On A Class of Uniform Algorithms

Sanat K. Basu

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 $\mathcal{O}(S(n)^2)$ steps where $S(n)$ is the spacebound of the S-machine. Corresponding results in terms of the standard Turing machine model are then derived.

1 Introduction

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 specification 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. [] 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 [] 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 [] 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 studied in [], [].

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 $\mathcal{O}(S(n)^2)$ steps. This solves a problem of Karp [].

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

The relation between time and space complexities have been studied extensively. Hopcroft et al. [] 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. []

The contents of this paper are organized into five sections. In section 2 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 3 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 4 we define a context graph. The node labels of the context graph contain the equivalence class of the crossing function of a 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 5 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 6 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 $\mathcal{O}(S(n)^2)$ steps where $S(n)$ is the space bound of the S-machine.

2 A nondeterministic sweep machine (S-machine)

In this section we introduce a nondeterministic Turing machine model called the sweep machine of 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 is 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 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 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 $k^2 + 2k$. 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 $M = \langle Q, \Sigma, \Gamma, \delta, \{q_0\}, H\rangle$. where Q is a finite set of *states*, and $Q = Q^L \bigcup Q^R \bigcup P^R \bigcup \{q_0\}$. Q^L is a set of *left moving 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.

Σ is a finite set of input symbols, excluding the symbols $\not\flat$, $\not\phi$, z. Γ is a finite set of *tape symbols*. Each element of Γ is an ordered pair $< x, y >$ of symbols:

$$\Gamma = \Big\{< x, y > | x \in \Sigma \bigcup \{\not\flat\}, y \in \{\not\flat, z\}\Big\} \bigcup \{< \not\phi, \not\phi >\}$$

$H \subseteq Q$ 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 $Q^R \bigcup P^R$. L denotes a left move and R is a right move.

(i) For each $q_0 \in \{q_0\}, < x, z > \in \Gamma$ there exists $< x_1, z >, \ldots, < x_k, z > \in \Gamma$ and $q^L_{i_1}, \ldots, q^L_{i_k} \in Q_L$ such that $(q_0, < x, z >) \mapsto \big\{(q^L_{i_1}, < x_1, z >), \ldots, (q^L_{i_k}, < x_k, z >)\big\}$ is in δ.

(ii) For each $q^L \in Q^L, < x, \not\flat > \in \Gamma$, there exists $< x_1, \not\flat >, \ldots, < x_k, \not\flat > \in \Gamma$ and $q^L_{i_1}, \ldots, q^L_{i_k} \in Q^L$ such that $(q^L, < x, \not\flat >) \mapsto \big\{(q^L_{i_1}, < x_1, \not\flat >), \ldots, (q^L_{i_k}, < x_l, \not\flat >)\big\}$ is in δ.

(iii) For each $q^L \in Q^L$, there exists $q^R \in Q^R$ such that $(q^L, < \not\phi, \not\phi >) \mapsto (q^R, < \not\phi, \not\phi >)$ is in δ.

(iv) For each $q^R \in Q^R, < x, \not\flat > \in \Gamma$, there exists $q^R_{i_1}, \ldots, q^R_{i_k} \in Q^R, < x_1, \not\flat >, \ldots, < x_k, \not\flat > \in \Gamma$ such that $(q^R, < x, \not\flat >) \mapsto \big\{(q^R_{i_1}, < x_1, \not\flat >) \ldots (q^R_{i_k}, < x_k, \not\flat >)\big\}$ is in δ.

(v) For each $q^R \in Q^R, < x, z > \in \Gamma$, there exists $p^R_{i_1}, \ldots, p^R_{i_k} \in P^R$ and $< x_1 \not\flat >, \ldots, < x_k, \not\flat > \in \Gamma$ such that $(q^R, < x, z >) \mapsto \big\{p^R_{i_1}, < x_1, \not\flat >), \ldots, (p^R_{i_k}, < x_k, \not\flat >)\big\}$ is in δ.

(vi) For each $p^R \in P^R$ and $< x, \not\flat > \in \Gamma$, there exists $q^L_{i_1}, \ldots, q^L_{i_k} \in Q^L$ and $< x_1, z >, \ldots, < x_k, z > \in \Gamma$ such that $(p^R_i, < x, \not\flat >) \longrightarrow \big\{(q^L_{i_1}, < x_1, z >), \ldots, (q^L_{i_k}, < x_k, z >)\big\}$ 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 $\not\phi$ 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 $\not\flat$ plays a special role. We assume that M does not enter a halt state on scanning a $< \not\flat, \not\flat >$ or $< \not\flat, z >$ tape symbol. The machine M performs all of 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

d) move one cell to the left or right.

