Acceptance Trees

M. HENNESSY

University of Edinburgh, Edinburgh, Scotland

Abstract. A simple model, AT, for nondeterministic machines is presented which is based on certain types of trees. A set of operations, Σ, is defined over AT and it is shown to be completely characterized by a set of inequations over Σ. AT is used to define the denotational semantics of a language for defining nondeterministic machines. The significance of the model is demonstrated by showing that this semantics reflects an intuitive operational semantics of machines based on the idea that machines should only be differentiated if there is some experiment that differentiates between them.

Categories and Subject Descriptors: F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*logics of programs*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*algebraic approaches to semantics*; *denotational semantics*

General Terms: Theory

Additional Key Words and Phrases: Axioms, nondeterministic machines, testing equivalence, trees

1. *Introduction*

There is a well-developed and very successful theory of (nondeterministic) machines based on their ability to accept strings. For the most part, this has been developed by formal language theorists who view machines as formal mechanisms for recognizing languages. A large body of work exists relating various methods of generating languages, in the form of grammars, with methods of recognizing languages, in the form of machines [20]. Indeed, it can be argued that this connection with grammars has greatly influenced our traditional view of machines.

Consider the three finite automata, M_1, M_2, M_3, in Figure 1. We use the notation of [20] so that q_0 is the initial and final state in each machine. Each accept the same language, which can be described as the language represented by the regular expression $(a(b + c))^*$. Consequently, the machines are deemed to be equivalent. To be more precise, the semantic domain for interpreting these machines consists of sets of strings over the input alphabet. One associates with each machine an object of that domain, namely, the set of strings that it accepts. Note that this semantic mapping, from machines to domain, is operational in nature; to find out the semantic object associated with a particular machine, one must know how to run the machine. Then one says that two machines are equivalent if they denote the same semantic object. The three machines M_1, M_2, M_3 are equivalent since they denote the same set of strings.

Author's address: Department of Computer Science, University of Edinburgh, James Clark Maxwell Building, The Kings Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland.

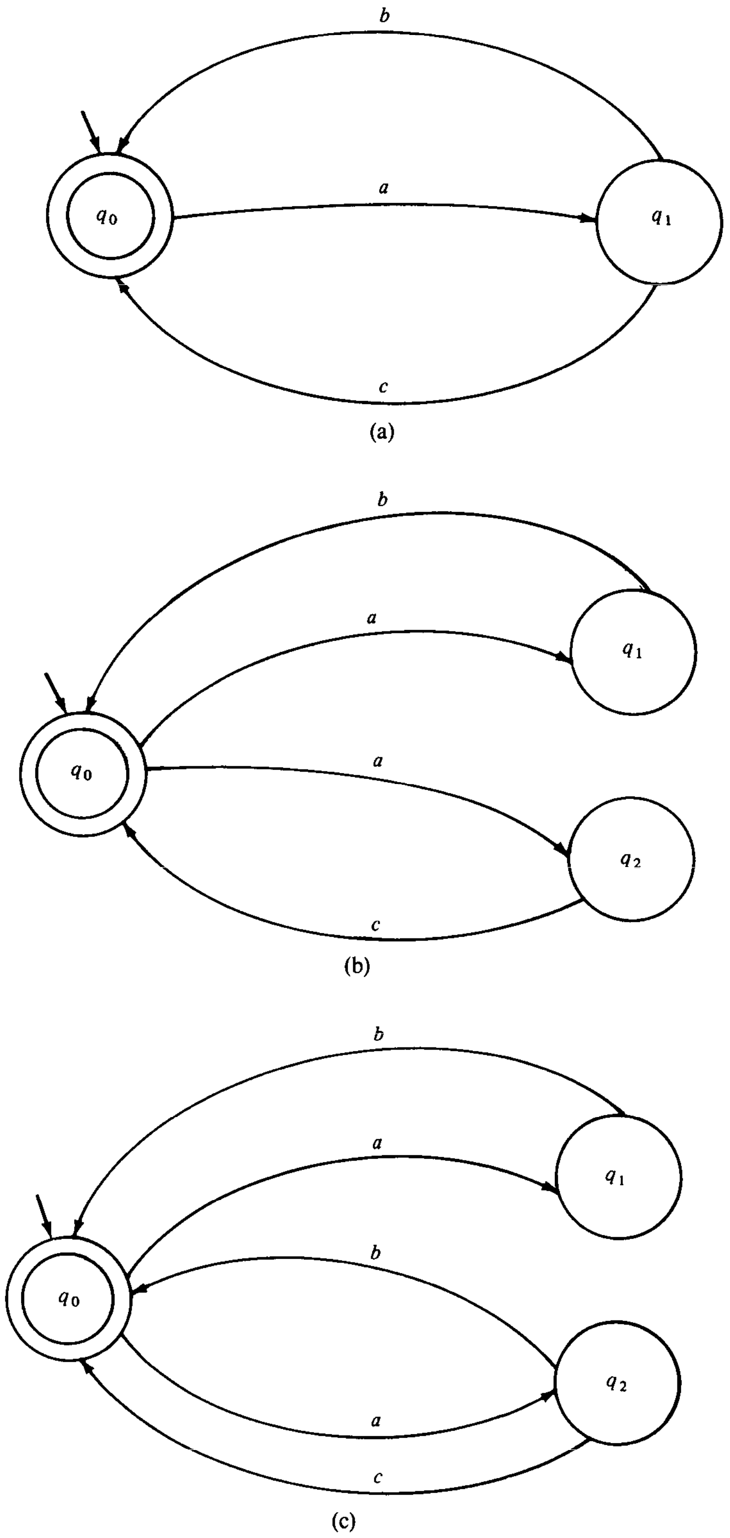


FIG. 1. Three finite automata. (a) Machine M_1. (b) Machine M_2. (c) Machine M_3.

However, there are many ways in which they are not equivalent. For example, M_1 and M_2 have a different number of states, and M_2 and M_3 may traverse different sequences of states when accepting certain strings, such as *ab*. These are *internal* differences and should not necessarily imply inequivalence. Inequivalence should only come about from behavioral differences that have external manifestations. However, it can be argued that the three machines M_1, M_2, M_3 have differences that are detectable by an external observer.

Consider a user of these machines. He "uses" it by giving some string as input and waiting for an answer—either YES, the string is accepted, or NO, the string is not accepted. For example, if the user proffers the string *ab* to the machine M_1, then, since it starts in state q_0, it

—accepts a and moves to state q_1,
—accepts b and moves to state q_0.

Since the input has been exhausted and q_0 is an accept state, the string *ab* is said to be accepted. Note that there is only one possible sequence of states that M_1 can go through when accepting *ab*. On the other hand, M_3, when presented with *ab*, can either

—accept a and move to state q_1,
—accept b and move to state q_0;

or

—accept a and move to state q_2,
—accept b and move to state q_0.

However, both possibilities lead to an accepting state and therefore, like M_1, M_3 always accepts the string *ab*, although its response is nondeterministic.

When M_2 is presented with *ab* it can

—accept a and move to state q_1,
—accept b and move to state q_0;

or

—accept a and move to state q_2.

In state q_2, b cannot be accepted, so this possible sequence of events leads to a rejection of *ab*. We can sum up these experiences by saying that, when presented with the string *ab*

—the machines M_1 and M_3 will always accept,
—the machine M_2 may accept or reject.

This may be rephrased by saying that, at least for the input *ab*, M_2 is more nondeterministic than M_1 and M_3.

A similar difference may be seen between M_1 and M_3 for the input *ac*; M_1 will always accept it, whereas M_3 may accept or may reject. Indeed, if one wanted a machine that accepts the language $a(b + c)^*$, then M_1 is infinitely superior to either M_2 or M_3 for the simple reason that M_1 will always accept any string in the language. On the other hand, the response from M_2 (or M_3) is more nondeterministic; it may accept the string proffered or it may not. M_3 is also to be preferred over M_2 since its behavior is less erratic. There is a subset of the language that M_3 will always accept, whereas with M_2 the response to any input may either be YES or NO. These ideas lead to a more stringent comparison between machines. For each potential input string, one can say whether a machine *may* accept it or *must* accept it. The latter may be more correctly rendered as "may not reject it." Then machines are compared by collating both the set of strings it *may* accept and the set of strings it *must* accept. This form of comparison will distinguish between M_1, M_2, and M_3.

Before developing a theory of machines, one must have a clear idea of what external behavior is to be considered important. It should now be apparent that the traditional theory of machines, as exemplified by [20], only takes into consideration the set of strings that *may* be accepted. It has been argued elsewhere (for instance, [24]) that this is inadequate for many applications. In particular, in recently developed theories for communicating machines [6, 19, 24], it is important to be able to distinguish between machines such as M_1, M_2, M_3. Informally, the reason is as follows: If M_1 and M_2 are to be equivalent, then it should be possible to replace M_1, when it is used as a subsystem of a larger system, by M_2, without affecting the overall behavior of the larger system. However, it is easy to construct a deadlock-free system that uses M_1 as a subsystem such that, if M_1 is replaced by M_2, then a possible deadlock is introduced into the overall system.

In this paper we explain a new theory of machines that takes into account both the strings that *may* be accepted and the strings that *must* be accepted, together with some information about possible deadlocks. In Section 2, we introduce the semantic domain. Sets of strings will no longer be adequate to represent machines; they retain too little information. Instead, they are replaced by certain kinds of *labeled trees*. These have a particularly simple structure. There are some minor complications due to the fact that we wish to represent machines that may only be partially defined. Such partially defined objects have greatly facilitated the elaboration of semantic theories for the λ-calculus [31]. In the present context, their presence enables us to show that our model, AT (Acceptance Trees), is a continuous partial order (cpo). The ordering on AT is designed to represent the intuitive notion of "less deterministic than." A machine M will be more nondeterministic than machine M' if (approximately)

(i) they both *may* accept exactly the same set of strings,
(ii) M *must* accept a string, then M' *must* accept it also.

We also define various continuous operations over AT. The set of operators will be denoted by Σ. The most important are two nondeterministic operators $+$ and $\oplus$.

In Section 3, we explain one of the main results of the paper, the Algebraic Characterization Theorem. We show that AT is in fact the initial Σ-cpo in the class of Σ-cpos that satisfy a particular set of equations. This result has some interesting consequences. For example, it shows that AT is isomorphic to the model I_1 in [15]. This, in turn, is fully abstract with respect to an operational semantics based on the idea of communicating processes experimenting on each other. This is explained in Section 4.2 in which the Operational Characterization Theorem is given. More important, the Algebraic Characterization Theorem gives a complete proof system for AT. This consists of the set of axioms, together with a very general form of induction, called *General Induction* in [8]. Indeed, this is the main import of initiality. This proof system is analogous to those given in [30] for regular expressions. The system F_1 of that paper also consists of a set of axioms and a simple inductive rule. If we restricted our attention to simple subsets of AT, such as that corresponding to finite machines, one might also be able to restrict the form of induction used to something, such as fixpoint induction [34].

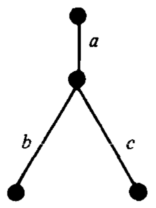
These consequences are not elaborated on in the paper. Instead, we address the problem of how one uses the model AT to give semantics to machines. With the simpler model, one merely associated with each machine the set of strings it may accept. This operational approach is not strictly necessary. In [8], a language for machines was given, and it was shown how one could associate with each machine a set of strings in a more abstract way, using fixpoint semantics: Each machine gives rise to an equation over the model and one then associates with the machine the least solution of the corresponding equation. Moreover, it was shown that this solution coincided with the set of strings that one obtains in the more operational semantics. In Section 4, we mimic this approach for a simple language for nondeterministic machines. With each expression in the language, which represents a machine, we associate an object in AT, in the usual denotational way [10, 12]. We then try to formalize an operational semantics. To do this, we must say which strings a machine *may* accept, which strings a machine *must* accept and which machines are underdefined. In Section 4.2, we formalize this using the idea of performing experiments on machines and thereby trying to detect differences between specific machines. As stated previously, this leads to the Operational Characterization Theorem. In Section 4.3, we give an alternative formalization based on a property language $\mathscr{P}$, a modal language, and two satisfaction relations between machines and properties. The first one states when a machine may satisfy a property and the second when it must satisfy a property. These properties deal, of course, with the ability to accept strings but they also capture some information about possible deadlocks. If we let $\mathscr{P}_o(M)$, $\mathscr{P}_c(M)$ be the set of properties that a machine M may (respectively, must) satisfy, then these two sets encapsulate an operational semantics of the machine. The third main result of the paper, the Modal Characterization Theorem, states that the denotational and operational semantics coincide; that is, two machines M_1, M_2 denote the same object in AT if and only if $\mathscr{P}_o(M_1) = \mathscr{P}_o(M_2)$ and $\mathscr{P}_c(M_1) = \mathscr{P}_c(M_2)$. These results indicate that our model adequately reflects the behavioral aspects of machines that we deemed important.

The first three sections may be read without any knowledge of previous work in this area. However, some acquaintance with abstract algebra and continuous algebras is assumed, particularly to understand the proofs. The required prerequisites may be found in [13]. Section 5, which contains the proofs of the three main theorems, relies heavily on knowledge of the results and techniques in [15]. This has the advantage of keeping the section extremely short. It is followed by a brief comparison between our work and other models recently proposed.

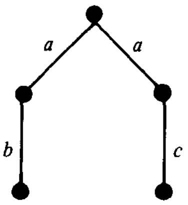
2. *A Description of the Model*

2.1. The model consists of certain kinds of rooted trees. Both the branches and the nodes are labeled. In this informal introduction to the model, we assume a nonempty set of actions A, which the machines under consideration can perform. Alternatively, if we view the machines as accepting automata A may be taken to be the input alphabet. The branches of the trees are labeled by elements of A.

