] In section 4 we relate the information in the nodes of G_i with (i,i+1) sweeps of M.

We now estimate the number of equivalence classes of $\bar{G}_i$ under $\equiv$. Let $r_1, r_2 \in [r]$. Then $r_1 \equiv r_2 \Rightarrow r_1 \equiv_0 r_2$ and hence $D_{r_1} = D_{r_2}$, $E_{r_1} = E_{r_2}$ and $F_{R_{r_1}} * E_{r_1} = F_{R_{r_2}} * E_{r_2}$. Since $F_t = F_{R_t} * E_t$ for any node t, the condition above yields

$$D_{r_1} = D_{r_2},\ E_{r_1} = E_{r_2} \text{ and } F_{r_1} = F_{r_2}.$$

Therefore any two members of the class [r] must have same D,E,F components. Suppose now $R_{r_1} = R_{r_2}$. Then vertices r_1, r_2 of $\bar{G}_i$ have same F,D,R,E components and hence $r_1 = r_2$. Therefore the number of equivalence classes of the vertices of $\bar{G}_i$ is no more than the number of nodes of G_{i-1}. By construction, G_1 has at most 2^p nodes, since each node must contain a distinct eq symbolset. Assuming inductively that the number of nodes of G_{i-1} is $< 2^p$, the number of equivalence classes of $\bar{G}_i$ and hence the nubmers of nodes of G_i is $< 2^p$.

Lemma 3.7: The number of steps required to compute G_i from $\bar{G}_i$ is at most $2n^2 2^{2m+7p}$.

Proof: We first consider the construction of $\bar{G}_i$ from G_{i-1}. The number of nodes of G_{i-1} is at most 2^p and thus the number of edges of G_{i-1} is at most $n2^p$. Since the function edgesearch searches the set edges, the maximum number of steps required by edgesearch (t,x)

---

[1] q^i here denotes a string of i q's.

is $\leq n.2^p$. To obtain $\bar{G}_i$ we need to make as many calls to procedure S as the number of edges of $\bar{G}_i$. Each node of $\bar{G}_i$ has the components F,D,R,E. Since R can have at most 2^p values, the number of nodes of $G_i \leq 2^m.2^p.2^p.2^p = 2^{m+3p}$. Thus the number of edges of $\bar{G}_i \leq n2^{m+3p}$.

From procedure S, a call S(s,t,[q]) needs $\leq n2^p + c$ steps, since in addition to the call to edgesearch, only a constant number of steps are required. The total number of steps required to compute $\bar{G}_i$ is

$$\leq (n2^p+c)\ n2^{m+3p} \leq 2n^2 2^{m+4p}$$

since c is small. We next consider the construction of G_i from $\bar{G}_i$. The nubmer of equivalence classes of nodes of $\bar{G}_i \leq 2^p$. To compute a partition P_i from P_{i-1}, we may have to compare all the q-successor of each pair of nodes, which would require $\leq n2^{2m+6p}$ steps. The maximum number of partitions that may have to be computed is $\leq 2^p$. Thus computation of the equivalence classes needs $\leq n2^{2m+7p}$ steps. Therfore the total number of steps required to compute G_i is

$$\leq 2n^2 2^{m+4p} + n2^{2m+7p}$$

$$\leq 2n^2 2^{2m+7p}.$$

From the construction of G_i we observe that G_i is obtained form G_{i-1}, P_i and $\sum_i'$. A constant number of steps are therefore required for computation of G_i. The number of steps does not depend on i.

Section 4: Properties of G_i:

In this section we show that G_i contains the required information about the computations of the sweep machine in its node and edge labels.

Let $\alpha = q_{i_1} \ldots q_{i_k}$ be an arbitrary sequence of entry statesets. In section 2, we hve defined the symbols $w_{i,\alpha}$ $f_{w_{i,\alpha}}$ and $\sigma_{i,\alpha}$. Thus, $w_{i,\alpha}$ is the string of symbolsets to the left of the i-boundary and $f_{w_{i,\alpha}}$ the crossing function of this string at the end of the (i,i+1) secondary sweep sequence α. $\sigma_{i,\alpha}$ is the symbol at cell i at the end of this sequence. At boundary i, the sequence of entry statesets including the (i,i+1) primary sweep is $p_i, q_{i_1}, \ldots, q_{i_k}$. By definition, at the end of the (i-1,i) primary sweep, the symbolset at cell i-1 is $\sigma_{i-1,\Lambda}$ and the crossing function of the string of symbolsets to the left of the (i-1) boundary is $f_{w_{i-1,\Lambda}}$.

We now define two functions, denoted α' and sym(i-1,α'). For given α (and i), α' is the sequence of (secondary) entry statesets at boundary i-1. sym(i-1,α') is the symbolset at cell (i-1) at the end of (i,i+1) secondary sweep sequence α. We observe that if $\alpha = \Lambda$, α' and sym(i-1,α') are defined by p_i. If we consider all (i,i+1) sweeps, then the least value of α is p_i.

Definition 4.1 Let $\alpha = \Lambda$. Then $\alpha' = p_i * \sigma_{i-1,\Lambda}$ and sym(i-1,α') = $(\sigma_{i-1,\Lambda} * p_i) * f_{w_{i-1,\Lambda}}(\alpha')$.

Assume now that α' and sym(i-1,α') have been defined for all α of length n.

Then $(\alpha q)' = \alpha' q * syn(i-1,\alpha')$

$= \alpha' q'$ where $q' = q * syn(i-1,\alpha')$

and $syn(i-1,(\alpha q)') = (syn(i-1,\alpha') * q) * f_{w_{i-1,\alpha'}}(q')$

Following lemma shows that α' and $syn(i-1,\alpha')$ have the properties described above.

Lemma 4.1: Let $\alpha = q_{i_1} \ldots q_{i_m}$ be a sequence of (entry) statesets. If m (i,i+1) secondary sweeps are performed with entry statesets defined by α, then α' is the sequence of entry statesets for the corresponding (i-1,i) secondary sweeps and $syn(i-1,\alpha')$ is the symbolset at cell (i-1) at the end of the (i,i+1) secondary sweep sequence α.

Proof: by induction on m. For $m=0$, $\alpha=\Delta$.

For $\alpha=\Delta$, the entry stateset at i is p_i and since the symbolset in cell (i-1) is $\sigma_{i-1,\Delta}$, the entry stateset at i-1 is $p_i * \sigma_{i-1,\Delta} = \alpha'$. The exit stateset at (i-1) boundary then is $f_{w_{i-1,\Delta}}(\alpha')$. Since the symbolset at (i-1) is $(\sigma_{i-1,\Delta} * p_i)$, the symbolset at (i-1) when the (i,i+1) primary sweep exits at i is $(\sigma_{i-1,\Delta} * p_i) * f_{w_{i-1,\Delta}}(\alpha') = syn(i-1,\alpha')$.

