FUNCTIONAL BEHAVIOR OF NONDETERMINISTIC AND CONCURRENT PROGRAMS

by

Michael Main* and David B. Benson**

CU-CS-246-83 June, 1983

*Department of Computer Science, University of Colorado, Boulder, CO 80309 USA

**Department of Computer Science, Washington State University, Pullman, WA 99164-1210 USA

This research was supported in part by grant MCS-8003433 and a grant from the University of Colorado on Research and Creative Work.

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR AND DO NOT NECESSARILY REFLECT THE VIEWS OF THE NATIONAL SCIENCE FOUNDATION.

FUNCTIONAL BEHAVIOR OF NONDETERMINISTIC AND CONCURRENT PROGRAMS*

Michael G. Main
Department of Computer Science
University of Colorado
Boulder, CO 80309 USA

David B. Benson
Department of Computer Science
Washington State University
Pullman, WA 99164-1210 USA

ABSTRACT

The functional behavior of a deterministic program segment is a function $f:D \to D$, where D is some set of states for the computation. This notion of functional behaviors can be extended to nondeterministic and concurrent programs using techniques from linear algebra. In particular, the functional behavior of a nondeterministic program segment is a linear transformation $f:A \to A$, where A is a free semiring module. Other notions from linear algebra carry over into this setting. For example, weakest preconditions and predicate transformers correspond to well-studied concepts in linear algebra. Using multilinear algebra, programs with tuples of inputs and outputs can be handled. For nondeterministic concurrent programs, the functional behavior is a linear transformation $f:A \to A$, where A is a free semiring algebra. In this case, f *may* also be an algebra morphism, which indicates that the program involves no interprocess communication. Finally, we study a model of syntax for programs whose semantics is given using linear algebra. It is shown that in this model free interpretations (essentially Herbrand universes) do not generally exist.

*A summary of part of this paper appears under the title "Functional Behavior of Nondeterministic Programs" in the Proceedings of the 1983 Foundations of Computation Theory Conference, held in Borgholm, Sweden.

1. INTRODUCTION

In denotational semantics, the behavior of a program segment is a function $f:D \rightarrow D$, where D is a set of states for some abstract machine. If nonterminating computations are possible, then D may contain a special element $\perp \in D$ to represent this fact. A tuple of inputs or outputs to a program segment can be represented by a set of tuples $D^n = \{(d_1, \cdots d_n) \mid d_i \in D\}$. In particular, the functional behavior of a program segment with n inputs and p outputs is a function $f:D^n \rightarrow D^p$. We will generally refer to such "program segments" as just "programs".

The purpose of this paper is to extend our understanding of the functional behavior of programs to nondeterministic and concurrent situations. For nondeterminism, we consider three views. In the first view, the only property of interest is the set of possible outputs a nondeterministic program may produce from a given input. This is the widely studied standard notion of nondeterminism [6,15,34,35,36,39]. An alternative is to record with each possible output state the number of different computation paths which lead to it [1]. A third view of nondeterminism is related to probabilistic models of computation [20,38]. In this view, we record with each state some predicate which must hold in order for that state to be a possible output. The predicates may involve timing considerations which can affect a computation, or they may be statements about nondeterministic choices made by a program.

These three notions of nondeterminism are developed in section 2. The unifying idea behind the three models is a semiring module. In all three cases, we show how the functional behavior of a nondeterministic program segment is a linear transformation $f:A \rightarrow A$, where A is a free semiring module generated by the set D of states.

Semiring modules are a generalization of vector spaces. In section 3, we show how some other ideas from linear algebra generalize to our setting. In particular, elements of the dual module of A correspond to "conditions" in the sense of Dijkstra [6,7]. Weakest preconditions are developed using kernels of linear transformations. Predicate transformers are linear functional transformers.

Programs with tuples of inputs and outputs are considered in section 4, using a suggestion of Hennessy and Plotkin [15]. The important observation is that the behavior of a nondeterministic program is multilinear with respect to its many inputs. As a result, the functional behavior of a nondeterministic program with an n-tuple of inputs and a p-tuple of outputs is a linear transformation $f : \bigotimes^{n} A \to \bigotimes^{p} A$, where the domain and codomain are iterated tensor products of a free semiring module A.

These ideas are extended to nondeterministic concurrent programs by placing additional structure on the underlying set of deterministic states. In particular, we consider a deterministic state to consist of any finite number of data elements or messages existing concurrently. With this model, the data space for computations forms a semiring algebra -- which is a generalization of ring algebras studied in linear algebra. However, the functional behavior of a program segment is not always an algebra morphism. Specifically, the behavior of a program segment is an algebra morphism if and only if no communication occurs between its several inputs.

Finally, we look at an abstraction of this sort of functional behavior, based on algebraic theories. The abstraction consists of a set of uninterpreted nondeterministic program segments. An interpretation associates a linear transformation with each program segment in a way that preserves certain operations. Our focus is on the universal properties of the model -- we show that in general, universal (or free) interpretations do not exist.

Our study is limited in a number of ways. First, we consider mainly finite nondeterminism and finite concurrency. Any given program deals with only a finite number of nondeterministic possibilities, and for each of these, there is only a finite number of concurrent inputs or outputs. For nondeterminism, our results carry over to the infinite case, and occasionally we point out how to do this. The second restriction is that at the moment we do not consider recursively defined or iterative programs. We plan to handle such programs in the future using two techniques. One possibility is to use ordered semiring modules and least fixed-point techniques [15,34,39]. Another approach is based on the observation that the linear transformations we are studying are matrixes with respect to a basis of the semiring module. Finally, in our study of concurrency, there is no explicit consideration of time. Models of concurrent computation which account for time ordering (or partial ordering [37]) of events could be included in our model by using an implicit ordering of messages. We expect to report on this topic subsequently.

2. NONDETERMINISM, SEMIRINGS AND SEMIRING MODULES

We start from a set of deterministic states for a computation, denoted by D. The term "state" is entirely neutral -- for example, if continuation semantics is intended, then the states may be continuations. For concurrent processes, the states may also have additional structure, which we discuss later. But for the moment, view states as having no internal structure.

2.1. Nondeterministic Distributions of States

Nondeterminism occurs when the output of a computation is not uniquely determined by its input. In this case, a record may be kept of the possible output states which could arise from a given input state. Such a record is called a *nondeterministic distribution of states*, or simply a *distribution*.

There are a variety of notions for a nondeterministic distribution of states. Manes discusses many in his development of "distributional set theories" [29]. We discuss three such ideas, concentrating on *finite* nondeterminism -- i.e., each computation has only a finite number of output choices for a given input.

The first possibility is to list all the different deterministic states which may arise in a nondeterministic computation starting from a given input state. In this case, a distribution consists of a finite subset of D. If infinite nondeterminism is allowed in the model, then infinite subsets of D are also suitable distributions. In either case, the collection of all distributions has set union as the semantic correspondent to the **or** operator available in some programming languages. For reasons made clear later, it is convenient to use the addition sign to represent the union of distributions: for distributions x and y, the distribution $x+y$ is read "x or y" and is the set union of x and y. The collection of all distributions has the structure of a commutative monoid with respect to $+$. The empty set is the identity for this monoid.

There remains the question of the meaning of the empty set of states. We take the view that the empty set is not a possible outcome of a nondeterministic computation. That is, every computation must produce some state as its output. If a computation never terminates, the "output state" may be represented by a special element $\perp \in D$. Despite the fact that the empty distribution is an impossible output, we keep it in the collection of distributions in order to answer questions such as: what inputs to a particular program give a correct output? In general, the answer to this question will be a nondeterministic distribution of states. Sometimes there may be no input which makes a program correct, in which case the answer to the above question will be "the empty distribution". Questions like these are the subject of section 3.

If the issue is simply which states may be reached by a nondeterministic computation, the subset notion of distributions suffices. Another choice is to count the number of different computation paths which give rise to each state in a final distribution [1]. In this case, we need a *multiset* of deterministic states to record the number of computation paths leading to each possible state. For example, the multiset $\{d_1,d_1,d_2\}$

represents two paths leading to state d_1 and one path leading to state d_2. A finite distribution is any finite multiset with elements from D. For infinite nondeterminism, infinite multisets are allowed.

Additive notation is a convenient way of denoting multisets. Let $\sum_{d \in D} n_d d$ be a formal sum running over all the states in D. Each term $n_d d$ consists of the count $n_d \geq 0$ associated with the state $d \in D$. Thus, $\sum_{d \in D} n_d d$ denotes the multiset containing n_d copies of d for each $d \in D$. The empty multiset, denoted by 0, represents an impossible outcome, as did the empty set in the subset version of nondeterministic distributions. We may also view a sum $\sum_{d \in D} n_d d$ as an element in a vector space with unit vectors d and coefficients n_d. This is not strictly correct, as the coefficients are natural numbers, rather than arbitrary real numbers. The correct formalism -- a semiring module -- is given later in this section.

For finite nondeterminism, each coefficient n_d in a sum $\sum_{d \in D} n_d d$ is an element of $\mathbf{N} = \{0,1,2,\ldots\}$. Furthermore, only a finite number of the n_d are nonzero. If infinite nondeterminism is of interest, additive notation may still be used by extending the allowable coefficients to $\mathbf{N}^\infty = \mathbf{N} \cup \{\infty\}$. Infinite nondeterminism is further developed elsewhere [1].

A third view of nondeterminism is an extension of the idea of recording the path count of a state. Indeed, it also generalizes some aspects of Kozen's work on probabilistic nondeterminism [20]. Let L be any distributive lattice with a supremum (1) and an infimum (0). The elements of L may be thought of as conditions or predicates having an effect on what outputs are possible from a nondeterministic computation. For example, the predicates could make statements about timing considerations or external stimuli which could affect a computation. The "least upper bound" operation of the lattice corresponds to the **or** of two predicates, while "greatest lower bound" is **and**. The supremum is the always **true** predicate, and the infimum is the **false** predicate. Every Boolean algebra is a distributed lattice.

In this view of nondeterminism, we record the condition $c_d \in L$ which must hold for each state $d \in D$ to be a possible output. A distribution is a formal sum $\sum_{d \in D} c_d d$ with the coefficients drawn from L. For finite nondeterminism, only a finite number of the coefficients are nonzero in any sum. Results relating this view of nondeterminism to problems in concurrent computation are given elsewhere [2].

Of course, this last view includes the subset version of distributions. Let the lattice of conditions be the two-element Boolean algebra $\mathbb{B}$, so that formal sums $\sum_{d \in D} b_d d$ represent subsets of D.

The systems of coefficients mentioned above are all instances of commutative positive semirings. While specific details vary in the examples, there is a great deal of commonality best exposed in the general setting described below.

2.2. Semirings

Let k be a set with two distinguished elements 0 and 1 ($0 \neq 1$) and two binary operations on k, denoted $+$ and $\cdot$. The operation $+$ is called *addition* and $\cdot$ is *multiplication*. The tuple $<k,+,\cdot,0,1>$ is a *semiring* if the following conditions hold:

(1) $<k,+,0>$ is a commutative monoid.

(2) $<k,\cdot,1>$ is a monoid.

(3) Multiplication distributes over addition. That is, for all $r,s,t \in k$:

$$r\cdot(s+t) = (r\cdot s)+(r\cdot t) \quad \text{and} \quad (s+t)\cdot r = (s\cdot r)+(t\cdot r).$$

(4) $r\cdot 0 = 0 = 0\cdot r$, for all $r \in k$.

We frequently write k for $<k,+,\cdot,0,1>$. The identity for addition (0) is called the *zero* and the identity for multiplication (1) is the *unit*. A semiring is *commutative* if its multiplication is commutative. If every element has an additive inverse, then k is a *ring*. A semiring is *positive* if for all $r,s \in k$:

$$r+s = 0 \quad \text{implies} \quad r = s = 0.$$

No positive semiring is a ring.

Examples: The natural numbers, **N**, and the natural numbers extended to infinity, $\mathbf{N}^{\infty}$, are commutative positive semirings, with the usual operations. The set of all integers and the set of all reals are commutative semirings, but not positive. The two-element Boolean algebra $\mathbf{B} = <\{0,1\},\vee,\wedge,0,1>$ is a commutative positive semiring, as is any distributive lattice with an infimum and supremum $(0 \neq 1)$.

A function $f:k \to k'$ from one semiring k to another k' is a *semiring morphism* if it is a monoid morphism for both addition and multiplication. For example, the map from **N** to **B** taking 0 to 0 and all other numbers to 1 is a semiring morphism.

2.3. Semiring Modules

Let k be a semiring. A k*-module* consists of a commutative monoid $<A,+,0>$ and a function from $k \times A$ to A (the image of $<r,x>$ being written rx). The operations are subject to the axioms:

$$(r+s)x = (rx)+(sx)$$
$$r(x+y) = (rx)+(ry)$$
$$(r \cdot s)x = r(sx)$$
$$0x = 0$$
$$1x = x.$$

We ordinarily write A for the module $<A,+,0>$ over the semiring k. The operation from $k \times A$ to A is called *scalar multiplication*. Note that the symbols + and 0 are each used in two different ways: the addition and zero for the semiring k and also the monoid operation and identity for A. If k is a ring, then the definition of a semiring k-module coincides with the definition of a ring k-module [25, V]. A subset X of a k-module A is said to *span* A provided that every element of A can be expressed as a finite sum $r_1x_1 + \cdots + r_nx_n = \sum_{i=1}^{n} r_ix_i$, where (r_i) is a sequence of elements from k and (x_i) is a sequence from X. If every element of A can be expressed as such a sum in exactly one way (apart from commutativity of +), then X is a *basis* for A.

Other authors term a "semi-module" what we have defined as a module. Since we will only consider modules over semirings, the shorter term is preferable.

Examples: (i) Let $<A,+,0>$ be a commutative idempotent monoid. Then A is a $\mathbb{B}$-module with scalar multiplication defined by $1x = x$ and $0x = 0$ for all $x \in A$.

(ii) Every commutative semiring $<k,+,\cdot,0,1>$ is a commutative monoid $<k,+,0>$. This monoid is a k-module. The scalar multiplication is just the multiplication of the semiring.

(iii) If k is a semiring and D is a set, then the collection of formal sums of the form $\sum_{d \in D} r_d d$ is a k-module. Addition and scalar multiplication are defined pointwise, so that:

$$\sum_{d \in D} r_d d + \sum_{d \in D} s_d d = \sum_{d \in D} (r_d + s_d) d$$

$$s\left(\sum_{d \in D} r_d d\right) = \sum_{d \in D} (s \cdot r_d) d$$

The collection of formal sums remains a k-module if we include only those sums where a finite number of coefficients are nonzero. In both cases, the formal sum with all zero coefficients, denoted 0, is the identity for +.

