JOURNAL OF COMPUTER AND SYSTEM SCIENCES 27, 29–50 (1983)

Port Automata and the Algebra of Concurrent Processes*

MARTHA STEENSTRUP AND MICHAEL A. ARBIB

Computer and Information Science Department, University of Massachusetts, Amherst, Massachusetts 01003

AND

ERNEST G. MANES

Mathematics and Statistics Department, University of Massachusetts, Amherst, Massachusetts 01003

Received October 20, 1981; revised February 19, 1982

We model a *process*—the unit of concurrent processing—as a *port automaton*, a certain type of nondeterministic sequential machine. We show how to compute the response of these automata, and study the properties of their port interconnections. We are motivated by the work of Milne and Milner, but eschew their use of domains, continuous functions, and power-domains. Instead we can use sets, maps and subsets by applying our theory of *greatest fixpoints* to the study of the response of nondeterministic machines.

INTRODUCTION

Within the past 25 years, concurrent computation has moved from the realm of theoretical speculation to that of practical implementation. During this period of time, numerous theoretical models of concurrency have appeared in the literature: Dijkstra [5], Petri [13], Campbell and Habermann [6], Hewitt [7], Brinch–Hansen [3], Hoare [8], and Pnueli [15], to mention a few. The purposes for which the various models have been developed include the following: to facilitate the investigation of the behavior of concurrent processes, to aid in the designing of concurrent systems, to illustrate specific process synchronization problems, and to verify the correctness of parallel programs.

We have chosen the synchronized communication paradigm as the basis for our model of concurrency because it is not too difficult to provide a formal definition of the behavior of such communicating processes, and because most other forms of

* The research reported in this paper was supported in part by a grant from the National Science Foundation, No. MCS–8005112, with co-principal investigators M. A. Arbib and E. G. Manes. We thank the referee for helpful criticism.

0022-0000/83 $3.00

communication between processes may be captured in this particular model. (It is not clear whether asynchronous message-passing using queues, such as that utilized in actor systems (Hewitt [7]), can be successfully modelled using this framework. In this case, the difficulties arise because the order in which messages are sent is not necessarily the order in which they arrive.) Several of the extant concurrency models are based upon this paradigm, in particular communicating sequential processes (CSP) (Hoare [8]) and the processes of Milne and Milner [10]. We shall concern ourselves with the latter, in which a process is defined as a certain fixed point of a particular powerdomain equation. The powerdomain construction, utilized by several authors as a formal means for capturing the behavior of nondeterministic processes, is rather complicated and does not afford a particularly intuitive view of communicating sequential processes. We will discuss this issue in more detail in a later section. It was this aspect of the Milne–Milner model that motivated us to search for an alternative representation.

Milner [11] introduces an elegant calculus of synchronized communicating processes to express the behaviors of concurrent systems up to observation equivalence (see below), and to provide various proof techniques. Milner's approach to the semantics of concurrency is considerably more operational than the approach used in Milne and Milner [10]. In fact, he mentions [11, p. 162] that a problem with their process model was that the denotation did not properly match the operational meaning. We submit that our model retains the denotational flavor of the Milne–Milner model, yet is operational in nature owing to the fact that it is automata-theoretic; and that both the denotational and operational meanings coincide.

The Model

Our formal model of concurrent computation parallels that of Milne and Milner, and was originally designed to provide a simple theoretical framework in which to study concurrency. We recognize that a process can be represented as a particular type of nondeterministic sequential machine, which we call a port automaton. Before we present the formal definitions, however, we introduce our informal notion of a process, in order to provide the reader with an intuitive basis from which to examine the formal model.

We consider a process to be a formal machine which is capable of communicating with its environment. Each process possesses a set of ports through which communications take place. Communication consists of a simultaneous exchange of values through a port, that is, a process receives a value from the environment while at the same time the process sends a value to the environment; both the sending and the receiving occurring at the same port. (Actually, in all instances that we consider, the flow of communication is essentially undirectional, in that one of the exchanged values is always the trivial value, which we denote by "$\neq$".)