According to the description above, the direction of move in d) is known.

There is a lot to unpack here, and the original paper is pretty terse. So let's start by looking at the various pieces of the formal definition.

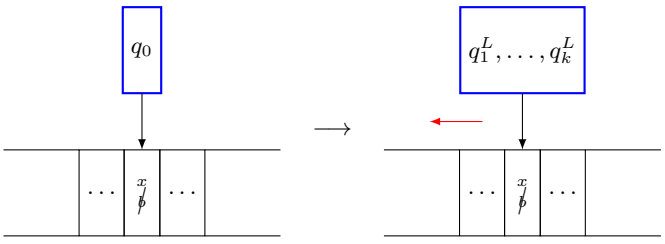
First, the structure of each cell in the tape consists of a set of tape symbols. I.e., the contents of cell $c(i) \in \Gamma^*$, the power set of tape symbols. We will see that only a restricted set of these tape symbols is permissible in a moment.

The state of the S-machine will be some $q \in Q$. This state can be one of four special types of state:

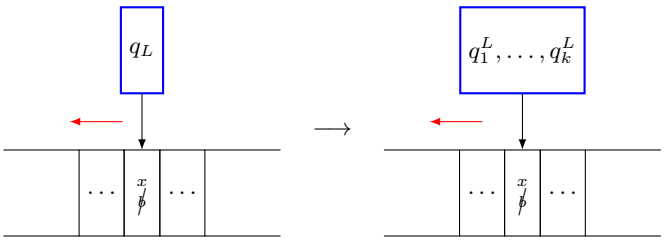
- $q \in Q^L$: this is a left moving state, meaning that when the S-machine advances, it will move left.
- $q \in q_0$: these are start states. The S-machine will start in one of these states. Per (i), the machine will start moving left, provided that the marker for the start state is blank.
- $q \in Q^R$: this is a right moving state, meaning that when the S-machine advances, it will move right.
- $p \in P^R$: this is also a set of right moving states, but they have some additional properties. From (v), it follows that if the S-machine is in the state $p \in P^R$, and it reads a cell consisting of an input symbol x, and a blank marker $\not\phi$, then it will write the z symbol in the marker position, and transition to a left moving state.

Now, the set of properties laid out above suggest that not all tape symbol sets are actually permissible. In particular, the state of the marker part of the tape symbol must be synchronized. Let's revisit the rules in a bit more detail.

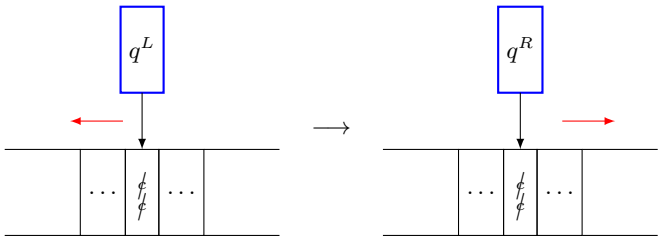
(i) If the S-machine is in an initial state q_0, the tape symbol active has some symbol x and a blank marker z, then the machine will leave the blank marker alone, and enter a set of left moving state.



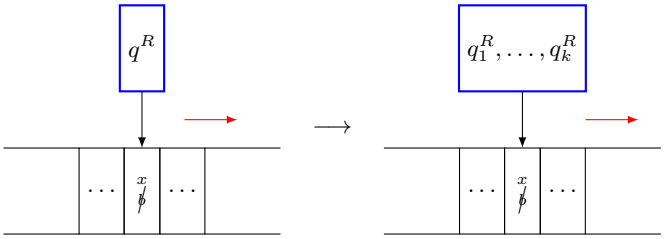
(ii) If the S-machine is in a left moving state, and the marker is $\not\phi$, then if will remain in a left moving set of states. In other words, if the cell is marked with $\not\phi$, the S-machine is guaranteed to move left, and leave the marker as $\not\phi$.