(iv) A function $f: A \to B$, from one k-module A to another B, is a *linear transformation* if it is a monoid morphism from the monoid A to the monoid B, compatible with scalar multiplication. These facts may be recorded in the single equation:

$$f(rx + sy) = r(f(x)) + s(f(y)), \text{ for all } r,s \in k \text{ and } x,y \in A.$$

The set of linear transformations from A to B form a k-module with pointwise operations. Thus, if f and g are linear transformations from A to B, then:

$$(f+g)(x) = f(x) + g(x)$$
$$(rf)(x) = r(f(x)),$$

for all $x \in A$ and $r \in k$.

2.4. Free Modules

Let k be a semiring and D be any set. As described above, the collection of formal sums of the form $\sum_{d \in D} r_d d$, with coefficients from k is a k-module. It remains a k-module if we include only those sums with a finite number of nonzero coefficients. In the following we describe the universal property of this module of finite formal sums, denoted $k^{(D)}$.

Consider the function $\varepsilon : D \to k^{(D)}$ which maps each $d \in D$ to the formal sum ε_d whose coefficient at d is 1 and all other coefficients are zero. The sums of the form ε_d form a basis for $k^{(D)}$ which is *free* in the following sense: Suppose A is a k-module and $f : D \to A$ is a function. Then there is a unique linear transformation $\overline{f} : k^{(D)} \to A$ making the following triangle commute:

$$\begin{tikzcd}
D \arrow[r, "\varepsilon"] \arrow[dr, "f"'] & k^{(D)} \arrow[d, dashed, "\overline{f}"] \\
 & A
\end{tikzcd}$$

In particular, $\overline{f}\left(\sum_{d \in D} r_d d\right) = \sum_{d \in D} r_d(f(d))$. The fact that each formal sum in $k^{(D)}$ has a finite number of nonzero coefficients guarantees that the sum $\sum_{d \in D} r_d(f(d))$ in A is defined.

Because of this property, we call $k^{(D)}$ a *free* k-module over D, with *insertion* $\varepsilon: D \to k^{(D)}$. The practical significance of a free module is that every linear transformation $g: k^{(D)} \to A$ is completely determined by its image on the basis elements ε_d. Moreover, any function from these basis elements to A extends uniquely to a linear transformation from $k^{(D)}$ to A.

Each linear transformation from a free module to a free module may be described by a matrix over the semiring of scalars. For example, let the semiring k be a Boolean algebra with elements $a, \bar{a} \in k$. Let $D = \{d_1, d_2\}$ and define $f_1: k^{(D)} \to k^{(D)}$ by:

$$f_1(d_1) = 1d_1$$

$$f_1(d_2) = ad_1 + \bar{a}d_2.$$

The d_1 and d_2 on the left of these equalities are abbreviations for ε_{d_1} and ε_{d_2}, which we will frequently use. The right side of each equation is a formal sum in $k^{(D)}$. The linear transformation f_1 may be described by the matrix:

$$\begin{array}{c c} & \begin{array}{cc} d_1 & d_2 \end{array} \\ \begin{array}{c} d_1 \\ d_2 \end{array} & \begin{bmatrix} 1 & 0 \\ a & \bar{a} \end{bmatrix} \end{array}$$

With this representation, the addition and composition of linear transformations are performed using the usual rules for matrix addition and multiplication. Throughout the remainder of the paper we will occasionally refer back to this particular example of a linear transformation.

2.5. Program Behaviors

At the start of this section, we gave three views of nondeterminism, starting from a set D of deterministic states. For finite nondeterminism, the collection of nondeterministic distributions was always a free k-module $k^{(D)}$ for some commutative positive semiring k (see table 1). The basis elements $\varepsilon_d \in k^{(D)}$ correspond to deterministic states. The formal sum 0, with all zero coefficients, is an impossible outcome, representing no state. The addition operation in the k-module corresponds to the **or** operator available in some programming languages.

Suppose k is any commutative positive semiring and $A = k^{(D)}$ is a free k-module over D, considered as the collection of nondeterministic distributions. The functional behavior of a nondeterministic program segment is a linear transformation $f : A \rightarrow A$. Linearity implies that for all $r \in k$ and $x, y \in A$:

$$f(rx) = r(f(x))$$

$$f(x+y)=f(x)+f(y).$$

The first axiom means that if r is the requirement for x to be input, then r is again the requirement for $f(x)$ to be output. The second axiom means that when the input is x **or** y, then the output is $f(x)$ **or** $f(y)$.

View of Nondeterminism	Semiring of Scalars	Finite Nondeterministic Distributions
subsets	Boolean semiring, **B**	$\mathbf{B}^{(D)}$
multisets model	Natural numbers, **N**	$\mathbf{N}^{(D)}$
conditions model	Distributive lattice, L, with 0 and 1	$L^{(D)}$

Table 1. Models of Nondeterminism.

Not all of the linear transformations on A represent program behaviors. Since zero is not a possible output, we require

$$f(x) = 0 \quad \text{implies} \quad x = 0.$$

We call a linear transformation that meets this requirement a *positive transformation*. The previous example of f_1 is a positive transformation. In this example, the elements a and $\bar{a}$ in the semiring k correspond to conditions which affect the computation of f_1. Perhaps a means "console button a is pushed" and $\bar{a}$ is the negation of a. In this case, the behavior of f_1 is informally described as follows:

* if the input to f_1 is state d_1, then the output is also d_1.

* if the input to f_1 is state d_2 and console button a is pushed, then the output is d_1.

* if the input to f_1 is state d_2 and console button a is not pushed, then the output is d_2.

If we have computations that never terminate, then A may contain a special element $\perp$ to represent such a computation. In general, we do not require $f(\perp) = \perp$, since a program may give output from exogenous events not explicitly given in its input. A positive transformation which does have $f(\perp) = \perp$ is called *strict*.

For infinite nondeterminism, a similar model arises. The main difference is that the semiring of scalars and their modules need to have countably infinite sums defined. Moreover, the infinite sum operation needs to be associative and commutative, and multiplication must distribute over infinite sums. The extended natural numbers $\mathbf{N}^{\infty}$, form such a semiring. Any countably complete distributive lattice with countable distributivity is also such a semiring. In these infinite models, linear transformations must preserve infinite sums -- i.e., $f\left(\sum_{i \in I} x_i\right) = \sum_{i \in I} f(x_i)$, for any countable index set I.

3. DUAL MODULES, KERNELS AND WEAKEST PRECONDITIONS

Recall that every commutative semiring k is itself a k-module with scalar multiplication just the semiring multiplication (section 2.3). For any other k-module, A, the collection of all linear transformations from A to k is a k-module, denoted A^* and called the ***dual module*** of A. Scalar multiplication and addition are defined pointwise, as in the last example of section 2.3. If $A = k^{(D)}$ is the set of nondeterministic distributions for some set D of deterministic states, then A^* can be interpreted as "conditions" in the sense of Dijkstra [6,7]. The correspondence between conditions and A^* is given in this section.

We begin with the simple case when k is the Boolean semiring $\mathbf{B} = \{0,1\}$ and $A = \mathbf{B}^{(D)}$ is the free Boolean module over a set D of deterministic states. We will view A as the set of finite subsets of D, with set union as addition. A condition, in the sense of Dijkstra, is just a subset $R \subseteq D$ of deterministic states that are acceptable, according to some rule. Equivalently, a condition is a function $f: D \to \mathbf{B}$, where $f(d) = 0$ iff d is an acceptable state. Using 0 for an *acceptable* state may seem a bit backwards at first. The reason for this choice becomes clear when we extend f to all of A.

Since A is free over D, there is a unique linear transformation $\bar{f}: A \to \mathbf{B}$ which makes the following triangle commute (section 2.4):

$$\begin{tikzcd}
D \arrow[r, "\varepsilon"] \arrow[dr, "f"'] & A \arrow[d, dashed, "\bar{f}"] \\
 & \mathbf{B}
\end{tikzcd}$$

In particular, $\bar{f}: A \to \mathbf{B}$ takes a finite subset $S \subseteq D$ to $\sum_{d \in S} f(d)$. It follows that $\bar{f}(S) = 0$ iff $f(d) = 0$ for all $d \in S$. Thus, we consider a nondeterministic distribution $S \subseteq D$ to be acceptable iff each $d \in S$ is acceptable, and the set of acceptable distributions are those elements of A which are mapped to 0 by $\overline{f}$. In linear algebra, this subset of A is called the *kernel* of $\overline{f}$, denoted $\mathbf{ker}(\overline{f}) = \{S \in A \mid \overline{f}(S) = 0\}$.

So, each condition is a function $f : D \rightarrow \mathbf{B}$, which extends to a unique linear transformation $\overline{f} : A \rightarrow \mathbf{B}$. Moreover, since A is free over D, each linear transformation $g : A \rightarrow \mathbf{B}$ is uniquely determined by its image on the basis elements ε_d $(d \in D)$. So, a condition *is* a linear transformation from A to $\mathbf{B}$. The module A^*, consisting of all linear transformations from A to $\mathbf{B}$, is the collection of all possible conditions. For a condition $g : A \rightarrow \mathbf{B}$, the set of acceptable nondeterministic distributions is the kernel of g.

When $k = \mathbf{N}$ and $A = \mathbf{N}^{(D)}$ (the multiset model), the dual module A^* also corresponds to conditions, although in this case a condition $g : A \rightarrow \mathbf{N}$ records more than just the acceptability or unacceptability of each state. For $x \in A$, we interpret $g(x)$ to be a count of the number of different ways in which the states of x are unacceptable. For example, we could keep track of the number of different nondeterministic computation paths that lead to an unacceptable output with input x. The requirement that $g : A \rightarrow \mathbf{N}$ be a linear transformation formalizes the idea that the number of ways in which x **or** y is unacceptable is the sum of the number of ways that x and y are unacceptable on their own. Once again, the kernel of a condition $g : A \rightarrow \mathbf{N}$ corresponds to the set of states which are acceptable.

A third case is when k is a distributive lattice of predicates and $A = k^{(D)}$. In this case, a condition $g : A \rightarrow k$ records requirements for a state to be unacceptable. A distribution $x \in A$ is acceptable iff the predicate $g(x) \in k$ is **false**. The linearity of a condition $g : A \rightarrow k$ means that $x+y$ is acceptable iff x is acceptable and y is acceptable. The kernel of a condition corresponds to the set of states which are always acceptable.

In summary: when a k-module $A = k^{(D)}$ is considered as the set of nondeterministic distributions, then the dual module A^*, consisting of all linear transformations from A to

k, is the set of "conditions" in the sense of Dijkstra [6,7]. For a condition $g:A \to k$, the kernel of g (those elements mapped to 0) corresponds to the set of thoses states which meet the condition. As an example, the always **true** condition (every state is acceptable) corresponds to the linear transformation $0:A \to k$ which maps every $x \in A$ to $0 \in k$. The kernel of $0:A \to k$ is all of A -- i.e., every nondeterministic distribution is acceptable. On the other hand, the linear transformation $1:A \to k$ with $1(\varepsilon_d) = 1$ for all $d \in D$ corresponds to the always **false** condition -- i.e., it is impossible to have an acceptable state. In this case, $\mathbf{ker}(1) = \{0\}$ consists of only the impossible state.

Now, a nondeterministic program segment is a linear transformation $t:A \to A$. If $g:A \to k$ is a condition, then the composition $g \circ t:A \to k$ is also a condition. In Dijkstra's terminology, $g \circ t$ is the "weakest precondition" for g with program t. That is, if a distribution $x \in A$ satisfies the condition $g \circ t$ ($i.e.$, $x \in \mathbf{ker}(g \circ t)$), then carrying out the program t on x is certain to establish the truth of condition g ($i.e.$, $t(x) \in \mathbf{ker}(g)$). Moreover, $g \circ t$ is the "weakest" such condition, in that as many states as possible are acceptable (it has the largest possible kernel).

Dijkstra denotes this weakest precondition by $wp(t,g)$. The form $wp(t,-)$, with the post-condition left as a parameter, is called a predicate transformer. In our setting, a predicate transformer has the form $-\circ t$, called a linear functional transformer in linear algebra.

As an example, let k, $D = \{d_1,d_2\}$ and f_1 be as defined in section 2.4. Let $p_1:k^{(D)} \to k$ be the condition with $p_1(\varepsilon_{d_1}) = 0$ and $p_1(\varepsilon_{d_2}) = 1$, indicating that d_1 is the only acceptable state. The condition p_1 can be represented as the column vector $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. The weakest precondition $p_1 \circ f_1$ corresponds to the multiplication of the matrix for f_1 times this vector:

$$p_1 \circ f_1 = \begin{bmatrix} 1 & 0 \\ a & \bar{a} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{a} \end{bmatrix}$$

Thus, in the weakest precondition, state d_1 is always acceptable, and d_2 is acceptable iff console button a is pushed (*i.e.*, $\bar{a}$ is **false**). This may be informally read as: to establish state d_1 via program f_1, either start in state d_1 (in which case the position of console button a doesn't matter), or start in state d_2 with console button a pressed. If you're not sure whether the machine starts in state d_1 or d_2, then be sure to press console button a to establish state d_1.

4. MULTIPLE INPUT, MULTIPLE OUTPUT PROGRAM BEHAVIORS

Programs with multiple inputs or outputs require the use of tensor products as the appropriate algebraic structure in the face of nondeterminism. This use of tensor products is not new, but our presentation and motivation differ from previous uses [15,35,36]. Throughout this section, k is any commutative positive semiring.

