Distributed Event Algebras*

A. KLAPPER

University of Manitoba, Department of Computer Science,
Winnipeg, Manitoba R3T 2N2, Canada

Received October 12, 1989; revised February 25, 1990

A general model, the Distributed Event Algebra or D-algebra, for distributed computation is developed, generalizing Lynch's Event State Algebras. Such models are essential to the construction of effective and convincing proofs of distributed algorithms. D-algebras are designed to allow hierarchical proof techniques. Two notions of mapping between D-algebras are defined. One operates on the level of uninterpreted actions, the other on the level of actions with interpretations as operators on states. A hierarchical proof of correctness using D-algebras consists of the construction of a series of D-algebras, from high level to low level, connected by a series of correctness preserving maps, and a proof of the correctness of the high level D-algebra. D-algebras also incorporate the notion of execution of a system as a partially ordered set of actions, thus reducing overspecification of executions. This results in the state history of a system under a particular execution being modeled as a directed graph, thus capturing all possible state sequences in a single structure.

1. INTRODUCTION

As distributed computing becomes more widespread, the need is emerging for formal proof techniques for distributed algorithms. Correctness questions of many algorithms are too complex to be settled easily. For example, a reasonable proof of correctness of the distributed minimum cost spanning tree algorithm of Gallager et al. [3] first appeared in [14]. A reasonable intuitive argument can often be made straightforwardly, while rigorous proof requires massive effort and great complexity. Existing proofs are often forced to operate at very low levels of systems, thus losing the sense of the original intuitive arguments. Moreover, it is often difficult to state precisely what is meant by correctness in the absence of good mathematical models. Thus it is vital to develop models which abstract away irrelevant details (such as the precise times at which events occur) and allow desirable properties to be expressed in an abstract setting with ample tools for building proofs.

Several abstract models for distributed systems have enjoyed wide success, notably Hoare's Communicating Sequential Processes [6] and Petri Nets [10]. More recently, Lynch [7] successfully used an algebraic model called Event State Algebras to prove the correctness of a concurrency control algorithm. Essential to the proof technique was the idea of building a series of algebras connected by

* The research was performed while the author was at Northeastern University.
correctness preserving maps (hierarchical proof). Thus the correctness of the initial high level model of the algorithm implied the correctness of the final low level model. This approach has been further developed in various directions in [2, 4, 8, 9, 13]. A limited notion of hierarchical proof of distributed systems was also developed by Sere [12], but the underlying model was quite different.

This paper extends the Event State Algebra model to a more general model for distributed computation called Distributed Event Algebras, or D-algebras. D-algebras come with two notions of mapping to be used in hierarchical proofs. One (interpretations) operates on the level of actions (mapping executions to executions). The other (possibilities maps) operates on both the levels of actions and states (mapping each state of one system to a set of possible states of another).

An important feature of the D-algebra model is that it abstracts away considerations of precise time and deals instead with the order of operations. In fact, the order of operations local to different nodes may not need to be specified. In correctness proofs for sequential algorithms this issue does not arise but in distributed algorithms proofs often become bogged down in tricky timing considerations. D-algebras are less restrictive than many models of distributed computation in that they keep track only of those timing considerations that are truly relevant.

A D-algebra consists of a set of nodes, a set of states, and a set of partial operations from the global state space to itself. Each operation has a locality (the node performing the operation), and certain pairs of operations are said to be serial. That is, their order of execution is of importance. An execution of a D-algebra, rather than being a sequence, is a partially ordered set of operations. These definitions are made precise in Sections 2 and 3. The notion of executions of distributed systems as partially ordered sets has appeared in the context of temporal logic [11].

In Section 4 we construct the history of an execution, analogous to the sequence of states that occur in a sequential system. For D-algebras, however, a history is a labeling of the finite initial subsets of the execution. This is the minimal information required to represent the states that occur as a system executes. In Section 6 we show that an execution can be replaced by one with minimal restrictions on the order of operations. In the final section we define mappings between D-algebras that can be used in hierarchical correctness proofs, and derive some of their basic properties.

One difficulty in modeling distributed systems is that a communication is really two operations—transmission and reception—which may be interlaced with other operations. Thus the execution of an operation at a given point in time may depend on the state of the system at a previous time. We give an example in Section 5 of how this two-step nature of communication can be modeled using D-algebras.

2. DISTRIBUTED EVENT ALGEBRAS

In this section an algebraic model for distributed computing is presented. The purpose of this model is to provide a sufficiently abstract environment to carry out
proofs of properties of distributed systems. In particular, precise details of the
timing of events in such a system have been abstracted away, while the minimal
information required to order events is maintained. The model includes notions of
mappings between systems, with which properties can be pulled back or pushed
forward from simpler systems to more complex ones. One interesting feature is that
executions are viewed as partially ordered sets rather than sequences of operations.
The idea is that the order of execution of a pair of operations is often irrelevant.