Assume now inductively that at the end of the (i,i+1) sequence α, $|\alpha| = m$, the sequence of secondary entry statesets at (i-1) is α' and the symbol at cell (i-1) is $syn(i-1,\alpha')$. Consider now the next (i,i+1) secondary sweep with entry stateset q. Then the entry stateset at (i-1) is $q * syn(i-1,\alpha')$ and hence $(\alpha q)' = \alpha' q'$. When this (m+1) st secondary sweep moves to left of (i-1) boundary, the symbolset at (i-1)

cell is sym(i-1,α')*q. By inductive hypothesis, the crossing function of the string of symbolsets to left of (i-1) boundary is $f_{w_{i-1,\alpha'}}$.

Then, the exit state at (i-1) boundary is

$f_{w_{i-1,\alpha'}}(q')$ and hence the symbolset at (i-1) is

$(sym(i-1,\alpha')*q)*f_{w_{i-1,\alpha'}}(q') = sym(i-1,(\alpha q)')$.

This completes the inductive step and hence the lemma.

Consider now the call sweep (i) corresponding to the (i,i+1) primary sweep. On entry to the sweep function, stA(i) = p_i, symA(i) = σ'_i. On exit from the function, stA(i) = Q_{i+1}, symA(i) = $\sigma_{i,\Delta}$, where Q_{i+1} is the stateset in which the (i,i+1) primary sweep crosses the (i+1) boundary to the right.

The relationship between the symbols denoting configurations of the sweep machine are summarized in the lemma below.

Lemma 4.2: For all i,α,

1) $f_{w_{i,\alpha}} = f_{w_{i-1,\alpha'}} * sym(i-1,\alpha')$

2) $Q_{i+1} = f_{w_{i-1,\Delta}} * \sigma_{i-1,\Delta}(p_i) * \sigma'_i$

3) $\sigma_{i,\alpha q} = \sigma_{i,\alpha} * f_{w_{i,\alpha}}(q)$

4) $\sigma_{i,\Delta} = \sigma'_i * f_{w_{i-1,\Delta}} * \sigma_{i-1,\Delta}(p_i)$

5) $p_{i+1} = Q_{i+1} * \sigma_{i+1}$; $\sigma'_{i+1} = \sigma_{i+1} * Q_{i+1}$.

Proof: From lemma 4.1 above, at the end of the (i,i+1) secondary sweep sequence α, the symbol at cell (i-1) is sym(i-1,α'). Since α' is the (i-1,i) secondary sweep sequence at (i-1) boundary, $f_{w_{i-1,\alpha'}}$ is

the crossing function of the string of symbolsets to left of (i-1) boundary. From definition of crossing function and symbol product.

$f_{w_{i,\alpha}} = f_{w_{i-1,\alpha'}} * \mathrm{sym}(i-1,\alpha')$. Therefore(1).

At the beginning of the (i,i+1) primary sweep, the entry state set at boundary i is p_i, symbolset at cell i is σ'_i. At the end of the (i-1,i) primary sweep, the crossing function of the string of symbolsets to left of (i-1) boundary is $f_{w_{i-1,\Delta}}$ and the symbolset at (i-1) is $\sigma_{i=1,\Delta}$. Thus the exit stateset at i boundary at the end of (i,i+1) primary sweep is $f_{w_{i-1,\Delta}} * \sigma_{i-1,\Delta}(p_i)$. Since the symbolset at i is still σ'_i, the exit state set at (i+1) boundary $Q_{i+1} = f_{w_{i-1,\Delta}} * \sigma_{i-1,\Delta}(p_i) * \sigma'_i$. Hence (2).

It follows that the symbol set at cell i after the (i,i+1) primary sweep crosses the (i+1) boundary is

$\sigma_{i,\Delta} = \sigma'_i * f_{w_{i-1,\Delta}} * \sigma_{i-1,\Delta}(p_i)$. Hence (4).

(5) now follows since the symbolset at cell (i+1) is σ_{i+1} and the exit stateset at (i+1) boundary is Q_{i+1}. Consider now (3). By definition, at the end of (i,i+1) secondary sweep sequence α, the symbolset at cell i is $\sigma_{i,\alpha}$ and the crossing function of the string to left of i boundary is $f_{w_{i,\alpha}}$. If the entry stateset for the next (i,i+1) secondary sweep is q, then the exit stateset at i boundary is $f_{w_{i,\alpha}}(q)$ and hence

$\sigma_{i,\alpha q} = \sigma_{i,\alpha} * f_{w_{i,\alpha}}(q)$.

Let $\alpha = q_{i_1} q_{i_2} \ldots q_{i_m}$ be a sequence of statesets of length m, and let $[\alpha] = [q_{i_1}][q_{i_2}]\ldots[q_{i_m}]$ be the corresponding sequence of equivalence classes of statesets or eq statesets. We consider two properties of the graphs G_i and $\bar{G}_i$, called the vertical and horizontal property respectively. We refer to these as (V,i,m) and (H,i,m).

Definition 4.2:

(V,i,m): Let $\alpha = q_{i_1} \ldots q_{i_m}$ and v the $[\alpha]$ successor of the root in G_i.

Then $[f_{w_{i,\alpha}}] = F_v$ and $[\sigma_{i,\alpha}] = D_v$.

(H,i,m): Let $\alpha = q_{i_1} \ldots q_{i_m}$ and v the $[\alpha]$ successor of the root in $\bar{G}_i$.

Then $R_v = [\alpha']$ successor of root in G_{i-1}

$[f_{w_{i-1,\alpha'}}] = F_{R_v}$ and $[\mathrm{sym}(i-1,\alpha')] = E_v$.

The property (V,i,m) states that at the end of the (i,i+1) secondary sweep sequence α, the equivalence class of the crossing function of the symbolset string to the left of the i boundary and that of the symbolset in cell i are equal to the F and D components of the $[\alpha]$ successor of the root in G_i. The property (H,i,m) states that at the end of the (i,i+1) secondary sweep sequence α, the $[\alpha']$ successor of the root in G_{i-1} is the R component of the $[\alpha]$ successor of the root in $\bar{G}_i$. The F component of the $[\alpha']$ successor in G_{i-1} equals the equivalence class of the crossing function of the string of symbolsets to left of (i-1) boundary. The equivalence class of the symbolset at i-1 is equal to the E component of the $[\alpha]$ successor of $\bar{G}_i$.

We intend to show $(V,\forall i,\forall m)$. The plan of the proof is as follows: For basis, we show $(V,1,\forall m)$ and define $[p_2] = P_2$, $[\sigma_2'] = \sum_2'$.

We assume as induction hypothesis H1, $(V,i-1,\forall m)$ and $[p_i] = P_i$, $[\sigma_i'] = \sum_i'$. We show $(V,i,0)$.

We then assume as induction hypothesis H2, (V,i,m)

We show as basis $(H,i,0)$

We assume as induction hypothesis H3 (H,i,m) and show $(H,i,m+1)$

We thus have $(H,i,\forall m)$, using which and H2 we show $(V,i,m+1)$. Thus $(V,i,\forall m)$ and together with H1, we have $(V,\forall i,\forall m)$.

$(V,1,\forall m)$ has been proved in lemma 3.4. Q_2 is defined by the procedure S'. From lemma 4.2, $p_2 = Q_2 * \sigma_2$ and $\sigma_2' = \sigma_2 * Q_2$. We define $P_2 = [p_2]$ and $\sum_2' = [\sigma'_2]$.