Each port of a process may be in one of two modes at any given instant: inac-

tivated—not capable of communication, or activated—capable of communication; an activated port may be engaged in communication. Although more than one port of a process may be activated at one time, at most one port may actually be communicating. (This restriction was imposed in order to keep the complexity of the notation to a minimum. These processes are computationally equivalent to those which allow more than one communication at a time.) The receipt of an input value during communication effects a state transition in the process.

To achieve concurrency, we construct networks of these processes by connecting together ports belonging to separate processes. A network of processes may be regarded both as the set of its component processes and as a single process itself. When viewed as the latter, the connected ports are no longer visible to the environment. Let us consider, for this case, the state transition resulting from an external communication. This state change may involve several state transitions resulting from internal communications. The manner in which these internal transitions are effected is not necessarily deterministic, since the order in which the internal communications occur is not necessarily predetermined. Thus, the state transition caused by an external communication in a network is inherently nondeterministic.

We now proceed to formalize our notion of a process by giving the definition of a port automaton.

DEFINITION. A *port automaton* P is a collection of objects and maps $(L, Q, \tau, \delta, \beta, X, Y)$, where

L is the set of ports,

Q is the set of states,

$\tau \in 2^Q$ is the set of initial states,

$X = (X_i : i \in L)$, where X_i is the input set for port i,

$Y = (Y_i : i \in L)$, where Y_i is the output set for port i,

$\delta : Q \times \bigsqcup_{i \in L} X_i \to 2^Q$ is the transition map, where $\bigsqcup_{i \in L} X_i = \{(x, i) : x \in X_i\}$ is the disjoint union of the X_i's,

$\beta = (\beta_i : i \in L)$, where $\beta_i : Q \to Y_i$ is the output map for port i,

all subject to the axiom that for each $q \in Q$

$$\{x \in X_i : \delta(q, (x, i)) \neq \varnothing\} = \varnothing \quad \text{or} \quad X_i.$$

This axiom is added for technical convenience. It ensures that in any state $q \in Q$, for any port i, either all elements of the input set X_i will be capable of being accepted or none of them will.

DEFINITION. We say that port i of P is *activated* in state q if for all $x \in X_i$, $\delta(q, (x, i)) \neq \varnothing$. Otherwise, we say port i is *inactivated.*

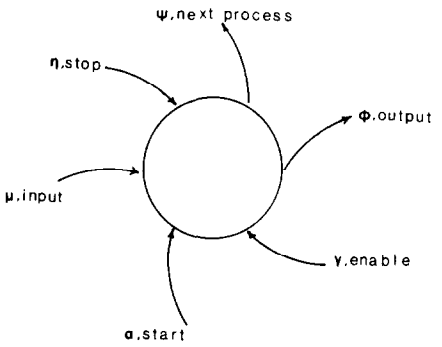

FIGURE 1

Now that the reader has been acquainted with both the intuitive and the formal descriptions of a process, we present a representative example of a process modelled as a port automaton. The example has been adapted from Milner [11]. We will study this example again in more detail when we discuss networks of processes; it is presented here in a somewhat simplified form in order to aid the reader in relating it to the formal definition. Figure 1 is a schematic depiction of the port automaton, the labels coupled with the descriptions indicating the port names and the nature of the communication passing in the given direction, respectively.

The port automaton in Fig. 1 functions as follows. Input arriving through γ enables the automaton, which then waits for a start signal through α. Once this is received, the automaton waits for an integer input through μ and begins processing. At this phase, the automaton may either send an enable signal through ψ or receive a stop processing signal through η. If ψ is sent first, then the automaton waits for the stop signal through η. When a stop signal is received, integer output is sent through ϕ. Then a signal to enable the next automaton is sent through ψ, provided it has not already been sent previously. Pictorially, we may represent this by using nondeterministic state graph notation, where each node corresponds to a particular state of the automaton, and each arc corresponds to the port (and input value) responsible for the state change (Fig. 2). (Note that to be precise, the arc labelled μ really should be represented by multiple arcs leading to multiple states, one for each input value.)

Our next goal is to deduce the reachability map and the response map; in reference to formal machines, the reachability map determines, given an initial state and a set of input values, the state of the machine after receiving these values, while the response map determines, given an initial state and a set of input values, the output of the machine after receiving these values. To accomplish this goal, we must first indicate the order in which inputs from various ports are received. Inputs to a port automaton can be thought of as a set of queues, each queue corresponding to the values to be input at a specific port. Using our automaton of Fig. 1, we might have the arrangement of input queues shown in Fig. 3.