4.1. Nondeterministic Pairs and Tensor Products

Suppose A is a k-module, considered as a set of nondeterministic distributions, as in section 2. The module A may be free, but we do not require this at the moment. Consider the set of all ordered pairs of distributions:

$$A \times A = \{(x,y) \mid x \in A \text{ and } y \in A\}.$$

This set is a k-module with $(x_1,y_1)+(x_2,y_2) = (x_1+x_2,y_1+y_2)$ and $r(x,y) = (rx,ry)$. For nondeterministic programs with two inputs or outputs, we want an element such as $(x_1,y_1)+(x_2,y_2)$ to be the nondeterministic choice between the pairs (x_1,y_1) and (x_2,y_2). Now, $(x_1,y_1)+(x_2,y_2)$ is an element of $A \times A$, but it is not the element we want, because in $A \times A$:

$$(x_1,y_1)+(x_2,y_2) = (x_1,y_2)+(x_2,y_1).$$

In short, there is no way to tell which distributions are paired together. If we choose some other k-module P to represent nondeterministic pairs of distributions, we want each pair $(x,y)\in A\times A$ to be contained in P. This can be achieved by a *function* $\otimes:A\times A\to P$ which takes each pair $(x,y)\in A\times A$ to an element $x\otimes y$ in P (note the infix notation). Following a suggestion of Hennessy and Plotkin [15], this function should be ***bilinear;*** that is, for all $x,y,z \in A$ and $r \in k$:

$$x\otimes(y+z) = (x\otimes y)+(x\otimes z),$$

$$(x+y)\otimes z = (x\otimes z)+(y\otimes z),$$

$$(rx\otimes y) = r(x\otimes y) = (x\otimes ry).$$

Intuitively, the first axiom can be interpreted as follows: If a pair of distributions has x as its first component and either y or z as its second component, then that is the same as the pair (x,y) or the pair (x,z). The second axiom has a similar meaning. The third axiom means that a requirement that is necessary for one component of a pair to occur is also needed for the pair as a whole to occur.

So, what we want is a k-module representing nondeterministic pairs of states, together with a bilinear function mapping $A\times A$ into this k-module. The tensor product provides a free or universal way to achieve this. Specifically, the tensor product of A with itself is a k-module $A\otimes A$ together with a bilinear function $\otimes:A\times A\to A\otimes A$. The function $\otimes$ is universal in the following sense: suppose P is some k-module and $g:A\times A\to P$ is a bilinear function. Then there is a unique k-module morphism $h:A\otimes A\to P$ such that the following triangle commutes:

$$\begin{tikzcd}
A\times A \arrow[r, "\otimes"] \arrow[dr, "g"'] & A\otimes A \arrow[d, dashed, "h"] \\
& P
\end{tikzcd}$$

Because of this universal property among bilinear functions, we use the k-module $A\otimes A$ as the set of nondeterministic distributions of pairs of states. The element $x\otimes y$ represents the pair of distributions (x,y).

4.2. Tensor Products of Free Modules

In general, the construction of the tensor product $A\otimes A$ involves reducing the free k-module $k^{(A\times A)}$ by an appropriate quotient [13,14,28]. However, in one important case -- when A is free -- the construction of $A\otimes A$ is straightforward and sheds additional light on why $A\otimes A$ is an appropriate space for nondeterministic distributions of pairs of states. For this case, we start with a set D of deterministic states. The set of nondeterministic distributions is the free k-module $k^{(D)}$, as in section 2. Following this line of thought, a *deterministic* pair of states is an element of

$$D\times D = \{(d,e) \mid d \in D \text{ and } e \in D\}.$$

The set of nondeterministic distributions of pairs of states is the free k-module $k^{(D\times D)}$ over $D\times D$. The following theorem justifies our use of tensor products by showing $k^{(D)}\otimes k^{(D)} = k^{(D\times D)}$. It is a generalization of a well-known theorem for ring modules [25].

Theorem 4.1. *For any set D, the free k-module $k^{(D\times D)}$ is a tensor product $k^{(D)}\otimes k^{(D)}$. The universal bilinear map $\otimes : k^{(D)}\times k^{(D)} \to k^{(D\times D)}$ is defined by*

$$\left(\sum_{d\in D} r_d d\right)\otimes\left(\sum_{e\,\in\, D} s_e e\right) \;=\; \sum_{(d,e)\,\in\, D\times D} (r_d\cdot s_e)(d,e).$$

Proof: First note that the above equation defines the bilinear map $\otimes$ completely since every element of $k^{(D)}$ is of the form $\sum_{d\in D} r_d d$. From the definition, it is easy to show that $\otimes$ is bilinear. We must show its universal property. Toward this end, let P be a k-module and $g : k^{(D)}\times k^{(D)} \to P$ a bilinear function. Now, g determines a function $f : D\times D \to P$ defined by $f(d,e) = g(\varepsilon_d, \varepsilon_e)$, where $\varepsilon_d, \varepsilon_e \in k^{(D)}$ are the basis elements corresponding to d and e. That is, f is the restriction of g to the basis elements. Since $k^{(D\times D)}$ is free over $D\times D$, the function f extends uniquely to a linear transformation $\bar{f} : k^{(D\times D)} \to P$. Standard algebraic manipulations show that $\bar{f}$ is the unique linear transformation making the triangle

$$\begin{tikzcd}
k^{(D)}\times k^{(D)} \arrow[r, "\otimes"] \arrow[dr, "g"'] & k^{(D\times D)} \arrow[d, dashed, "\bar{f}"] \\
& P
\end{tikzcd}$$

commute, as required. □

As a concrete example, suppose $x = r_1 d_1 + r_2 d_2$ and $y = s_1 d_1 + s_2 d_2$ are distributions. From theorem 4.1, we have:

$$\begin{aligned}
(x\otimes y) &= (r_1 d_1 + r_2 d_2)\otimes(s_1 d_1 + s_2 d_2) \\
&= (r_1 d_1 \otimes s_1 d_1) + (r_1 d_1 \otimes s_2 d_2) + (r_2 d_2 \otimes s_1 d_1) + (r_2 d_2 \otimes s_2 d_2) \\
&= r_1\cdot s_1 (d_1\otimes d_1) + r_1\cdot s_2 (d_1\otimes d_2) + r_2\cdot s_1 (d_2\otimes d_1) + r_2\cdot s_2 (d_2\otimes d_2).
\end{aligned}$$

In the last line, $(d_i\otimes d_j)$ is an abbreviation for $(\varepsilon_{d_i}\otimes\varepsilon_{d_j})$. The element $(x\otimes y)$ expresses all of the possible pairings of a deterministic state the distribution x with one in y. If we have additional information that only the d_1 states are paired together, and the d_2

states are similarly paired, then we can express this knowledge as:

$$(r_1 d_1 \otimes s_1 d_1) + (r_2 d_2 \otimes s_2 d_2) = r_1 \cdot s_1 (d_1 \otimes d_1) + r_2 \cdot s_2 (d_2 \otimes d_2)$$

within the tensor product. Expression of such information is not possible in the Cartesian product.

4.3. Iterated Tensor Products

Iterated tensor products can be used for nondeterministic n-tuples of states in the same way that $A \otimes A$ has been used for pairs. We begin with two arbitrary k-modules A and B, and the set of all pairs:

$$A \times B = \{(x,y) \mid x \in A \text{ and } y \in B\}$$

A function $f : A \times B \to P$ to a k-module P is called *bilinear* provided that for all $x, x' \in A$, $y, y' \in B$ and $r \in k$:

$$f(x+x',y) = f(x,y) + f(x',y),$$
$$f(x,y+y') = f(x,y) + f(x,y'),$$
$$f(rx,y) = r(f(x,y)) = f(x,ry).$$

The tensor product of A and B is a k-module $A \otimes B$ together with a universal bilinear function $\otimes : A \times B \to A \otimes B$; that is, if $g : A \times B \to P$ is a bilinear function, then there is a unique linear transformation $h : A \otimes B \to P$ such that $h(x \otimes y) = g(x,y)$ for all $x \in A$ and $y \in B$.

As before, a tensor product can be constructed by taking a quotient of the free module $k^{(A \times B)}$, but we only need the universal property of $A \otimes B$. From this universal property, we can show that any two tensor products of A and B are isomorphic. That is, if $f_1 : A \times B \to P_1$ and $f_2 : A \times B \to P_2$ are both universal bilinear functions, then $P_1 \cong P_2$. The isomorphism maps $f_1(x,y)$ to $f_2(x,y)$. For this reason, we generally speak of *the* tensor product $A \otimes B$.

Example: Let A be a k-module and also consider k as a k-module, as shown in section 2.3. The function $f : k\times A \to A$ which takes each pair (k,x) to $kx \in A$ is bilinear. Now, suppose $g : k\times A \to P$ is another bilinear function. Define a linear transformation $h : A \to P$ by $h(x) = g(1,x)$. Then for any pair $(k,x) \in k\times A$, we have $g(k,x) = g(1,kx) = h(kx) = h(f(k,x))$. Thus, the triangle

$$\begin{tikzcd}
k\times A \arrow[r, "f"] \arrow[dr, "g"'] & A \arrow[d, dashed, "h"] \\
 & P
\end{tikzcd}$$

commutes. Moreover, h is the unique linear transformation with this property. To see this, suppose $h' : A \to P$ is a linear transformation that makes the triangle commute. Then for any $x \in A$ we have $h(x) = h(f(1,x)) = g(1,x) = h'(f(1,x)) = h'(x)$, which implies $h = h'$. Thus, A is the tensor product of k and A, with $k\otimes x = kx$.

If $f : A \to A'$ and $g : B \to B'$ are two linear transformations, then we can define a linear transformation $f\otimes g : A\otimes B \to A'\otimes B'$ called the *tensor product* of f with g. To define $f\otimes g$, we first define a function $(f\times g) : A\times B \to A'\otimes B'$ which takes each pair $(x,y) \in A\times B$ to $f(x)\otimes g(y) \in A'\otimes B'$. This function is bilinear, so by universality, there is a unique linear transformation $f\otimes g : A\otimes B \to A'\otimes B'$ with $(f\otimes g)(x\otimes y) = (f\times g)(x,y) = f(x)\otimes g(y)$, for all $x \in A$ and $y \in B$. The linear transformation $f\otimes g$ is the function corresponding to the idea of executing f and g in parallel.

For example, suppose $f(d_1) = e_1$ and $f(d_2) = e_2$. Then:

$$(f\otimes f)(d_1\otimes d_2) = f(d_1)\otimes f(d_2) = e_1\otimes e_2$$

Example: Let $D=\{d_1,d_2\}$ and $f_1{:}k^{(D)}\to k^{(D)}$ be as in the previous example of section 2.4. Let $D'=\{d_3,d_4\}$ and define $g_1{:}k^{(D')}\to k^{(D')}$ as the unique linear transformation with $g_1(d_3)=1d_3$ and $g_1(d_4)=\bar{a}d_3+ad_4$. The matrix for g_1 is:

$$\begin{array}{c|cc} & d_3 & d_4 \\ \hline d_3 & 1 & 0 \\ d_4 & \bar{a} & a \end{array}$$

Now, $k^{(D)}\otimes k^{(D')}=k^{(D\times D')}$, which has four basis elements. We will denote these basis elements in $k^{(D)}\otimes k^{(D')}=k^{(D\times D')}$ by

$$\{d_1\otimes d_3,d_1\otimes d_4,d_2\otimes d_3,d_2\otimes d_4,\},$$

where $d_i\otimes d_j$ is an abbreviation for the basis element $\varepsilon_{(d_i,d_j)}$ (or equivalently $\varepsilon_{d_i}\otimes\varepsilon_{d_j}$). We can write $f_1\otimes g_1$ using these basis elements. For example:

$$\begin{aligned}(f_1\otimes g_1)(d_2\otimes d_4) &= f_1(d_2)\otimes g_1(d_4)\\ &= (ad_1+\bar{a}d_2)\otimes(\bar{a}d_3+ad_4)\\ &= a\cdot\bar{a}\,(d_1\otimes d_3)+a\cdot a\,(d_1\otimes d_4)+\bar{a}\cdot\bar{a}\,(d_2\otimes d_3)+\bar{a}\cdot a\,(d_2\otimes d_4)\\ &= a\,(d_1\otimes d_4)+\bar{a}\,(d_2\otimes d_3).\end{aligned}$$

The final equality is because k (in this example) is a Boolean algebra with $\bar{a}$ the negation of a -- *i.e.*, $a\cdot\bar{a}=\bar{a}\cdot a=0$ and $a\cdot a=a$ and $\bar{a}\cdot\bar{a}=\bar{a}$. The value of $f_1\otimes g_1$ at the other three basis elements is given in this matrix:

$$\begin{array}{c|cccc} & d_1\otimes d_3 & d_1\otimes d_4 & d_2\otimes d_3 & d_2\otimes d_4 \\ \hline d_1\otimes d_3 & 1 & 0 & 0 & 0 \\ d_1\otimes d_4 & \bar{a} & a & 0 & 0 \\ d_2\otimes d_3 & a & 0 & \bar{a} & 0 \\ d_2\otimes d_4 & 0 & a & \bar{a} & 0 \end{array}$$

Compare this matrix with the individual matrices for f_1 and g_1 to see how the tensor product causes f_1 and g_1 to act "in parallel". In this tensor product, f_1 and g_1 do not explicitly communicate with one another by sending messages or otherwise modifying each other's state. Both accept the condition from the outside world and act in concert depending upon the condition. This is not communication between f_1 and g_1, but rather from the outside world to both f_1 and g_1. An alternative view of essentially the same idea is given by Kurshan and Gopinath [21,22].

Now we can define iterated tensor products, which will be used for nondeterministic distributions of n-tuples, much as $A \otimes A$ was used for pairs. If C is a k-module, then the tensor product of C with $A \otimes B$ can be formed, denoted $C \otimes (A \otimes B)$. This k-module is called an *iterated tensor product* and is isomorphic to the tensor product $(C \otimes A) \otimes B$. The isomorphism maps each $x \otimes (y \otimes z)$ to $(x \otimes y) \otimes z$. Because of this isomorphism, we generally write $(x \otimes y \otimes z) \in (C \otimes A \otimes B)$, without specifying the order of the $\otimes$ operations.

For any k-module A, we can form a sequence of iterated tensor products as follows:

$$\bigotimes^0 A = k\,,$$

$$\text{for } n \geq 1\text{:} \quad \bigotimes^n A \;=\; (\bigotimes^{n-1} A) \otimes A \;=\; A \otimes \cdots \otimes A \ (n \text{ factors}).$$

Note that $\bigotimes^1 A = k \otimes A = A$, from the example given earlier. More generally, $\bigotimes^{p+q} A = (\bigotimes^p A) \otimes (\bigotimes^q A)$. This last equality gives a way of combining two linear transformations $f : \bigotimes^n A \to \bigotimes^p A$ and $g : \bigotimes^l A \to \bigotimes^q A$. Specifically, the tensor product of these two linear transformations is a linear transformation $f \otimes g : (\bigotimes^{n+l} A) \to (\bigotimes^{p+q} A)$.