**Definition 1.** A distributed event algebra (or D-algebra) is a 6-tuple \( A = \langle P, S, I, \mathcal{F}, \lambda, \sim \rangle \), where

1. \( P \) is a finite set of nodes.
2. \( S \) is a finite set of states.
3. \( I \subseteq S \) (the initial states).
4. \( \mathcal{F} \) is a set of partial maps from \( S \) to itself (operations). The domain of an
   operation \( f \in \mathcal{F} \) will be denoted by \( \delta(f) \).
5. \( \lambda: \mathcal{F} \rightarrow P \) is called the locality map.
6. \( \sim \) is a symmetric binary relation on \( \mathcal{F} \). If \( f \sim f' \), then \( f \) is said to be serial
to \( f' \). The complement of \( \sim \) is denoted by \( \not\sim \).

A distributed event algebra must satisfy

\( D1: \) If \( f, g \in \mathcal{F} \), and \( \lambda(f) = \lambda(g) \) then \( f \sim g \).
\( D2: \) If \( f \not\sim g \) and \( s \in \delta(f) \), then \( s \in \delta(g) \) if and only if \( f(s) \in \delta(g) \).
\( D3: \) If \( f \not\sim g \) and \( s \in \delta(f) \cap \delta(g) \) then \( f(g(s)) = g(f(s)) \).

As an example of how D-algebras can model distributed systems, we may con-
sider \( P \) to be a set of processes in a distributed system. Each process \( p \) has a set
of states \( S_p \), and \( P = \prod_{\rho \in P} S_p \). Each \( p \) also has a set of operations it can perform
to change states—those operations \( f \) whose locality \( \lambda(f) \) is \( p \). For any given state
\( s_p \) at \( p \), a fixed set of operations is enabled, independent of the states of other
processes—those \( f \) for which \( \lambda(f) = p \) and \( \exists t \in \delta(f): t_p = s_p \) or, equivalently,
\( \forall t: t_p = s_p \Rightarrow t \in \delta(f) \). Execution of an operation at \( p \), however, may change the state
at other processes, typically by transmission of messages over communications
channels.

The semantics of distributed systems are, in general, highly independent of the
order of execution of many operations. In a D-algebra, operations for which the
order of execution is significant are said to be serial. The first three axioms give
minimum requirements for operations to be serial. By \( D1 \), operations are serial if
they have the same locality. \( D2 \) says they are serial if they affect each other's
enabling. \( D3 \) says they are serial if the final state depends on the order in which
they were executed.

In this setting we might add axioms to reflect the local nature of operations. For
element, analogous to Lynch's local domains and local changes axioms [7], we
might have
D4: If \( s \in \delta(f) \) and \( s_{\lambda(f)} = t_{\lambda(f)} \), then \( t \in \delta(f) \) (local domains).

D5: For any \( p \in P \) and \( f \in \mathcal{F} \), if \( s, t \in \delta(f) \), \( s_{\lambda(f)} = t_{\lambda(f)} \), and \( s_p = t_p \), then \( f(s)_p = f(t)_p \) (local changes).

A consequence of the local nature of the execution of operations in such a system is that a process must be able to tell from its local state whether an operation is enabled. This is implied by axiom D4. The local state change at a node \( p \) caused by an operation executed at a node \( q \) should depend only on the original states at \( p \) and \( q \), thus D5.

In such a setting, the set \( P \) of processes is generally given, possibly with some restrictions on which processes may communicate (hence which processes can affect each other's states). Specification of a particular type of system consists of imposing some restrictions on local state spaces and operations. For example, to model a system of communicating processes with bounded buffers, one needs local states that reflect the states of the buffers, operations that perform message transmission, and a restriction that these are the only operations that affect nonlocal states. The operations of transmission of a message and reception of a message would be required to be serial. We see an example of such a model in Section 5.

Specification of an algorithm then consists of the further specification of the state spaces and the local operations with their domains. Such a system executes nondeterministically, hence correctness must generally be stated in the form "for all possible executions, property \( X \) holds." We make the notion of an execution of a D-algebra precise in the next section.

In case \( P \) is a singleton set and \( I \) contains a single element this definition is equivalent to Lynch's definition of an Event State Algebra. In general, however, even if every pair of operations is serial, D-algebras plus axioms D4 and D5 is a more restrictive model than Lynch's Event Algebras Distributed over \( P \), due to axiom D5 when \( p \neq \lambda(f) \). Moreover, the notion of serial operations is new here.