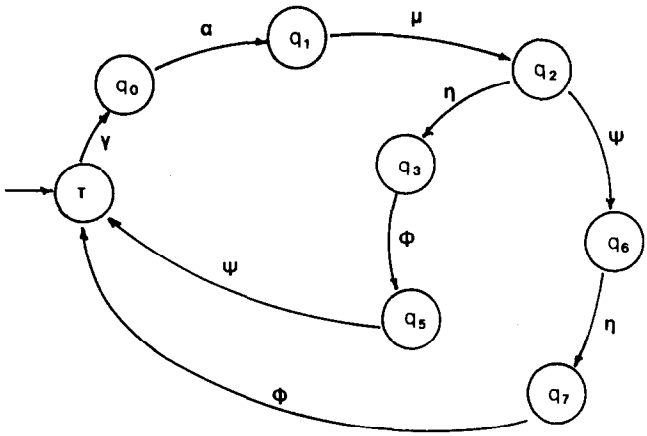

FIGURE 2

The order in which values are stripped from these queues is, in one sense, dependent upon the order in which the ports become activated. Following our concept of a process, a port automaton may receive an input at any activated port of the automaton, and if more than one port is activated at a given instant, then it is not possible to predetermine which port will actually engage in communication. Formally, inputs to a port automaton are of the form

$w = (w_1,..., w_n)$, where $L = \{1,..., n\}$ and for $1 \leqslant i \leqslant n$, $w_i \in X_i{}^*$. This implies that $w \in \prod_{i \in L} X_i{}^*$.

We refer to a specific ordering of the ports at which the input values arrive as an *interleaving*. The order in which inputs are stripped from the various $X_i{}^*$'s can also depend upon the interleaving specified. Note that the ordering thus specified may not coincide with the ordering of the port activations; this discrepancy may lead to

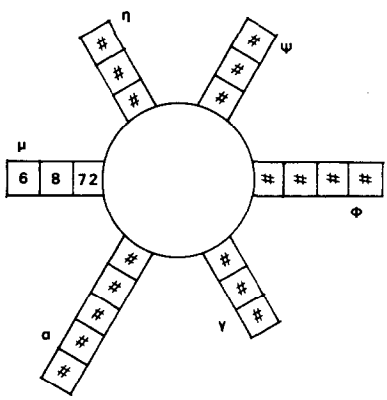

FIGURE 3

undefined states in the computation sequences. An interleaving s may be regarded as an element of L^*. If we pluck input values from the various queues and concatenate them according to the order specified by s, we obtain a string in $(\bigsqcup_{i\in L} X_i)^*$. Thus, each interleaving s induces a mapping from $\prod_{i\in L}(X_i^*)$ to $(\bigsqcup_{i\in L} X_i)^*$, formally described as follows.

For $s \in L^*$,

$$\phi_s : \prod_{i\in L} (X_i^*) \to \left(\bigsqcup_{i\in L} X_i\right)^*,$$

where

$$\phi_\Lambda(w) = \Lambda, \qquad \text{for all} \quad w \in \prod_{i\in L} (X_i^*),$$

$$\phi_{j\cdot t}(w) = \phi_t(w), \qquad \text{if} \quad w_j = \Lambda,$$

$$= x \cdot \phi_t(w_1, \ldots, w_{j-1}, v, w_{j+1}, \ldots, w_n), \qquad \text{if} \quad w_j = x \cdot v.$$

At last, we are ready to define the reachability map and the response map.

DEFINITION. Let $f: A \to 2^B$ be a function. The *extension by union* $f^\#$ of f is the map $f^\# : 2^A \to 2^B$ defined by $f^\#(S) = \bigcup \{f(s) : s \in S\}$.