Assume now as induction hypothesis (H1) that for all α, if v is the $[\alpha]$ successor, in G_{i-1} then,

$(V,i-1,\forall m)$: $[f_{w_{i-1,\alpha}}] = F_v$, $[\sigma_{i-1,\alpha}] = D_v$ and $[p_i] = P_i$, $[\sigma_i'] = \sum_i'$.

We show $(V,i,0)$, and $[p_{i+1}] = P_{i+1}$, $[\sigma'_{i+1}] = \sum_{i+1}'$;

Let r_1 be the root of G_{i-1} and r_2 that of $\bar{G}_i$. From the construction of $\bar{G}_i$, r_2 is defined by the procedure call $S(s,t,P_i,Q')$ where $F_s = F_{r_1} * D_{r_1}$, $E_s = D_{r_1}$, $R_s = r_1$ and $D_s = \sum_i'$, and on exit $t = r_2$.

From $(V,i-1,0)$, $[f_{w_{i-1,\Delta}}] = F_{r_1}$ and $[\sigma_{i-1,\Delta}] = D_{r_1}$ (4.1)

and $[p_i] = P_i$, $[\sigma_i'] = \sum_i'$ (4.2)

From procedure body S,

$$A = P_i * D_{r_i} = [p_i] * [\sigma_{i-1,\Delta}] \text{ form (4.1) and (4.2) above}$$

$$= [p_i * \sigma_{i-1,\Delta}] \text{ by lemma 2.6}$$

$$B = D_{r_1} * P_i = [\sigma_{i-1,\Delta} * P_i], \text{ similarly.}$$

Using the ' function of definition 4.1, $A = [\Delta']$. Let v_1 be the A-successor in G_{i-1}.

From H1, $[f_{w_{i-1,\Delta'}}] = F_{v_1}$ and $[\sigma_{i-1,\Delta'}] = D_{v_1}$

From procedure S, $R_{r_2} = v_1$, $X = F_{v_1}$ and $Y = F_{R_s} = F_{r_1}$

Therefore, $C = F_{r_1}(A) = [f_{w_{i-1,\Delta}}]\ ([p_i * \sigma_{i-1,\Delta}])$

$$= [f_{w_{i-1,\Delta}}\ (p_i * \sigma_{i-1,\Delta})] \text{ by lemma 2.6}$$

Then, $E_{r_2} = B * C$

$$= [\sigma_{i-1,\Delta} * p_i] * [f_{w_{i-1,\Delta}}\ (p_i * \sigma_{i-1,\Delta})]$$

$$= [(\sigma_{i-1,\Delta} * p_i) * f_{w_{i-1,\Delta}}\ (p_i * \sigma_{i-1,\Delta})] \text{ by lemma 2.6}$$

$$= [\text{sym}(i-1,\Delta')] \text{ by definition 4.1} \qquad (4.3)$$

Further,

$D_{r_2} = D_s * F_{r_1} * D_{r_1}(P_i)$. From equations 4.1 and 4.2 above,

$$F_{r_1} * D_{r_1}(P_i) = [f_{w_{i-1,\Delta}}] * [\sigma_{i-1,\Delta}]\ ([p_i])$$

$$= [f_{w_{i-1,\Delta}} * \sigma_{i-1,\Delta}(p_i)] \text{ by lemma 2.6} \qquad (4.4)$$

Since $D_s = \sum_i' = [\sigma_i']$ by H1,

$$D_{r_2} = [\sigma_i' * f_{w_{i-1,\Delta}} * \sigma_{i-1,\Delta}(p_i)] \text{ by lemma 2.6}$$

$= [\sigma_{i,\Delta}]$. by lemma 4.2 (4.5)

Also, $F_{r_2} = F_{v_1} * E_{r_2}$

$= [f_{w_{i-1,\Delta'}}] * [sym(i-1,\Delta')]$

$= [f_{w_{i-1,\Delta'}} * sym(i-1,\Delta')]$

$= [f_{w_{i,\Delta}}]$ by lemma 4.2, 2.6. (4.6)

Using procedure S and equation (4.4) above,

$$Q' = F_s(P_i) * D_s = F_{r_1} * D_{r_1}(P_i) * \sum_i'$$

From program segment P, and equation (4.2),

$$P_{i+1} = Q' * [\sigma_{i+1}]$$

$= [f_{w_{i-1,\Delta}} * \sigma_{i-1,\Delta}(p_i) * \sigma_i' * \sigma_{i+1}]$

$= [p_{i+1}]$ by lemma 4.2 (4.7)

and similarly,

$$\sum_{i+1}' = [\sigma_{i+1}] * Q' = [\sigma_{i+1}'].$$ (4.8)

Equations (4.5) to (4.8) show (V,i,0), by construction of G_i from $\bar{G}_i$.

Assume now as induction hypothesis H2, (V,i,m). i.e. for $\alpha = q_{i_1} q_{i_2} \ldots q_{i_m}$,

$$[f_{w_{i,\alpha}}] = f_{[\alpha]},\ [\sigma_{i,\alpha}] = D_{[\alpha]} \text{ in } G_i$$

where $F_{[\alpha]}(D_{[\alpha]})$ denote the F and D components of the (unique) $[\alpha]$ successor of the root in G_i. We now show (H,i, m). For basis, (H,i,0), is already shown. For m=0, $\alpha=\Delta$ and $[\Delta]$ successor of the root in $\bar{G}_i$ is r_2. Then,

$R_{r_2} = v_1$ where v_1 = A-successor of the root in G_{i-1}

$= [\Delta']$ successor in G_{i-1}, since $A = [\Delta']$.

$E_{r_2} = [sym(i-1,\Delta')]$ from equation (4.3)

and

$F_{R_{r_2}} = F_{v_1} = [f_{w_{i-1,\Delta'}}]$ which demonstrates the basis.

Assume now as inductive hypothesis (H3), (H,i,m) i.e. if $\alpha = q_{i_1} \ldots q_{i_m}$, v the $[\alpha]$ successor in $\bar{G}_i$ then $R_v = [\alpha']$ successor in G_{i-1},

$$[f_{w_{i-1,\alpha'}}] = F_{R_v},\ [sym(i-1,\alpha')] = E_v \qquad (4.9,\ 4.10)$$

To show (H,i,m+1),

let $\alpha_1 = \alpha q$. Then $\alpha_1' = \alpha' q'$ where

$$q' = q * sym\ (i-1,\alpha') \text{ by definition 4.1} \qquad (4.11)$$

Let v_1 be the $[\alpha][q]$ successor in $\bar{G}_1$. Then by construction v_1 is defined by the procedure call S(v,t,[q],Q') where, on exit $t=v_1$.

From procedure body,

$A = [q] * E_v = [q] * [sym(i-1,\alpha')]$ from equation (4.10)

$= [q * sym(i-1,\alpha')]$ by lemma 2.6

$= [q']$ from equation (4.11)

$B = E_v * [q] = [sym(i-1,\alpha') * q]$

R_{v_1} = A-succ of R_v in G_{i-1}

$= [q']$ succ of R_v in G_{i-1}

and $X = F_{R_{v_1}}$, $Y = F_{R_v}$, $C = Y(A) = F_{R_v}(A)$

$= [f_{w_{i-1,\alpha'}}]([q*sym(i-1,\alpha')])$ by equation (4.9)

$= [f_{w_{i-1,\alpha'}}(q*sym(i-1,\alpha'))]$ by lemma 2.6-2.8.