3. Executions of a D-Algebra

The operations of a sequential system are executed in some (total) order. The operations of a distributed (nonsequential) system only seem to be executed in order when viewed in a particular space–time coordinate system. In fact, if we change space–time coordinates we may see a different sequence of events. Thus it is appropriate that an execution of a D-algebra be as independent of space–time coordinates as possible. This motivates the following definition.

**Definition 2.** An execution of a D-algebra \( A \) is a triple \( \mathcal{E} = (E, <, l) \), where \( E \) is a set, \(<\) is an irreflexive partial order on \( E \), and \( l: E \to \mathcal{F} \) such that

E1: For every \( x \in E \), \( \{ y \in E : y < x \} \) is finite.

E2: \( l(x) \sim l(y) \) implies \( x = y \) or \( x < y \) or \( y < x \).
Thus \((E, <)\) consists of a partially ordered set of actions, and \(l\) determines an interpretation of these actions by operations of the D-algebra. From these axioms it can be inferred that given a node \(p\) and execution \(\mathcal{E}\), the operations of \(\mathcal{E}\) that are local to \(p\) are executed sequentially. If \(\mathcal{E} = (E, <, l)\) is an execution and \(x \in E\), we sometimes abuse the notation and say \(x \in \mathcal{E}\).

**Lemma 1.** \(\mathcal{E}\) is an execution of \(A\) and \(p \in P\), then \(\mathcal{E}_p = \{ x \in E : \lambda(l(x)) = p \}\) is totally ordered by \(<\).

**Proof.** Let \(x, y \in \mathcal{E}_p\). \(\lambda(l(x)) = \lambda(l(y))\) so \(l(x) \sim l(y)\) by D1. Therefore, \(x = y \lor x < y \lor y < x\) by E2.

Recall that a partially ordered set is well founded if it has no infinite decreasing sequence, and hence that every subset has at least one initial element. We next show that executions are well founded. This implies that \(\mathcal{E}\) is filtered by its finite initial subsets which will allow us to define the history of an execution on a state.

**Proposition 1.** Every execution \(\mathcal{E} = (E, <, l)\) of a D-algebra is well founded. Every subset of \(E\) has finitely many initial elements.

**Proof.** That \(\mathcal{E}\) is well founded follows immediately from axiom E1.

Let \(G \subseteq E\). Suppose \(B = \{ x \in G : \neg \exists y \in G : y < x \}\), the set of initial elements in \(G\), is infinite. \(P\) is finite so there is a \(p \in P\) and an infinite subset \(B'\) of \(B\) such that for every \(x \in B'\), \(\lambda(l(x)) = p\). By Lemma 1, \(B'\) is totally ordered by \(<\), hence contains at most one initial element, a contradiction.

Note that if \(\mathcal{E}\) is an execution, then the function \(v(x)\) whose value is the number of predecessors of \(x\) is a *bound function*, in the sense of Gries [5], that is, \(x < y\) implies \(v(x) < v(y)\).

### 4. The History of an Execution

Executions are objects on the level of uninterpreted actions, independent of any interpretation of the operations as functions on state spaces. In this section we interpret executions as operations on the states of D-algebras. In sequential systems the notion of a state sequence makes sense. In an execution of such a system, from a given state there is always a well defined next operation, and hence a well defined next state. In a distributed system this is no longer true. Just as the operations only occur in a partial order, so must the states occur in a partial order. For a given execution, from a given state there may be several possible next states.

Recall that a subset \(F\) of a partially ordered set \((E, <)\) is initial if for all \(x \in E\), and \(y \in F\), if \(x < y\) then \(x \in F\). Each initial subset \(F\) determines an execution, and at times we do not distinguish \(F\) from the execution it determines. We denote by \(N_\mathcal{E}\) the set of finite initial subsets of \(\mathcal{E}\). \(N_\mathcal{E}\) is partially ordered by inclusion. Each element of \(N_\mathcal{E}\) corresponds to a possible finite stage in the execution of \(\mathcal{E}\).
If \( f \) is a partial function from \( S \) to itself, we extend \( f \) to a total function as follows. Let \( \omega \) be an element not in \( S \). Define \( f(x) = \omega \) if \( x \notin \delta(f) \) or \( x = \omega \).

**Theorem 1.** For every execution \( \mathcal{E} = (E, <) \) there is a unique function \( \sigma_{\mathcal{E}} : N_\mathcal{E} \to \{ f : S \cup \{ \omega \} \to S \cup \{ \omega \} \} \) such that

1. \( \sigma_{\mathcal{E}}(\emptyset) = \text{identity} \),
2. \( \sigma_{\mathcal{E}}(F) = l(x) \circ \sigma_{\mathcal{E}}(F - \{ x \}) \) whenever \( x \in F \), \( F \in N_\mathcal{E} \), and \( F - \{ x \} \in N_\mathcal{E} \).