DEFINITION. The *reachability map* for a port automaton P is $r = (r_s : s \in L^*)$, where $r_s : \prod_{i\in L}(X_i^*) \to 2^Q$ is the map $\tilde{r} \cdot \phi_s$, and where $\tilde{r} : (\bigsqcup_{i\in L} X_i)^* \to 2^Q$ is the conventional reachability map:

$$\tilde{r}(\Lambda) = \tau,$$

$$\tilde{r}(v \cdot (x, i)) = \delta^\#(\tilde{r}(v), (x, i)), \qquad \text{where}$$

$$i \in L, \qquad x \in X_i, \qquad v \in \left(\bigsqcup_{i\in L} X_i\right)^*,$$

Note. In the definition of the response map below, the quotient set $(\bigsqcup_{i\in L} 2^{Y_i})/\varnothing$ is equivalent to $(\bigsqcup_{i\in L}(2^{Y_i} - \{\varnothing\})) + \{\varnothing\}$. We have adopted this notation to make explicit for each power set 2^{Y_i}, the identification of its empty subset with a single empty set.

DEFINITION. The *response map* for a port automaton P is $f = (f_s : s \in L^*)$, where $f_s : \prod_{i\in L}(X_i^*) \to (\bigsqcup_{i\in L} 2^{Y_i})/\varnothing$ is the map $\tilde{f} \cdot \phi_s$, where $\tilde{f} : (\bigsqcup_{i\in L} X_i)^* \to (\bigsqcup_{i\in L} 2^{Y_i})/\varnothing$ is the map

$$\tilde{f}(\Lambda) = \varnothing,$$

$$\tilde{f}(v \cdot (x, i)) = \beta_i^\#(\tilde{r}(v)), \qquad \text{where} \quad i \in L, \quad x \in X_i, \quad v \in \left(\bigsqcup_{i\in L} X_i\right)^*.$$

Let us now see how Milner's notion of observation equivalence (Milner [11]) relates to our response function. Informally, Milner defines two behavior expressions (which we will refer to as processes) to be observation equivalent (denoted by $\approx$) if and only if they are indistinguishable by observation in any context. Observations are the results of performing s-experiments, where an s-experiment is a time-ordered string of actions, each action either a value submitted by or received by the observer. We write $P \Rightarrow^s P'$, if process P' is the result of performing an s-experiment on process P. Note that unobservable actions resulting from internal communications may occur between any two actions of the s-experiment. Under observation equivalence, a process P_1 containing a possibly infinite number of unobservable actions and a process P_2 containing no unobservable actions are observation equivalent iff the following criteria are met [11, p. 99]: $P_1 \approx_K P_2\ \forall k \geqslant 0$, where $P_1 \approx_0 P_2$ is always true, and $P_1 \approx_{K+1} P_2$ if and only if for all $s \in (\coprod_{i \in L} X_i \cup \coprod_{i \in L} Y_i)^*$ the following hold:

(i) if $P_1 \Rightarrow^s P_1'$, then for some P_2', $P_2 \Rightarrow^s P_2'$ and $P_1' \approx_K P_2'$;

(ii) if $P_2 \Rightarrow^s P_2'$, then for some P_1', $P_1 \Rightarrow^s P_1'$ and $P_1' \approx_K P_2'$.

Recall our definition of response function—a set of response mappings, one associated with each interleaving, yielding a set of possible output values given a particular set of input queues. In our model, we can define an equivalence relation on the set of processes, such that two processes are equivalent iff they have the same response function, that is, that the response maps coincide on each interleaving. However, this equivalence relation is not as strong as Milner's observation equivalence, since an s-experiment yields only one sequence of actions whereas a response map gives all possible outputs for a given interleaving. To adapt observation equivalence to our framework, we redefine an s-experiment as an interleaving interwoven with possible response values. That is, following each input value in the s-experiment is one of the associated output values. Thus, after n inputs in the s-experiment, the next immediate value is one of a set of possible responses for the given subinterleaving. We can then use Milner's notion of observation equivalence in our model.