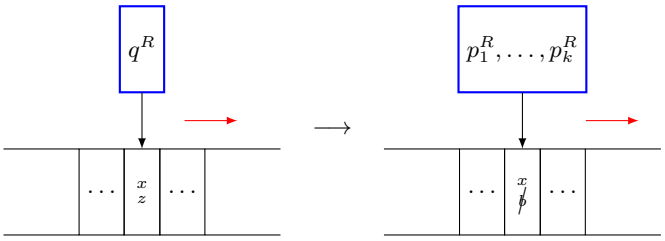
(iii) If the S-machine is in a left moving state, and it encounters $< \not\epsilon, \not\epsilon >$, it will enter a right moving state, and leave the $\not\epsilon$ marker in place.



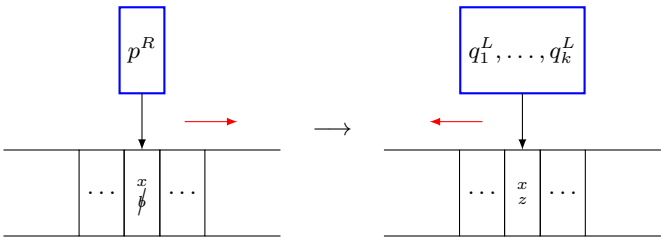
(iv) If the S-machine is in a right moving state, and the marker is $\not\flat$, it will remain in a right moving set of states.



(v) If the S-machine is in a right moving state, and the marker is z (blank), then it will enter a state in P^R and mark the cell as $\not\flat$.



(vi) If the s-machine is in state p^R $in P^R$, and the cell is marked with $\not\flat$, then it will mark the cell as z (blank) and enter a left moving state.



While stated above, it is important to realize that no matter how the machine operates, all the set of states are synchronized with respect to their motion and the marker state of the cell. So let us define first an augmented set of tape symbols that includes $\not\phi$:

$$\Sigma^+ = \Sigma \bigcup \not\phi$$

and let us also define a set of marker symbols

$$Y = \{\not\phi, z\}$$

So it is more accurate to describe the contents of the tape cell as

$$\Gamma^* = \left\{ \{< x_1, y >, \ldots, < x_k, y >\} | < x_i, y > \in \Sigma^+ \times Y \bigcup < \not\epsilon, \not\epsilon > \right\}$$

In words, this means each cell contains a 2-tuple, consisting of a set x_i and a marker y, where

- The set x_i comes from either the input symbols or $\not\phi$.
- The marker y is either $\not\phi$ or z.
- A special case symbol $< \not\epsilon, \not\epsilon >$, can appear, as can $< \not\epsilon \ldots \not\epsilon, \not\epsilon >$.

No other combination can appear. It may be helpful to think of $\not\phi$ as a `NULL` symbol.

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 the set of states and B a set of tape symbols of M. We introduce the following operator $*$ called *product*.

$$A * B = \{q | \exists x \in \Gamma, q_1 \in A, a_1 \in B \ni (q, x) \in \delta(q_1, a_1)\}$$

$$B * A = \{x | \exists q \in Q, q_1 \in A, a_1 \in B \ni (q_1, x) \in \delta(q_1, a_1)\}.$$

Typo? Should be

$$A * B = \{q | \exists x \in \Gamma, q_1 \in A, a_1 \in B \ni (q, x) \in \delta(q_1, a_1)\}$$
$$B * A = \{x | \exists q \in Q, q_1 \in A, a_1 \in B \ni (q, x) \in \delta(q_1, a_1)\}.$$

This $A * B$ 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 scanned cell a member of B.

Henceforth, we shall refer to a set of states of M as a *stateset* and a set of tape symbols of M as a *symbolset* 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.

Lemma 1. *For any statesets q, q_1, q_2 and symbolsets a, a_1, a_2*

$$q * (a_1 + a_2) = q * a_1 + q * a_2$$
$$(q_1 + q_2) * a = q_1 * a + q_2 * a$$
$$(a_1 + a_2) * q = a_1 * q + a_2 * q$$
$$a * (q_1 + q_2) = a * q_1 + a * q_2$$

Proof. From the definition above,

$$\begin{aligned}
p \in q * (a_1 + a_2) &\iff \exists p' \in q, x \in \Gamma, b \in a_1 + a_2 \ni (p, x) \in \delta(p', b) \\
&\iff \exists p' \in q, x \in \Gamma, b \in a_1 \ni (p, x) \in \delta(p', b) \vee \exists p' \in q, x \in \Gamma, b \in a_2 \ni (p, x) \in \delta(p', b) \\
&\iff p \in q * a_1 \vee p \in q * a_2 \\
&\iff p \in q * a_1 + q * a_2
\end{aligned}$$