The tree



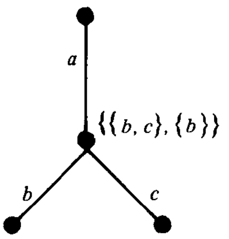
could be taken to represent a machine that performs the action a and then can perform either the action b or the action c but not both. We view this machine as being *deterministic.* At each point in time, its behavior depends entirely on the user or, more generally, on its environment. Initially, it can only accept the input a. Then, if presented with the input b, it must accept it, and if presented with the input c, it must accept it. Thus, its behavior will depend entirely on the input string it is asked to accept. Trees of the form



will not be allowed even though they can represent machines with inherently nondeterministic behavior. Instead, we stipulate

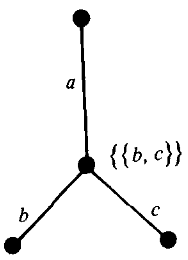
(S1) For every $a \in A$, every node in the tree has at most one successor branch labeled by a.

Because of this condition every node in the tree is uniquely identified by a string in A^*. If t is a tree, we let $L(t)$ denote this set of strings and, for $s \in L(t)$, we let $t(s)$ denote the node uniquely identified by s. Note that $L(t)$ is always prefix-closed. The set of actions labeling the successor branches of a node n is called its *successor set* and is denoted by $S(n)$. To model the nondeterministic behavior, we label the nodes by nonempty subsets of $\mathscr{P}(S(n))$, the set of subsets of the set $S(n)$. The tree



is a typical example. This represents a machine that accepts the strings ab, ac. After accepting the symbol a, it can be in one of two internal states represented by the sets $\{b, c\}$, $\{b\}$, respectively. If the machine is in the state represented by $\{b, c\}$, then, when presented with b or c as input, it must accept. If it is in the state represented by $\{b\}$, then, when it is presented with b as input, it must accept; however, in this state, it will not accept c. Consequently, it must accept the string ab, whereas it may or may not accept ac.

The tree



on the other hand, represents a machine that will accept the same two strings, but is more deterministic; after accepting a, there is only one internal state represented by the set $\{b, c\}$, which must accept when one of b and c is presented to it. Thus, it must accept both ab and ac, which makes it less uncertain than the previous machine.

The sets that label the nodes are called *acceptance sets*, and the acceptance set of a node n is denoted by $\mathscr{A}(n)$. Acceptance sets satisfy certain consistency requirements.

(A1) $S \in \mathscr{A}(n)$ implies $S \subseteq S(n)$.

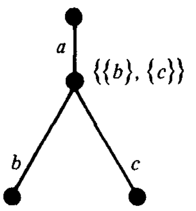
This is understandable since every S in $\mathscr{A}(n)$ represents an internal state and we identify an internal state with the set of symbols that can be accepted when the machine is in that state.

(A2) If $a \in S(n)$ then there exists $S \in \mathscr{A}(n)$ such that $a \in S$.

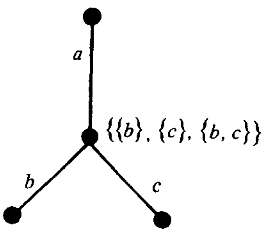
This just means that, if a is a successor label of a node, then there is at least one internal state associated with the node that can accept a.

(A3) $\mathscr{A}(n)$ is closed under union, that is, if $S_1, S_2 \in \mathscr{A}(n)$ then $S_1 \cup S_2 \in \mathscr{A}(n)$.

This condition is to ensure that the ordering "is less deterministic than" is in fact a partial order on trees. If we did not have this condition, then the tree t_1



would be allowed and would be a different tree than t_2



However, both t_1 and t_2 both accept the same language, and the only string that either must accept is a. Consequently, each is trivially "less deterministic than" the other. So this relation would not be antisymmetric.

(A4) $\mathscr{A}(n)$ is convex-closed; that is, if $X, Z \in \mathscr{A}(n)$ and $X \subseteq Y \subseteq Z$, then $Y \in \mathscr{A}(n)$.

This has the same justification as A3. If it were allowed, we would have trees that were different but that could not be differentiated using the comparison "less deterministic than."

Conditions A3 and A4 are summarized by saying that $\mathscr{A}(n)$ is saturated: A set $\mathscr{A} \subseteq \mathscr{P}(A)$ is *saturated* if it is nonempty, closed under union, and convex-closed. We let Sat(A) be the saturated subsets of $\mathscr{P}(A)$ and, for $\mathscr{S} \subseteq \mathscr{P}(A)$, we let $c(\mathscr{S})$ be the least saturated set containing $\mathscr{S}$.

The condition that $\mathscr{A}(n)$ be saturated has as a consequence that $S(n) \in \mathscr{A}(n)$. If this is the only set in $\mathscr{A}(n)$, then the tree is deterministic at that node. When describing trees, we have used the convention that if it is deterministic at a node (i.e., the only label is $\{S(n)\}$), we omit the label from the node. This makes the trees more readable. Note that the trivial tree consisting of a single node

$$\bullet$$

is in fact under this convention representing the tree

$$\bullet\ \{\{\emptyset\}\}$$

Saturated sets tend to be large in comparison to the amount of information they contain. When describing trees, we generally do not list out all the elements of $\mathscr{A}(n)$, but instead give a minimal subset that generates, that is, a minimal $\mathscr{S}$ such that $c(\mathscr{S}) = \mathscr{A}(n)$. So, for example,

$$\{\{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}, \qquad \{\{b\}, \{c\}, \{a, c\}, \{b, c\}, \{a, b, c\}\},$$

are given as

$$\{\{a\}, \{b\}, \{c\}\}, \qquad \{\{b\}, \{c\}, \{a, c\}\},$$

respectively. To make these enumerations more readable we give, when possible, the minimal sets that generate an acceptance set as sequences. So the two examples will be rendered as

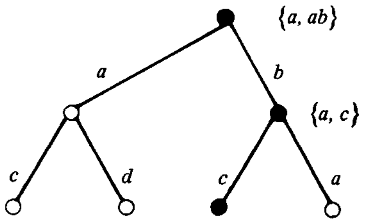
$$\{a, b, c\}, \qquad \{b, c, ac\}.$$

Partially defined trees are defined using a new type of node ○. The node ● is called *closed*, whereas ○ is called *open*. The first requirement on open nodes is that

(O1) Open nodes are *not* labeled by any acceptance sets.

Intuitively open nodes describe parts of machines that are not fully defined. Since they are not fully defined, the internal states at that point cannot be elaborated.

For example, the tree



describes a machine whose behavior is not fully defined. In particular, what happens after it performs an a action is not fully defined. All we know is that, after performing a, it may (in certain unknown circumstances) perform c and d. We have two further conditions about open nodes.

(O2) Every descendant of an open node is open.
(O3) If $S(n)$ is infinite, n is open.

For the moment, we are only interested in machines that exhibit bounded nondeterminism. If such a machine can accept an infinite number of different symbols, then it can also spend all of its time deciding which one to accept; that is, it can diverge. In our framework, this internal behavior will have no explicit representation except that "internal divergence" will be represented by open nodes. As an immediate consequence of these assumptions, we have condition O3. Note that O3 implies that, for every closed node n, $\mathscr{A}(n)$ is finite.

An alternative way of stating O2 is to say that the subset of $L(t)$, which identifies closed nodes, must be prefix-closed. We use $\mathrm{CL}(t)$ to denote this set and $\mathrm{OL}(t)$ to denote $\mathrm{L}(t)/\mathrm{CL}(t)$. Condition O2 is somewhat more difficult to motivate than O3. The reason for it will be seen more clearly in the next section in which we define a partial order on trees. Roughly speaking, we can improve on a tree by adding new subtrees at an open node. In general, such additions will have the effect of improving on successors of the open node in question. For these effects to be represented in the model, these successor nodes must be open.

We have now completed our informal description of the model. To conclude this section, we recapitulate on the definition.

Definition 2.1.1. For a set of symbols A, let $\mathrm{AT}(A)$ be the set of rooted, finite, or infinitely branching trees such that

(i) every branch is labeled by an element of A,
(ii) every node is either open (o) or closed (●),
(iii) every closed node is labeled by a saturated subset of $\mathscr{P}(A)$,

and which satisfies the conditions S1, A1, A2, O2, and O3, given above.

Since A will remain fixed, $\mathrm{AT}(A)$ will be abbreviated by AT. The variables t, n, a, s will be used to range over AT, the set of nodes, A and A^*, respectively. Throughout the paper, we use the notation introduced in this section. In addition, if a labels a branch from the root of t, that is, $a \in S(t(\epsilon))$ or $a \in L(t)$, then t/a will denote the subtree of t whose root is the node $t(a)$. Also, by a slight abuse of notation, we write $S(t)$, $\mathscr{A}(t)$ in place of $S(t(\epsilon))$, $\mathscr{A}(t(\epsilon))$. These notations make it very easy to define precisely a particular tree in AT. Every tree is uniquely determined by

—$\mathrm{L}(t)$, a nonempty prefix-closed subset of A^*, representing the nodes of the tree,
—$\mathrm{CL}(t)$, a prefix-closed subset of $\mathrm{L}(t)$, representing the closed nodes,
—A total mapping $\mathscr{A}$: $\mathrm{CL}(t) \to \mathrm{Sat}(A)$.

Consequently, to define a tree, we need only give these two sets and an appropriate mapping. However, we must ensure that the conditions A1, A2, and O3 are satisfied.

2.2. In this section we describe a partial order $\leq$ on AT that will, in fact, turn out to be a complete partial order, that is, directed sets of trees will have least upper bounds.

We are primarily interested in totally defined trees, that is, trees that contain no open nodes. So we begin by describing the partial order as applied to these. In the introduction, we stated that for two nondeterministic machines M_1, M_2, the machine M_1 would be less deterministic than M_2 if

(i) both M_1 and M_2 accept the same language, and
(ii) for a given string in the language, there are fewer uncertainties about M_2 accepting the language than there are about M_1.

The first condition will be formalized by demanding that, if $t_1 \leq t_2$, then

(C1) $\mathrm{L}(t_1) = \mathrm{L}(t_2)$.

The second condition will be rendered as

(C2) For $s \in \mathrm{L}(t_1)$, $\mathscr{A}(t_1(s)) \supseteq \mathscr{A}(t_2(s))$.

That is, at each pair of corresponding points in the trees t_1 and t_2, the acceptance sets of t_2 are contained in the acceptance sets of t_1. If one thinks of the correspondence between acceptance sets and internal states this seems natural.

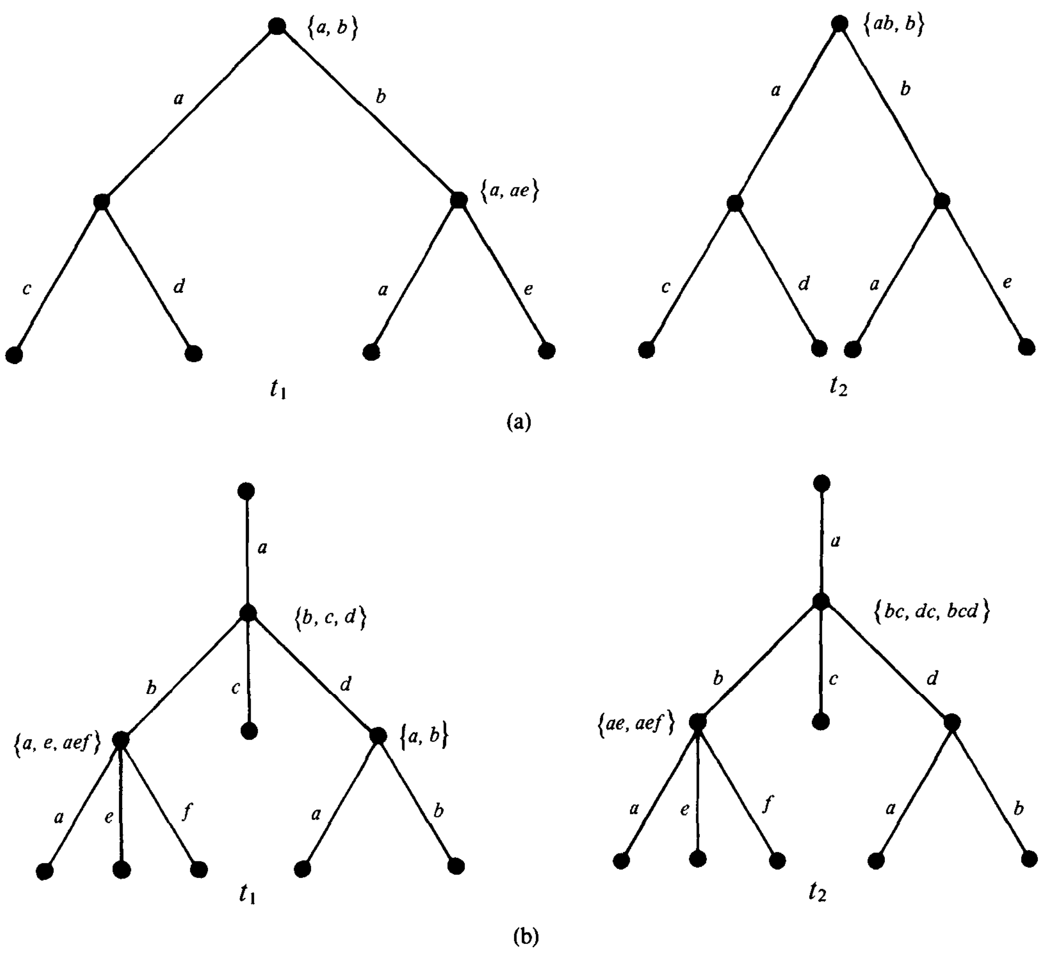


FIG. 2. Examples of trees with closed nodes. In (a) and (b), $t_1 \leq t_2$.