4.4. Nondeterministic n-tuples

The importance of $\bigotimes^n A$ to us is this: for each $n \geq 0$, we want to have nondeterministic distributions of n-tuples. In general, the k-module $A^n = A \times \cdots \times A$ (n factors) is unsuitable for the same reason that $A \times A$ is not appropriate for pairs. What we need is a k-module P and a mapping $\otimes : A^n \to P$, which is linear in each of its n components. For example, if $n = 3$, then we require:

$$(w+x)\otimes y\otimes z=(w\otimes y\otimes z)+(x\otimes y\otimes z),$$

$$x\otimes(w+y)\otimes z=(x\otimes w\otimes z)+(x\otimes y\otimes z),$$

$$x\otimes y\otimes(w+z)=(x\otimes y\otimes w)+(x\otimes y\otimes z),$$

$$r(x\otimes y\otimes z)=(rx\otimes y\otimes z)=(x\otimes ry\otimes z)=(x\otimes y\otimes rz).$$

Such a function is called *multilinear*.

The iterated tensor product $\overset{n}{\otimes}A$ provides a universal multilinear function with domain A^n. That is, there is a universal multilinear function $\otimes: A^n \to \overset{n}{\otimes}A$, with the image of $(x_1, \cdots, x_n) \in A^n$ being written $(x_1\otimes \cdots \otimes x_n)$. This multilinear function is free in exactly the same way that $\otimes: A\times A \to A\otimes A$ is free for bilinear functions. Thus, if $g: A^n \to P$ is a multilinear function, then there is a unique linear transformation $h: \overset{n}{\otimes}A \to P$ with $h(x_1\otimes \cdots \otimes x_n) = g(x_1, \cdots, x_n)$.

Because of this universal property, we use the k-module $\overset{n}{\otimes}A$ as the set of nondeterministic distributions of n-tuples of states. The tensor product $\overset{n}{\otimes}A$ is particularly simple when $A = k^{(D)}$ is free over some set D:

Theorem 4.2. *For any set D, the free k-module $k^{(D^n)}$ is a tensor product $\overset{n}{\otimes} k^{(D)}$. The universal multilinear map $\otimes: \left[k^{(D)}\right]^n \to k^{(D^n)}$ is defined by*

$$\left[\varepsilon_{d_1}\otimes \cdots \otimes\varepsilon_{d_n}\right] \quad = \quad \varepsilon_{<d_1, \cdots, d_n>},$$

where the ε_x are the appropriate basis elements in the free k-modules.

Proof: Generalize theorem 4.1. □

4.5. Program Behaviors

Given a k-module A as a set of nondeterministic distributions, the k-module $\overset{n}{\otimes}A$ is the set of nondeterministic distributions of n-tuples of states. The functional behavior of a program with an n-tuple of inputs and a p-tuple of outputs is a linear transformation $f:\overset{n}{\otimes}A \to \overset{p}{\otimes}A$. The requirement of linearity is motivated by the same reasoning used in section 2.5 for single-input, single-output programs. Not all linear transformation are programs. Since zero is not a possible output, a linear transformation that represents a program behavior must be positive.

Notice that because of multilinearity, zero has an annihilator property in distributions of tuples. That is $x_1 \otimes \cdots \otimes 0 \otimes \cdots \otimes x_n = 0$. This means that if any one component of a tuple is impossible, then the entire tuple is impossible. The converse is not true -- that is $x_1 \otimes \cdots \otimes x_n$ may be zero even if none of the x_i are zero. In this case, $(x_1, \cdots, x_n)$ can be thought of as an inconsistent tuple. For example, suppose r and s are elements of the semiring k with $r \cdot s = 0$. This might occur if k is a distributive lattice of predicates, with r and s mutually incompatible. For any $x, y \in A$ we have

$$(rx \otimes sy) = (r \cdot s)(x \otimes y) = 0(x \otimes y) = 0.$$

This means that the distribution rx can never be paired with the distribution sy. In our running example, ad_1 can never be paired with $\bar{a}d_3$. Either the console button a is pushed or it isn't, irrespective of the machine state.

One advantage of this view of program behaviors is that it indicates several different methods of specifying a program $f:\overset{n}{\otimes}A \to \overset{p}{\otimes}A$ with n inputs and p outputs. These methods are listed here:

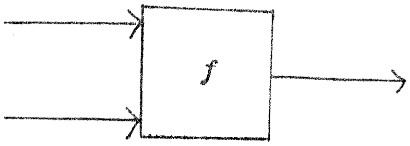
(1) **Deterministic inputs method:** If $A = k^{(D)}$ is a free k-module over D, then $\overset{n}{\otimes}A = k^{(D^n)}$ is free over the set D^n. Therefore, every function $f:D^n \to \overset{p}{\otimes}A$ determines a unique linear transformation $\bar{f}:\overset{n}{\otimes}A \to \overset{p}{\otimes}A$. Moreover, from the fact that k is positive, it can be shown that $\bar{f}$ is a positive transformation iff $f(d_1, \cdots, d_n) \neq 0$ for all n-tuples $(d_1, \cdots, d_n) \in D^n$. This method is equivalent to specifying a nondeterministic program by giving its result for any deterministic input.

(2) **Parallel method:** Suppose $f : \overset{n}{\otimes} A \to \overset{p}{\otimes} A$ and $g : \overset{l}{\otimes} A \to \overset{q}{\otimes} A$ are linear transformations. Then their tensor product $f \otimes g : (\overset{n+l}{\otimes} A) \to (\overset{p+q}{\otimes} A)$ is a linear transformation which may be thought of as f and g acting in parallel with no communication. If f and g are both positive transformations, this does not guarantee that $f \otimes g$ is also a positive transformation. In particular, suppose $f, g : A \to A$ with $f(x) = ry$ and $g(x) = sy$ for some $x, y \in A$ and $r, s \in k$. If $r \cdot s = 0$ in k, then $f \otimes g(x \otimes x) = ry \otimes sy = 0$. This indicates that the conditions under which f may accept x as input are mutually incompatible with the conditions under which g may accept x -- i.e., $r \cdot s = 0$.

(3) **Multilinear method:** Suppose $g : A^n \to \overset{p}{\otimes} A$ is a multilinear function. Then g extends to a unique linear transformation $h : \overset{n}{\otimes} A \to \overset{p}{\otimes} A$ with $h(x_1 \otimes \cdots \otimes x_n) = g(x_1, \cdots, x_n)$. Thus, we can specify a linear transformation $h : \overset{n}{\otimes} A \to \overset{p}{\otimes} A$ by giving its corresponding multilinear function $g : A^n \to \overset{p}{\otimes} A$. The linear transformation h is a positive transformation iff $g(x_1, \cdots, x_n) = 0$ implies $(x_1 \otimes \cdots \otimes x_n) = 0$. This follows from the fact that k is positive and $\overset{n}{\otimes} A$ is spanned by elements of the form $(x_1 \otimes \cdots \otimes x_n)$.

5. CONCURRENT COMPUTATIONS

The notion of concurrent computation that we examine here is a fixed network of nodes interconnected by data paths, similar to MacQueen's model [26] or that of Faustini [10]. A node is a server, accepting messages at its input ports and producing messages at its output ports. A server is entirely functional and carries no internal state. Graphically, a server with two input ports and one output port may be drawn like this:



For alternate views of servers which have internal states, see [30,31,32,33,40]. We develop an algebraic structure on the set of possible inputs to a node and we characterize a node as "communicating" or "noncommunicating" depending on its functional behavior with respect to this algebraic structure.

5.1. Message Configurations

We begin with a set M of messages which may traverse the network. These messages are deterministic entities whose internal structure is unimportant. They may be simple integers, or complex "active" messages, as Wall advocates [42]. Any finite number of messages may exist concurrently on a data path. In other words, a data path contains a finite multiset of elements from the set M.

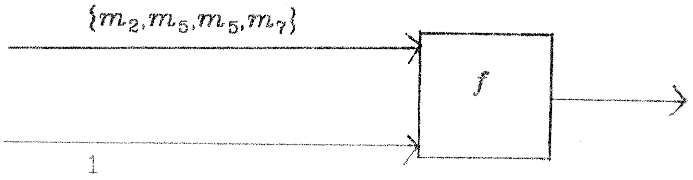
A finite multiset of messages is called a *configuration*. For example, a data path might contain the configuration $\{m_2, m_5, m_5, m_7\}$. This means that m_2, m_7 and two copies of m_5 exist *concurrently* on the path. The occurrences of these four messages are not explicitly ordered in time, although they could implicitly be ordered through some structure on M. (For example, Pratt has considered such ordering [37].)

Given a set M of messages, the collection of all possible configurations is a commutative monoid under the operation of multiset union. We denote this monoid by $\mathbf{M}(M)$ and use multiplicative notation for union. For example,

$$\{m_2, m_5\} \cdot \{m_5, m_7\} = \{m_2, m_5, m_5, m_7\}.$$

In general, for configurations c and d, the union configuration $c \cdot d$ consists of the messages of c existing concurrently with those of d. For this reason, multiplication is called the *concurrency operation*. For consistency with multiplicative notation, we use 1 for the empty multiset.

Example:



The diagram illustrates a node or server, f, with the configuration $\{m_2,m_5,m_5,m_7\}$ awaiting service at the first input port and no message awaiting service at the second input port.

In this analysis we equate 1 (the empty multiset) with $\perp$ (nontermination, with no output). Here is the reason: If we can only observe the output of a server without knowing the internal processing, we only see the multiset of states from those messages which have been served. Hence, $\perp$ is not observably different from the empty multiset.

5.2. Communicating and Noncommunicating Behaviors

A single input, single output deterministic server has as its behavior a function

$$f : \mathbf{M}(M) \rightarrow \mathbf{M}(M)$$

In general, a server behavior is not a monoid morphism. That is, it may be that for some $c, d \in \mathbf{M}(M)$, $f(c \cdot d) \neq f(c) \cdot f(d)$. Also, $f(1)$ need not be 1. The first inequality will occur if the presence of c effects the computation on d or vice versa. Considered as active messages, c and d may communicate. The inequality $f(1) \neq 1$ occurs if the server is capable of producing output without receiving input. In general, a behavior $f : \mathbf{M}(M) \rightarrow \mathbf{M}(M)$ is called *noncommunicating* if it is a monoid morphism; otherwise it is called *communicating*. This terminology fits our interpretation of 1, since $f(1) \neq 1$

implies that the server has some implicit communication with the world which is not represented by in its single input.

5.3. Nondeterminism

The situation at a port may be nondeterministic. A nondeterministic distribution of configurations formalizes this situation. Explicitly, we take the set $\mathbf{M}(M)$ of configurations as our set of deterministic states. Then, the set of nondeterministic distributions of configurations is a free k-module, $k^{(\mathbf{M}(M))}$, for some commutative positive semiring k (as shown in section 2).

A basis element $\varepsilon_d \in k^{(\mathbf{M}(M))}$, whose coefficient at $d \in \mathbf{M}(M)$ is 1 and all other coefficients are zero, corresponds to a deterministic configuration. The concurrency operation can be defined on these deterministic configurations by $\varepsilon_c \cdot \varepsilon_d = \varepsilon_{c \cdot d}$. To extend the operation to all of $k^{(\mathbf{M}(M))}$, we take the axioms:

$$x \cdot (y+z) = (x \cdot y)+(x \cdot z),$$

$$(rx) \cdot y = r(x \cdot y).$$

The first axiom means that if there are two concurrent configurations, one of which is x and the other is y **or** z, then that is the same as either $x \cdot y$ **or** $x \cdot z$. The second equality assures that a condition on one component of a configuration is a condition for the entire configuration. With these axioms, there is a unique way to extend the concurrency operation to all of $k^{(\mathbf{M}(M))}$. The extension is:

$$\sum_{d \in \mathbf{M}(M)} r_d d \cdot \sum_{c \in \mathbf{M}(M)} s_c c = \sum_{d \in \mathbf{M}(M)} \sum_{c \in \mathbf{M}(M)} (r_d \cdot s_c)(d \cdot c), \qquad 5.1$$

The identity for the operation is ε_1, which we usually denote by just 1.

This definition means that $<k^{(\mathbf{M}(M))},+,\cdot,0,1>$ is a commutative positive semiring. Furthermore, it is a commutative k-algebra, as defined here: Let k be a commutative semiring. A k-*algebra* is a semiring $<A,+,\cdot,0,1>$ such that the monoid $<A,+,0>$ is also a k-module with scalar multiplication satisfying

$$(rx)\cdot y = r(x\cdot y) = x\cdot(ry)$$

for all $r \in k$ and $x,y \in A$.

A k-algebra is commutative if the semiring multiplication is commutative. A function between k-algebras is a k-algebra morphism provided it is simultaneously a semiring morphism and a linear transformation of k-modules.