Processes Do Not Need Powerdomains

Milne and Milner require a great deal of algebraic structure for their definition of process. We argue in this section that such structure is unnecessary, but first we recall their setting. (The exact definitions are not required for the present discussion; they may be found in Milne and Milner [10], pp. 304–305.) A *domain* D is an ω-algebraic complete partial order (cpo). The *cartesian product* of two domains is a domain. For any denumerable indexing set L and any family $(D_i : i \in L)$ of domains, the *indexed sum*

$$D_i = \{(i, d) : i \in L, d \in D_i\} \cup \{\perp_s\}$$

is also a domain, with $s \sqsubseteq s'$ iff $s = \perp_s$, or $s = (i, d)$, $s' = (i, d')$, and $d \sqsubseteq d'$. For any pair D, E of domains, the *function domain* $D \to E$ of continuous functions from D to E is a domain with the ordering

$$f \sqsubseteq f' \Leftrightarrow (\forall d \in D)(fd \sqsubseteq f'd).$$

Finally, for any domain D, there is a domain $\mathscr{P}(D)$, the powerdomain of D, whose members are a certain subset of the power set of D, and whose construction is a slight variant of that given by Smyth [16].

Given a finite set L of ports, Milne and Milner would (with slightly different notation) introduce a pair of domains X_i and Y_i for each i in L, where they call L a sort. They then define the domain P_L of processes of sort L as a particular solution of the isomorphism

$$P_L = \mathscr{P}\left(\coprod_{i \in L} (Y_i \times [X_i \to P_L])\right). \tag{1}$$

They note (p. 305) that "the existence of such isomorphisms (without using $\mathscr{P}$) is due to Scott; a major purpose in the work of Plotkin and Smyth was to justify the use of the powerdomain construction in such isomorphisms." Scott was motivated to use domains and continuous functions because one can then solve the isomorphism $D \simeq [D \to D]$, but one cannot solve $D \simeq D^D$ in the category *Set* of sets and maps, where D^D comprises *all* maps from set D to itself. Similarly, Smyth and Plotkin introduced the powerdomain construct so that they could solve the isomorphism $D \simeq \mathscr{P}(D)$, since $D \simeq 2^D$ is not solvable.

Our critique is that it provides an unwieldy conceptual apparatus which makes the specification of even a simple process far less intuitive than the formal machine approach presented above, whereas, we claim, domains, continuous functions, and powerdomains may be bypassed using suitable constructions in the category of sets. To support the latter claim, we must introduce the notion of *greatest fixpoint* from Arbib and Manes [2].

Let X be any endofunctor of the category of sets. We introduce a category $CoDyn(X)$ whose objects (Q, Δ) are *X-codynamics*, that is maps $\Delta : Q \to XQ$, and whose morphisms $h : (Q, \Delta) \to (Q', \Delta')$ are maps $h : Q \to Q'$ for which we have commutativity of

$$\begin{array}{ccc} Q & \xrightarrow{\Delta} & XQ \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle Xh} \\ Q' & \xrightarrow{\Delta'} & XQ' \end{array}$$

We say that X has a *greatest fixpoint* (M, θ) if (M, θ) is terminal in the category $CoDyn(X)$, that is, if for each X-codynamics (Q, Δ), there is a unique $h : Q \to M$ such that

$$\begin{array}{ccc} Q & \xrightarrow{\Delta} & XQ \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle Xh} \\ M & \xrightarrow{\theta} & XM \end{array}$$

Such a θ, if it exists, is an isomorphism $M \simeq XM$. Moreover, the solution (M, θ) is itself unique up to isomorphism in $CoDyn(X)$.

The crucial point for our discussion is the first example given by Arbib and Manes. They note that a finite automaton (without specified initial state) can be given by a pair of maps $\delta : Q \times A \to Q$ (the next-state map, transition map, or dynamics) and $\beta : Q \to Y$ (the output map) or, equivalently, by a single map

$$\Delta_{\beta,\delta} : Q \to (Y \times [A \to Q]), q \mapsto (\beta(q), \delta(q, \cdot)). \tag{2}$$

In other words, such an automaton can be presented as a *codynamics* of the functor X defined by

$$QX = Y \times [A \to Q].$$

They then observe that the greatest fixpoint for this functor is the set of all response maps $[A^* \to Y]$, with the isomorphism

$$[A^* \to Y] \xrightarrow{\cong} Y \times (A \to [A^* \to Y]) \tag{3}$$

given by $h \to (h(\Lambda), a \mapsto hL_a)$, where $hL_a(w) = h(aw)$. The verification that (3) is the greatest fixpoint of our X is given by checking (corresponding to (2)) that there is a unique σ satisfying

$$\begin{array}{ccc} Q & \xrightarrow{\Delta_{\beta,\delta}} & XQ \\ {\scriptstyle \sigma}\downarrow & & \downarrow{\scriptstyle X\sigma} \\ [A^* \to Y] & \xrightarrow{\cong} & X[A^* \to Y] \end{array} \tag{4}$$

In fact, we can easily check that σ is the *observability map* which sends each q to its response map $\sigma(q) : A^* \to Y$ which specifies, for each input sequence w, the output $\sigma(q)(w) = \beta(\delta^*(q, w))$ (where $\delta^*(q, w)$ is the state reached from q by applying w) that the automaton would emit after reading in w from q as initial state.