C1 and C2 completely determine the partial order over trees that have no open nodes. This implies that, if two such trees are comparable, they must have the same structure; only the acceptance sets are different. Examples are given in Figure 2.

The presence of an open node indicates that the tree is not fully defined at that point. So it can be improved upon at an open node by grafting on any subtree. This leads to the demand that, if $t_1 \leq t_2$, then

(O1) $L(t_1) \subseteq L(t_2)$.

For trees that contain only open nodes, O1 will completely determine the partial order. Examples are given in Figure 3.

In general, trees will have both open and closed nodes and the definition of $\leq$ is a mixture of the conditions O1, C1, and C2. The definition is obtained by seeing what these conditions demand at each individual node and from the extra requirement that "open nodes are less defined than closed nodes."

Definition 2.2.1. For $t_1, t_2 \in \mathrm{AT}$, $t_1 \leq t_2$, if

(i) $L(t_1) \subseteq L(t_2)$,

(ii) $CL(t_1) \subseteq CL(t_2)$,

(iii) for every $s \in CL(t_1)$, $\mathscr{A}(t_2(s)) \subseteq \mathscr{A}(t_1(s))$.

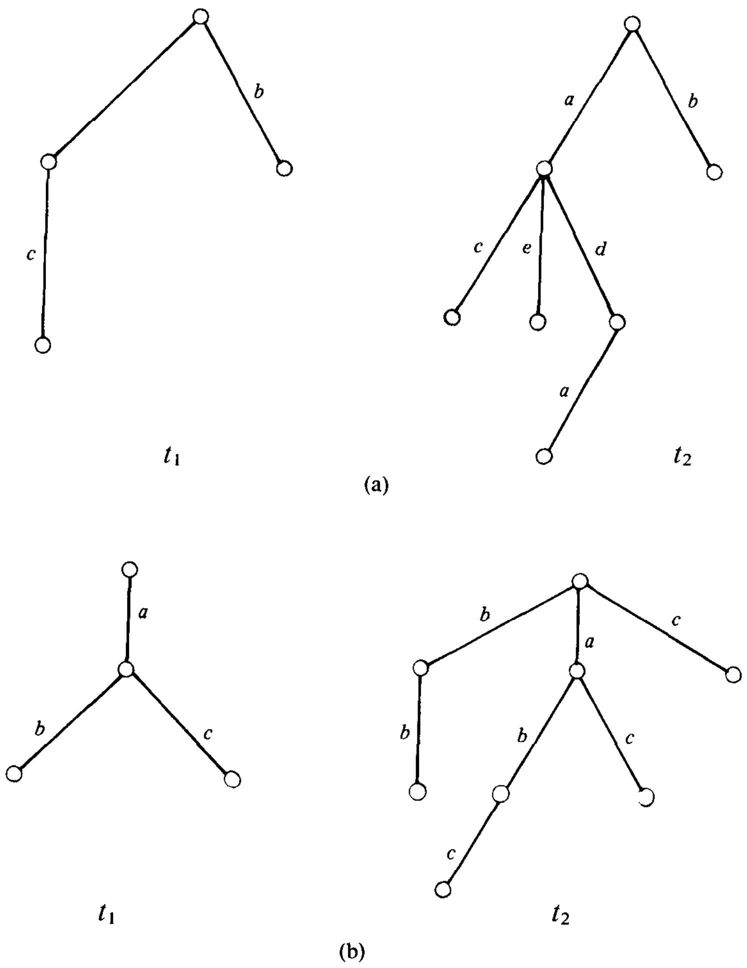


FIG. 3. Examples of trees with open nodes. In (a) and (b), $t_1 \leq t_2$.

One can show that if t_1 contains only closed nodes, that is, $\mathrm{L}(t_1) = \mathrm{CL}(t_1)$, and $t_1 \leq t_2$, then t_2 only contains closed nodes and the conditions C1 and C2 are satisfied. Conversely, if both t_1 and t_2 contain only closed nodes and they satisfy C1 and C2, then $t_1 \leq t_2$. Similarly, if t_1 and t_2 contain only open nodes (i.e., $\mathrm{CL}(t_1) = \mathrm{CL}(t_2) = \varnothing$), then $t_1 \leq t_2$, if and only if they satisfy O1. Examples of $\leq$ between trees that have both open and closed nodes are given in Figure 4.

THEOREM 2.2.2

(*a*) *$\langle AT, \leq \rangle$ is a complete partial order.*
(*b*) *$\langle AT, \leq \rangle$ is an algebraic complete partial order whose finite elements are all those trees that have a finite number of nodes.*

PROOF

(a) To show that $\leq$ is a partial order is a matter of simple calculation.
Let D be a directed set of trees. Define a new tree t as follows:

(i) $\mathrm{L}(t) = \{s \mid s \in \mathrm{L}(t') \text{ for some } t' \in D\}$,
(ii) $\mathrm{CL}(t) = \{s \mid s \in \mathrm{CL}(t') \text{ for some } t' \in D\}$.

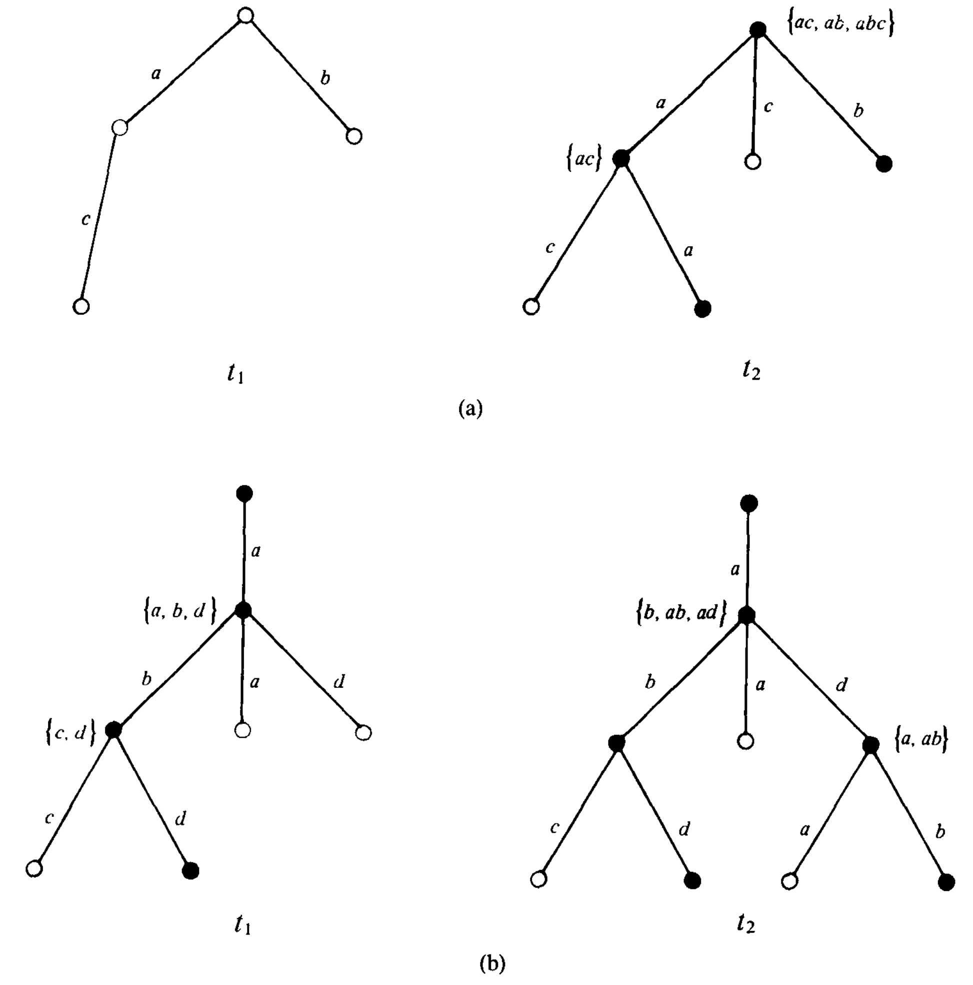


FIG. 4. Examples of trees with open and closed nodes. In (a) and (b), $t_1 \leq t_2$.

We must also associate with each closed node of the tree, $t(s)$, an acceptance set $\mathscr{A}(t(s))$. If $s \in \mathrm{CL}(t)$, then we let

$$\mathscr{A}(t(s)) = \{A \mid A \in \mathscr{A}(t'(s)) \text{ for all but a finite number of } t' \in D\}.$$

Some routine calculations suffice to prove that $\mathscr{A}(t(s))$ is indeed an acceptance set.

To show that we have, in fact, defined an element of AT, we must show that conditions A1, A2, and O3 are satisfied. These are easily deduced from the remarks

(i) $S(t(s)) = \bigcup \{S(t'(s)) \mid t' \in D, s \in L(t')\}$,
(ii) if $t(s)$ is closed, then $\mathscr{A}(t(s)) = \mathscr{A}(t'(s))$ and $S(t(s)) = S(t'(s))$, for all but a finite number of t' in D.

Having defined the tree t, we must now show that it is the least upper bound of D; that is, if $t' \leq u$ for every t' in D, then $t \leq u$. This however follows in a straightforward manner from remarks (i) and (ii).

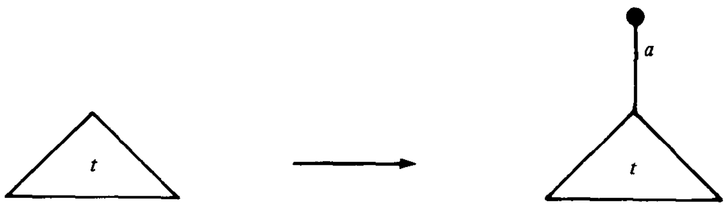
Finally, the least element of $\langle \mathrm{AT}, \leq \rangle$ is the trial tree o.

(b) Left to the reader. The arguments are straightforward but rather lengthy. □

We can isolate at least two interesting subsets of AT. Let DAT be the set of *deterministic* trees, that is, those trees whose acceptance sets at every node is a singleton set. More formally, $t \in \mathrm{DAT}$ if, for every $s \in \mathrm{CL}(t)$, $\mathscr{A}(t(s))$ consists of the unique element $S(t(s))$. Elements of DAT have no internal behavior. It is quite easy to show that $\langle \mathrm{DAT}, \leq \rangle$ is also a complete partial order. Let FDAT be the set of *fully defined trees*, that is, those trees all of whose nodes are closed; if $t \in \mathrm{FDAT}$, then $\mathrm{L}(t) = \mathrm{CL}(t)$. $\langle \mathrm{FDAT}, \leq \rangle$ is not a complete partial order since it lacks a minimal element. We end this section with another set of examples, given in Figure 5, of trees t_1, t_2 such that $t_1 \not\leq t_2$.

2.3. In this section, we describe some operations on trees.

The first operation is of prefixing a tree by a specific action a. This may be described diagrammatically as



Following the conventions of the previous sections, this indicates that the acceptance set associated with the root of the new tree is simply $\{\{a\}\}$. We now describe this operation more formally.

For $a \in A$, $t \in \mathrm{AT}$, let $a.t$ be the new tree described by

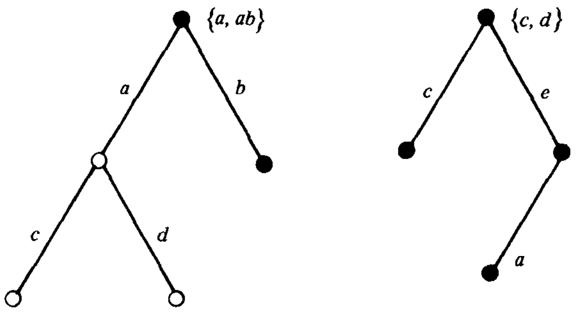
(i) $\mathrm{L}(a.t) = \{\epsilon\} \cup \{as \mid s \in \mathrm{L}(t)\}$,
(ii) $\mathrm{CL}(a.t) = \{\epsilon\} \cup \{as \mid s \in \mathrm{CL}(t)\}$,
(iii) $\mathscr{A}(a.t(\epsilon)) = \{\{a\}\}$
$\mathscr{A}(a.t(as)) = \mathscr{A}(t(s))$.

It is easy to check that this does indeed define a tree. In fact we have

PROPOSITION 2.3.1. *For every $a \in A$, the operation $a.\_: AT \rightarrow AT$ is continuous.*

We shall usually render $a.t$ as at.