Then

$E_{v_1} = B*C = [(sym(i-1,\alpha')*q) * f_{w_{i-1,\alpha'}}(q*sym(i-1,\alpha'))]$ using lemma 2.6

$= [sym(i-1,(\alpha q)')]$ by definition 4.1 (4.12)

$F_{v_1} = F_{R_{v_1}} * E_{v_1}$ from procedure S. Also,

R_{v_1} = [q'] successor of R_v in G_{i-1} and

R_v = [α'] successor of root in G_{i-1} from (H,i,m).

Then R_{v_1} = [$\alpha'q'$] successor in G_{i-1} = [$(\alpha q)'$] successor in G_{i-1} (4.13)

From H1,

$F_{R_{v_1}} = [f_{w_{i-1,\alpha'q'}}] = [f_{w_{i-1,(\alpha q)'}}]$ (4.14)

Equations (4.12) to (4.14) show (H,i,m+1). From these equations,

$F_{v_1} = [f_{w_{i-1,(\alpha q)'}}] * [sym(i-1,(\alpha q)']$

$= [f_{w_{i-1,(\alpha q)'}} * sym(i-1,(\alpha q)')]$

$= [f_{w_{i,\alpha q}}]$ by lemma 4.2 (4.15)

By lemma 3.6, the [α] successors of G_i and $\bar{G}_i$ have same F, D components. Then

$F_{[\alpha]} = F_v$; $D_{[\alpha]} = D_v$ and

$F_{[\alpha q]} = F_{v_1}$; $D_{[\alpha q]} = D_{v_1}$.

Then, from S(v,t,[q],Q'), since $t = v_1$ on exit,

$D_{v_1} = D_v * F_v([q]) = D_{[\alpha]} * F_{[\alpha]}([q])$

$= [\sigma_{i,\alpha}] * [f_{w_{i,\alpha}}(q)]$ by H2

$= [\sigma_{i,\alpha q}]$ by lemma 2.6

Since in G_i, $F_{[\alpha q]} = F_{v_1} = [f_{w_{i,\alpha q}}]$ from equation (4.15) and

$$D_{[\alpha q]} = D_{v_1} = [\sigma_{i,\alpha q}],$$ we have $(V, i, m+1)$

and therefore $(V, \forall_i, \forall_m)$

We therefore have the following,

Theorem 4.1 Let $\alpha = q_{i_1} q_{i_2} \cdots q_{i_m}$ be a (possibly empty) sequence of entry statesets, and w be an arbitrary input. If G_i is the context graph at boundary $i (i \geq 1)$, at the end of the $(i, i+1)$ primary sweep and v the $[\alpha]$ successor of the root of G_i, then

$$[f_{w_{i,\alpha}}] = F_v, \quad [\alpha_{i,\alpha}] = D_v.$$

Section 5: Uniformity of $\bar{P}_2$:

Consider the program $\bar{P}_1$ defined in section 1. Let w be an input of length n . With the array A initialized in the manner defined in section 1, the program $\bar{P}_1$ halts on input w, the halt stateset is accepting if $w \in L$ and rejecting if $w \notin L$. In section 3, we have defined the context graph and the procedure for it's computation. From the proof of theorem 4.1 and the procedure graphbar, it follows that in computing G_i we need three parameters. These are G_{i-1}, $\sum_i'$ and P_i. However, $\sum_i' = [\sigma_i']$ and $P_i = [p_i]$. The symbolset σ_i' and the stateset p_i are contained in symA(i) and stA(i) at the end of the (i-1,i) primary sweep. We have also shown that each context graph G_i is a finite grpah, its size independent of i.

We shall assume that the cells of the array A have a third component, that stores the context graph. Thus, at the end of the (i-1,i) primary sweep, graph A(j) = G_j for $1 \leq j < i$, graph (A(j)) = $\emptyset$ for $j \geq i$. The procedure for computation of G_i from G_{i-1} can now be included in the function sweep. This function called newsweep needs only one parameter, the array A. It locates the sweep marker by scanning the array A from left.

```
function newsweep (A:array):stateset;
var i : integer;
begin i := 1; while symA(i) unmarked do i := i+1;
      sweep(i);
      compute G_i; graph A(i) := G_i;
end.
```

Given input w, $|w| = n$, $w = a_1 a_2 \ldots a_n$, we initialize array A as in section 1, and perform the (1,2) primary sweep by the call sweep (1). Computation of G_1 is defined in section 3. We let graph A(1) = G_1; and graph A(i) = $\emptyset$ for i > 1. With this value of A as the initial value, we have

```
P̄2 : var R : state set;
     begin R := nil; initialize A;
          while R ∩ H <> nil do
               R := newsweep (A);
     end.
```

For every input w, $\bar{P}_2$ terminates. The halting stateset on exit is accepting if w ε L and rejecting otherwise. We show that $\bar{P}_2$ is uniform.[1]

For a given w, let $\bar{w}$ be obtained by adding enough ♭ symbols to w, so that its length is S(n)+1. i.e., $\bar{w} = ¢\, a_1 a_2 \ldots a_{S(n)+1}$ where $a_i = ♭$ for $n < i \leq S(n)+1$. Given $\bar{w}$, initialization of A would proceed as defined above. A configuration C is a 5-tuple (W,G,p,σ',δ) where W, δ are strings of symbol sets, G a context grpah, p a stateset and σ' a symbolset, satisfying the following property: for some input w and some value of i,

$[W] = [w_{i-1}, \Lambda, \sigma_{i-1}, \Lambda]$, $G = G_{i-1}$, $[\sigma'] = [\sigma'_i]$ and $[p] = [p_i]$.

δ is an arbitrary string of symbolsets of the form $\gamma\{\langle ♭,♯\rangle\}^k$ where γ is a (possibly empty) string over $\{\langle x,♭\rangle\}$, $x \in \sum - \{♯\}$. δ preserves the format of the initial string of symbolsets defined in section 1. During the computations of $\bar{P}_2$, additional symbolsets $\{\langle ♭,♯\rangle\}$ will be appended to δ as required. This implies that δ at the least consists of the symbolset $\{\langle ♭,♯\rangle\}$.

---

[1] The formalization used in this paper was defined as the well-behaved property in [2].

Given a 5-tuple of the format described above, we can decide if it represents a configuration by starting with finitely many initial configurations with $|w| = |W|+1$ and performing $|W|-1$ newsweeps.

Let $C = (W,G,p,\sigma',\delta)$ be a configuration. The function newsweep is defined on C and yields a unique output, if we initialize array A as follows. All components of A are set equal to $\emptyset$ before initialization. Let $W = W_0W_1\ldots W_{i-1}$ and $\delta = \delta_1\delta_2\ldots\delta_k$. Then we let

$$\begin{aligned} \text{symA}(j) &= W_j \text{ for } 0 \leq j < i \\ &= \sigma' \text{ for } j = i \\ &= \delta_{j-i} \text{ for } i < j \leq i + k \\ &= \{\langle \not b, \not b \rangle\} \text{ for } j > i + k, \end{aligned}$$

stA(i+1) = p, graph A(i) = G

and then call newsweep (A). We shall show subsequently that the output is also a configurations. A configuraiton C is a <u>null configuration</u> if p is a halting stateset. Let null(Y) denote the set of all null configurations. From the definitions of halting statesets and symbolsets, it follows that newsweep maps null configurations to null configurations.