Examples: Let k be a commutative semiring. Then k is a k-algebra by defining the scalar multiplication to be the semiring multiplication of k. We denote this k-algebra by k. Let C be a commutative monoid and $k^{(C)}$ be the free k-module over the set C. The set of basis elements of $k^{(C)}$ are a commutative monoid with $\varepsilon_c \cdot \varepsilon_d = \varepsilon_{c\cdot d}$. The multiplication extends to all of $k^{(C)}$ by equation 5.1, making $k^{(C)}$ a commutative k-algebra. This is exactly how $k^{(\mathbb{M}(M))}$ was made into a k-algebra above. Moreover, the commutative k-algebra $k^{(\mathbb{M}(M))}$ is free over the set M, as shown in the following theorem:

Theorem 5.1. *Let $\eta: M \to k^{(\mathbb{M}(M))}$ be the function which maps each $m \in M$ to the formal sum η_m whose coefficient is 1 at the multiset $\{m\} \in \mathbb{M}(M)$ and 0 elsewhere. Suppose A is a commutative k-algebra and $f: M \to A$ is any function. Then there exists a unique k-algebra morphism $\bar{f}: k^{(\mathbb{M}(M))} \to A$ with $\bar{f}(\eta_m) = f(m)$, as in the following commuting triangle:*

$$\begin{tikzcd}
M \arrow[r, "\eta"] \arrow[dr, "f"'] & k^{(\mathbb{M}(M))} \arrow[d, dashed, "\bar{f}"] \\
 & A
\end{tikzcd}$$

Proof: First, we extend the domain of f to any multiset in $\mathbb{M}(M)$ by defining $f(\{m_1, \cdots m_n\}) = f(m_1) \cdot \ldots \cdot f(m_n)$ and $f(1) = 1$. Now, let $\sum_{d \in \mathbb{M}(M)} r_d d$ be an element of $k^{(\mathbb{M}(M))}$. We define $\overline{f}(\sum_{d \in \mathbb{M}(M)} r_d d)$ to be the sum $\sum_{d \in \mathbb{M}(M)} r_d(f(d))$ in A. This exists in A since only a finite number of the coefficients r_d are nonzero in any formal sum. For any $m \in M$ we have $\overline{f}(\eta_m) = f(\{m\}) = f(m)$, so the triangle commutes. Standard techniques show that $\overline{f}$ is a k-algebra morphism, and the only one which makes the triangle commute. □

The behavior of a single-input, single output nondeterministic server is a linear transformation

$$f : k^{(\mathbb{M}(M))} \to k^{(\mathbb{M}(M))}.$$

As in the deterministic case, f does not need to preserve the concurrency operator or 1, so in general f is not a k-algebra morphism. If f is a k-algebra morphism, then we call it *noncommunicating;* otherwise it is *communicating*. The following theorem gives a necessary and sufficient condition for a behavior to be noncommunicating:

Theorem 5.2. *Let A and B be k-algebras and suppose $X \subseteq A$ spans the k-module A. A linear transformation $f : A \to B$ is a k-algebra morphism iff $f(1) = 1$ and $f(x \cdot y) = f(x) \cdot f(y)$ for all $x, y \in X$.* •

Proof: Clearly the conditions on f are necessary. To show that they are sufficient, we must show that f preserves arbitrary multiplication whenever it preserves multiplication of elements from X. Toward this end, let $\sum_{i=1}^{p} r_i x_i$ and $\sum_{j=1}^{q} s_j y_j$ be arbitrary elements of A, where (x_i) and (y_j) are sequences from X. (Recall that every element of A must have this form since X spans the k-module A.) Then:

$$f\left[\sum_{i=1}^{p} r_i x_i \cdot \sum_{j=1}^{q} s_j y_j\right]$$

$$= f\left[\sum_{i=1}^{p}\sum_{j=1}^{q}(r_i x_i)\cdot(s_j y_j)\right] \qquad \text{(multiplication distributes over +)}$$

$$= f\left[\sum_{i=1}^{p}\sum_{j=1}^{q}(r_i\cdot s_j)(x_i\cdot y_j)\right] \qquad \text{(pull out scalars)}$$

$$= \sum_{i=1}^{p}\sum_{j=1}^{q}(r_i\cdot s_j)(f(x_i\cdot y_j)) \qquad \text{(f is linear transformation)}$$

$$= \sum_{i=1}^{p}\sum_{j=1}^{q}(r_i\cdot s_j)(f(x_i)\cdot f(y_j)) \qquad \text{(f preserves multiplication on X)}$$

$$= \sum_{i=1}^{p} r_i f(x_i) \cdot \sum_{j=1}^{q} s_j f(y_j)$$

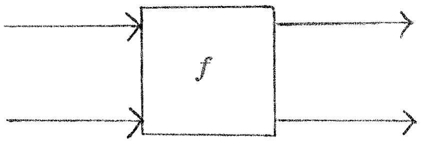
$$= f\left[\sum_{i=1}^{p} r_i x_i\right] \cdot f\left[\sum_{j=1}^{q} s_j y_j\right].$$

Therefore, f preserves multiplication and is a k-algebra morphism. □

For the k-module $k^{(\mathbf{M}(M))}$, the deterministic configurations ε_d ($d \in \mathbf{M}(M)$) form a basis. Therefore, a linear transformation $f: k^{(\mathbf{M}(M))} \to k^{(\mathbf{M}(M))}$ is noncommunicating iff it is noncommunicating for deterministic inputs.

5.4. Multiple Ports

Servers with multiple entry and exit ports require some development to explicate their functional behavior. To begin, consider a server with two entry and two exit ports:



A message $m \in M$ either enters f via the first port, denoted $<1,m>$, or via the second, $<2,m>$. Thus, the set of all possible deterministic inputs to f is the disjoint union (or coproduct):

$$M \sqcup M = \{<i,m> \mid m \in M \text{ and } (i=1 \text{ or } i=2)\}.$$

This idea of using coproducts to record the data line of an input is from Elgot [8,9] and the intuition is developed by Lorentz and Benson [23].

To allow more than one input message concurrently, we take the monoid $\mathbb{M}(M \sqcup M)$. This is the collection of all finite multisets with elements from $M \sqcup M$, such as $\{<1,m_2>,<2,m_2>,<2,m_5>,<2,m_5>\}$. Similarly, the set of deterministic outputs for f is the monoid $\mathbb{M}(M \sqcup M)$. So, in the deterministic case, the functional behavior of the server is a function $f : \mathbb{M}(M \sqcup M) \rightarrow \mathbb{M}(M \sqcup M)$. In the nondeterministic case, the behavior is a linear transformation $f : k^{(\mathbb{M}(M \sqcup M))} \rightarrow k^{(\mathbb{M}(M \sqcup M))}$, for some commutative positive semiring k.

More generally, the possible messages that can arise on n data lines are elements of the set

$$n \bullet M = \{<i,m> \mid m \in M \text{ and } 1 \leq i \leq n\}.$$

The set of possible configurations for n data lines is the monoid $\mathbb{M}(n \bullet M)$, consisting of all finite multisets with elements from $n \bullet M$. Finally, a nondeterministic distribution is an element of the free k-module $k^{(\mathbb{M}(n \bullet M))}$ for some commutative positive semiring k. This k-module is a k-algebra by extending the multiplication of $\mathbb{M}(n \bullet M)$ to all of $k^{(\mathbb{M}(n \bullet M))}$, as shown in the last section.

The functional behavior of a nondeterministic multiprocess server with n entry ports and p exit ports is a linear transformation:

$$f : k^{(\mathbb{M}(n \bullet M))} \rightarrow k^{(\mathbb{M}(p \bullet M))}.$$

If f is also a k-algebra morphism, then it is a noncommunicating behavior; otherwise it is communicating.

5.5. Isomorphic Algebras

In the last section, we showed how nondeterministic distributions of configurations form a k-algebra $k^{(\mathbb{M}(n \bullet M))}$, where n is the number of data lines or ports, and M is the set of possible messages. We will show that $k^{(\mathbb{M}(n \bullet M))}$ is isomorphic to two other k-algebras.

Let $\mathbb{M}(M)^n$ be the n-fold product monoid, consisting of all n-tuples:

$$\mathbb{M}(M)^n = \{<d_1, \cdots, d_n> \mid d_i \in \mathbb{M}(M) \text{ for } 1 \leq i \leq n\}.$$

Multiplication is by components $<c_1, \cdots, c_n> \cdot <d_1, \cdots, d_n> = <c_1 \cdot d_1, \cdots, c_n \cdot d_n>$ and the identity is $<1, \cdots, 1>$. The monoid $\mathbb{M}(M)^n$ is commutative since $\mathbb{M}(M)$ is. The k-module $k^{(\mathbb{M}(M)^n)}$ is made into a k-algebra by extending the multiplication of $\mathbb{M}(M)^n$, as shown in section 5.3.

Theorem 5.3. *The k-algebras $k^{(\mathbb{M}(n \bullet M))}$ and $k^{(\mathbb{M}(M)^n)}$ are isomorphic.*

Proof outline: The isomorphism follows immediately from the fact that the commutative monoid $\mathbb{M}(n \bullet M)$ is isomorphic to the monoid $\mathbb{M}(M)^n$. The monoid isomorphism maps each singleton multiset $\{<i,m>\}$ to the n-tuple $<1, \cdots, 1, \{m\}, 1, \cdots, 1>$, where the $\{m\}$ appears in the i^{th} component. □

Let $\bigotimes^n k^{(\mathbb{M}(M))}$ be the iterated tensor product of the module $k^{(\mathbb{M}(M))}$ with itself n times, as in section 4.3. There is a unique way to make the module $\bigotimes^n k^{(\mathbb{M}(M))}$ into a k-algebra with:

$$(x_1 \otimes \cdots \otimes x_n) \cdot (y_1 \otimes \cdots \otimes y_n) = (x_1 \cdot y_1 \otimes \cdots \otimes x_n \cdot y_n).$$

The complete construction of this k-algebra is identical to the case of ring algebras [25, XVI.4], but the only fact we need is the above equality.

Theorem 5.4. *The k-algebras $k^{(\mathbb{M}(M)^n)}$ and $\bigotimes^n k^{(\mathbb{M}(M))}$ are isomorphic.*

Proof: We have already shown that the function $\delta : k^{(\mathbb{M}(M)^n)} \to \bigotimes^n k^{(\mathbb{M}(M))}$ mapping each basis element $\varepsilon_{<d_1, \cdots, d_n>}$ to $\varepsilon_{d_1} \otimes \cdots \otimes \varepsilon_{d_n}$ is an isomorphism of k-modules (theorem 4.2). Now, we will show that it is also a k-algebra morphism, hence an isomorphism. First note that δ does preserve the identity for multiplication -- specifically $\delta(\varepsilon_{<1, \cdots, 1>}) = \varepsilon_1 \otimes \cdots \otimes \varepsilon_1$ is the multiplicative identity in $\bigotimes^n k^{(\mathbb{M}(M))}$. Also, δ preserves multiplication of basis elements since:

$$
\begin{aligned}
\delta(\varepsilon_{<c_1,\cdots,c_n>}\cdot\varepsilon_{<d_1,\cdots,d_n>}) &= \delta(\varepsilon_{<c_1\cdot d_1,\cdots,c_n\cdot d_n>}) \\
&= \varepsilon_{c_1\cdot d_1}\otimes\cdots\otimes\varepsilon_{c_n\cdot d_n} \\
&= \varepsilon_{c_1}\cdot\varepsilon_{d_1}\otimes\cdots\otimes\varepsilon_{c_n}\cdot\varepsilon_{d_n} \\
&= (\varepsilon_{c_1}\otimes\cdots\otimes\varepsilon_{c_n})\cdot(\varepsilon_{d_1}\otimes\cdots\otimes\varepsilon_{d_n}) \\
&= \delta(\varepsilon_{<c_1,\cdots,c_n>})\cdot\delta(\varepsilon_{<d_1,\cdots,d_n>}).
\end{aligned}
$$

Finally, by theorem 5.2, this implies that δ is a k-algebra morphism. □

Combining these last two theorems gives a final isomorphism:

Theorem 5.5. *The k-algebras* $k^{(\mathbb{M}(n\bullet M))}$ *and* $\bigotimes^n k^{(\mathbb{M}(M))}$ *are isomorphic.* □

These isomorphisms are used in the next section to give several ways of specifying the functional behavior of multiprocess servers.

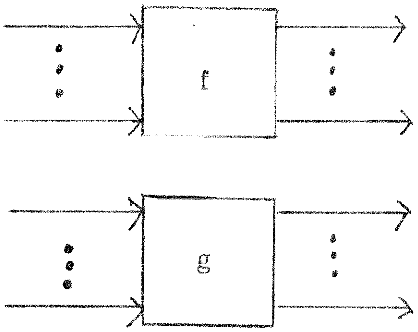
5.6. Specifying Multiprocess Behaviors

The functional behavior of a nondeterministic multiprocess server with n input ports and p output ports is a linear transformation from the free commutative k-algebra $k^{(M(n\bullet M))}$ to the free commutative k-algebra $k^{(M(p\bullet M))}$. If it is also a k-algebra morphism, then it is a noncommunicating behavior. From the isomorphisms of the last section, the behavior can alternately be viewed as a linear transformation $f:\bigotimes^n k^{(\mathbb{M}(M))}\to\bigotimes^p k^{(\mathbb{M}(M))}$ or $f:k^{(\mathbb{M}(M)^n)}\to k^{(\mathbb{M}(M)^p)}$. Here are four ways of specifying such a behavior:

(1) **Noncommunicating Method:** The domain $k^{(\mathbb{M}(n\bullet M))}$ is the free commutative k-algebra over $n\bullet M$ (theorem 5.1). This means that every k-algebra morphism $f:k^{(\mathbb{M}(n\bullet M))}\to k^{(\mathbb{M}(p\bullet M))}$ is completely specified by giving a function from the set $n\bullet M$ to $k^{(\mathbb{M}(p\bullet M))}$. Any noncommunicating behavior may be specified in this way.

(2) **Communicating Method:** The domain $k^{(\mathbb{M}(M)^n)}$ is the free k-module over the set $\mathbb{M}(M)^n$. Therefore, every linear transformation $f: k^{(\mathbb{M}(M)^n)} \to k^{(\mathbb{M}(M)^p)}$ is completely specified by giving a function from $\mathbb{M}(M)^n$ to $k^{(\mathbb{M}(M)^p)}$. A behavior specified in this way is noncommunicating iff the underlying function from $\mathbb{M}(M)^n$ to $k^{(\mathbb{M}(M)^p)}$ is a monoid morphism.

(3) **Tensor Method:** Let $f: \bigotimes^n k^{(\mathbb{M}(M))} \to \bigotimes^p k^{(\mathbb{M}(M))}$ and $g: \bigotimes^l k^{(\mathbb{M}(M))} \to \bigotimes^q k^{(\mathbb{M}(M))}$ be linear transformations. Then $f \otimes g: \bigotimes^{n+l} k^{(\mathbb{M}(M))} \to \bigotimes^{p+q} k^{(\mathbb{M}(M))}$ is also a linear transformation. This can be viewed as the servers f and g acting in parallel, with no communication between them. As a diagram, it can be drawn like this:



The behavior $f \otimes g$ is noncommunicating iff both f and g are.

(4) **Multilinear Method:** From the definition of tensor products, each multilinear function $g: (k^{(\mathbb{M}(M))})^n \to \bigotimes^p k^{(\mathbb{M}(M))}$ extends to a unique linear transformation $h: \bigotimes^n k^{(\mathbb{M}(M))} \to \bigotimes^p k^{(\mathbb{M}(M))}$ with $h(x_1 \otimes \cdots \otimes x_n) = g(x_1, \cdots, x_n)$.

Examples: (i) a server called **fork** has one input and two outputs. Intuitively, it copies any message at its single input port to both of its output ports. The **fork** server is noncommunicating, so it suffices to define $\mathbf{fork}(m) = m \otimes m$, for each $m \in M$. (Here, and later, we let $m \in M$ also denote the formal sum $\varepsilon_{\{m\}} \in k^{(\mathbb{M}(M))}$.)

(ii) a server called **choice** also has one input and two outputs. An input presented to its input port is to be copied to one or the other of its output ports, but not both. The choice as to which output port is nondeterministic. This is given by defining **choice**$(m) = (1 \otimes m) + (m \otimes 1)$, for each $m \in M$.