**Proof.**

1. Uniqueness: If \( \sigma_{\mathcal{E}} \) is such a function, and \( F \in N_\mathcal{E} \), let \( x_1, \ldots, x_n \) be a total ordering of \( F \) compatible with \( < \). Thus for each \( i = 1, \ldots, n \), \( \{ x_1, \ldots, x_i \} \in N_\mathcal{E} \). Then \( \sigma_{\mathcal{E}}(F) = l(x_n) \circ l(x_{n-1}) \circ \cdots \circ l(x_1) \).

2. Existence: We can go from one total ordering of the elements of \( F \) compatible with \( < \) to another by repeatedly interchanging consecutive pairs, all the while maintaining compatibility with \( < \). Such a pair \( x, y \) must have nonserial labels by axiom E2. A state \( s \) is in the domain of the composition of \( l(x) \) and \( l(y) \) in one order if and only if it is in the domain of their composition in the other order, by axiom D2. In that case, \( l(x) \circ l(y)(s) = l(y) \circ l(x)(s) \) by axiom D3. If \( s \) is not in the domain of this composition, then \( l(x) \circ l(y)(s) = l(y) \circ l(x)(s) = \omega \). Thus (extended) \( l(x) \) and \( l(y) \) commute on all states. It follows that the definition of \( \sigma_{\mathcal{E}}(F) \) in the first part of this proof is independent of the choice of ordering of \( F \), so \( \sigma_{\mathcal{E}} \) is well defined.

The function \( \sigma_{\mathcal{E}} \) is referred to as the history function of \( \mathcal{E} \). We say that an execution \( \mathcal{E} \) is valid from a state \( s \) if for every \( F \in N_\mathcal{E} \), \( \sigma_{\mathcal{E}}(F)(s) \neq \omega \). If \( \mathcal{E} \) is finite, it suffices that this hold for \( F = E \). The set of states from which \( \mathcal{E} \) is valid is called its domain, denoted \( \delta(\mathcal{E}) \). If \( s \in \delta(\mathcal{E}) \), then we write, for each \( F \in N_\mathcal{E} \), \( \sigma_{\mathcal{E}, s}(F) = \sigma_{\mathcal{E}}(F)(s) \) (or simply \( \sigma_{\mathcal{E}}(F) \)). The function \( \sigma_{\mathcal{E}, s} \) is called the history of \( \mathcal{E} \) from \( s \). It is analogous to the sequence of states that occur when a sequential system is executed. Note that if \( F \in N_\mathcal{E} \), then \( F \) determines a valid execution, \( \delta(\mathcal{E}) \subseteq \delta(F) \), and \( \sigma_{\mathcal{E}, s}(F) = F(s) \) if \( s \in \delta(\mathcal{E}) \).

If \( \mathcal{E} \) is finite and valid from \( s \), then we define the value \( \mathcal{E}(s) \) of \( \mathcal{E} \) at \( s \) to be \( \sigma_{\mathcal{E}}(E) \). If \( R \subseteq S \), then \( \mathcal{E}(R) = \{ \mathcal{E}(s) : s \in R \wedge s \in \delta(\mathcal{E}) \} \). If \( R \subseteq \delta(\mathcal{E}) \), then \( \mathcal{E} \) is valid from \( R \). If \( \mathcal{E} \) is valid from \( I \), we simply say it is valid. Given \( R \subseteq S \), \( T \subseteq S \) is computible from \( R \) if \( \forall t \in T : \exists \mathcal{E}, \text{ finite, } s \in R : t = \mathcal{E}(s) \). \( T \) is computable if it is computable from \( I \). The computable states are those that can be reached as the result of valid computations starting from initial states.

### 5. An Example: Communication Over a Channel

To make these definitions concrete, we describe a D-algebra that models a protocol for communication over a channel. In this example, a sequence of symbols is being transmitted from one node to another. We assume that symbols are received in the order in which they are sent and are taken from some finite alphabet.
The possibility of data loss exists if a symbol is received before the previous symbol has been saved. When this occurs, the receiver is capable of sending a restart signal to the transmitter. To account for transmission delays, we model the transmission of one symbol by two operations, a send and a receive. We then incorporate the sequence of symbols that has been sent, but not yet received, as the second component of the state space of the receiver. In the following, a state is a pair of pairs. We abuse the notation and write \((s, t, u, v)\) for the pair of pairs \(((s, t), (u, v))\). \(\mathbb{N}\) denotes the set of natural numbers.