Now compare the fixpoint definition (1) of the *domain* of processes with our definition (3) and (4) of the *set* of response maps as a greatest fixpoint. We may say that the Milne–Milner approach to processes is like a theory of automata which only talks of response maps $[A^* \to Y]$, and never discusses implementations $(\delta : Q \times A \to Q, \beta : Q \to Y)$. Our approach allows us to exploit the benefits of both interpretations, adopting whichever is more convenient to the purpose at hand, and moving back and forth where necessary.

With this background, let us re-analyze our definition of a port automaton, and of the response map of such an automaton. The port automaton $(L, Q, \tau, \delta, \beta, X, Y)$ may be re-presented as a codynamics

$$\Delta_{\beta,\delta} : Q \to \coprod_{i \in L} (Y_i \times [X_i \to 2^Q]), \tag{5}$$

which makes explicit the linkage which requires that the input and output must use the same port i in any activation. The form corresponding to (1) would be

$$Q \to \mathscr{P}\left(\coprod_{i \in L} (Y_i \times [X_i \to Q])\right), \tag{6}$$

which differs from (5) in the use of $\mathscr{P}$ rather than the subset operator, and in allowing nondeterminism to couple a choice of output with a choice of deterministic next-state map. We shall not discuss the latter difference further. The crucial question is "Can we afford to work with sets and 2^Q, or must we work with domains and replace 2^Q by $\mathscr{P}(Q)$ in (5)?"

For the purpose of the present section (but not for the study of interconnection), we limit ourselves to the case in which L is a singleton, and consider

$$\Delta_{\beta,\delta} : Q \to Y \times [A \to 2^Q]. \tag{7}$$

Note that the functor implicit here *cannot* have a greatest fixpoint, since we can solve

$$Q \simeq Y \times [A \to 2^Q]$$

in *Set* only if Y is empty, as we see by a simple cardinality argument. However, all is not lost, for we may replace the $\Delta_{\beta,\delta}$ of (7) by the map

$$\hat{\Delta}_{\beta,\delta} : \bar{Q} \to 2^Y \times [A \to \bar{Q}], \tag{8}$$

where $\bar{Q} = 2^Q$, and we have (if pr_i represents the projection map from a product onto its ith component)

$$\mathrm{pr}_1 \cdot \hat{\Delta}_{\beta,\delta}(p) = \{\mathrm{pr}_1 \cdot \Delta_{\beta,\delta}(q) : q \in p\},$$

while

$$(\mathrm{pr}_2 \cdot \hat{\Delta}_{\beta,\delta}(p))(a) = \bigcup \{(\mathrm{pr}_2 \cdot \Delta_{\beta,\delta}(q))(a) : q \in p\}.$$

Arbib and Manes [1] have given a general theory of nondeterministic dynamics $QX \to QT$, where QT may be 2^Q, probability distributions over Q, fuzzy sets on Q, (or even the power domain $\mathscr{P}(Q)$ [D. Lehmann, personal communication, 1976]). The correct setting for studying observability maps for nondeterministic processes $(Q \times A \to 2^Q)$ then proves to be at the level of (8) rather than that of (7). We have chosen not to develop the general theory here, but we note that in (8) both $\bar{Q} = 2^Q$

and 2^Y are semilattices under set inclusion; that the collection of all set-theoretic maps, $[A \to 2^Q]$, then inherits a semilattice structure; and then that the cartesian product $2^Y \times [A \to \bar{Q}]$ is a semilattice. (The general theory requires that T be a fuzzy theory (= monad = algebraic theory), and that semilattices be replaced by T-algebras.) The greatest fixpoint we seek, then, is not for the endofunctor $QX = Y \times [A \to 2^Q]$ of *Set* (we have already seen that $QX \cong Q$ cannot be satisfied for this X), but rather for the endofunctor $\bar{Q}\hat{X} = 2^Y \times [A \to \bar{Q}]$ in the category of semilattices and their homomorphisms. The general theory of Arbib and Manes [1] then specializes to the observation that X *does* have a greatest fixpoint, namely $[A^* \to 2^Y]$, equipped with the semilattice isomorphism