The next operation takes two trees and "glues" them together at their roots. The acceptance set, when it exists, at the new root is simply the closure of the union of the acceptance sets at the roots of the original two trees. For example, when the operation is applied to



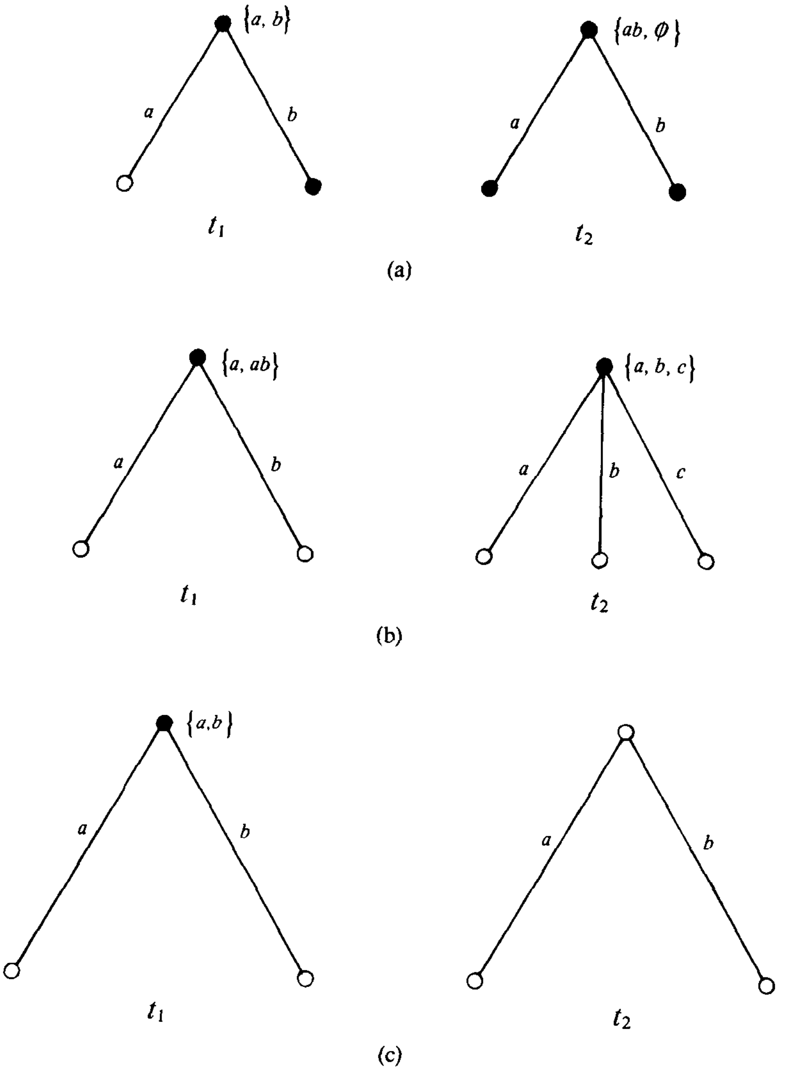
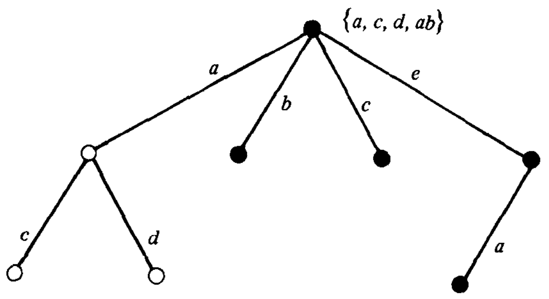
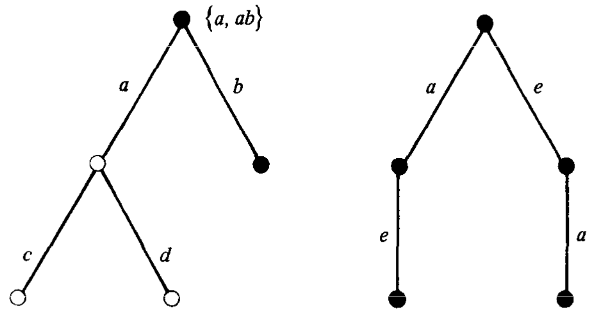


FIG. 5. Three examples of trees in which $t_1 \not\leq t_2$.

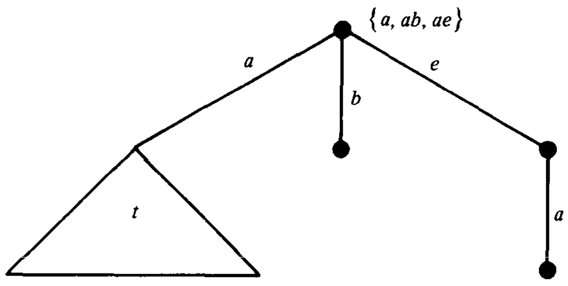
we obtain the tree



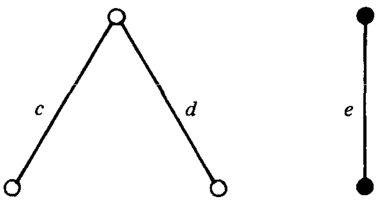
If either of the roots are open, then the root of the constructed tree is also open. This is a reflection of the fact that open nodes are considered to be "less defined" than closed nodes. The only problem occurs when we try to glue together two trees that have an initial action in common. For example,



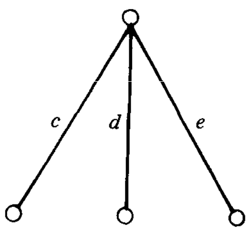
In this case, we proceed as before gluing together the two trees at the root. However, the a-subtree of the new tree is now obtained by applying the same operation to the two a-subtrees of the original trees. This results in the following tree:



where t is the tree obtained by gluing together the subtrees



So t is



This new operation is therefore seen to be recursive in nature. However, it can be defined quite straightforwardly in the following way: If t_1, t_2 are trees, let $t_1 \oplus t_2$ be the tree t where

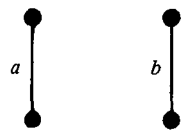
$$\begin{aligned} \mathrm{L}(t) &= \mathrm{L}_1(t) \cup \mathrm{L}_2(t), \\ \mathrm{CL}(t) &= \{s \in \mathrm{L}(t), \text{ for } i = 1, 2, s \in \mathrm{L}(t) \text{ implies } s \in \mathrm{CL}(t_i)\}, \\ \mathscr{A}(t(s)) &= c(\mathscr{A}(t_1(s)) \cup \mathscr{A}(t_2(s))), \\ &\quad \text{with the convention that if } s \notin \mathrm{CL}(t_i) \text{ then } \mathscr{A}(t_i(s)) = 0. \end{aligned}$$

PROPOSITION 2.3.2. *$\oplus$: $AT \times AT \to AT$ is continuous.*

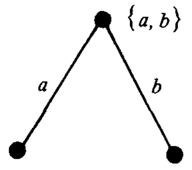
PROOF. Straightforward. □

We can define a slightly different operator, +, that represents external nondeterminism; the machine represented by $t_1 + t_2$ will once more act either like t_1 or t_2, but the choice will depend on what symbol the machine is asked to accept (or what action it is asked to perform). A simple example will explain the difference beween these two operators

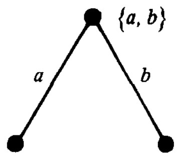
Let t_1, t_2 be



Then $t_1 \oplus t_2$ is



and $t_1 + t_2$ will be



Note that $t_1 \oplus t_2$ is nondeterministic, whereas $t_1 + t_2$ is deterministic. The definition of + is nearly identical to that of $\oplus$. The only difference occurs at the root when both the parameters have closed nodes.

If t_1, t_2 are trees, let $t_1 + t_2$ be the tree t where

$$\begin{gathered}\mathrm{L}(t),\ \mathrm{CL}(t) \text{ are as in the definition of } t_1 \oplus t_2,\\ \mathscr{A}(t) = (A_1 \cup A_2, A_1 \in \mathscr{A}(t_1), A2 \in \mathscr{A}(t_2)\},\\ \mathscr{A}(t(s)) \text{ is as above in the definition of } t_1 \oplus t_2, \text{ whenever } s \neq \epsilon.\end{gathered}$$

PROPOSITION 2.3.3. *+: $AT \times AT \to AT$ is continuous.*

PROOF. Straightforward. □

Examples of these operators are given in Figures 6 and 7. It should be pointed out that + does not always preserve determinism; that is, if t_1, t_2 are deterministic, then $t_1 + t_2$ may in general be nondeterministic. However, if $S(t_1) \cap S(t_2) = \emptyset$, determinism is preserved.

3. *Characterization of the Model*

The operations defined in the previous section satisfy many interesting properties. For example, $\oplus$ satisfies the axioms

$$\begin{gathered}X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z,\\ X \oplus Y = Y \oplus X,\\ X \oplus X = X.\end{gathered}$$

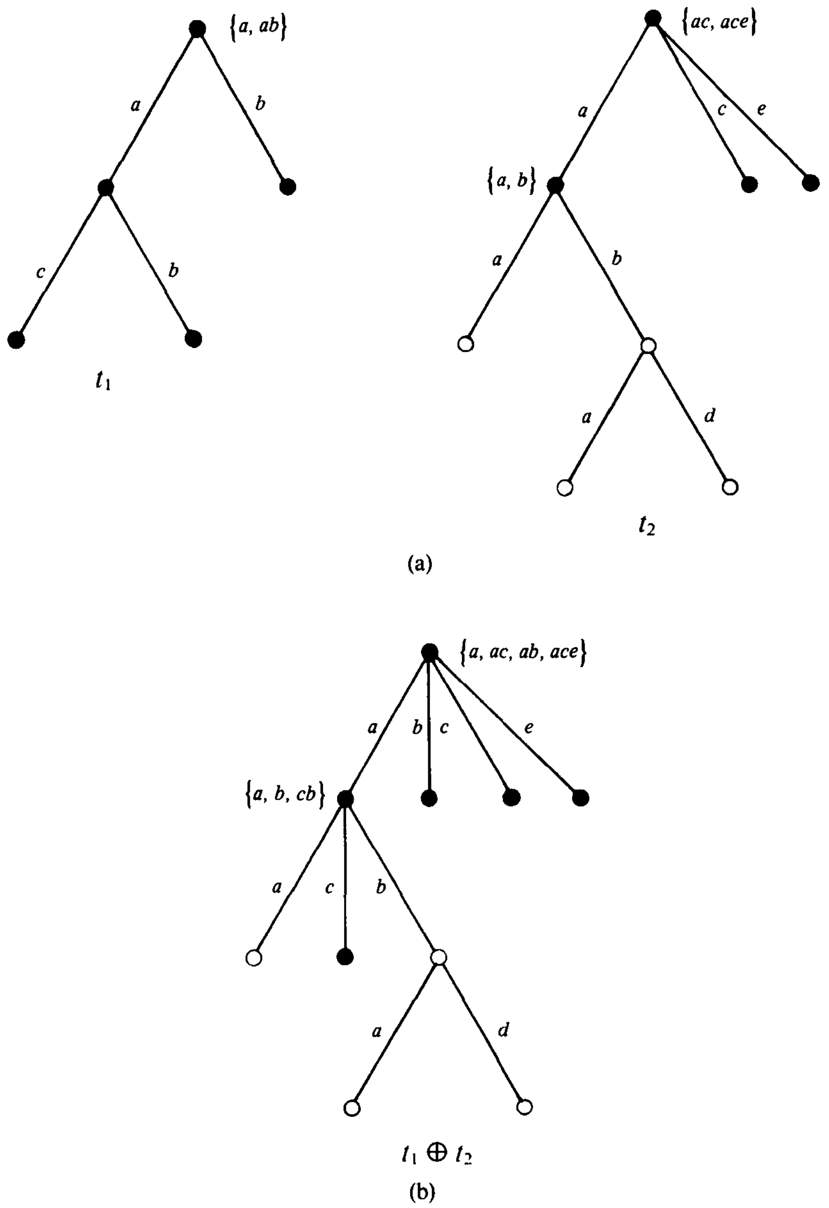


FIG. 6. Application of the operator $\oplus$. (a) Trees t_1 and t_2. (b) Tree $t_1 \oplus t_2$.

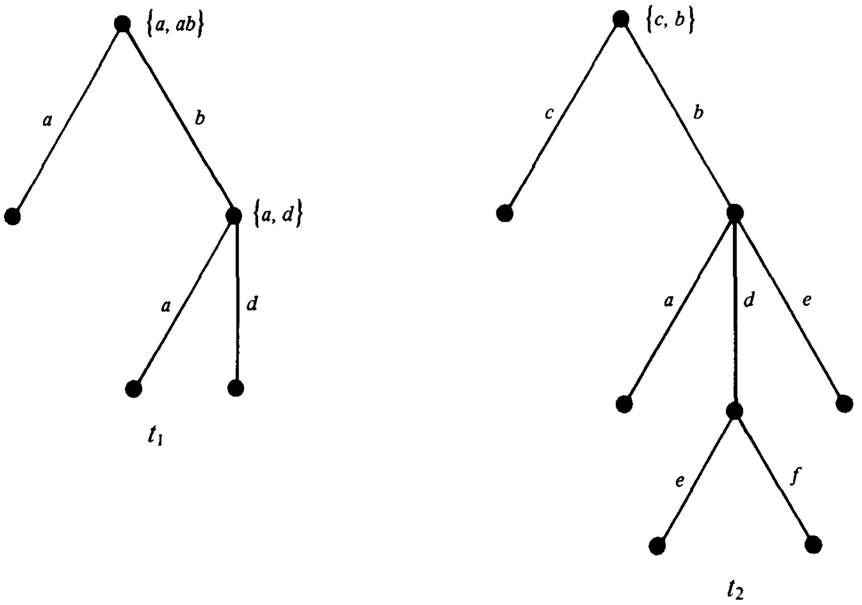
These three axioms can be summarized by saying that

1. (AT, $\oplus$) is an idempotent Abelian semigroup.

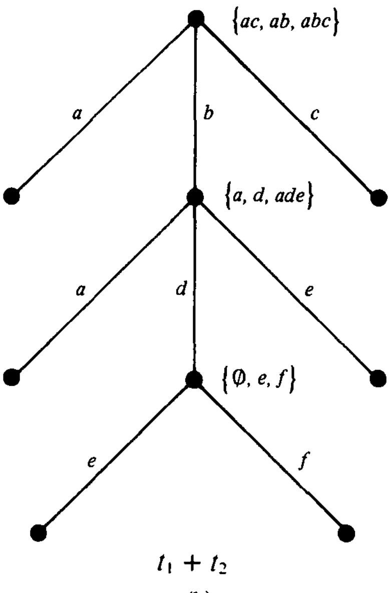
Equivalently one can say that (AT, $\prec$) is a semilattice when $t \prec t'$ if there exists a t'' such that $t \oplus t'' \leq t'$. For details, see [21]. The operator + also satisfies these axioms. In addition, if we use the nullary operator $\mathbb{O}$ to denote the trivial tree

we also have the axiom

$$X + \mathbb{O} = X.$$



(a)



(b)

FIG. 7. Application of the new operator. (a) Trees t_1 and t_2. (b) Tree $t_1 + t_2$.