\[
P = \{p_1, p_2\}
\]

\[
S_1 = \{\text{write, restart}\} \times \mathbb{N}
\]

\[
S_2 = \{\text{read, err, } b \in \Sigma\} \times \Sigma^*
\]

\[
I = \{(\text{write, 0, read, } A)\}
\]

\[
\mathcal{F} = \{\text{recv, save, send}_{\text{restart}}, \text{recv}_{\text{restart}}, \text{recover, send}_{b: b \in \Sigma}\}
\]

\[
\delta(\text{send}_s) = \{(\text{write, } t)\} \times S_2
\]

\[
\delta(\text{recv}_{\text{restart}}) = \{(s, k): k \geq 1\} \times S_2
\]

\[
\delta(\text{recover}) = \{(\text{restart, } t)\} \times S_2
\]

\[
\delta(\text{recv}) = S_1 \times \{(u, x): x \in \Sigma^+\}
\]

\[
\delta(\text{save}) = S_1 \times \{(b, v): b \in \Sigma\}
\]

\[
\delta(\text{send}_{\text{restart}}) = S_1 \times \{(\text{err, } v)\}
\]

\[
\text{send}_b(\text{write, } t, u, \alpha) = (\text{write, } t, u, \alpha b)
\]

\[
\text{recv}(s, t, \text{read, } b\alpha) = (s, t, b, \alpha)
\]

\[
\text{recv}(s, t, c, b\alpha) = \text{recv}(s, t, \text{err, } b\alpha) = (s, t, \text{err, } \alpha)
\]

\[
\text{save}(s, t, b, v) = (s, t, \text{read, } v)
\]

\[
\text{send}_{\text{restart}}(s, k, \text{err, } v) = (s, k + 1, \text{read, } v)
\]

\[
\text{recv}_{\text{restart}}(s, k, u, v) = (\text{restart, } k - 1, u, v)
\]

\[
\text{recover}(\text{restart, } t, u, v) = (\text{write, } t, u, v)
\]

\[
\lambda(\text{send}_s) = \lambda(\text{recv}_{\text{restart}}) = \lambda(\text{recover}) = p_1
\]

\[
\lambda(\text{recv}) = \lambda(\text{save}) = \lambda(\text{send}_{\text{restart}}) = p_2
\]

It is straightforward to check that \(A = \langle P, S_1 \times S_2, I, \mathcal{F}, \lambda, \sim \rangle\) is a D-algebra. For example, \(\text{send}_b \sim \text{save}\), so we must check D2 and D3. We have \(\delta(\text{send}_b) = \{(\text{write, } \alpha, u, v)\}\), and \(\text{send}_b(\text{write, } \alpha, u, v) = (\text{write, } \alpha b, u, v)\), while both these states
are in \( \delta(\text{save}) \) if and only if \( u = c \) for some \( c \). This proves D2 for \( f = \text{send}_b, \ g = \text{save} \). \( \delta(\text{send}_b) \cap \delta(\text{save}) = \{ (\text{write}, t, c, \alpha) \} \), and these operations commute on these states, which proves D3. The remaining axioms are proved similarly. The serial relations \( \text{send}_b \sim \text{recv} \) and \( \text{send}_{\text{restart}} \sim \text{recv}_{\text{restart}} \) are forced by axiom D2.

In Fig. 1 an execution of \( A \) is described by a directed vertex labeled graph. Figure 2 gives the history of this execution.

6. MINIMIZATION OF EXECUTIONS

In this section it is shown that attention can be restricted to a certain class of executions, the minimal executions. These are executions in which as few restrictions as possible are put on the order of operations.

**DEFINITION 3.** An execution \( \mathcal{E} \) of \( A \) is minimal if whenever \( x \) is an immediate predecessor of \( y \) (i.e., \( x < y \land \neg \exists z : x < z < y \)), \( l(x) \sim l(y) \).

**THEOREM 2.** For every execution \( \mathcal{B} \), there is a minimal execution \( \rho(\mathcal{E}) \) such that

1. \( \rho(\mathcal{E}) \) has the same underlying set of actions \( E \) and labeling \( l \) as \( \mathcal{E} \).
2. The partial order in \( \mathcal{E} \) is a refinement of the partial order in \( \rho(\mathcal{E}) \).
3. \( \delta(\mathcal{E}) = \delta(\rho(\mathcal{E})) \).
4. If \( \mathcal{E} \) is finite and \( s \in \delta(\mathcal{E}) \), then \( \rho(\mathcal{E})(s) = \delta(s) \).

**Proof.** Let \( \mathcal{E} = (E, <, l) \). We define \( <' \) to be the transitive closure of \( \{(x, y) : x, y \in E \land x < y \land l(x) \sim l(y)\} \), and let \( \rho(\mathcal{E}) = (E, <', l) \). Clearly \( <' \) is an irreflexive partial order (since \( < \) is), and \( \rho(\mathcal{E}) \) satisfies the first two assertions of the theorem. We must show that \( \rho(\mathcal{E}) \) satisfies axioms E1 and E2 for an execution, and the remaining two conclusions of the theorem.