We consider now the set of all intermediate configurations of $\bar{P}_2$, denoted by D. Let w be any input and A_w the array obtained by the initialization procedure defined in the paragraph immediately preceeding $\bar{P}_2$. Let C_w be the corresponding configuration. Let $\mathcal{C}$ be the set of all configurations and $F_s: \mathcal{C} \dashrightarrow \mathcal{C}$ the total function on $\mathcal{C}$ defined by the loop body of $\bar{P}_2$.

Let $D = \{F_s^k(C_w) | k \geq 0\}$. D consists of precisely those configurations that can arise during successive iterations of $\bar{P}_2$ with

the initial input as A_w. Since each element of D is a configuration, $D \subseteq \mathcal{C}$.

Consider now the entire program $\bar{P}_2$. For every input w, with A initialized to A_w, $\bar{P}_2$ terminates, the output being a null configuration. If instead A is initialized to an element $F_s^k(C_w)$ of D, then $\bar{P}_2$ would terminate, the output configuration will be the same as that obtained with initialization A_w. Therefore the function $F : D \longrightarrow null(Y)$ defined by $\bar{P}_2$ is total on D. We now define an equivalence relation on $\mathcal{C}$. First we consider strings of symbolsets δ. If $\delta = \delta_1 \ldots \delta_k$, then $[\delta] = [\delta_1] \ldots [\delta_k]$. $\delta_1 \approx \delta_2$ if $[\delta_2] = [\delta_1]\ [[\langle b, b\rangle]]^k$ or $[\delta_1] = [\delta_2]\ [[\langle b, b\rangle]]^k$ for some k. If $\delta_1 = a\delta_1'$, $\delta_2 = b\delta_2'$ and $\delta_1 \approx \delta_2$ then $[a] = [b]$ and $\delta_1' \approx \delta_2'$.

Definition 5.1 Let $C_1 = (W_1, G_1, p_1, \sigma_1', \delta_1)$ and $C_2 = (W_2, G_2, p_2, \sigma_2', \delta_2)$ be two configurations. C_1 and C_2 are strongly equivalent ($C_1 \equiv C_2$) if $[W_1] = [W_2]$, $G_1 = G_2$, $[p_1] = [p_2]$, $[\sigma_1'] = [\sigma_2']$ and $\delta_1 \approx \delta_2$. C_1, C_2 are weakly equivalent ($\tilde{=}$) if only the last four conditions hold. We now have the following.

Lemma 5.1: If $C_1 \equiv C_2$ then $F_s(C_1) \equiv F_s(C_2)$.

Proof: Let $C_1 = (W_1, g_1, q_1, \sigma_1', \delta_1)$ be a configuration. By definition there must exist a configuration $C_1' \in D$, $C_1' = (w_{i-1,\Delta}\sigma_{i-1,\Delta}, G_{i-1}, p_i, \sigma_i', \delta_1)$ where $[W_1] = [w_{i-1,\Delta}\sigma_{i-1,\Delta}]$, $g_1 = G_{i-1}$, $[q_1] = [p_i]$ and $[\sigma_1'] = [\sigma_i']$. Let $W_1 = W'a$. Then $[W'] = [w_{i-1,\Delta}]$, $[a] = [\sigma_{i-1,\Delta}]$. We note that W_1 is a string of symbolsets that may not occur during any computation of $\bar{P}_2$ with proper input. From lemma 2.9 $[f_{W'}] = [f_{w_{i-1,\Delta}}]$. By theorem 4.1, if r_0 is the root of G_{i-1}, then

$$F_{r_0} = [f_{w_{i-1,\Delta}}] = [f_{W'}] \text{ and } D_{r_0} = [\sigma_{i-1,\Delta}] = [a].$$

Consider $F_s(C_1')$. Let $\delta_1 = b\delta_1'$. From lemma 4.2,

$$Q_{i+1} = f_{w_{i-1,\Delta}} * \sigma_{i-1,\Delta}(p_i) * \sigma_i'$$

Using lemma 2.6 to 2.8,

$$[Q_{i+1}] = [f_{w_{i-1,\Delta}}] * [\sigma_{i-1,\Delta}]([p_i]) * [\sigma_i']$$

$$= F_{r_0} * D_{r_0}([p_i]) * [\sigma_i']$$

Since $\sigma_{i+1} = b$,

$$[p_{i+1}] = [Q_{i+1}] * [b] \text{ and } [\sigma_{i+1}'] = [b] * [Q_{i+1}]$$

where $F_S(C_1') = (w_{i,\Delta}\sigma_{i,\Delta}, G_i, p_{i+1}, \sigma_{i+1}', \delta_1')$ and G_i is defined from G_{i-1}, $[p_i]$ and $[\sigma_i']$.

Consider now $F_s(C_1)$ obtained by applying the newsweep function to the configuration C_1. Using the notation introduced above, we can write the output configuration as $\bar{C}_1 = (W_2, g_2, q_2, \sigma_2', \delta_1')$. At the end of the (i,i+1) sweep of C_1, since the entry stateset at i boundary is q_1 the exit stateset at (i+1) boundary is

$$Q_{i+1}' = f_{W_1}(q_1) * \sigma_1'$$

Since $W_1 = W'a$, we have

$$[Q_{i+1}'] = [f_{W'a}]([q_1]) * [\sigma_1']$$

$$= [f_{W'}] * [a]([q_1]) * [\sigma_1'] \text{ by lemma 2.8}$$

$$= F_{r_0} * D_{r_0}([p_i]) * [\sigma_i'] = [Q_{i+1}].$$

Then

$$[q_2] = [Q_{i+1}' * b] = [p_{i+1}] \text{ and}$$

$$[\sigma_2'] = [b * Q_{i+1}'] = [\sigma_{i+1}']$$

Consider now g_2. This is computed from g_1, $[q_1]$ and $[\sigma_1']$. Since $g_1 = G_{i-1}$, $[q_1] = [p_i]$ and $[\sigma_1'] = [\sigma_i']$, $g_2 = G_i$. It remains to show that $[W_2] = [w_{i,\Delta}\ \sigma_{i,\Delta}]$.

Consider now the configuration C_1'. In obtaining $F_s(C_1')$ we perform the (i,i+1) primary sweep on C_1'. The entry stateset at i is p_i and by definition 4.1 and lemma 4.1, the entry stateset at i-1 is p_i'. At the end of this (i-1,i) secondary sweep the string of symbol-sets to left of (i-1) boundary is $w_{i-1,p_i'}$. At the end of the (i,i+1) primary sweep, by lemma 4.1, the symbolset at (i-1) is $sym(i-1,p_i')$. We therefore have

$$w_{i,\Delta} = w_{i-1,p_i'}\ sym(i-1,p_i')$$

and hence $[w_{i,\Delta}] = [w_{i-1,p_i'}]\ [sym(i-1,p_i')]$.