$$L : [A^* \to 2^Y] \to 2^Y \times [A \to [A^* \to 2^Y]],$$

which pairs each $h \in [A^* \to 2^Y]$ with $h(\Lambda) \subset 2^Y$ and $(a \mapsto hL_a)$ in $[A \to [A^* \to 2^Y]]$. The unique semilattice homomorphism $\sigma: 2^Q \to [A^* \to 2^Y]$ satisfying

$$\begin{array}{ccc} 2^Q & \xrightarrow{\hat{\Delta}_{\beta,\delta}} & 2^Y \times [A \to 2^Q] \\ {\scriptstyle\sigma}\downarrow & & \downarrow{\scriptstyle\hat{X}\sigma} \\ [A^* \to 2^Y] & \xrightarrow{L} & 2^Y \times [A \to [A^* \to 2^Y]] \end{array}$$

is indeed the nondeterministic observability map.

The key point in all this is that we move from the unsolvable $Q \simeq 2^Y \times [A \to 2^Q]$ to the solvable $\bar{Q} \simeq 2^Y \times [A \to \bar{Q}]$ subject to the restriction that this be a semilattice isomorphism. We note the following:

(i) We use arbitrary subsets 2^Y and maps $A \to \bar{Q}$, with no restriction to powerdomains or continuous maps.

(ii) The greatest fixpoint $[A^* \to 2^Y]$ is *not* of the form 2^B for any B, yet the Milne–Milner approach seeks the domain B of a least fixpoint, not the general form of this set of response maps.

(iii) The use of semilattices is the result of the choice $QT = 2^Q$ to express the nondeterminism in our processes. We can just as easily handle stochastic nondeterminism by replacing semilattices by T-algebras (see Arbib and Manes [1] for the general theory) for QT = probability distributions over Q.

Machine Interconnection

Up until now, we have been laying a formal foundation upon which to build our understanding of concurrent computation. We now introduce concurrency in the form of networks of communicating processes, realized as connections of port automata.

We make distinctions between types of port automata depending upon what properties each possesses. The kind of port automaton we define below permits no nondeterminism when viewed as a single entity.

DEFINITION. A port automaton P is said to be *deterministic* if τ is a singleton and δ is a partial function.

We wish to regard deterministic port automata as the atomic elements in a network of port automata, and we shall give a formal theorem to this effect in the next section. Before we begin discussing machine interconnection, however, we illustrate two examples of deterministic port automata.

Note. We have selected a state graph depiction of port automata in an effort to facilitate comprehension. The state graph scheme is to be interpreted as follows:

1. Each node is labelled in the following format:

state | {activated port : output set} | {inactivated port : output set}.

Thus, $q\,|\alpha_1 : T_1, \alpha_2 : T_2|\,\gamma_1 : U_1, \gamma_2 : U_2, \gamma_3 : U_3$ is a typical node labelling.

2. Each arc is labelled in the following format:

activated port in previous state: input value. Thus, $\mu_3 : V_3$ is a typical arc labelling.

3. If T_j, U_j, or $V_j = 1$, then the labelling is abbreviated, using only the port label.

The first example we consider is that of a binary semaphore (Dijkstra [4]). Two ports, α and γ, are necessary; the P operation is performed through port α, and the V operation is performed through port γ. (Were we to simulate the situation in which n processes access a semaphore, then our port automaton model would have to have $2n$ ports, one α and one γ dedicated to each of the n processes.) Initially, the semaphore is in a state in which a P operation may be performed. A V operation may be performed at any time. Refer to Fig. 4.

The second example we have chosen is a register capable of holding values from a set $A = \{a_1, a_2, \ldots\}$, which may or may not be finite. The register has two ports, α and γ; a value is written in at port α, and is read out (nondestructively) at port γ. In the initial state, the register is assumed to be empty, and if a read is attempted, the empty string, Λ, is returned. For each different $a_j \in A$ written into the register, we regard the register as being in a separate state, since intuitively the contents of the register determines its state (Fig. 5).