**E1:** If \( y <' x \), then \( y < x \), so the set of predecessors of \( x \) in \( \rho(\mathcal{E}) \) is a subset of the set of predecessors of \( x \) in \( \mathcal{E} \), which is finite.

**E2:**

\[
\begin{align*}
l(x) \sim l(y) &\Rightarrow x = y \lor x < y \lor y < x \\
\Rightarrow x = y \lor (x < y \land l(x) \sim l(y)) \lor (y < x \land l(y) \sim l(x)) \\
\Rightarrow x = y \lor x <' y \lor y <' x.
\end{align*}
\]
We now prove the last two assertions of the theorem. Every initial subset of \( \mathcal{E} \) relative to \( < \) is also an initial subset relative to \( <' \), so \( N_{\mathcal{E}} \subseteq N_{\mathcal{E}'} \). By the uniqueness of the history function, \( \sigma_{\mathcal{E}} = \sigma_{\mathcal{E}'} \) on \( N_{\mathcal{E}} \). In particular, \( \sigma_{\mathcal{E}}(E) = \sigma_{\mathcal{E}'}(E) \). The theorem follows.

The set of finite minimal executions of a D-algebra \( A \) is denoted by \( \mathcal{F}(A) \) (or simply by \( \mathcal{F} \) if there is no ambiguity). \( \mathcal{F} \) embeds naturally in \( \mathcal{F} \) as the set of one node executions. Let \( \mathcal{E} = (E, <, I) \) be a finite execution and let \( \mathcal{E}' = (E', <', I') \) be an arbitrary execution. We define the composition \( \mathcal{E}' \mathcal{E} \) as follows: Let \( x <' y \) if either \( x, y \in E \) and \( x < y \), or \( x, y \in E' \) and \( x <' y \), or \( x \in E \) and \( y \in E' \). Let \( l''(x) = l(x) \) if \( x \in E \) and \( l''(x) = l'(x) \) if \( x \in E' \). Let \( \mathcal{E}' \mathcal{E} \) be the execution \( (E \cup E', <'', I'') \).
is finite if $\mathcal{E}'$ is finite. Then $\mathcal{E}' = \rho(\mathcal{E}' \ast \mathcal{E})$. Composition restricts to a binary operation on $\hat{\mathcal{E}}$. Composition is associative and, with the empty execution as identity, $\hat{\mathcal{E}}$ forms a monoid.

**Proposition 2.** 1. If $s \in \delta(\mathcal{E})$ and $\delta(s) \in \delta(\mathcal{E}')$, then $s \in \delta(\mathcal{E}' \ast \mathcal{E})$. If $\mathcal{E}$ is finite as well, then $(\mathcal{E}' \ast \mathcal{E})(s) = \delta'(\delta(s))$.

2. If $s \in \delta(\mathcal{E})$ and $\delta(s) \in \delta(\mathcal{E}')$, then $s \in \delta(\mathcal{E}' \ast \mathcal{E})$. If $\mathcal{E}'$ is finite as well, then $(\mathcal{E}' \ast \mathcal{E})(s) = \delta'(\delta(s))$. Thus the map from $\hat{\mathcal{E}}$ to the set of partial maps on $S$ is a homomorphism of monoids.

**Proof.** 1. An initial subset of $\mathcal{E}' \ast \mathcal{E}$ is either an initial subset of $\mathcal{E}$ or of the form $E \cup F$, where $F$ is an initial subset of $\mathcal{E}'$. Hence $N_{\mathcal{E}' \ast \mathcal{E}} = N_{\mathcal{E}} \cup \{ E \cup F : F \in N_{\mathcal{E}'} \}$. We have

$$\sigma_{\mathcal{E}'}(F) = \sigma_{\mathcal{E}}(F) \quad \text{if } F \subseteq E$$

$$\sigma_{\mathcal{E}'}(E \cup F) = \sigma_{\mathcal{E}'}(F) \sigma_{\mathcal{E}}(E) \quad \text{if } F \subseteq E'.$$

The assertion follows.

2. Follows from the first assertion and Theorem 2.

7. Simulations and Possibilities Maps

Much of the power of the theory of D-algebras lies in the ability to successively abstract away properties of systems using hierarchical proofs of correctness. We start with a system $A$, modeled as a D-algebra, for which we would like to prove some property. We then design a sequence of successively simpler D-algebras $B_1$, $B_2$, ..., $B_n$, connected by a sequence of maps which preserve the property in question, and prove the property for $B_n$. This approach has been used successfully in correctness proofs for nested transaction systems [7] using the Event State Algebra model, as well as in proofs using quite different models [1, 12].