Since $[f_{W'}] = [f_{w_{i-1,\Delta}}]$, $[\sigma_{i-1,\Delta}] = [a]$ and $[p_i] = [q_1]$, from definition 4.1 it follows that $[p_i'] = [q_1']$ and $[sym(i-1,p_i')] = [sym'(i-1,q_1')]$ where sym' refers to the computation on C_1 and sym that on C_1'. From Lemma 4.2,

$$[\sigma_{i,\Delta}] = [\sigma_i'] * F_{r_0} * D_{r_0}([p_i])$$
$$= [\sigma_1'] * F_{r_0} * D_{r_0}([q_1]) = [a_2]$$

where $W_2 = W_2' a_1 a_2$. Also from the discussion above, $a_1 = sym'(i-1,q_1')$ and hence $[a_1] = [sym(i-1,p_i')]$. We have $[w_{i-1,\Delta}] = [W']$. $w_{i-1,p_i'}$ is obtained after a (i-1,i) secondary sweep with entry stateset p_i' on $w_{i-1,\Delta}$ and W_2' is obtained after a (i-1,i) secondary sweep with entry stateset q_1' on W'. Since $[p_i'] = [q_1']$, by lemma 2.9, $[w_{i-1,p_i'}] = [W_2']$. Then,

$$[W_2] = [W_2'][a_1][a_2] = [w_{i-1,p_i'}][\mathrm{sym}(i-1,p_i')][\sigma_{i,\Delta}]$$

$$= [w_{i,\Delta}][\sigma_{i,\Delta}] = [w_{i,\Delta}\sigma_{i\Delta}], \text{ as was to be}$$

shown. $F_s(C_1)$ is therefore a configuration.

The proof above shows that if $C_1 \cong C_1'$ then $F_s(C_1) \cong F_s(C_1')$, $C_1', F_s(C_1') \in D$. Suppose now $C_1 \cong C_2$ and $C_2' \in D$ such that $C_2 \cong C_2'$. Then $F_s(C_2) \cong F_s(C_2')$. Since $\cong$ is an equivalence relation, $C_1 \cong C_1'$ and $C_2 \cong C_2'$ implies $C_1' \cong C_2'$. Since C_1', $C_2' \in D$, $F_s(C_1')$, $F_s(C_2') \in D$ and from proof above, $F_s(C_1') \cong F_s(C_2')$. Since $F_s(C_1') \cong F_s(C_1)$ and $F_s(C_2') \cong F_s(C_2)$, we have $F_s(C_1) \cong F_s(C_2)$ and hence the lemma.

Lemma 5.1 shows that if C is any configuration then so is $F_s(C)$. Therefore the function F_s: $\mathcal{C}$ --> $\mathcal{C}$ is total on $\mathcal{C}$. Let F: $\mathcal{C}$ --> null(Y) be the function defined by $\bar{P}_2$. For $C \in \mathcal{C}$, F(C) is the output configuration if any, when A is initialized to the configuration C and the while do loop of $\bar{P}_2$ is executed on this array. If now, C is any configuration then there exists a $C' \in D$ such that $C \cong C'$. By lemma 5.1, for all k, $F_s^k(C) \cong F_s^k(C')$. Since $C' \in D$, for some value k, $\bar{P}_2$ must terminate with a halt configuration and therefore so must C. Therefore F: $\mathcal{C}$ --> null(Y) is total on $\mathcal{C}$.

Lemma 5.2: If $C_1 \cong C_2$ then $F_s(C_1) \cong F_s(C_2)$.

Proof: Let $C_1 = (W_1, G_1, p_1, \sigma_1', \delta_1)$ and $C_2 = (W_2, G_2, p_2, \sigma_2', \delta_2)$ be two configurations. Since $C_1 \cong C_2$, we have

$$G_1 = G_2, [p_1] = [p_2], [\sigma_1'] = [\sigma_2'] \text{ and } [\delta_1] \approx [\delta_2] .$$

Further, since C_1 is a configuration, for some i and input W,

$[W_1] = [W_{i-1,\Lambda}\sigma_{i-1,\Lambda}]$, $G_1 = G_{i-1}$, $[p_1] = [p_i]$ and $[\sigma_1'] = [\sigma_i']$.

Similarly, since C_2 is a configuration, for some j and input $\bar{W}$,

$[W_2] = [\bar{W}_{j-1,\Lambda}\bar{\sigma}_{j-1,\Lambda}]$, $G_2 = \bar{G}_{j-1}$, $[p_2] = [\bar{p}_j]$ and $[\sigma_2'] = [\bar{\sigma}_j']$.

Let $C_1' = (W_{i-1,\mathcal{L}}, \sigma_{i-1,\mathcal{L}}, G_{i-1}, p_i, \sigma_i', \delta_1)$ and $C_2' = (\bar{W}_{j-1,\mathcal{L}}, \bar{\sigma}_{j-1,\mathcal{L}}, \bar{G}_{j-1}, \bar{p}_j, \sigma_j', \delta_2)$.
Let $\delta_1 = a\delta_1'$ and $\delta_2 = b\delta_2'$. Since $\delta_1 \approx \delta_2$, [a] = [b] and $\delta_1' \propto \delta_2'$. Then

$F_s(C_1') = (W_{i,\mathcal{L}}, \sigma_{i,\mathcal{L}}, G_i, p_{i+1}, \sigma_{i+1}', \delta_1')$ and

$F_s(C_2') = (\bar{W}_{j,\mathcal{L}}, \bar{\sigma}_{j,\mathcal{L}}, \bar{G}_j, \bar{p}_{j+1}, \bar{\sigma}_{j+1}', \delta_2')$.

Let r_1 be the root of G_{i-1} and r_2 that of $\bar{G}_{j-1}$. Since $G_1 = G_2$, $F_{r_1} = F_{r_2}$ and $D_{r_1} = D_{r_2}$. By Theorem 4.1,

$F_{r_1} = [f_{W_{i-1,\mathcal{L}}}]$, $F_{r_2} = [f_{\bar{W}_{j-1,\mathcal{L}}}]$, $D_{r_1} = [\sigma_{i-1,\mathcal{L}}]$

and $D_{r_2} = [\bar{\sigma}_{j-1,\mathcal{L}}]$.