(iii) the concurrency operation is a server with two input ports and one output port. It copies any message at either input port to its single output port. This action is noncommunicating, and is the extension of the function $f : M \sqcup M \to k^{(\mathbb{M}(M))}$ defined by $f(<i,m>) = m$.

(iv) there are a number of ways to define other servers with two input ports and one output port. The motivation is that each pair of messages, one per input port, gives rise to a single message -- possibly nondeterministically. In general one has **join**$(m \otimes 1) = 1 =$ **join**$(1 \otimes m)$ for any $m \in M$ and **join**$(m_1 \otimes m_2)$ can be arbitrarily defined $(m_1, m_2 \in M)$. There are then numerous different ways of extending the domain of **join** to all of $k^{(\mathbb{M}(M))} \otimes k^{(\mathbb{M}(M))}$ [2,3]. Clearly each such **join** server is communicating.

6. BIGEBRAS

Let M be any set and $A = k^{(\mathbb{M}(M))}$ be a free commutative k-algebra over M for some commutative positive semiring k. As shown in the last section, the functional behavior of a multiprocess server with n inputs and p outputs is a linear transformation $f : \overset{n}{\otimes} A \to \overset{p}{\otimes} A$. If f is also a k-algebra morphism, then it is called noncommunicating. This section explores the algebraic properties of two particular noncommunicating behaviors **fork**$: A \to A \otimes A$ and **choice**$: A \to A \otimes A$. The **fork** corresponds to a message duplicating itself and continuing along two distinct data lines. The **choice** behavior corresponds to a message nondeterministically choosing one of two possible data lines.

Throughout this section, we consider M as a subset of A by identifying each $m \in M$ with the formal sum in A whose coefficient at $\{m\} \in \mathbb{M}(M)$ is 1 and all other coefficients are zero. The importance of this subset is that any k-algebra morphism $f : A \to B$ to a commutative k-algebra B is completely determined by its action on M, and conversely, any function from M to B extends to a unique k-algebra morphism (theorem 5.1).

6.1. A Semiring of Morphisms

Let B be any commutative k-algebra. The set of k-algebra morphisms from $A = k^{(\mathbf{M}(M))}$ to B forms a commutative semiring. For $f, g : A \to B$, we define $f+g$ and $f \cdot g$ by their action on an arbitrary $m \in M$:

$$(f+g)(m) = f(m)+g(m),$$

$$(f \cdot g)(m) = f(m) \cdot g(m).$$

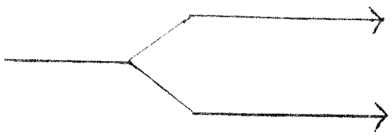
The zero for addition is the morphism that takes every $m \in M$ to the zero in B, while the unit is the morphism that takes every $m \in M$ to the unit in B. Viewed as a multiprocess server, $f+g$ is nondeterministic; it is either f or g, but not both. In a similar way, $f \cdot g$ consists of f and g acting concurrently on the *same* input.

This semiring of morphisms is characterized by two particular morphisms: **choice**:$A \to A \otimes A$ and **fork**:$A \to A \otimes A$. These are defined by their values at any $m \in M$:

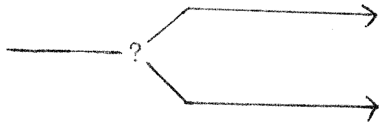
$$\textbf{choice}(m) = (m \otimes 1)+(1 \otimes m),$$

$$\textbf{fork}(m) = (m \otimes 1) \cdot (1 \otimes m) = (m \otimes m).$$

In concurrent processing, these are important processes. The **fork** morphism allows a message to split into two identical messages, each of which proceeds along a separate data path. As a flowchart, it can be drawn:



Any data entering at the left is duplicated at the fork and each copy continues independently. The **choice** morphism sends a state to one or the other of its output lines, but not both. The decision as to which output line is taken is nondeterministic. As a flowchart, **choice** can be drawn:

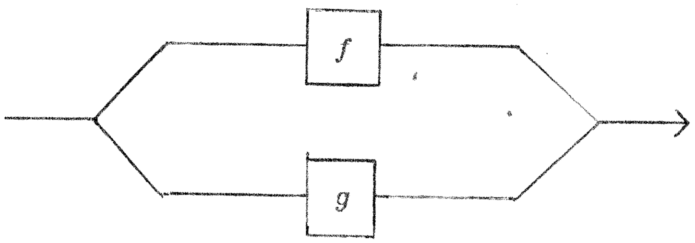


Let **concur**:$B \otimes B \to B$ be the multiplication which takes each $(x \otimes y) \in B \otimes B$ to $x \cdot y \in B$. We have the following identity for all k-algebra morphisms $f, g : A \to B$:

$$\mathbf{concur} \circ (f \otimes g) \circ \mathbf{fork} = f \cdot g.$$

$$\begin{tikzcd}
A \arrow[r, "\mathbf{fork}"] \arrow[d, "f \cdot g"'] & A \otimes A \arrow[d, "f \otimes g"] \\
B & B \otimes B \arrow[l, "\mathbf{concur}"]
\end{tikzcd}$$

The flowchart for this situation is:

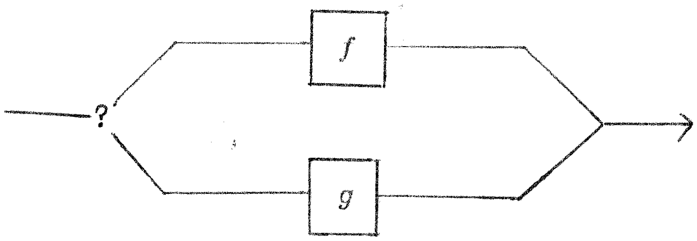


A second identity is:

$$\mathbf{concur} \circ (f \otimes g) \circ \mathbf{choice} = f + g.$$

$$\begin{tikzcd}
A \arrow[r, "\textbf{choice}"] \arrow[d, "f+g"'] & A \otimes A \arrow[d, "f \otimes g"] \\
B & B \otimes B \arrow[l, "\textbf{concur}"]
\end{tikzcd}$$

In this case, the flowchart is:



The algebraic structure obtained in each of these two cases has been previously studied in algebraic topology, group representation theory and the theory of linear operators. We given the necessary definitions to relate **fork** and **choice** to existing mathematics in the next section.

6.2. Cogebras and Bigebras

An alternate definition of a k-algebra can be given in terms of diagrams. Let B be a k-algebra. The semiring multiplication in B distributes over addition, hence it is a bilinear map from $B \times B$ to B. By the universal properties of tensor products (section 4.1) there is a unique linear transformation $\psi : B \otimes B \to B$ with $\psi(x \otimes y) = x \cdot y$. (In fact, this is the **concur** map we used in the last section.) The unit 1 in B also defines a linear transformation $\mu : k \to B$, with $\mu(r) = r1$ for all $r \in k$. (Here, k is considered as a k-module, as in the examples of section 2.3.)

In these terms, the definition of a k-algebra is a k-module B together with two linear transformations $\psi:B\otimes B\rightarrow B$ and $\mu:k\rightarrow B$, such that the following two diagrams commute:

$$\begin{tikzcd}
B\otimes B\otimes B \arrow[r, "I\otimes\psi"] \arrow[d, "\psi\otimes I"'] & B\otimes B \arrow[d, "\psi"] \\
B\otimes B \arrow[r, "\psi"'] & B
\end{tikzcd}$$

$$\begin{tikzcd}
k\otimes B \arrow[r, phantom, "\cong"] \arrow[d, "\mu\otimes I"'] & B \arrow[r, phantom, "\cong"] & B\otimes k \arrow[d, "I\otimes\mu"] \\
B\otimes B \arrow[ru, "\psi"'] & & B\otimes B \arrow[lu, "\psi"]
\end{tikzcd}$$

The map $I:B\rightarrow B$ is the identity. The isomorphism $\cong:k\otimes B\rightarrow B$ maps each $(r\otimes x)$ to rx and similarly for $\cong:B\otimes k\rightarrow B$. The top diagram asserts that multiplication is associative. The bottom diagram asserts that $1\in B$ is the identity for multiplication. The proof that this defines a k-algebra is similar to the case for ring algebras [25, XI.12]. We call ψ the ***multiplication map*** and μ the ***unit map***. This k-algebra is denoted by $<B,\psi,\mu>$.

The advantage of this definition is that its dual is easily defined by reversing the arrows. Specifically, a ***k-cogebra*** [4] is a k-module C together with two linear transformations $\rho:C\rightarrow C\otimes C$ and $\eta:C\rightarrow k$, such that the following diagrams commute:

$$\begin{tikzcd}
C\otimes C\otimes C & C\otimes C \arrow[l, "I\otimes\rho"'] \\
C\otimes C \arrow[u, "\rho\otimes I"] & C \arrow[l, "\rho"] \arrow[u, "\rho"']
\end{tikzcd}$$

$$\begin{tikzcd}
\mathrm{k}\otimes \mathrm{C} \arrow[r, phantom, "\cong"] & \mathrm{C} \arrow[dl, "\rho"] \arrow[dr, "\rho"'] \arrow[r, phantom, "\cong"] & \mathrm{C}\otimes \mathrm{k} \\
\mathrm{C}\otimes \mathrm{C} \arrow[u, "\eta\otimes \mathrm{I}"] & & \mathrm{C}\otimes \mathrm{C} \arrow[u, "\mathrm{I}\otimes\eta"']
\end{tikzcd}$$

We denote this k-cogebra by $<C,\rho,\eta>$. The linear transformation ρ is the *comultiplication map* and η is the *counit map*.

The final structure defined here is simultaneously a k-algebra and a k-cogebra. Let B be a k-algebra and suppose $<B,\rho,\eta>$ a k-cogebra. Then B is a ***bigebra*** [4] with comultiplication ρ and counit η if these conditions hold:

(1) $\rho:B\to B\otimes B$ is a k-algebra morphism,

(2) $\eta:B\to k$ is a k-algebra morphism.

The k-algebras k in (2) and $B\otimes B$ in (1) are defined in section 5.3 and 5.5.

Now we return to the k-algebra $A = k^{(\mathbb{M}(M))}$ to demonstrate two bigebra structures. Let $0_k:A\to k$ map each $m \in M$ to 0 and $1_k:A\to k$ map each $m \in M$ to 1. As shown in section 3, these maps are "conditions" in the sense of Dijkstra. In particular, 1_k is the always **false** condition, and 0_k is the condition that accepts exactly those nondeterministic distributions which do not include 1 ($\perp$) as a possible outcome. (Note that in general $wp(t, 0_k)$ is not computable.) These two conditions are counit maps for cogebras over the module A:

Theorem 6.1. $(A, \mathbf{choice}, 0_k)$ *is a k-cogebra.*

Proof: First, the diagram

$$\begin{tikzcd}
A\otimes A\otimes A & A\otimes A \arrow[l, "I\otimes \mathbf{choice}"'] \\
A\otimes A \arrow[u, "\mathbf{choice}\otimes I"] & A \arrow[l, "\mathbf{choice}"] \arrow[u, "\mathbf{choice}"']
\end{tikzcd}$$

must commute. Both paths take an element $m \in M \subseteq A$ to

$$(m\otimes 1\otimes 1)+(1\otimes m\otimes 1)+(1\otimes 1\otimes m)$$

But since A is free over M, this means that the diagram commutes everywhere. The second diagram that must commute is:

$$\begin{tikzcd}
\mathrm{k}\otimes \mathrm{A} \arrow[r, phantom, "\cong"] & \mathrm{A} \arrow[dl, "\mathbf{choice}"] \arrow[dr, "\mathbf{choice}"'] \arrow[r, phantom, "\cong"] & \mathrm{A}\otimes \mathrm{k} \\
A\otimes A \arrow[u, "0_k\otimes I"] & & A\otimes A \arrow[u, "I\otimes 0_k"']
\end{tikzcd}$$

We show that the left side commutes; the right side follows by a parallel argument. The isomorphism $\cong$ takes each $x \in A$ to $1 \otimes x$. For any element $m \in M$:

$$\begin{aligned}(0_k \otimes I) \circ \mathbf{choice}(m) &= (0_k \otimes I)[(m \otimes 1) + (1 \otimes m)] \\ &= (0_k \otimes I)(m \otimes 1) + (0_k \otimes I)(1 \otimes m) \\ &= (0 \otimes 1) + (1 \otimes m) \\ &= 0 + (1 \otimes m) \\ &= (1 \otimes m).\end{aligned}$$

Thus, the diagram commutes for elements of M, hence it does for all of A. □

Theorem 6.2. *$(A, \mathbf{fork}, 1_k)$ is a k-cogebra.*

Proof: The proof is identical to that of the previous theorem, replacing **choice** with **fork**, 0_k with 1_k and $+$ with $\cdot$. □

Furthermore, **fork**, **choice**, 0_k, and 1_k are all k-algebra morphisms. This gives the following corollaries:

Corollary 6.3. *The k-algebra A is a bigebra with comultiplication* **choice** *and counit 0_k.* □

Corollary 6.4. *The k-algebra A is a bigebra with comultiplication* **fork** *and counit 1_k.* □

7. SYNTAX

In the study of programming languages, the division between the syntax of a programming language and the interpretation of that syntax is basic. The *syntax* is a set of uninterpreted functions or program segments. Some syntactic operations, such as program composition, may be defined on this set. For a given syntax, an *interpretation* associates a function with each uninterpreted program segment in such a way that the syntactic operations are preserved. This division, originally proposed by Ianov [16] is now standard in the study of sequential programs. This section introduces and studies such a model of syntax and interpretations suitable for programming constructs having the functional behavior described earlier. Additionally, the model provides an abstract syntax for semantic models such as Pratt's processes [37], Kahn-MacQueen dataflow nets [17,26], Milne and Milner's flow algebras [33] and others. The main result is that, in general, a universal or free interpretation does not exist.

7.1. Monoidal Theories

The model of syntax we introduce is a generalization of algebraic theories [11,41], which have been used as abstract syntax for deterministic functional languages. Essentially, an algebraic theory provides a set of *uninterpreted functions*, each with a fixed number of arguments and outputs. An uninterpreted function f with n arguments and p outputs is denoted $f:[n]\to[p]$. An *interpretation* α assigns to each uninterpreted function $f:[n]\to[p]$ a function $\alpha_f:A^n\to A^p$, where A is a set (called the *semantic domain)* and A^n is the product set consisting of all n-tuples of elements from A.