In this section we discuss two kinds of mappings between D-algebras. The first, interpretations, relate D-algebras on the level of executions, independent of their interpretation as state change functions. The second, possibilities maps, consist of a validity preserving interpretation and a map from global states to sets of global states. If we know the state of one system, then, upon weakening restrictions, we may only know several possibilities for the state of a second system.

**Definition 4.** An *interpretation* of a D-algebra $A = \langle P, S, I, \mathcal{F}, \lambda, \sim \rangle$ by a D-algebra $B = \langle Q, T, J, \mathcal{G}, \mu, \sim' \rangle$ is a map $h : \mathcal{F} \to \mathcal{G} \cup \{ \emptyset \}$ such that if $f, g \in \mathcal{F}$, $x \in h(f)$, $y \in h(g)$, and $l(x) \sim' l(y)$, then $f \sim g$.

Here we understand the empty set as defining the identity function on the state space of any D-algebra.
We can extend \( h \) to \( \hat{h} \). This is done by mapping an execution to the disjoint union of the images of its labels and adding enough relations to preserve the relations of the original execution. More precisely, let \( \mathcal{E} = (E, \prec, l) \) be an execution of \( A \). If \( h(l(x)) = \emptyset \) for every \( x \in E \), then \( h(\mathcal{E}) = \emptyset \). Otherwise, let \( h(\mathcal{E}) = (F, \prec', m) \), where

\[
F = \bigcup_{x \in E} \{ (x, y) : h(l(x)) = (E_x, \prec_x, l_x), y \in E_y \},
\]

\[
(x, y_1) \prec (x, y_2) \quad \text{if} \quad y_1 \prec_x y_2,
\]

\[
(x_1, y_1) \prec (x_2, y_2) \quad \text{if} \quad x_1 \prec x_2,
\]

and

\[
m(x, y) = l_x(y).
\]

The condition on serial operations is equivalent to the condition that this induced map take executions to executions. This condition does not hold for an arbitrary \( h \).

For example, consider the following interpretation, described pictorially by representing executions as labeled graphs, with the name of the element of an execution written above the vertex and its label below:

\[
h(f_1) = \frac{y_1}{x_1}, \quad h(f_2) = \emptyset, \quad h(f_3) = \frac{z_1}{x_2}.
\]

Then

Interpretations are not quite homomorphisms between the monoids of minimal executions, since minimal executions may not map to minimal executions. As we next see, however, when executions are interpreted as functions on states, interpretations do act as homomorphisms. The induced map \( \rho \circ h \) is a homomorphism of monoids.

**Proposition 3.** If \( h \) is an interpretation of \( A \) by \( B \), and \( \mathcal{E} \) and \( \mathcal{E}' \) are finite executions of \( A \), then

1. \( \rho(h(\rho(\mathcal{E}))) = \rho(h(\mathcal{E})). \)
2. For every \( t \in T \), \( h(\rho(\mathcal{E}))(t) = \rho(h(\mathcal{E}))(t) = h(\mathcal{E})(t). \)
3. \( p(h(\mathcal{E} \circ \mathcal{E}')) = p(h(\mathcal{E})) \cdot p(h(\mathcal{E}')) \).

4. For every \( t \in T \), \( h(\mathcal{E} \circ \mathcal{E}')(t) = h(\mathcal{E}) \cdot h(\mathcal{E}')(t) \).

**Proof.** 1. These two executions have the same underlying set, so it suffices to see that their partial orders are identical. This follows from the fact that the partial order on \( h(\mathcal{E}) \) is a refinement of the partial order on \( h(\rho(\mathcal{E})) \), which in turn follows from the fact that the partial order on \( \mathcal{E} \) is a refinement of the partial order on \( \rho(\mathcal{E}) \).

2. For every \( t \in T \), \( h(\rho(\mathcal{E}))(t) = \rho(h(\rho(\mathcal{E}))(t)) = \rho(h(\mathcal{E}))(t) = h(\mathcal{E})(t) \).

3. We have by construction that \( h(\mathcal{E} \circ \mathcal{E}') = h(\mathcal{E}) \circ h(\mathcal{E}') \), so \( p(h(\mathcal{E} \circ \mathcal{E}')) = p(h(\rho(\mathcal{E} \circ \mathcal{E}')))) = p(h(\mathcal{E}) \circ h(\mathcal{E}')) = h(\mathcal{E}) \cdot h(\mathcal{E}')(t) \).

4. For every \( t \in T \), \( h(\mathcal{E} \circ \mathcal{E}')(t) = p(h(\mathcal{E} \circ \mathcal{E}'))(t) = p(h(\mathcal{E})) \cdot p(h(\mathcal{E}'))(t) = h(\mathcal{E}) \cdot h(\mathcal{E}')(t) \).