Let $Q_{i+1}(\bar{Q}_{j+1})$ be the exit stateset at i+1 (j+1) boundary, at the end of (i, i+1) ((j, j+1)) sweep of $C_1'(C_2')$. By lemma 4.2 and 2.6, since $\sigma_{i+1} = a$, $\bar{\sigma}_{j+1} = b$, we have,

$$[Q_{i+1}] = [f_{W_{i-1,\mathcal{L}}}] * [\sigma_{i-1,\mathcal{L}}]\ ([p_i]) * [\sigma_i']$$
$$= [f_{\bar{W}_{j-1,\mathcal{L}}}] * [\bar{\sigma}_{j-1,\mathcal{L}}]\ ([\bar{p}_j]) * [\bar{\sigma}_j']$$
$$= [\bar{Q}_{j+1}].$$

Then

$$[p_{i+1}] = [Q_{i+1}] * [a] = [\bar{Q}_{j+1}] * [b] = [\bar{p}_{j+1}]$$

and

$$[\sigma_{i+1}'] = [a] * [Q_{i+1}] = [a] * [\bar{Q}_{j+1}] = [\bar{\sigma}_{j+1}'] .$$

From section 3, G_i is computed from $G_{i-1}, [p_i], [\sigma_i']$ and G_j is computed from $\bar{G}_{j-1}, [\bar{p}_j], [\bar{\sigma}_j']$. Since $G_{i-1} = \bar{G}_{j-1}$, $[p_i] = [\bar{p}_j]$, $[\sigma_i'] = [\bar{\sigma}_j']$, it follows that $G_i = \bar{G}_j$. Therefore $F_s(C_1') \tilde{\equiv} F_s(C_2')$.
By definition of $\equiv$, $C_1 \equiv C_1'$ and $C_2 \equiv C_2'$. By lemma 5.1, $F_s(C_1) \equiv F_s(C_1')$ and $F_s(C_2) \equiv F_s(C_2')$, and therefore by the definition of $\tilde{\equiv}$,

$F_s(C_1) \tilde{\equiv} F_s(C_1')$ and $F_s(C_2) \tilde{\equiv} F_s(C_2')$.

Since $\tilde{\equiv}$ is an equivalence relation, we have

$F_s(C_1) \tilde{\equiv} F_s(C_2)$ as was to be shown.

From lemma 5.2 it follows that if C_1, C_2 are any two configurations, $C_1 \tilde{\equiv} C_2$, then for all k, $F^k(C_1) \tilde{\equiv} F^k(C_2)$. If we choose a large k, both $F_s^k(C_1)$ and will be halt-configura-

tions, since F is total on C.

From the definitions of equivalence of statesets and halt configurations, it follows that both $F_s^k(C_1)$ and $F_s^k(C_2)$ will be accepting configurations or both will be rejecting configurations.

Let w be a given input of length n. Let C_{i+1} denote the configuration at the beginning of the ith iteration of the while-do loop of $\bar{P}_2$, $i \geq 1$. From the initialization of $\bar{P}_2$ and the definition of the function newsweep, C_{i+1} is the configuration at the end of the (i,i+1) primary sweep of M.

Let $C_i = (\mathcal{W}_i, G_i, p_i, \sigma_i', \delta_i)$.

From the definition of the space bound of M, for all $i \geq S(n)+1$,

$\sigma_i' = \{\langle \not b, z\rangle\}$ and δ_i consists of $\{\langle \not b, \not b\rangle\}$ symbolsets only.

Consider now a lexicographic ordering on strings of symbolsets. The least member of a subset of symbolset strings is therefore uniquely defined.

Let $T = \{\langle \mathcal{W}, G, p, \sigma', \delta\rangle \in \mathcal{C} \mid \sigma' = \{\langle \not b, z\rangle\}, \delta = \{\langle \not b, \not b\rangle\}$, and for given G, p, σ', δ w is the least string such that $\langle \mathcal{W}, G, p, \sigma', \delta\rangle \in \mathcal{C}\}$. Since only finitely many values are possible for G,p, T is a finite set. Further since $\{\langle \not b, \not b\rangle\} \approx \{\langle \not b, \not b\rangle\}^k$ for all k, there must exist unique $a \in T$ such that $C_{S(n)+1} \cong a$. We could execute $\bar{P}_2$ on each element of T and decide whether it is accepted or rejeted. Then to decide if W is accepted by $\bar{P}_2$, we execute $\bar{P}_2$ to obtain $C_{S(n)+1}$. This requires $\theta(S(n)^2)$ steps. We obtain $a = C_{S(n)+1}$ by searching the finite set T and then decide if W is accepted or rejected by $\bar{P}_2$.

We therefore have the following theorem. We assume the function value S(n) can be computed within $\theta(S(n)^2)$ steps.

Theorem 5.1: Let L be a language accepted by a S-machine within space S(n). Then there exists a deterministic algorithm that accepts L

within $\theta(S(n)^2)$ steps.

From remarks at the end of section 1, we have

Corollary 5.1: Let L be a language accepted by a nondeterministic Turing machine with space bound $S(n)$. Then there exists a deterministic algorithm that accepts L within $\theta((S(n)^2)$ steps.

Let PSPACE, PTIME (NPSPACE, NPTIME) be the classes of languages accepted by a deterministic (nondeterministic) Turing machine with a polynomially bounded space, time. We then have

Corollary 5.2: PSPACE = PTIME = NPSPACE = NPTIME.

Proof: From Theorem 5.1 and the relation between S-machine and deterministic Turing machine space bound, PSPACE $\subseteq$ PTIME. Since PTIME $\subseteq$ PSPACE by definition, we have PTIME = PSPACE. Since NPSPACE = PSPACE, we have PTIME = PSPACE = NPSPACE. Also, NPTIME $\subseteq$ NPSPACE = PTIME and PTIME $\subseteq$ NPTIME. Thus, PTIME = NPTIME = PSPACE = NPSPACE as was to be shown.

This solves a problem due to Karp[7].

We now proceed to show that $\bar{P}_2$ is uniform over a linear data domain defined by the configurations of $\bar{P}_2$. We use symbolism and notations introduced in [1], [2].

Let C be any configuration. The support set of C, S(C) is the set of all configurations that could yield a configuration equivalent to C after some finite number of applications of the sweep function i.e.

$$S(C) = \{C' \mid \text{for some } k,\ F_s^k(C') \cong C\}.$$

$C \in S(C)$ by definition.

Two configurations C_1, C_2 are compatible if $C_1 \in S(C_2)$. By lemma 5.2 above, if C_1, C_2 are compatible then so are C_1 and $F_S^k(C_2)$ for any

k. We define a binary operator $\circledast$ on $\mathcal{C}$. For $C_1, C_2 \in \mathcal{C}$,

$C_1 \circledast C_2 = F_s^k(C_1)$ if C_1, C_2 are compatible and $F_s^k(C_1) \cong C_2$.

$= C_1$ otherwise.

Then, the composition of any two configurations yields a configuration.

Let B_Y be the set of basis structures, defined as follows,

$B_Y = \{(w,G,p,\sigma,\delta) \in \mid w$ is the least string for given G, p, σ, $\delta\}$ $\cup$ all 2-configurations, where 'least' refers to the standard lexicographic ordering on strings of symbol-sets. It follows that for every $(W,G,p,\sigma,\delta) \in \mathcal{C}$, there is a unique element $(\omega,G,p,\sigma,\delta] \in B_Y$.

If $X_1, X_2 \subseteq \mathcal{C}$ then we define $X_1 \circledast X_2 = \bigcup\{C_1 \circledast C_2 \mid C_1 \in X_1, C_2 \in X_2\}$.

Let $B_{Y,1} = B_Y$ and $B_{Y,1} = B_{Y,i-1} * B_Y$ for $i > 1$. Let $clos(B_Y) = \bigcup_i B_{Y,i}$.

We have the following lemma.

Lemma 5.3: $clos\ (B_Y) \subseteq \mathcal{C}$.