So we can say that

2. (AT, +, Ⓓ) is an idempotent Abelian semigroup with zero.

Moreover, each of the binary operators distribute over the other. This can be expressed by saying that AT satisfies the following axioms:

(D1) $X \oplus (Y + Z) = (X \oplus Y) + (Y \oplus Z)$.
(D2) $X + (Y \oplus Z) = (X + Y) \oplus (X + Z)$.

All of these remarks are easily checked by examining the constructions of $\oplus$ and $+$. Similarly, we have, for each $a \in A$,

(N1) $aX + aY = a(X \oplus Y)$.
(N2) $aX \oplus aY = a(X \oplus Y)$.
(N3) $X \oplus Y \sqsubseteq X + Y$.

If we introduce a further nullary operator Ω to denote the trivial tree

$$\circ$$

we also have the axioms

(Ω1) $\Omega \sqsubseteq X$.
(Ω2) $X + \Omega \sqsubseteq X \oplus \Omega$.

There are many other properties of the operators that one might wish to consider, but this set is particularly interesting in that it completely determines AT. This can be explained as follows: If we let Σ denote the set of operators $\{\varnothing, \Omega, a$ (for all $a \in A$), $\oplus, +\}$, then AT can be considered as a Σ-cpo (Σ-complete partial order). We then have that

THEOREM 3.1. THE ALGEBRAIC CHARACTERIZATION THEOREM. *$\langle AT, \leq \rangle$ is isomorphic, as a Σ-cpo, to the initial Σ-cpo that satisfies* 1, 2, $D1$, $D2$, $N1$, $N2$, $N3$, *and* $\Omega 1$, $\Omega 2$.

The proof relies on a knowledge of [14]. It is given in Section 5.

This theorem has a number of interesting corollaries. For example, every finite tree can be denoted by a term over the operator set Σ. Moreover the set of axioms gives a complete proof system for the relation $\leq$ between trees. In fact, every tree can be considered to be a limit of such terms and, therefore, the axioms, together with a very general form of induction called *general induction* in [8], give a complete proof system for $\leq$ over arbitrary trees. The rule of general induction is not finitary, but if one replaces it by some finitary approximation to it, such as Scott Induction, then one obtains an effective proof system. Another immediate result of the characterization theorem is that AT is isomorphic to the model I_1 of [15]. This gives an operational significance to AT, which is discussed in Section 4.2. We do not have a corresponding characterization of DAT or FDAT.

4. *Using the Model as a Semantics*

4.1. We have shown how AT can be considered as a Σ-cpo. Using the approach of [10], [12], and [13], we can now interpret a language, which only uses these operators, in AT in a very straightforward way. For completeness sake, we define the language and its interpretation in this section, although it already has been given in [15, sect. 1.2].

Let X be a set of variables, ranged over by x. The set of recursive terms over Σ, RFC_Σ, ranged over by t, is then defined by the following BNF-like schema:

$$t ::= x \mid \mathrm{op}(t_1, \ldots, t_k), \qquad \mathrm{op} \in \Sigma^k \mid \mathrm{rec}\ x.t.$$

The term rec $x.t$ denotes a recursive definition that might also be rendered as

$$x \Leftarrow t.$$

We have the usual notions connected with this syntax. The operator, rec $x.$, binds occurrences of x in the subterm t of rec $x.t$ to rec $x.t$ itself. This gives rise, in the usual way, to the definition of free and bound occurrences of variables in terms. CREC_Σ denotes the set of *closed* terms, that is, terms with no free occurrences of variables. We use p, q to range over closed terms. FREC_Σ denotes the set of terms with no occurrence of rec x._, that is, the *finite* terms. A substitution is a mapping from X to REC_Σ and we use ρ to range over substitutions. If each $\rho(x)$ is in CREC_Σ, it will be called a *closed substitution.* We let $t\rho$ denote the result of substituting $\rho(x)$ for each free occurrence of x in t and $t[u/x]$ the result of substituting u into each free occurrence of x in t.

Given *any* Σ-cpo D, we can give a denotational semantics to our language in a natural way, following [10]. To cope with terms that are not closed, we need the notion of D-environments: let ENV_D be the set of mappings from X to D. Then the denotational semantics is in the form of a mapping:

$$\mathscr{M}_D\colon \mathrm{REC}_\Sigma \to (\mathrm{ENV}_D \to D).$$

It is defined by induction on terms. If e is a D-environment, then $e[d/x]$ is a new environment that differs from e only at x where it is defined to be d.

DEFINITION 4.1.1. Define $\mathscr{M}_D$ as follows:

(i) $\mathscr{M}_D(x)(e) = e(x)$,
(ii) $\mathscr{M}_D(\mathrm{op}(t))(e) = \mathrm{op}_D(\mathscr{M}_D(t_1)(e) \cdots \mathscr{M}_D(t_k)(e))$,
(iii) $\mathscr{M}_D(\mathrm{rec}\ x.t)(e) = Y\,\lambda d.\mathscr{M}_D(t)(e[d/x])$,

where Y represents the least fixpoint operator.

Note that, if t is a closed term, then $\mathscr{M}_D(t)$ is a constant function from ENV_D to D, which we identify with an element of D.

For *any* Σ-cpo D we now have an interpretation $\mathscr{M}_D$. We are, of course, only interested in the interpretation in our model AT. In Figure 8, we give three simple instances of $\mathscr{M}_{\mathrm{AT}}$. It is very easy to see that, if p does not contain an occurrence of the operator $\oplus$, then $\mathscr{M}_{\mathrm{AT}}(p)$ is in fact a deterministic tree. Another interesting set of trees we have previously discussed is FDAT, the set of fully determined trees; a tree is in FDAT if all of its nodes are closed. We now elaborate a condition on terms that ensure that they are interpreted as fully determined trees. The crucial point is already exhibited in Figure 8; the first two trees differ only in the color of the nodes. The open nodes in the second tree arise because of the presence of $+x$ in the recursive definition. Such an occurrence of a variable is called *unguarded*; the other two occurrences of x are *guarded*, the first by the action a, the second by b.

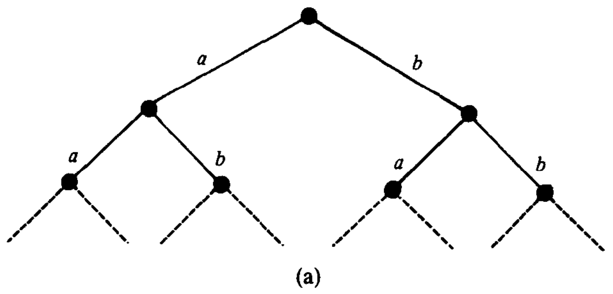
Definition 4.1.2. A closed term is *fully guarded* if

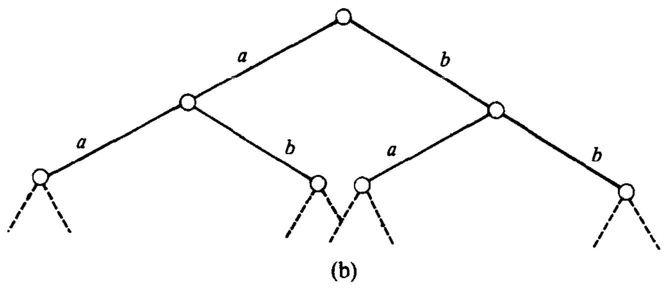
(i) it contains no occurrence of Ω,
(ii) in every subterm of the form rec $x.t$, every free occurrence of x in the t is guarded, that is, occurs within a subterm of the form $a.u$.

So, for example, rec $x.ax + bx$ is fully guarded but $a(\mathrm{rec}\ x.ax + x)$ is not, since in the subterm $ax + x$ the second occurrence of x is not guarded. Since Ω is interpreted in the same way as rec $x.x$, the first condition in the definition is reasonable.

THEOREM 4.1.3. *$\mathscr{M}_{\mathrm{AT}}(p)$ is fully determined if and only if p is fully guarded.*

PROOF. An arbitrary term t is called fully guarded if, for every closed substitution ρ such that $\rho(x)$ is fully guarded for every x in X, $t\rho$ is fully guarded.





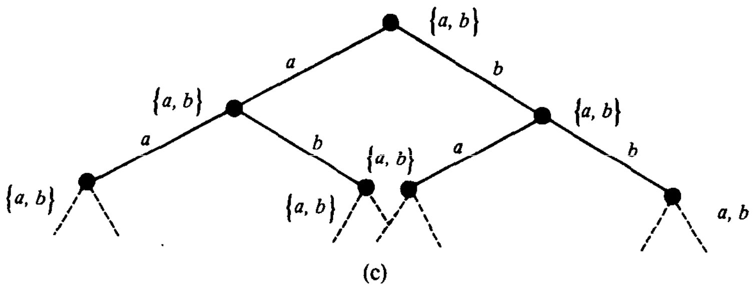


FIG. 8. Three simple instances of $\mathcal{M}_{\mathrm{AT}}$. (a) $\mathcal{M}_{\mathrm{AT}}$ (rec $x.ax + bx$). (b) $\mathcal{M}_{\mathrm{AT}}$ (rec $x.ax + bx + x$). (c) $\mathcal{M}_{\mathrm{AT}}$ (rec $x.ax \oplus bx$).

Suppose t is fully guarded. Then one can show by induction on t that $\mathcal{M}_{\mathrm{AT}}(t)(e) \in \mathrm{FDAT}$ whenever e is such that $e(x) \in \mathrm{FDAT}$ for every x in X. Since FDAT is closed under the operations $+$, $\oplus$, and $a.\_$, the only nontrivial case is when t is of the form rec $x.u$. In this case, we use the fact that $\mathcal{M}_{\mathrm{AT}}(t)(e) = V\{\mathcal{M}_{\mathrm{AT}}(t^n)(e), n \geq 0\}$, where t^n is obtained by unwinding the recursive definition n times. One can now prove by induction on n that all nodes of depth less than n in $\mathcal{M}_{\mathrm{AT}}(t^n(e))$ are closed. It follows by the construction of limits in AT that $\mathcal{M}_{\mathrm{AT}}(t)(e) \in \mathrm{FDAT}$.

Conversely, suppose p is not fully guarded. First note that if t has an unguarded occurrence of x, then $\mathcal{M}_{\mathrm{AT}}(\text{rec } x.t)(e)$ has an open root, for any environment e. Now if p is not fully guarded, it has a subterm whose denotation in AT has an open node. By induction on the depth of the occurrence of this subterm, one can also show that $\mathcal{M}_{\mathrm{AT}}(p)$ contains an open node, that is, $\mathcal{M}_{\mathrm{AT}}(p) \notin \mathrm{FDAT}$. □

One would not naturally write terms involving unguarded recursions since they involve purely circular definitions. This proposition states that, so long as we stick to "natural" terms, we will always obtain fully determined trees; or, in other words, the underdefined trees are only needed to take care of certain "unnatural" terms that we are allowed syntactically, but that are not the prime concern of the language.

4.2. The model AT equates certain terms in the language REC_Σ and distinguishes others. For the model to be of interest, there should be some computational or operational justification for these identifications and distinctions. This is the subject of this section.

In fact, we have already given such a justification in [14]. There we considered a language that was an extension of the present REC_Σ, in which we could model communicating processes. A number of operational preorders were defined, all based on the general idea of processes conducting experiments on each other. As was noted in the previous section, AT is isomorphic to the model I_1 of [15] and, consequently, Theorem 3.2.1 of that paper gives an operational justification for AT in terms of communicating processes.

To keep at least the expositary part of this paper self-contained, we now give an outline of this operational justification. It is based on the idea that closed terms represent machines for accepting strings of symbols. For example rec $x.ax + bx$ represents a machine that can accept any string over $\{a, b\}$. Indeed, the standard semantics for finite automata [20] are usually given in these terms. We may also think of this semantics in terms of experiments. To perform an experiment, we present a machine such as rec $x.a(bx + cx)$ with a string to accept, such as *abaca*. In this case the machine accepts the string; that is, the experiment is successful. The standard semantics now identifies a machine with the set of strings it accepts. A more precise rendition would be that it is identified with the set of strings that it may accept. For example, (rec $x.abx + acx$) accepts the string *aba* although there are computations from this machine that can deadlock when trying to accept this string. In other words, it is not true that rec $x.(abx + acx)$ must accept *aba*. On the other hand, rec $x.a(bx + cx)$ must accept *aba*. Whereas the standard semantics uses only the notion of "may accept," we use both "may accept" and "must accept." In fact, our operational view will allow us to perform more complicated experiments than simply "try to accept the string x." The langauge used in [15] is a superset of the present one and, within it, one can express the interaction between the experimenter and the machine. The experimenters are also machines. They may be simple deterministic ones performing the string-based experiments discussed above, or they may be more complicated, for example, by presenting alternatives to the machine being experimented upon. Here we explain this general notion of experimentation without recourse to the language of [15]. To do so, we need to introduce a number of concepts.

The first one is the notion of "accepting a symbol." For each symbol $a \in A$, we define a binary (infix) relation $\xrightarrow{a}$ with the intention that $p \xrightarrow{a} p'$ means that p may accept the symbol a and thereby transformed in p'.

For example, it will be true that

$$ab\Phi \xrightarrow{a} b\Phi,$$

$$\text{rec } x.(ax + b\Phi) \xrightarrow{a} \text{rec } x.(ax + b\Phi).$$

To define the relations $\xrightarrow{a}$, we need the notion of a state: A process accepts a sequence of symbols by starting in some state, accepting the first symbol and changing to a new state, accepting the next symbol and changing to a new state, etc. The operator $\oplus$ is intimately connected with the idea of state. For example, the term $a\Phi \oplus b\Phi$ represents a machine that is in one of two states $a\Phi$ or $b\Phi$. If it is in the former, then it can accept the symbol a; if in the latter, it can only accept b. On the other hand, the term $a\Phi + b\Phi$ represents a machine that has only one possible state in which it can accept a if it is offered and b if it is offered. The following definition formalizes the concept of "state."