Another related model of syntax is co-(algebraic theories), which have been used by Elgot and others as abstract syntax with an emphasis on flow of control [8,9,23]. A co-(algebraic theory) consists of a set of *uninterpreted flowcharts*, each with a fixed number of entry lines and exit lines. An interpretation α assigns to each uninterpreted flowchart $f:[n]\to[p]$ a function $\alpha_f:n\times A\to p\times A$, where $n=\{1,2,\cdots,n\}$ and $p=\{1,2,\cdots,p\}$ are sets of control line numbers and A is the semantic domain.

The model of syntax we introduce is called a *monoidal theory*. A monoidal theory is a set of uninterpreted *nondeterministic program segments*, each with a fixed number of inputs and outputs. An interpretation takes an uninterpreted program segment $f:[n]\to[p]$ in the theory to a linear transformation $\alpha_f:\bigotimes^n A\to\bigotimes^p A$, where A is some semiring module. The syntactic operations available in a monoidal theory correspond to composition and tensor product of linear transformations. Formally, the definition is as follows:

Definition. A *monoidal theory* $\mathbf{M}$ is a family of sets $<\mathbf{M}(n,p)>$, where n and p range over non-negative integers and the following operations are defined:

(1) **Composition:** Whenever $f \in \mathbf{M}(n,p)$ and $g \in \mathbf{M}(p,q)$ then $g \circ f \in \mathbf{M}(n,q)$. The operation $\circ$ is associative. For any non-negative integer n, the set $\mathbf{M}(n,n)$ has a special element 1_n which is a two-sided identity for $\circ$.

(2) **Multiplication:** Whenever $f \in \mathbf{M}(n,p)$ and $g \in \mathbf{M}(l,q)$ then $f \otimes g \in \mathbf{M}(n+l,p+q)$. The operation $\otimes$ is associative and 1_0 is the identity for $\otimes$. Furthermore, for any non-negative integers n and p, $1_n \otimes 1_p = 1_{n+p}$.

An additional condition is that multiplication preserves composition. That is:

(3) **Preservation of composition:** Whenever f,g,h and j are in $\mathbf{M}$ with $f \circ g$ and $h \circ j$ defined, then:

$$(f \circ g) \otimes (h \circ j) = (f \otimes h) \circ (g \otimes j).$$

For $f \in \mathbf{M}(n,p)$, we write $f:[n] \rightarrow [p]$. Such an element is an uninterpreted program segment with an n-tuple of inputs and a p-tuple of outputs. The syntactic operation of composition represents the composition of linear transformations, while multiplication of two elements $f \otimes g$ represents taking their tensor product.

If $\mathbf{M}$ and $\mathbf{L}$ are monoidal theories, then a *theory morphism* $\alpha : \mathbf{M} \rightarrow \mathbf{L}$ assigns to each $f:[n] \rightarrow [p]$ in $\mathbf{M}$ an element $\alpha_f:[n] \rightarrow [p]$ in $\mathbf{L}$, such that α preserves $\circ$, $\otimes$ and the identities. That is:

$$\alpha_{f \circ g} = \alpha_f \circ \alpha_g$$

$$\alpha_{f \otimes g} = \alpha_f \otimes \alpha_g$$

$$\alpha_{1_n} = 1_n$$

Let Ω be a family of sets $<\Omega(n,p)>$, where n and p range over non-negative integers. Think of Ω as a set of uninterpreted program segments, from which we will generate a monoidal theory. A function σ which assigns to each $f \in \Omega(n,p)$ an element

$\sigma_f:[n]\to[p]$ in a monoidal theory **M** is called an *insertion* of Ω into **M**. In this case we write $\sigma:\Omega\to\mathbf{M}$. By a *free* monoidal theory over Ω, we mean a monoidal theory **F** together with an insertion $\eta:\Omega\to\mathbf{F}$ such that if **M** is any monoidal theory and $\sigma:\Omega\to\mathbf{M}$ is an insertion, then there is a unique theory morphism $\alpha:\mathbf{F}\to\mathbf{M}$ such that the triangle

$$\begin{tikzcd}
\Omega \arrow[r, "\eta"] \arrow[dr, "\sigma"'] & \mathbf{F} \arrow[d, dashed, "\alpha"] \\
 & \mathbf{M}
\end{tikzcd}$$

commutes.

A free monoidal theory over Ω is analogous to a free algebraic theory [11]. It is the least constrained monoidal theory which is generated by Ω. Intuitively, it gives a free syntax for the set of uninterpreted program segments Ω.

Theorem 7.1. *Let Ω be a family of sets $<\Omega(n,p)>$, where n and p range over non-negative integers. A free monoidal theory over Ω exists.*

Proof outline. The proof is by construction. From Ω, we recursively construct a family of sets $<\Sigma(n,p)>$ of derived operators. An equivalence relation is defined on each set $\Sigma(n,p)$, so that the resulting equivalence classes form a monoidal theory which is free over Ω. The construction parallels the construction of a free algebraic theory [11] -- complete details are given elsewhere [27, section 5.1]. □

7.2. Interpretations

Let k be any positive commutative semiring. Any k-module A determines a monoidal theory denoted $MON{<}A{>}$. The elements of $MON{<}A{>}$ are the linear transformations from $\otimes^n A$ to $\otimes^p A$. For example, any linear transformation $f : A\otimes A \rightarrow A\otimes A$ is an element of $MON\langle A\rangle(2,2)$, and so on. Composition of two elements of $MON\langle A\rangle$ is just function composition; the multiplication of two elements is their tensor product.

An interpretation is a way of assigning linear transformations to the uninterpreted program segments of a monoidal theory. Intuitively, it gives a meaning to each uninterpreted element in **M**:

Definition. Let **M** be a monoidal theory. An **M**-*interpretation* is a k-module A together with a theory morphism $\alpha : \mathbf{M} \rightarrow MON\langle A\rangle$. The k-module A is the *semantic domain* and α is the *interpretation function*. Given that (A,α) and (B,β) are **M**-interpretations, an **M**-*homomorphism* from (A,α) to (B,β) is a linear transformation $h : A \rightarrow B$ such that for all $f : [n] \rightarrow [p]$ in **M**, the following square commutes:

$$\begin{tikzcd}
\otimes^n A \arrow[r, "\alpha_f"] \arrow[d, "\otimes^n h"'] & \otimes^p A \arrow[d, "\otimes^p h"] \\
\otimes^n B \arrow[r, "\beta_f"'] & \otimes^p B
\end{tikzcd}$$

The morphisms $\otimes^n h$ and $\otimes^p h$ are iterated tensor products defined as $\otimes^n h = h \otimes \cdots \otimes h$ (n factors).

One important class of interpretations are the *free* interpretations. Informally, a free interpretation of a theory gives a way of carrying out symbolic execution. This is because any other interpretation is a homomorphic image of the free interpretation, in the same way that a symbolic interpretation of a program can be mapped onto any other interpretation. As a result, free interpretations provide a way of reasoning about a whole class of interpretations at once. Initial interpretations, used by the ADJ group [12] are a special case of free interpretations.

Specifically, if A is a k-module, then by a free $\mathbf{M}$-interpretation over A we mean an $\mathbf{M}$-interpretation (F,γ) together with a linear transformation $f:A \to F$ such that if (B,β) is any $\mathbf{M}$-interpretation and $g:A \to B$ is a linear transformation, then there is a unique $\mathbf{M}$-homomorphism $h:(F,\gamma) \to (B,\beta)$ with $h \circ f = g$, as in the following commuting triangle:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g"'] & F \arrow[d, dashed, "h"] \\
& B
\end{tikzcd}$$

The linear transformation $f:A \to F$ is called the *insertion* of A into F.

Given this background, an important question is: Do free interpretations always exist? The remainder of this section shows that in general they do not. The first step toward showing this is to define a function on the elements of any tensor product. If A is any k-module and x is in $A \otimes A$, then we define $\| x \|_A$ to be the smallest integer n such that there exist sequences $y_1, \ldots, y_n$ and $z_1, \ldots, z_n$ of elements from A with:

$$x = \sum_{i=1}^{n} (y_i \otimes z_i).$$

The value of $\| x \|_A$ is always finite since $A \otimes A$ is spanned by the elements $(y \otimes z)$ with $y,z \in A$. However, for some k-modules, the value of $\| x \|_A$ may be arbitrarily large, as shown in the following:

Theorem 7.2. *Let $N = k^{(\mathbf{N})}$ be the free k-module generated by the set of natural numbers* $\mathbf{N} = \{0, 1, 2, \ldots\}$. *For any positive integer n, there exists an element $x \in N \otimes N$ with $[\![x]\!]_N = n$.*

Proof. Recall from theorem 4.1 that $N \otimes N$ is the free k-module generated by $\mathbf{N} \times \mathbf{N}$. Hence, elements of $N \otimes N$ are formal sums $\sum_{d \in \mathbf{N} \times \mathbf{N}}{}' r_d d$. For any $i \in \mathbf{N}$, let $\varepsilon_{(i,i)}$ be the formal sum in $N \otimes N$ which has 1 as the coefficient at (i,i) and zero coefficients elsewhere. It is easy to show that the element $x = \sum_{i=1}^{n} \varepsilon_{(i,i)}$ has $[\![x]\!]_N = n$. □

A second preliminary result is that a linear transformation of the form $h \otimes h$ cannot increase the value of the function $[\![\bullet]\!]$ at any point:

Theorem 7.3. *Let $h : A \to B$ be a linear transformation and x be an element of $A \otimes A$. Then $[\![h \otimes h(x)]\!]_B \leq [\![x]\!]_A$.*

Proof. Suppose $[\![x]\!]_A = n$. Then there are sequences $y_1, \ldots, y_n$ and $z_1, \ldots, z_n$ of elements of A with

$$x = \sum_{i=1}^{n} (y_i \otimes z_i).$$

Therefore:

$$h \otimes h(x) = h \otimes h\left(\sum_{i=1}^{n} (y_i \otimes z_i)\right)$$
$$= \sum_{i=1}^{n} (h(y_i) \otimes h(z_i)),$$

and $[\![h \otimes h(x)]\!]_B \leq n$. □

Now we can show that, in general, monoidal theories do not have free interpretations. In particular, if M is a free monoidal theory over a set Ω and $f : [n] \to [p] \in \Omega$ with $n > 1$, then M does not have free interpretations. This is formalized in the following theorem:

Theorem 7.4. *Let A be a nonempty free k-*module and suppose $\mathbf{M}$ *is a free monoidal theory over a family of sets $<\Omega(n,p)>$. If $\Omega(n,p)$ is nonempty for at least one value of $n>0$ and $p>1$, then there is no free $\mathbf{M}$-interpretation over A.*

Proof. For concreteness, we assume that $\Omega(1,2)$ is nonempty and $d:[1]\rightarrow[2]$ is the insertion of some element from $\Omega(1,2)$ into $\mathbf{M}$. It is straightforward to verify that the theorem remains valid for other values of n and p. In order to derive a contradiction, assume (F,γ) is a free $\mathbf{M}$-interpretation over A with insertion $f:A\rightarrow F$. Let y be a basis element of A and suppose $i = \llbracket \gamma_d(f(y)) \rrbracket_F$. Also, let N be the free k-module $k^{(\mathbf{N})}$, generated by the natural numbers $\mathbf{N}=\{0,1,\ldots\}$, and let z be a basis element of N. We will construct an $\mathbf{M}$-interpretation (N,β) which will lead to the contradiction.

By theorem 7.2, $N\otimes N$ contains some element x with $\llbracket x \rrbracket_N > i$. Now, let (N,β) be any $\mathbf{M}$-interpretation with $\beta_d(z) = x$. Such an interpretation exists since z is a basis element of N. Also, since y is a basis element of A, there is a linear transformation $g:A\rightarrow N$ with $g(y)=z$. For this linear transformation g, there must be a unique $\mathbf{M}$-homomorphism $h:(F,\gamma)\rightarrow(N,\beta)$ with $h\circ f = g$. Now we can derive the contradiction:

$$\begin{aligned}
i &= \llbracket \gamma_d(f(y)) \rrbracket_F && \text{(definition } i\text{)}\\
&\geq \llbracket h\otimes h(\gamma_d(f(y))) \rrbracket_N && \text{(theorem 7.3)}\\
&= \llbracket \beta_d(h(f(y))) \rrbracket_N && (h \text{ is } \mathbf{M}\text{-homomorphism})\\
&= \llbracket \beta_d(g(y)) \rrbracket_N && (h\circ f = g)\\
&= \llbracket \beta_d(z) \rrbracket_N &&\\
&= \llbracket x \rrbracket_N &&\\
&> i. &&
\end{aligned}$$

By this contradiction, we conclude there is no free $\mathbf{M}$-interpretation over A. □

Intuitively, this theorem means that free interpretations are prevented by uninterpreted functions whose output is not confined to a single output line.

7.3. Monoidal theories for communicating processes

Monoidal theories provide an algebraic characterization of processes which have a finite number of inputs and outputs. The two operations which are abstracted are sequential composition and parallel composition. This section shows how models of communicating systems which use these two operations are monoidal theories. Our first example is a dataflow model, as used by Kahn and MacQueen [17,26]. The processes of Pratt [37] and the flow algebras of Milner and Milne [30,31,32,33] provide two other examples. We finish with a comparison of monoidal theories to Winkowski's algebraic characterization based on Petri nets [43,44].

A basic concept in the dataflow model of computation is a *stream* of data. For a given set A, an *A-stream* is any countable sequence of elements from A. The set of all A-streams, denoted $A^{\dagger}$, has a useful partial order defined on it. If x and y are A-streams, then $x \leq y$ iff x is a prefix of y. A process with one input and one output is a function $f : A^{\dagger} \rightarrow A^{\dagger}$. Because of the sequential nature of the input stream, we require the axiom:

$$x \leq y \text{ implies } f(x) \leq f(y), \text{ for all } x, y \in A^{\dagger}.$$

Intuitively, this axiom means that when a process is given additional input, it can only produce additional output and not retract anything it has already sent to its output stream. Any function which meets this axiom is called *continuous*. MacQueen [26] describes how continuous functions can be specified.

The set of all n-tuples of A-streams, denoted $(A^{\dagger})^n$, is a partial order with componentwise ordering. That is,

$$(a_1, \ldots, a_n) \leq (b_1, \ldots, b_n) \text{ iff } a_i \leq b_i \text{ for all } 1 \leq i \leq n.$$

A process with n-inputs and p-outputs is a continuous function $f:(A^\dagger)^n \to (A^\dagger)^p$.