Proof: From definition above, $B_Y \subseteq \mathcal{C}$. Assume as induction bypothesis that $B_{Y,i} \subseteq \mathcal{C}$. Then

$B_{Y,i+1} = B_{Y,i} \circledast B_Y$. Let $X \in B_{Y,i}$; $a \in B_Y$.

If X, a are compatible, then $X \circledast a = F_s^k(X)$ for some k. By lemma 5.1, since $X \in \mathcal{C}$, $F_s^k(X) = X \circledast a \in \mathcal{C}$. If X, a are not compatible, then $X \circledast$

$a = X \varepsilon \mathcal{C}$. Then, $B_{Y,i+1} \subseteq$ and hence $clos(B_Y) \subseteq \mathcal{C}$.

A <u>n-configuration</u> is any configuration in which the first component is a string of symbolsets of length n.

<u>Lemma 5.4:</u> $D \subseteq clos(B_Y)$

<u>Proof:</u> By definition every 2-configuration of D is in B_Y and hence in $clos(B_Y)$.

Let $C_n = (w,G,p,\sigma',\delta)$ be a n-configuration in D. Assume as induction hypothesis that $C_n \varepsilon clos(B_Y)$.

Let $F_s(C_n) = (w_1,G_1,p_1,\sigma_1',\delta_1)$. By definition there exists a configuration $a = (w',G_1,p_1,\sigma_1',\delta_1) \varepsilon B_Y$. Further C_n, a are compatible since $F_s(C_n) \cong a$. Then $C_n \circledast a = F_s(C_n) \varepsilon clos(B_Y)$. $F_s(C_n)$ is an (n+1) configuraiton. Further every (n+1) configuration in D must be obtained from a n-configuration in D . Hence the lemma.

<u>Theorem 5.2:</u> Let $X \varepsilon clos(B_Y)$, $a \varepsilon B_Y$. If X,a are compatible then

$$F(X * a) \cong X \circledast F(a).$$

<u>Proof:</u> Since $X \varepsilon clos(B_Y)$ implies $X \varepsilon \mathcal{C}$, $\bar{P}_2$ must terminate on X. Then $F(X) = F_s^k(X)$ for some $k \geq 0$. Now since X,a are compatible, $X \circledast a \cong a$ and by Lemma 5.2,

$$F_s^i(X \circledast a) \cong F_s^i(a) \text{ for all } i.$$

Since $X \circledast a \varepsilon clos(B_Y)$, $F(X \circledast a) = F_s^k(X \circledast a) = F_s^k(a)$ for some k. Let $F(a) = F_s^m(a)$, for some m. Since X,a are compatible, so are X and $F_s^m(a)$. Hence $X \circledast F(a) = X \circledast F_s^m(a) \cong F_s^m(a)$. Let $k \geq m$. Both $F_s^m(a)$ and $F_s^k(a)$ are halt configurations. If we choose both m and k sufficiently large, so that the δ component consists of $\langle \not b, \not b \rangle$ only, then from the definition of halt configurations, $F_s^{m+j}(a) \cong F_s^m(a)$. Choosing $j = k - m$ yields $F_s^k(a) \cong F_s^m(a)$ and then

$$F(X \circledast a) \cong F_s^k(a) \cong X * F(a).$$

Theorem 5.2 shows $\bar{P}_2$ is uniform over $clos(B_Y)$. We show that $clos(B_Y)$ is a linear data domain using notatons of [1]. We impose a lexicographic order on the set of all configurations; so that a set of n-configurations has a unique least n-configuration. Let $X = (w,G,p,\sigma,\delta) \in clos(B_Y)\text{-null}(Y)$ be a n-configuration. We define head(X) to be the least c-configuration $(w',G,p,\sigma,\delta) \in B_Y$ and tail(X) to be the least (n-1) configuration X' such that $F_s(X') = X$. tail(X) $= X$ if $X \in B_Y$. We note that c is a constant independent of n, but defined by G, p, σ, δ. $clos(B_Y)$ with the head and tail functions form a linear data domain. If we let Y denote a configuration, we can rewrite $\bar{P}_2$ as **while** $Y \notin$ null(Y) **do** $Y := F_s(Y)$; then we have from theorem 5.2,

$a \in B_Y$ and $Y = a[\bar{P}_2]\ Y = F(a)$ implies

$$Y = Y_1 \text{ and } head(Y_1) = a[Y := Y_1;\ \underline{while}\ Y \notin null(Y)\ \underline{do}\ Y := F_S(Y)]\ Y \cong tail(Y_1)*F(a).$$

From the proof of Theorem 5.2, $tail(Y_1)$ and F(a) are compatible. By the definition of $\circledast$,

$$tail(Y_1) \circledast F(a) \cong F(a)$$

and therefore $\bar{P}_2$ either accepts or rejects both configurations Y and a. The algorithm described prior to theorem 5.1 uses this fact.

We have shown that the notion of uniformity of program behavior on a data domain is a useful one and can lead to efficient algorithms. In general, search for efficient algorithms must take into account the programming process and the structure of the data domain on which it is intended to operate.

References

1. Basu, S. K., 'A note on synthesis of inductive assertions', IEEE Trans on S.E. Vol. SE-6 No. 1 (Jan. 1980).

2. Basu, S.K., 'Development of iterative proggrams from function specifications', IEEE Trans on S.E. Vol. SE-6 No. 2 (Mar. 1980).

3. Cook, S.A., 'An overview of computational complexity', Comm. ACM, 26,6 (June 1983), 400-407.

4. Dahl, O.J., Dijkstra, E.W., and Hoare, C.A.R., Structured Programming, Academic Press, New York (1972).

5. Floyd, R.W., 'Assigning meanings to programs', Proc. Symp. Appl. Math. Amer. Math. Soc. 19 (1967), 19-32.

6. Hopcroft, J.E., Paul W., and Valiant, L., 'On time versus space', J. Assoc. Comm. Mach., 24 (1977), pp. 332-337.

7. Karp, R.M., 'Reducibility among combinatorial problems', Complexity of Computer Coputations, R.E. Miller and J.W. Thatcher, eds., Plenum Press, New York (1972), 85-104.

8. Mis ra, J. and Basu, S.K., 'Some classes of naturally provable programs', Proc. 2nd Int. Conf. on Software Eng., San Francisco (Oct., 1976).

9. Paul, W. and Reischuk, R., 'On time versus space II', Jour. Comp. System Sci., 22 (1981), pp. 312-327.

10. Rabin, M.O., 'Degree of difficulty of computing a function and a partial ordering of recursive sets', Tech. report No. 1, O.N.R., Jerusalem (1960).

11. Savitch, W.J. 'Relationships between nondeterministic and deterministic tape complexities', Jour.Comp and Syst. Sciences.4:2,pp177-192 (1970).

Acknowledgement: The author is greatly indebted to Albert R. Meyer for having introduced him to the problems of time and space complexity in the late 60's. Recently, he pointed out authors erroneous formulation of the main result in an earlier version of this paper. Author would also like to thank Deb Heckens and her colleagues at the Department of Computer Science, University of Nebraska, Lincoln; for preperation of this manuscript.