Definition 4.2.1. For each term t, let $\mathrm{ds}(t)$, the set of terms representing states of t, be defined by:

(i) $\mathrm{ds}(t \oplus u) = \mathrm{ds}(t) \cup \mathrm{ds}(u)$,
(ii) $\mathrm{ds}(at) = \{at\}$, $\mathrm{ds}(\Phi) = \{\Phi\}$, $\mathrm{ds}(x) = \{x\}$, $\mathrm{ds}(\Omega) = \{\Omega\}$,
(iii) $\mathrm{ds}(t + u) = \{t' + u',\ t' \in \mathrm{ds}(t),\ u' \in \mathrm{ds}(u)\}$,
(iv) $\mathrm{ds}(\mathrm{rec}\ x.t) = \{u[\mathrm{rec}\ x.t/x],\ u \in \mathrm{ds}(t)\}$.

So, for example, if t is $\mathrm{rec}\ x.ax \oplus bx$, then $\mathrm{ds}(t) = \{at, bt\}$. We may now define the next state relations $\xrightarrow{a}$.

Definition 4.2.2. For each $a.A$, let $\xrightarrow{a}$ be the least binary relation of CREC_Σ that satisfies

(i) $ap \xrightarrow{a} p'$ if $p' \in \mathrm{ds}(p)$;
(ii) (a) $p \xrightarrow{a} p'$ implies $p + q \xrightarrow{a} p'$
$\qquad p \oplus q \xrightarrow{a} p'$;
(*b*) $q \xrightarrow{a} q'$ implies $p + q \xrightarrow{a} q'$
$\qquad p \oplus q \xrightarrow{a} q'$;
(iii) $t[\mathrm{rec}\ x.t/x] \xrightarrow{a} q$ implies $\mathrm{rec}\ x.t \xrightarrow{a} q$.

As stated above, a machine accepts a string by starting in some state and then moving from state to state by virtue of accepting individual symbols. To show this, we extend the relations $\xrightarrow{a}$ to relations $\xrightarrow{s}$ where s ranges over A^* in the obvious way:

(i) $p \xrightarrow{\epsilon} q$ if $q \in \mathrm{ds}(p)$,
(ii) $p \xrightarrow{as} q$ is $p \xrightarrow{a} p_1$ and $p_1 \xrightarrow{s} q$.

The notions "may accept the string s" and "must accept the string s" can be defined in terms of these relations $\xrightarrow{s}$. For example, if p is $\mathrm{rec}\ x.abx + acx$, then p may accept aba since $p \xrightarrow{aba} bp$, whereas it is not true that p must accept aba since $p \xrightarrow{a} cp$ and the term cp cannot accept ba.

However, these relations will still not be able to distinguish terms such as Φ and Ω, or a and a^Ω. In the model they are differentiated by the color of the nodes of the corresponding tree. Syntactically they differ because in each pair one is fully guarded and the other is not. We can axiomatize the absence of unguarded variables at the topmost level in the following way.

Definition 4.2.3

(i) Let $\downarrow$ be the least (postfix) predicate over closed terms that satisfies
 (a) $\Phi\downarrow$, $ap\downarrow$;
 (b) $p\downarrow$, $q\downarrow$ implies $(p + q)\downarrow$, $(p \oplus q)\downarrow$;
 (c) $t[\mathrm{rec}\ x.t/x]\downarrow$ implies $\mathrm{rec}\ x.t\downarrow$.
(ii) Let $\uparrow$ denote the complement of $\downarrow$.

As examples, we have that $a\Omega\downarrow$ $(\mathrm{rec}\ x.ax + bx)\downarrow$ and $(\mathrm{rec}\ x.ax + bx + x)\uparrow$. The relation $\downarrow$ can be extended in a natural way to define $\downarrow s$ by

(i) $p\downarrow\epsilon$ if $p\downarrow$,
(ii) $p\downarrow as$ if $p\downarrow$ and $p \xrightarrow{a} p'$ implies $p'\downarrow s$.

So $p \downarrow s$ if one cannot uncover unguarded recursions or occurrences of Ω by performing the actions in s. Incidentally, $\downarrow$ can be used to give a more formal definition of fully guarded since it is easy to see that p is fully guarded if and only if $p \downarrow s$ for every s in A^*.

We are now ready to give our operational view of experimentation. Let w be a new distinguished action symbol. Intuitively performing the action w can be interpreted as *reporting successes.* Then an *experiment* is any closed term of the language that may also use this special symbol w. So, for example,

$$abw\mathbb{O}, \tag{e_1}$$
$$a(bw\mathbb{O} + cw\mathbb{O}), \tag{e_2}$$
$$a(bw\mathbb{O} + c\mathbb{O}), \tag{e_3}$$

are all experiments. For example, e_2 will report success if the machine being examined can perform either of the string of actions ab or ac. To apply the experiment e to the machine p, we use the notation

$$e \parallel p.$$

The application of this experiment proceeds by both e and p evolving (because of interaction between them) until e reaches a state in which it can report success. The rules that govern this evolution are

(i) if $e \xrightarrow{a} e'$ and $p \xrightarrow{a} p'$, then $e \parallel p \rightarrow e' \parallel p'$;
(ii) if $e \rightarrow e'$, then $e \parallel p \rightarrow e \parallel p'$.

The definition of a successful application is somewhat complicated by the presence of partial machines, such as $ab\Omega$.

A computation

$$e \parallel p \rightarrow e_1 \parallel p_1 \rightarrow \cdots \rightarrow e_n \parallel p_n$$

is *successful* if

(i) it is maximal, that is, there exists no $e' \parallel p'$ such that

$$e_n \parallel p_n \rightarrow e' \parallel p';$$

(ii) e_n can report success, that is, $e_n \xrightarrow{w} e'$ for some e'.

If, in addition, $p_n\downarrow$, then it will be called *strongly successful.*

Examples

(i) $abw\mathbb{O} \parallel (ab\mathbb{O} + ac\mathbb{O}) \rightarrow bw\mathbb{O} \parallel b\mathbb{O} \rightarrow w\mathbb{O} \parallel \mathbb{O}$ is strongly successful, whereas $abw\mathbb{O} \parallel (ab\mathbb{O} + ac\mathbb{O}) \rightarrow bw\mathbb{O} \parallel c\mathbb{O}$ is not.
(ii) $aw\mathbb{O} \parallel a\Omega \rightarrow w\mathbb{O} \parallel \Omega$ is successful. There is no strongly successful application of the experiment $aw\mathbb{O}$ to $a\Omega$. □

In general, there are various outcomes to the application of an experiment to a machine. So machines can be compared by tabulating those possible outcomes.

Definition 4.2.4

(i) p *may* e if there exists a successful computation from $e \parallel p$.
(ii) p *must* e if every computation from $e \parallel p$ is successful.
(iii) p *must*$\downarrow$ e if every computation for $e \parallel p$ is strongly successful.
(iv) For two closed terms p, q, $p \sqsubseteq_{op} q$, if for every experiment e,

  (a) p *may* e implies q *may* e,
  (b) p *must* e implies q *must* e,
  (c) p *must*$\downarrow$ e implies q *must*$\downarrow$ e.

Examples

(i) $ab\Phi + ac\Phi \underset{\sim}{\subset}_{\mathrm{op}} a(b\Phi + c\Phi)$ but $a(b\Phi + c\Phi) \not\underset{\sim}{\subset}_{\mathrm{op}} ab\Phi + ac\Phi$. To see the latter, consider the experiment $e = abw\Phi$. Then $a(b\Phi + c\Phi)$ *must* e whereas $ab\Phi + ac\Phi$ ~~*must*~~ e because of the computation $abw\Phi \,\|\, ab\Phi + ac\Phi \rightarrow bw\Phi \,\|\, c\Phi$.

(ii) Let p, q, e denote $ad\Phi + a(b\Phi + c\Phi) + ab\Phi$, $a(b\Phi + c\Phi) + ad\Phi$, $a(cw\Phi + dw\Phi)$, respectively. Then $p \underset{\sim}{\subset}_{\mathrm{op}} q$ but $q \not\underset{\sim}{\subset}_{\mathrm{op}} p$. The latter follows because q *must* e, whereas p ~~*must*~~ e.

(iii) $a\Omega \underset{\sim}{\subset}_{\mathrm{op}} a\Phi$ but $a\Phi \not\underset{\sim}{\subset}_{\mathrm{op}} a\Omega$ *because* $a\Phi$ *must*$\downarrow$ aw, whereas $aw\Phi \,\|\, a\Omega \rightarrow w\Phi \,\|\, a\Omega$ is not strongly successful. □

We are now ready to state the second characterization theorem, which states informally that the model AT differentiates between two machines if and only if they can be differentiated using experiments.

THEOREM 4.2.5. OPERATIONAL CHARACTERIZATION THEOREM

$$\mathcal{M}_{\mathrm{AT}}(p) < \mathcal{M}_{\mathrm{AT}}(q) \quad \textit{if and only if} \quad p \underset{\sim}{\subset}_{\mathrm{op}} q.$$

The proof is outlined in Section 5 and requires detailed knowledge of [15].

Each of the components of the definition of $\underset{\sim}{\subset}_{\mathrm{op}}$ is required for the theorem to be true. If the *may* component were omitted, then $ab\Phi + ac\Phi$ would be considered less then $ab\Phi$, whereas, interpreted in AT, this is not true. If the *must* component were omitted then $ab\Omega$ would be less than $ac\Phi + ab\Phi$, which is not true in AT. Without the *must*$\downarrow$ component, we would have Φ less than Ω, which is also false in AT. Finally, note that for fully guarded terms, which are interpreted as elements of FDAT the relations *must* and *must*$\downarrow$ coincide. These two variations are required only to deal with partial machines.

4.3. In this section we give an alternative characterization based on a modal language. It is formulated in terms of processes having certain properties (e.g., the ability to accept a string of symbols). These properties are defined as formulas of a property language $\mathscr{P}$. We then define *two* satisfaction relations $\models_o$, $\models_c$ over $\mathrm{CREC}_\Sigma \times \mathscr{P}$; $p \models_o \psi$ means that p may (optionally) satisfy the property expressed by ψ and $p \models_c \psi$ that p must (compulsorily) satisfy the property. The most interesting property is that of accepting a string. However, to capture the model exactly, we need to be able to ask simple questions about what happens after the string has been accepted. As explained in [19], when describing the failures model for processes, these questions are concerned with the possibility of deadlocks within the machine.

Let $\mathscr{L}$ be the language defined by

(i) true $\in \mathscr{L}$;
(ii) $\langle a \rangle \in \mathscr{L}$ for every $a \in A$;
(iii) $\psi_1, \psi_2\ \mathscr{L}$ implies $\psi_1 \vee \psi_2 \in \mathscr{L}$.

Let $\mathscr{P}$ be the set of formulas of the form

$$\textcircled{s}\psi \quad \text{where} \quad s \in A^* \quad \text{and} \quad \psi \in \mathscr{L}.$$

Definition 4.3.1

(i) $p \models$ true for every $p \in \mathrm{CREC}_\Sigma$.
(ii) $p \models \psi$ implies $p \models \psi \vee \psi'$, $p \models \psi' \vee \psi$.
(iii) $p \models \langle a \rangle$ if $p \xrightarrow{a} p'$ for some p'.

(iv) For $\textcircled{s}\psi \in \mathscr{P}$
  (a) $p \models_o \textcircled{s}\psi$ if $p \xrightarrow{s} p'$ for some p' such that $p' \models \psi$,
  (b) $p \models_c \textcircled{s}\psi$ if $p \downarrow s$ and $p \xrightarrow{s} p'$ implies $p' \models \psi$.

Examples

(i) Let p_1, p_2 denote rec $x.(abx + acx)$, rec $x.a(bx + cx)$, respectively. Then $p_2 \models_c \psi$, $p_1 \not\models_c \psi$, where ψ is $\textcircled{a}\langle b\rangle$. One can show, however, that for any $\psi \in \mathscr{P}$, $p_1 \models_o \psi$ if and only if $p_2 \models_o \psi$.
(ii) Let p_3 denote rec $x.a(bx \oplus cx)$. Then it is also true that $p_3 \not\models_c \psi$. One can also show that $p_1 \models_c \psi$ if and only if $p_3 \models_c \psi$, for any ψ in $\mathscr{P}$.
(iii) Let p_4 denote rec $x.a(bx + cx) + x$. Then $p_4 \not\models_c \psi$. The reason for this is that p_4 is a machine that is not fully defined because of the presence of unguarded recursion. It would be possible to "improve" on p_4 by giving it an extra capability to accept a. However, this capability might leave p_4 in a state that cannot accept the symbol b. In short, one can improve on p_4 in such a way that the improved machine need not accept ab.
(iv) $\Phi \models_c \textcircled{a}$ true, whereas $\Omega \not\models_c \textcircled{a}$ true. This is another example of the phenomenon in Example 3.
(v) $ab\Phi \not\models_o \textcircled{ac}$ true. Informally, we can deduce from this that, if the machine represented by the term $ab\Phi$ is offered the string ac, it will deadlock. □

The satisfaction relation $\models_c$ is more discriminating than $\models_o$. Note, however, that "must accept the string s" is not a primitive notion and cannot be represented by a formula in $\mathscr{P}$. Its effect can be obtained by a number of different formulas, whose composition depends on the composition of s itself. Informally, one can say, for example, that

$$\begin{aligned} p \text{ "must accept } a\text{" if and only if } & p \models_c \langle a\rangle, \\ p \text{ "must accept } ab\text{" if and only if } & p \models_c \langle a\rangle \\ \text{and } & p \models_c \textcircled{a}\langle b\rangle, \\ p \text{ "must accept } abc\text{" if and only if } & p \models_c \langle a\rangle \\ \text{and } & p \models_c \textcircled{a}\langle b\rangle \\ \text{and } & p \models_c \textcircled{ab}\langle c\rangle. \end{aligned}$$

Let

$$\mathscr{P}_o(p) = \{\psi \in \mathscr{P}, p \models_o \psi\},$$
$$\mathscr{P}_o(p) = \{\psi \in \mathscr{P}, p \models_c \psi\}.$$

The third main result of the paper is

THEOREM 4.3.2. MODAL CHARACTERIZATION THEOREM

$\mathscr{A}_{\mathrm{AT}}(p) < \mathscr{A}_{\mathrm{AT}}(q)$ *if and only if* $\mathscr{P}_o(p) \subseteq \mathscr{P}_o(q)$ *and* $\mathscr{P}_c(p) \subseteq \mathscr{P}_c(q)$.