Now, let $A(n,p)$ be the set of all continuous functions from $(A^\dagger)^n$ to $(A^\dagger)^p$. It is easy to see how these sets form a monoidal theory. Composition is merely function composition. If $f:(A^\dagger)^n \to (A^\dagger)^p$ and $g:(A^\dagger)^l \to (A^\dagger)^q$, then we define $f \otimes g:(A^\dagger)^{n+l} \to (A^\dagger)^{p+q}$ to be the product function which takes each tuple $(s_1, \ldots, s_n, s_{n+1}, \ldots, s_{n+l})$ to $f(s_1, \ldots, s_n) \times g(s_{n+1}, \ldots, s_{n+l})$ in $(A^\dagger)^{p+q}$. (Note: the composition of continuous functions is continuous, and similarly for their product.) The identity in $A(n,n)$ is simply the identity function.

This monoidal theory is in fact an algebraic theory, essentially because there is no nondeterminism. Several attempts have been made to extend the basic dataflow model to handle nondeterminism, including Kosinski [18,19], Brock-Ackerman [5] and Pratt [37]. In the following, we show how processes of finite sort in Pratt's model form a monoidal theory.

Pratt's model of communicating processes begins with the set-theoretic notion of the graph of a function. If $f:A \to A$ is a function, then its graph is the set of pairs $\{(x,y) \mid f(x)=y\}$. An ordered pair (x,y) in this set is viewed as a pair of events, linearly ordered in time. Specifically, the first event is the arrival of an input x, followed by the production of the output $y=f(x)$. Such ordered pairs of events are enough to completely describe functions -- even nondeterministic functions, or relations. To describe communicating processes, the notions of events and ordered pairs of events are generalized.

We begin with a set A of data elements, a denumerable set $I=\{I_1,I_2,\ldots\}$ of "input ports" and a denumerable set $O=\{O_1,O_2,\ldots\}$ of "output ports". An *input event* is an ordered pair from $A \times I$ and represents the arrival of a datum $x \in A$ at a port $i \in I$. Similarly, an *output event* is a pair from $A \times O$. For a function f, the events are all of the form (x, I_1) and $(f(x), O_1)$. Furthermore, the event (x, I_1) strictly precedes the event $(f(x), O_1)$.

The generalization of this is as follows: a *trace* is any partially ordered multiset of events. A *process* is any set of traces. The intended meaning of a process is that if the input events of one of its traces occur in the given order, then the output events of that trace occur. If events of more than one trace may occur, then the output is selected nondeterministically.

A trace is of sort (n,p) if all of its input events have ports selected from $\{I_1, \ldots, I_n\}$ and all of its output events have ports from $\{O_1, \ldots, O_p\}$. A process is of sort (n,p) provided all of its traces are of sort (n,p).

Now we can show how processes of finite sort form a monoidal theory. Let A be a set of data elements and define $A(n,p)$ to be all the processes of sort (n,p) over this data set. To define multiplication of processes, we first define two operations on traces:

(1) **Union:** If τ and σ are traces, then their union, written $\tau \cup \sigma$ is the multiset union of τ and σ. The partial order on $\tau \cup \sigma$ is inherited from τ and σ independently -- that is, for two events e_1 and e_2 in $\tau \cup \sigma$, $e_1 \leq e_2$ iff

$$e_1, e_2 \in \tau \text{ and } e_1 \leq e_2 \text{ in } \tau, \text{ or,}$$
$$e_1, e_2 \in \sigma \text{ and } e_1 \leq e_2 \text{ in } \sigma.$$

Note that when τ is of sort (n,p) and σ is of sort (l,q), then $\tau \cup \sigma$ is of sort $(\mathbf{max}(m,l), \mathbf{max}(n,q))$.

(2) **Shifts:** If σ is a trace, then $\sigma_{n,p}$ is the trace obtained from σ by adding n to the subscript of each input port and adding p to the subscript of each output port. The partial order on $\sigma_{n,p}$ is inherited from σ. Thus, if $\sigma = \{(x, I_1), (y, O_1)\}$ with $(x, I_1) \geq (y, O_1)$, then $\sigma_{n,p} = \{(x, I_{1+n}), (y, O_{1+p})\}$, with $(x, I_{1+n}) \geq (y, O_{1+p})$. Note that if σ is of sort (l,q), then $\sigma_{n,p}$ is of sort $(l+n, q+p)$.

If $f \in A(n,p)$ and $g \in A(l,q)$ are processes, then their multiplication is defined as

$$f \otimes g = \{\tau \cup (\sigma_{n,p}) \mid \tau \in f \text{ and } \sigma \in g\}.$$

It is easy to see that $f \otimes g$ is of sort $(n+l,p+q)$ and the operation is associative. Its identity is the process $\{\phi\}$ of sort (0,0), which contains only the empty trace. The meaning of $f \otimes g$ is that the two processes are executed side-by-side, with the ports of g appropriately shifted.

The composition of two processes is a generalization of the composition of relations. If $r, r' \subseteq A \times A$ are relations, then their composition is the relation

$$r \circ r' = \{(x,z) \mid \exists y \text{ such that } (x,y) \in r \text{ and } (y,z) \in r'\}.$$

To generalize this to processes, we need a new set $C = \{C_1, C_2, \ldots\}$, of *communication ports*. A *communication event* is a pair from $A \times C$, and will play the role that y fulfills in the definition of the composition of relations. A *communication trace* is a partially ordered multiset of input events, communication events and output events. It is analogous to an ordered triple (x,y,z). If τ is a communication trace, it can be restricted to an ordinary trace in three ways:

τ_{IO} is the trace obtained by eliminating all communication events from τ. It is analogous to obtaining the pair (x,z) from the triple (x,y,z).

τ_{IC} is the trace obtained by eliminating all output events from τ and changing each communication event (x,C_i) to the output event (x,O_i). It is analogous to restricting the triple (x,y,z) to the pair (x,y).

τ_{CO} is the trace obtained by eliminating all input events from τ and changing each communication event (x,C_i) to the input event (x,I_i). It is analogous to restricting the triple (x,y,z) to the pair (y,z).

Using these restrictions, we define the composition of two processes f and g as

$$f \circ g = \{\tau_{IO} \mid \tau \text{ is a communciation trace and } \tau_{IC} \in f \text{ and } \tau_{CO} \in g\}.$$

For any non-negative integer n, the identity process $1_n \in A(n,n)$ is

$1_n = \{\tau \mid \tau$ is a trace of sort (n,n) and the partially ordered submultiset of output events of τ is obtained from its submultiset of input events by replacing each input port I_i with $O_i\}$.

In particular, $1_0 = \{\phi\}$ is the identity for $\otimes$. It is straightforward to check that composition is associative and distributes over $\otimes$.

Thus, the processes of finite sort form a monoidal theory. In a similar way, certain of the flow algebras of Milner and Milne [30,31,32,33] form a monoidal theory, although it is more complicated than Pratt's model because of the names of ports.

Winkowski [43,44] also constructs an algebra for communicating systems, based on Petri nets. His algebras are not generally monoidal theories -- the principal reason is that multiplication is only partially defined. However, his two basic operations are the same as ours: parallel composition $(f \otimes g)$ and sequential composition $(f \circ g)$. Winkowski mentions that these two operations have nearly the same properties as the operations in a strict monoidal category [24]. A monoidal theory is a strict monoidal category, and a theory morphism is a monoidal functor.

In summary, we emphasize that wherever the operations of parallel and sequential composition have appeared, the monoidal theory axioms are met (or nearly so). For this reason, we believe monoidal theories are important to the study of communicating processes.

ACKNOWLEDGEMENTS The authors wish to acknowledge support of this research by NSF grant MCS-8003433. Additionally, M. Main has been partially supported by a grant from the University of Colorado Council on Research and Creative Work.

References

(1) D.B. Benson. Counting paths: nondeterminism as linear algebra, Washington State University Technical Report CS-82-084, Pullman, WA 99164 (1982), to appear in *IEEE Trans. Software Engineering*.

(2) D.B. Benson. Studies in fork-join parallelism, Washington State University Technical Report CS-82-101, Pullman, WA 99164 (1982).

(3) D.B. Benson. In Scott-Strachey style semantics, parallelism implies nondeterminism, *Math. Systems Theory* 15 (1982), 267-275.

(4) N. Bourbaki. *Algebra I* (translated from the French), (Addison-Wesley, 1974).

(5) J.D. Brock and W.B. Ackerman. Scenerios: A model of non-determinate computation, in: *Formalization of Programming Concepts*, LNCS 107, (Springer-Verlag, 1981), 252-259.

(6) E.W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs, *Comm. ACM* 18 (1975), 453-457.

(7) E.W. Dijkstra. *A Discipline of Programming*, (Prentice-Hall, 1976).

(8) C.C. Elgot. Algebraic theories and program schemes, in *Symp. on Semantics of Algorithmic Languages* (E. Engler, Ed.), Lecture Notes in Mathematics 188, (Springer-Verlag, 1971), 71-88.

(9) C.C. Elgot. Monadic computations and iterative algebraic theories, in: *Logic Colloquium 73* (H.E. Rose and J.C. Sheperdson, Eds.), (North-Holland Publishing Co., 1975), 175-230.

(10) A.A. Faustini. An operational semantics for pure dataflow, University of Warwick Theoretical Computing Report 38 (1981).

(11) J.A. Goguen, J.W. Thatcher, E.G. Wagner and J.B. Wright. An introduction to categories, algebraic theories and algebras, IBM Research Report RC 5369, Yorktown Heights, NY 10598 (1975).

(12) J.A. Goguen, J.W. Thatcher, E.G. Wagner and J.B. Wright. Initial algebra semantics and continuous algebras, *JACM* 24 (1977), 68-95.

(13) P.A. Grillet. The tensor product of semi-groups, *Trans. Amer. Math. Soc.* 138 (1969), 267-280.

(14) P.A. Grillet. The tensor product of commutative semi-groups, *Trans. Amer. Math. Soc.* 138 (1969), 281-293.

(15) M.C.B. Hennessy and G.D. Plotkin. Full abstraction for a simple parallel programming language, in: *Mathematical Foundations of Computer Science 79*, LNCS 74, (Springer-Verlag, 1979), 108-120.

(16) I. Ianov. The logical schemes of algorithms, in: *Problems of Cybernetics*, (Pergamon Press, 1960).

(17) G. Kahn. The semantics of a simple language for parallel programming, in *Proceedings of the IFIP Congress 74* (J.L Rosenfeld, Ed.), (North-Holland Publishing Co., 1974), 471-475.

(18) P.R. Kosinski Mathematical semantics and data flow programming, in: *Third Annual ACM Symposium on Principles of Programming Languages*, (1976), 175-184.

(19) P.R. Kosinski. A straightforward denotational semantics for nondeterminate dataflow programs, in: *Fifth Annual ACM Symposium on Principles of Programming Languages*, (1978), 214-219.

(20) D. Kozen. Semantics of probabilistic programs, *J. Comput. Sys. Sci.* 22 (1981), 328-350.

(21) R.P. Kurshan and B. Gopinath. The selection/resolution model for concurrent processes, preprint, Bell Laboratories, Murray Hill, NJ 07974 (undated).

(22) R.P. Kurshan and B. Gopinath. Modelling fully distributed (asynchronous) concurrent processes, preprint, Bell Laboratories, Murray Hill, NJ 07974 (April 1982).

(23) R.J. Lorentz and D.B. Benson. Deterministic and nondeterministic flowchart interpretations, Washington State University Technical Report CS-80-066, Pullman, WA 99164 (1980).

(24) S. MacLane. *Categories for the Working Mathematician*, (Springer-Verlag, 1971).

(25) S. MacLane and G. Birkhoff. *Algebra*, (MacMillan Publishing Co., 1979).

(26) D.B. MacQueen. Models for distributed computing, INRIA Technical Report 351, Paris (1979).

(27) M.G. Main. Interpretations of dataflow theories, Ph.D. Thesis, Washington State University, Pullman, WA 99164 (1982).

(28) M.G. Main and D.B. Benson. An algebra for nondeterministic distributed processes, Washington State University Technical Report CS-82-087, Pullman, WA 99164 (1982).

(29) E.G. Manes. A Class of Fuzzy Theories, *J. Math. Analysis and Applications* 85 (1982), 409-451.

(30) R. Milner. *A Calculus of Communicating Systems*, LNCS 92, (Springer-Verlag, 1980).

(31) R. Milner. Calculi for synchrony and asynchrony, University of Edinburgh Internal Report CSR-104-82, Edinburgh, Scotland (1982).

(32) R. Milner. Four combinators for concurrency, in: *Proceedings ACM Symposium on Principles of Distributed Computing*, (1982).

(33) G. Milne and R. Milner. Concurrent processes and their syntax, *JACM* 26 (1979), 302-321.

(34) G.D. Plotkin. A powerdomain construction, *SIAM J. Computing* 5 (1976), 452-487.

(35) A. Poigne. Using least fixed points to characterize formal computations of nondeterminate equations, in: *Formalizations of Programming Concepts* (J. Diaz and I. Ramos, Eds.), LNCS 107, (Springer-Verlag, 1981), 447-459.

(36) A. Poigne. On effective computations of nondeterministic schemes, in: *5th International Symposium on Programming*, LNCS 137, (Springer-Verlag, 1982), 323-336.

(37) V.R. Pratt. On the composition of processes, in: *Ninth Annual ACM Symposium on Principles of Programming Languages*, (1982).

(38) N. Saheb-Djahromi. CPO's of measures for nondeterminism, *Theoret. Comput. Sci,* 12 (1980), 19-37.

(39) M. Smyth. Powerdomains, *J. Comput. Sys. Sci.* 16 (1978),

(40) M. Steenstrup, M.A. Arbib and E.G. Manes. Port automata and the algebra of concurrent processes, COINS Technical Report 81-25, University of Massachusetts at Amherst (1981). TenG R.D. Tennent. The denotational semantics of programming languages, *Comm. ACM* 19 (1976), 437-453.

(41) E.G. Wagner, J.W. Thatcher and J.B. Wright. Programming languages as mathematical objects, in: *Mathematical Foundations of Computer Science 78*, LNCS 64, (Springer-Verlag, 1978), 84-101.

(42) D. Wall. Messages as active agents, in: *Ninth Annual ACM Symposium on Principles of Programming Languages*, (1982).

(43) J. Winkowski. An algebraic characterization of the behavior of nonsequential systems, *Info. Proc. Letters* 6 (1977), 105-109.

(44) J. Winkowski. Behaviors of concurrent systems, *Theoret. Comput. Sci.* 12 (1980), 39-60.