**DEFINITION 5.** A **possibilities map** from a D-algebra \( A = \langle P, S, I, \mathcal{F}, \lambda, \sim \rangle \) to a D-algebra \( B = \langle Q, T, J, \mathcal{G}, \mu, \sim \rangle \) is a pair \( \pi = (\sigma, h) \), where \( h \) is an interpretation of \( A \) by \( B \) and \( \sigma \) is a function from the state space of \( A \) to the set of nonempty subsets of the state space of \( B \), i.e., \( \sigma : S \to 2^T - \{\emptyset\} \), satisfying

P1: \( h(\mathcal{E}) \) is a valid execution whenever \( \mathcal{E} \) is a valid execution.

P2: \( J \subseteq \sigma(I) \).

P3: If \( s \in \delta(f) \) and \( t \in \sigma(s) \), then \( t \in \delta(h(f)) \) and \( h(f)(t) \in \sigma(f(s)) \) (see Fig. 3).

By induction on the size of a finite execution, we can prove the following extension of axiom P3.

**LEMMA 2.** If \((\sigma, h)\) is a possibilities map, \( \mathcal{E} \) is an execution of its domain, \( s \in \delta(\mathcal{E}) \), and \( t \in \sigma(s) \), then \( t \in \delta(h(\mathcal{E})) \). If, moreover, \( \mathcal{E} \) is finite, then \( h(\mathcal{E})(t) \in \sigma(f(s)) \).

A possibilities map models complex executions of its range by single operations on its domain. Given the D-algebra \( B \) for which we wish to prove some property, we find a D-algebra \( A \) which is simpler than \( B \), and a possibilities map \( \pi : A \to B \) of a type preserving the property in question. We are then reduced to proving the
property for $A$. We may in fact need to repeat this reduction several times. For example, certain classes of simulations and possibilities maps can be shown to preserve various reliability properties of distributed systems.

**Proposition 4.** Let $(\sigma, h): A \rightarrow B$ be a possibilities map, $A$ and $B$ as above.

1. If $\mathcal{E}$ is a valid finite execution of $A$, then $h(\mathcal{E})(s) \subseteq \sigma(\mathcal{E}(s))$.

2. If $(\tau, k): B \rightarrow C = \langle R, U, K, \mathcal{H}, \eta, \sim^\tau \rangle$ is a second possibilities map, then $(\tau \circ \sigma, k \circ h): A \rightarrow C$ is a possibilities map.

**Proof.**

1. Let $t \in J$. By P2, $t \in \sigma(s)$ for some $s \in I$. By Lemma 2, $h(\mathcal{E})(t) \in \sigma(\mathcal{E}(s)) \subseteq \sigma(\mathcal{E}(I))$.

2. By $\tau \circ \sigma$ we mean the map that takes $s \in S$ to $\bigcup_{t \in \delta(s)} \tau(t)$. To see that $h$ is an interpretation, let $f, g \in \mathcal{F}$, $(x_1, y_1) \in k \circ h(f)$, $(x_2, y_2) \in k \circ h(g)$, and $l((x_1, y_1)) \sim^\tau l((x_2, y_2))$. This means that $x_1 \in h(f)$, $y_1 \in E_{x_1}$, where $k(l(x_1)) = (E_{x_1}, x_1, l_{x_1})$, and similarly for $(x_2, y_2)$. Also, $l_{x_1}(y_1) \sim^\tau l_{x_2}(y_2)$. From the fact that $k$ is an interpretation, it follows that $l(x_1) \sim^\tau l(x_2)$. Then from the fact that $h$ is an interpretation it follows that $f \sim g$.

P1: Let $\mathcal{E}$ be a valid execution of $A$. Then $h(\mathcal{E})$ is a valid execution of $B$, so $k(h(\mathcal{E}))$ is a valid execution of $C$.

P2: $\tau \subseteq \tau(J) \subseteq \tau(\sigma(I))$.

P3: Let $f \in \mathcal{F}$, $s \in \delta(f)$, and $u \in \tau \circ \sigma(s)$. Then $u \in \tau(t)$ for some $t \in \delta(s)$. By P3 for $(\sigma, h)$, $t \in \delta(h(f))$ and $h(f)(t) \in \sigma(f(s))$. By Lemma 2 for $(\tau, k)$, $u \in \delta(k(h(f(t))))$ and $k(h(f))(u) \in \tau(k(t)) \subseteq \tau \circ \sigma(s)$.

It follows that D-algebras with possibilities maps as morphisms form a category, the possibilities algebras (or $p$-algebras).

**Acknowledgments**

I thank Nancy Lynch for useful discussions in the early stages of this work, an anonymous referee for suggestions that resulted in major revision of and substantial improvement to this paper, and John Casey for several useful comments.

**References**