We end this section with a discussion of the formulation of the language $\mathscr{P}$. If we omitted the formulas $\langle b\rangle$ from the language $\mathscr{L}$ we could not distinguish $ab\Phi + ac\Phi$ from $a(b\Phi + c\Phi)$. We also need $\vee$ as a connective in $\mathscr{L}$ for, otherwise, we could not distinguish $(a\Phi + b\Phi) \oplus c\Phi$ from $a\Phi \oplus (b\Phi + c\Phi)$. If $\textcircled{a}\langle b\rangle$ were not in $\mathscr{L}$, we would have $c\Phi$ approximating $a\Phi + c\Phi$. In short, all of the power of $\mathscr{L}$ is needed. On the other hand, if we try to obtain a more natural language by amalgamating the definitions of $\mathscr{L}$ and $\mathscr{P}$, then we gain too much power. For example, if we allow $\vee$ as a connection in $\mathscr{P}$, the axiom N3 would no longer be valid. If ψ denotes $\textcircled{\epsilon}(\textcircled{a}\Diamond_b \cup \textcircled{b}\Diamond_c)$, then

$$(ab\Phi + bc\Phi) + (ac\Phi + be\Phi) \models_c \psi,$$
$$(ab\Phi + bc) + (ac\Phi + be\Phi) \not\models_c \psi.$$

5. *Proof of the Characterization Theorems*

In this section we assume detailed knowledge of the notation and results of [15] and the algebraic constructions of [10], [12], and [34]. We let F_Σ be the finite terms over the operator set Σ and $<$ denote provability from the axioms of Section 3. To be more precise, $<$ is the least relation that satisfies these axioms and that is preserved by all of the operators in Σ. We let $=$ denote the equivalence generated by $<$.

5.1 This section is devoted to proving Theorem 3.1. In the terminology of [13] the initial Σ-cpo which satisfies the axioms can be described as $(F_\Sigma/<)^\infty$. We must show that this is isomorphic, as a Σ-cpo, to AT. The results of [15, sect. 4.3] enable us to give a much simpler description of $(F_\Sigma/<)$. Let N denote the set of normal forms, defined in [15, def. 4.3.2] and $<_1$ the relation over N defined in [15, def. 4.3.4]. Then, $(F_\Sigma/<)$ is isomorphic as a Σ-po to $N/<_1$. This follows from [15, corollary 4.3.8 and lemmas 4.3.5 to 4.3.7]. Consequently, it is sufficient to show that AT is isomorphic to $(N/<_1)^\infty$. Now both of these are algebraic Σ-cpos, which are determined completely by their finite elements. Let FAT denote the set of finite trees. Then we must show that FAT is isomorphic as a Σ-po to $N/<_1$.

LEMMA 5.1.1. *NT satisfies all of the axioms of Section* 3.

PROOF. It is sufficient to show that FAT satisfies the axioms. To prove that 2. and 1. hold, it is necessary to use induction on the size of trees. The axioms N1, N2, N3, $\Omega 1$, $\Omega 2$, all follow by simple calculations. This leaves D1, D2.

(D1) To show that $t_1 \oplus (t_2 + t_3)$ has the same acceptance set at the root (if it exists) as $(t_1 \oplus t_2) + (t_1 \oplus t_3)$ one needs that for arbitrary saturated sets $\mathscr{A}_1, \mathscr{A}_2, \mathscr{A}_3$:

$$c(\mathscr{A}_1 \cup \{A_1 \cup A_2,\ A_1 \in \mathscr{A}_2,\ A_2 \in \mathscr{A}_3\})$$
$$= \{B_1 \cup B_2,\ B_1 \in c(\mathscr{A}_1 \cup \mathscr{A}_2),\ B_2 \in c(\mathscr{A}_1 \cup \mathscr{A}_3)\}.$$

This is easily derived from the fact that

$$B \in c(\mathscr{A}_1 \cup \mathscr{A}_2) \qquad \text{if and only if} \qquad \cup\{A_i,\ i \in I\} \subseteq B \subseteq \cup\{A_j,\ j \in J\} \qquad (*)$$

where I, J are finite index sets and for each $i \in I$, $j \in J$

$$A_i \in \mathscr{A}_1 \cup \mathscr{A}_2, \qquad A_2 \cup \mathscr{A}_1 \cup \mathscr{A}_2.$$

Now to show that $t_1 \oplus (t_2 + t_3)$ is the same as $(t_1 \oplus t_2) + (t_1 \oplus t_3)$ is a simple case analysis on whether or not the intersections of the three sets $S(t_i)$, $i = 1, 2, 3$, are empty.

(D2) In this case also, the nontrivial part is to show that the acceptance sets on the respective roots are identical. The necessary result, which is derivable from $*$, is that for any saturated sets $\mathscr{A}_i$, $i = 1, 2, 3$,

$$\{A_1 \cup A_2,\ A_1 \in \mathscr{A}_1 \text{ and } A_2 \in c(\mathscr{A}_2 \cup \mathscr{A}_3)\} = c(\mathscr{B}_1 \cup \mathscr{B}_2)$$

where

$$\mathscr{B}_1 = \{B_1 \cup B_2,\ B_i \in \mathscr{A}_i\}, \qquad \mathscr{B}_2 = \{B_1 \cup B_3,\ B_i \in \mathscr{A}_i\}. \qquad \square$$

Every normal form (and indeed finite term) can be considered as an object in the language REC_Σ. This gives a natural mapping, h, from N to FAT, defined by

$$h(n) = \mathscr{M}_{\mathrm{AT}}(n).$$

LEMMA 5.1.2. *h induces a Σ-homomorphism from $N/<_1$ to FAT.*

PROOF

(i) We must show that h preserves $<_1$. Suppose $n <_1 n'$. Then $n < n'$ and, by the previous lemma, it follows that $h(n) < h(n')$.

(ii) We must show that h preserves every operator in Σ, that is, $\mathrm{op}_{\mathrm{AT}}(h(n_1), \ldots, h(n_k)) = h(\mathrm{op}_N(n_1, \ldots, n_k))$. If nf denotes the normal form function on finite terms, then op_N can be defined by $\mathrm{op}_N(n_1, \ldots, n_k) = \mathrm{nf}(\mathrm{op}(n_1, \ldots, n_k))$. Now $h(\mathrm{op}(n_1, \ldots, n_k)) = \mathrm{op}(h(n_1), \ldots, h(n_k))$ and, therefore, the result follows once more by the previous lemma. $\square$

We now define an inverse for h. Let $k\colon \mathrm{FAT} \to N$ be defined by

(i) if $t(\epsilon)$ is open, then

$$k(t) = \Omega + \Sigma\{ak(t/a),\ a \in S(t)\}.$$

(ii) if $t(\epsilon)$ is closed, then

$$k(t) = \textstyle\bigoplus\{\Sigma\{ak(t/a),\ a \in L\}\ L \in \mathscr{A}(t(\epsilon))\}.$$

LEMMA 5.1.3. *k induces a Σ-homomorphism from FAT to $N/<_1$.*

PROOF

(i) It is necessary to show that, if $t_1 \sqsubseteq t_2$, then $k(t_1) <_1 k(t_2)$. A simple case analysis will show that $k(t_1) <_1 k(t_2)$ and the result then follows by induction.

(ii) It is also necessary to show that

$$k(\mathrm{op}_{\mathrm{AT}}(t_1, \ldots, t_n)) = \mathrm{op}_{\mathrm{N}}(k(t_1, \ldots, t_n));$$

that is,

$$k(\mathrm{op}_{\mathrm{AT}}(t_1, \ldots, t_n)) = \mathrm{nf}(\mathrm{op}(k(t_1, \ldots, t_n))).$$

For the operator a. This is straightforward, but for the binary operators + and $\oplus$ this requires structural induction on terms and knowledge of the procedure for reducing terms to normal forms, which is given in [15, Appendix 1.] $\square$

COROLLARY 5.1.4. (THEOREM 3.1). *$(F_\Sigma/<)^\infty$ is isomorphic to AT as a Σ-cpo.*

PROOF. As noted above, it is sufficient to show that $(N/<_1)$ is isomorphic to FAT. This now follows from the previous two lemmas since h and k are inverses. $\square$

The notion of testing used in [14] is slightly different than that outlined in Section 4 of this paper. This led to the relation $\sqsubseteq_1^\dagger$ between terms, which has a different formulation from our relation $\sqsubseteq$ op. However, the differences are not significant, since they can both be shown to coincide with a third relation $\sqsubseteq_1'$ defined in [15]. Using this fact, the proof of the Operational Characterization Theorem is straightforward.

COROLLARY 5.1.5. (THEOREM 4.2.5)

$$\mathscr{M}_{\mathrm{AT}}(p) < \mathscr{M}_{\mathrm{AT}}(q) \qquad \textit{if and only if} \qquad p \underset{\sim}{\sqsubset}_{op} q.$$

PROOF. Consider the relation $\underset{\sim}{\sqsubset}'_1$ between terms defined in [15, sect. 4.1]. From [15, theorems 3.1.2 and 4.1.1] and our Algebraic Characterization Theorem, it follows that

$$\mathscr{M}_{\mathrm{AT}}(p) < \mathscr{M}_{\mathrm{AT}}(q) \qquad \text{if and only if} \qquad p \underset{\sim}{\sqsubset}' q.$$

However, the proof of [15, theorem 4.1.1] can easily be adapted to show that

$$p \underset{\sim}{\sqsubset}_{\mathrm{op}} q \qquad \text{if and only if} \qquad p \underset{\sim}{\sqsubset}'_1 q. \qquad \square$$

5.2. In this section, we prove the modal characterization theorem. We rely very heavily on the results of [15, sect. 4.1], where an alternative characterization of the model AT is given in terms of preorders $\underset{\sim}{\sqsubset}'_i$, $i = 1, 2, 3$. In fact, the modal characterization theorem will follow from the fact that $p \underset{\sim}{\sqsubset}'_3 q$ if and only if $\mathscr{P}_{\mathrm{o}}(p) \subseteq \mathscr{P}_{\mathrm{o}}(q)$ and $p \underset{\sim}{\sqsubset}'_2 q$ if and only if $\mathscr{P}_{\mathrm{c}}(p) \subseteq \mathscr{P}_{\mathrm{c}}(q)$.

PROPOSITION 5.2.1. *If $\mathscr{P}_o(p) \subseteq \mathscr{P}_o(q)$ and $\mathscr{P}_c(p) \subseteq \mathscr{P}_c(q)$, then $\mathscr{M}_{AT}(p) < \mathscr{M}_{AT}(q)$.*

PROOF. We know from [15] that $\mathscr{M}_{\mathrm{AT}}(p) < \mathscr{M}_{\mathrm{AT}}(q)$ if and only if $p \underset{\sim}{\sqsubset}'_1 q$. However, $\underset{\sim}{\sqsubset}'_1$ is simply the intersection of $\underset{\sim}{\sqsubset}'_3$ and $\underset{\sim}{\sqsubset}'_2$. Consequently, it is sufficient to prove that $p \underset{\sim}{\sqsubset}'_3 q$ and $p \underset{\sim}{\sqsubset}'_2 q$. We leave it to the reader to prove that $\mathscr{P}_{\mathrm{o}}(p) \subseteq \mathscr{P}_{\mathrm{o}}(q)$ implies $p \underset{\sim}{\sqsubset}'_3 q$, and we show that $\mathscr{P}_{\mathrm{c}}(p) \subseteq \mathscr{P}_{\mathrm{c}}(q)$ implies $p \underset{\sim}{\sqsubset}'_2 q$. Suppose $p \downarrow s$. We show by induction on s that

(i) $q \downarrow s$,
(ii) $S(s, q) \subseteq S(s, p)$, and
(iii) $\mathscr{A}(s, q) \not\subseteq \mathscr{A}(s, p)$.

(a) s is ϵ.
   (i) Let a be such that $a \notin S(\epsilon, p)$ and b not appear in q. Then vacuously $p \models_{\mathrm{c}} \textcircled{a}\langle b\rangle$. Since $\mathscr{P}_{\mathrm{c}}(p) \subseteq \mathscr{P}_{\mathrm{c}}(q)$, it follows that $q \models_{\mathrm{c}} \textcircled{a}\langle b\rangle$, that is, $q\downarrow$.
   (ii) Suppose $a \in S(\epsilon, q)$. If $a \notin S(\epsilon, p)$ we can proceed as in (i) to obtain a contradiction.
   (iii) Let $A \in \mathscr{A}(\epsilon, q)$. Suppose that for every $B \in \mathscr{A}(\epsilon, p)$ there exists some a_B such that $a_B \in B$, $a_B \notin A$. Then $p \models_{\mathrm{c}} \textcircled{\epsilon}\psi$, $q \models_{\mathrm{c}} \textcircled{\epsilon}\psi$, where ψ denotes $V\{\langle a_B\rangle, B \in \mathscr{A}(\epsilon, p)\}$. This, however, contradicts the fact that $\mathscr{P}_{\mathrm{c}}(p) \subseteq \mathscr{P}_{\mathrm{c}}(q)$. It follows that, for some $B \in \mathscr{A}(\epsilon, p)$, $B \subseteq A$; that is, $\mathscr{A}(s, q) \subseteq \mathscr{A}(s, p)$.