□

The original proof of Lemma 1 is somewhat terse. However, the concept is fairly simple. From the definition of the product operator $*$, we have that $q * (a_1 + a_2)$ is the set of all states reachable from q, with a tape cell containing either a_1 or a_2. If p is reachable from q given a symbol from a_1 or a_2, then there must be an initial state $p' \in q$ such that we transition to p with the symbol b drawn from $a_1 + a_2$. We can then consider if the symbol b comes from a_1 (in which case) $p \in q * a_1$, or if the symbol b comes from a_2, in

which case $p \in q * a_2$. If the symbol is common to both a_1 and a_2, the result is unaffected. As a result, we can claim that $p \in q * a_1 + q * a_2$.

A stateset q is *left moving* if $q \subseteq Q^L$, *right moving* if $q \subseteq Q^R$ and halting if $q \cap H \neq \emptyset$.

Should this not be $q \subseteq Q^R \cup P^R$? The subset P^R contains right moving states as well.

A symbolset a is *marked* 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 computation 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. $q \in Q_H$ is *accepting* if $q \cap H_A \neq \emptyset$ and *rejecting* if $q \cap H_A = \emptyset$.

We shall assume that there exist symbols $a_1, a_2 \in \Gamma$ such that

$$q \in H_A \implies \forall x \in \Gamma, \exists q' \in H_A \ni (q', a_1) \in \delta(q, x)$$

and

$$q \in H_R \implies \forall x \in \Gamma, \exists q' \in H_R \ni (q', a_2) \in \delta(q, x).$$

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

$$\Sigma_A = \{a \subseteq \Gamma | a_1 \in a\}$$
$$\Sigma_R = \{a \subseteq \Gamma | a_2 \in a\}$$
$$Q_A = \{q \subseteq Q | q \cap H_A \neq \emptyset\}$$
$$Q_R = \{q \subseteq Q | q \cap H \neq \emptyset \wedge q \cap H \neq \emptyset\}$$
$$\{Q\} = \{q | q \subseteq Q^L \vee q \subseteq Q^R\}.$$

3 Equivalence of statesets and symbol sets

4 The left context graph

5 Properties of G_i

6 Uniformity of $\bar{P}_2$

7 Appendix

7.1 Proofs for Lemma 1

The proofs of the various parts of Lemma 1 were left out of the original paper as they are straightforward. In the interest of completion, we include those proofs in this appendix subsection. First, we prove that for any statesets q, q_1, q_2 and symbolsets a, a_1, a_2

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

Proof.

$$\begin{aligned} p \in (q_1 + q_2) * a &\iff \exists p' \in (q_1 + q_2), x \in \Gamma \ni (p, x) \in \delta(p', a) \\ &\iff p' \in q_1, x \in \Gamma \ni (p, x) \in \delta(p', a) \vee p' \in q_2, x \in \Gamma \ni (p, x) \in \delta(p', a) \\ &\iff p \in q_1 * a \vee p \in q_2 * a \\ &\iff p \in q_1 * a + q_2 * a \end{aligned}$$

□

Next, we prove

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

Proof.

$$\begin{aligned} b \in (a_1 + a_2) * q &\iff \exists p' \in Q, p \in q, x \in (a_1 + a_2) \ni (p', b) \in \delta(p, x) \\ &\iff \exists p' \in Q, p \in Q, x \in a_1 \ni (p', b) \in \delta(p, x) \vee \exists p' \in Q, p \in Q, x \in a_2 \ni (p', b) \in \delta(p, x) \\ &\iff b \in a_1 * q \vee b \in a_2 * q \\ &\iff b \in a_1 * q + a_2 * q \end{aligned}$$

□

Finally, we prove

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

Proof.

$$\begin{aligned} b \in a * (q_1 + q_2) &\iff \exists p' \in Q, p \in (q_1 + q_2), x \in a \ni (p', b) \in \delta(p, x) \\ &\iff \exists p' \in Q, p \in q_1, x \in q \ni (p', b) \in \delta(p, x) \vee \exists p' \in Q, p \in q_2, x \in q \ni (p', b) \in \delta(p, x) \\ &\iff b \in a * q_1 \vee b \in a * q_2 \\ &\iff b \in a * q_1 + a * q_2 \end{aligned}$$

□