(b) s is $s'a$.
   If $S(s'a, p)$ is empty (i.e., $a \notin S(s', p)$), then it follows by induction that $S(s'a, q)$ is empty and the results are vacuously true. So we can assume $S(s'a, p)$ is not empty and, in this case, the proof is similar to part (a) with $s'a$ in place of ϵ. $\square$

PROPOSITION 5.2.2. *If $\mathscr{M}_{AT}(p) < \mathscr{M}_{AT}(q)$, then $\mathscr{P}_o(p) \subseteq \mathscr{P}_o(p)$ and $\mathscr{P}_c(p) \subseteq \mathscr{P}_c(q)$.*

PROOF. Once more, we leave it to the reader to show that $p \underset{\sim}{\sqsubset}'_3$ implies $\mathscr{P}_{\mathrm{o}}(p) \subseteq \mathscr{P}_{\mathrm{o}}(q)$ and we prove $p \underset{\sim}{\sqsubset}'_2$ implies $\mathscr{P}_{\mathrm{c}}(p) \subseteq \mathscr{P}_{\mathrm{c}}(q)$. The proposition then follows from the characterization of AT in [15]. Suppose $p \underset{\sim}{\sqsubset}'_2 q$ and $p \models_{\mathrm{c}} \textcircled{s}\psi$. We show $q \models_{\mathrm{c}} \textcircled{s}\psi$.

(i) ψ is true.
Since $p \downarrow s$, if and only if $p \models_c Ⓢ$ true and $p \;\underset{\approx}{\sqsubset}'_2\; q$, it follows that $q \models_c Ⓢ$ true.

(ii) Otherwise, ψ can be taken to be $\vee\{\langle a_i \rangle, 1 \le i \le n\}$. We must show that, if $q \xrightarrow{s} q'$, then $q' \xrightarrow{a_i} q''$ for some i, $1 \le i \le n$. Since $p \;\underset{\approx}{\sqsubset}'_2\; q$, there exists some p' such that $S(p') \subseteq S(q')$. It follows that $p' \xrightarrow{a_i} p''$ for some i and, therefore, $q' \xrightarrow{a_i} q''$. □

The modal characterization theorem now follows from these two propositions.

6. *Conclusion*

We have presented a simple model for nondeterministic machines called AT, which is based on particular kinds of trees. By defining some basic operations on AT, we have seen that it can be characterized by a simple set of axioms. Finally, we have tried to motivate the model from an operational point of view by examining the semantics of a simple language in the model. We have shown that this semantics reflects behavioral properties of programs that can be written in a simple modal language.

Our model represents a particular view of the behavior of machines. A very different viewpoint motivates another type of model, based on the notion of bisimulation. These models appear in [16] and [24–26]. In general, they are much more discriminating than AT. For example, if applied to our language of Section 4, they would differentiate

$$a(bc① + bd①), \qquad abc① + abd①.$$

Some of these models, such as [24–26], are based on the notion of equivalence between processes. However, these models do not have any algebraic characterization or representation that is independent of their operational definitions. For example, in [24], synchronization trees are not fully abstract with respect to strong equivalence. More recently, attempts have been made to use metric spaces to elucidate these kinds of equivalences [1, 2, 6, 11]. For example, in [1], complete metric spaces that satisfy some natural axioms are discussed. So far, the operational significance of these models have not been investigated, but they should illuminate the various modifications of the basic notion of observational equivalence from [16] and [24].

In [14] and [17], observational preorders, based on the notion of bisimulation, are studied. These lead to fully abstract models, which are, in fact, term models. These may be characterized equationally, but they have no satisfactory representation. All we know about these models is that they can be obtained by factoring trees using equations. This factoring process is rather complicated, and one has no idea what the objects in the model actually look like.

In [15], six different models for processes were introduced, three synchronous models I_1, I_2, I_3 and three asynchronous models J_1, J_2, J_3. We have chosen to concentrate on I_1 in this paper and, in fact, we have shown that AT is a representation of I_1. The other two synchronous models I_2 and I_3 also have convenient representations, which we now outline. Let WAT (weak acceptance trees) be the set of trees with *no* closed nodes. Elements of WAT are essentially sets of sequences, and the ordering given by Definition 2.2.1 is simply subset inclusion. Thus, WAT is a Σ-cpo, which is isomorphic (as a cpo) to the traces model of [19]. Its algebraic characterization can be obtained in the same way as that of AT by adding the axiom

$$X \sqsubseteq X \oplus Y. \tag{WN3}$$

As might be expected, the presence of this axiom is quite powerful. For example, one can use it in conjunction with the other axioms to prove

$$X \oplus Y = X + Y, \qquad a(X + Y) = aX + aY.$$

An operational characterization of the denotational semantics of our language in WN3 can be obtained as in Theorem 4.2.5 by omitting the *must* and *must*$\downarrow$ clauses from the definition of $\sqsubseteq$op.

On the other hand, let SAT (strong acceptance trees) be the set of all trees in AT with the property that *only* leaves may be open. SAT can be made into a cpo by omitting clauses (i) and (ii) from Definition 2.2.1. The algebraic characterization of SAT is obtained by adding the axiom

$$X \oplus Y \sqsubseteq X. \tag{SN3}$$

The axiom can be used in conjunction with the others to derive

$$X \oplus \Omega = \Omega, \qquad X + \Omega = \Omega,$$

which shows that, in this model, these operators are strict. Similarly, an operational characterization can be obtained by omitting the *may* clause from the definition of $\sqsubseteq_{\text{op}}$.

The asynchronous models J_i (and, indeed, RT_i of [8]) are very similar to those models. The differences occur because, in these models, it is necessary to model the internal behavior of processes definable in asynchronous CCS.

The refusal sets model for communicating processes was introduced in [5] and is discussed, together with its variations, in [3], [19], [22], [23], [27], and [29]. This is quite similar to AT, except that they label the nodes of the trees with refusal sets instead of acceptance sets. Consequently, this model is much larger than necessary in that there are many elements that cannot be defined in the languages they consider. The refusal sets model lack an equational characterization and because of the point just mentioned would be difficult to obtain.

If we assume that the set of actions is *finite* then one can prove that the refusal sets model is isomorphic to SAT as cpos. Moreover, in this case, the usual operations on refusal sets [5] are quite similar to ours. Their $a \rightarrow$ is the same as our prefixing and their $\Box$ coincides with $+$. However, their operation $\sqcap$ differs from $\oplus$ in the way that they treat the least element. As indicated previously, $\oplus$ is strict on SAT, whereas $\sqcap$ is not strict. It has been suggested in [3] and [7] that the properties of the least element in the refusal set model, CHAOS, be changed so as to make it more amenable to equational reasoning. The net effect would be to make $\oplus$ and $\sqcap$ coincide, thereby making SAT and this model isomorphic as Σ-cpos (at least, when A is finite). In general, when A is infinite, the refusal set model could be modified so that it only contains elements that, when viewed as trees, have bounded out-degree. So they correspond to processes that exhibit bounded nondeterminism. This modified model, bounded by refusal sets, is then isomorphic to SAT. For details, see [7]. Finally, we should point out that no connection has been made between the original refusal sets model and the operational behavior of machines. However, this point is discussed in [3] and [27].

A modal characteristic of observational equivalence is given in [16] and is further elaborated in [4] and [32]. Our modal characterization theorem is modeled on [35], but is actually a minor variation on a result first proved by C. Stirling [33].

REFERENCES

(Note: References [18] and [28] are not cited in text.)

1. Bergstra, J.A., and Klop, J.W. Fixed point semantics in process algebras. Tech. Rep. IW 206/82, Mathematisch Centrum, Amsterdam, The Netherlands, 1982.
2. Bergstra, J.A., and Klop, J.W. Process algebra for communication and mutual exclusion. Tech. Rep. IW 218/83, Mathematisch Centrum, Amsterdam, The Netherlands, 1983.
3. Brookes, S.D. A model for communicating sequential processes. Ph.D. dissertation. Oxford Univ., Oxford, England, 1983.
4. Brookes, S.D. and Rounds, W.C. Behavioural equivalence relations induced by programming logic. In *Proceedings of ICALP 1983*, Lecture Notes in Computer Science, vol. 154. Springer-Verlag, New York, 1983.
5. Brooks, S.D., Hoare, C.A.R., and Roscoe, A.W. A theory of communicating sequential processes. *J. ACM 31*, 3 (July 1984), 560–599.
6. de Bakker, J.W., and Zucker, J.I. Processes and the denotational semantics of concurrency. Tech. Rep. IW 209/82, Mathematisch Centrum, Amsterdam, The Netherlands, 1982.
7. De Nicola, R. A complete set of axioms for a theory of communicating sequential processes. In *Proceedings of FCT '83*. Lecture Notes in Computer Science, vol. 158. Springer-Verlag, New York, 1983, pp. 115–126; *Inf. Control*, to appear.
8. De Nicola, R., and Hennessy, M. Testing equivalences for processes. *Theor. Comput. Sci. 34*, 1, 2 (1984), 83–135.
9. Ginsburg, S., and Rice, H.G. Two families of languages related to ALGOL. *J. ACM 9*, 3 (July 1962), 350–371.
10. Goguen, J.A., Thatcher, J.W., Wagner, E.G., and Wright, J.B. Initial algebra semantics and continuous algebras. *J. ACM 24*, 1 (Jan. 1977), 68–95.
11. Golson, W.G., and Rounds, W.C. Connections between two theories of concurrency: Metric spaces and synchronisation trees. *Inf. Control 57* (1983), 102–124.
12. Gordon, M. *The Denotational Description of Programming Languages*, Springer-Verlag, New York, 1979.
13. Guessarian, I. *Algebraic semantics*. In Lecture Notes in Computer Science, vol. 99. Springer-Verlag, New York, 1981.
14. Hennessy, M. A term model for synchronous process. *Inf. Control 51*, 1 (Oct. 1981), 58–75.
15. Hennessy, M. Synchronous and asynchronous experiments on processes, *Inf. Control 59*, 1-3 (1983), 36–83.
16. Hennessy, M., and Milner, R. Algebraic laws for nondeterminism and concurrency, *J. ACM 32*, 1 (Jan. 1985), 137–162.
17. Hennessy, M., and Plotkin, G. A term model for CCS. In Lecture Notes in Computer Science, vol. 88. Springer-Verlag, New York, 1980, pp. 261–274.
18. Hoare, C.A.R. Communicating sequential processes. *Commun. ACM 21*, 8 (Aug. 1978), 666–676.
19. Hoare, C.A.R. A model for communicating sequential processes. Tech. Monograph Prg.-22. Comput. Lab., Univ. of Oxford, Oxford, England, 1981.
20. Hopcroft, J., and Ullman, J. *Formal Languages and Their Relation to Automata*. Addison-Wesley, Reading, Mass., 1969.
21. Howie, J. An Introduction to Semigroup Theory. Academic Press, Orlando, Fla., 1976.
22. Kennaway, J.K. Formal semantics of nondeterminism and parallelism, Ph.D. dissertation. Univ. of Oxford, Oxford, England, 1981.
23. Kennaway, J.K., and Hoare, C.A.R. *A theory of nondeterminism*. In *Proceedings of ICALP 1980*. Lecture Notes in Computer Science, vol. 85. Springer-Verlag, New York, 1980, pp. 338–350.
24. Milner, R. *A calculus of communicating systems*. In Lecture Notes in Computer Science, vol. 92, Springer-Verlag, New York, 1980.
25. Milner, R. On relating synchrony and asynchrony, Tech. Rep. CSR-75-80, Univ. of Edinburgh, Edinburgh, Scotland, 1980.
26. Milner, R. Calculi for synchrony and asynchrony. *Theor. Comput. Sci. 25*, (1983), 267–310.
27. Olderog, E.R., and Hoare, C.A.R. *Specification-oriented semantics for communicating processes*. In Lecture Notes in Computer Science, vol. 154. Springer-Verlag, New York, 1983.
28. Plotkin, G. A structural approach to operational semantics, Lecture Notes. Aarhus University, Aarhus, Denmark, 1981.
29. Rounds, W.C., and Brookes, S.D. Possible futures, acceptances, refusals, and communicating processes. In *Proceedings of the 22nd Foundations of Computer Science Annual Symposium* (Nashville, Tenn., Oct.). IEEE, New York, 1981.

30. SALOMAA, A. Two complete axiom systems for the algebra of regular events. *J. ACM 13*, 1 (Jan. 1966), 158–169.
31. SCOTT, D. Data types as lattices. *SIAM J. Comput. 5* (1976), 522–587.
32. STIRLING, C. A proof theoretic characterisation of observational equivalence, Tech. Rep. CSR-132-83. Univ. of Edinburgh, Edinburgh, Scotland, 1983; *Theor. Comput. Sci.*, to be published.
33. STIRLING, C. Unpublished manuscript.
34. STOY, J. *Denotational Semantics: The Scott–Strachey Approach to Programming Language Theory.* MIT Press, Cambridge, Mass., 1977.
35. WINSKEL, G. On powerdomains and modality. *Theor. Comput. Sci. 36* (1985), 127–137.

RECEIVED JULY 1983; REVISED FEBRUARY 1985; ACCEPTED MARCH 1985