Recall that we had previously informally claimed that a network of processes could be regarded as a single process in itself. We will show formally that the inter-

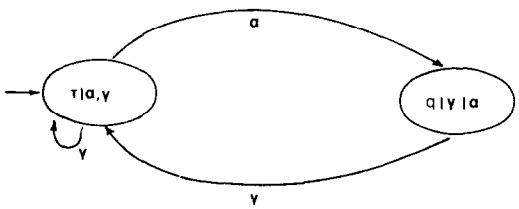

FIGURE 4

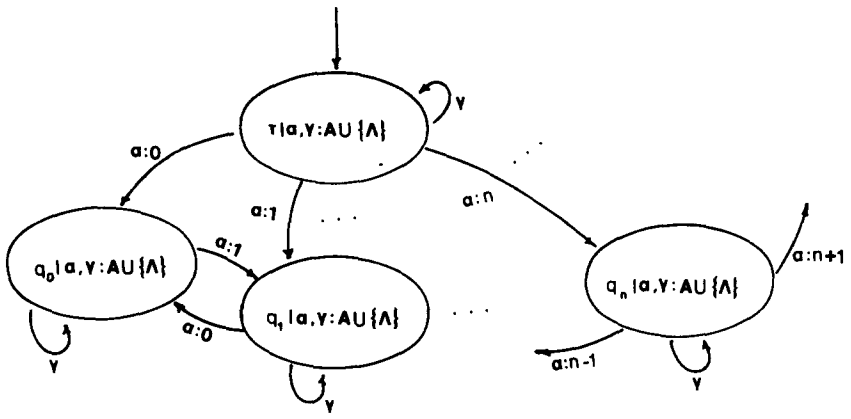

FIGURE 5

connection of two port automata is indeed a port automaton. For the immediate discussion, we will adopt the following convention:

Let P^1 and P^2 be two port automata, where

$$P^j = (L^j, Q^j, \tau^j, \delta^j, \beta^j, X^j, Y^j) \qquad \text{for} \quad j = 1, 2.$$

In order to interconnect two port automata, one or more ports from each automaton must be joined, under the constraint that the input set for each such port is the same as the output set for the corresponding port. We thus have

DEFINITION. Let $S^j \subseteq L^j$. The map $c : S^1 \to S^2$ is a *port connection map* if c is a bijection such that for each $i \in S^1$, $X_i = Y_{c(i)}$ and $Y_i = X_{c(i)}$. We call $c(i)$ the *complement* of i, and we say that i and $c(i)$ are *complementary* ports.

A port connection of two port automata yields a port automaton, such that the connected ports are subsumed and no longer visible to the environment. For the most part, the formal definition of a port connection is straight forward. As we shall see, it is the internal communications via the connected ports that make the behavior of the composite port automaton nontrivial.

DEFINITION. The *port connection* of P^1 and P^2 for a port connection map c, $P^1 \|_c P^2$, is a port automaton $P = (L, Q, \tau, \delta, \beta, X, Y)$, where

$$L = (L^1 - S^1) \cup (L^2 - S^2),$$
$$Q = Q^1 \times Q^2,$$
$$\tau = \tau^1 \times \tau^2 \in 2^Q,$$
$$X = (X_i : i \in L),$$

where

$$X_i = X_i^1, \quad \text{for} \quad i \in L^1 - S^1,$$
$$= X_i^2, \quad \text{for} \quad i \in L^2 - S^2,$$
$$Y = (Y_i : i \in L),$$

where

$$Y_i = Y_i^1, \quad \text{for} \quad i \in L^1 - S^1,$$
$$= Y_i^2, \quad \text{for} \quad i \in L^2 - S^2,$$
$$\beta = (\beta_i : i \in L),$$

where

$$\beta_i = Q \xrightarrow{\text{pr}_1} Q^1 \xrightarrow{\beta_i^1} Y_i^1, \quad \text{for} \quad i \in L^1 - S^1,$$
$$= Q \xrightarrow{\text{pr}_2} Q^2 \xrightarrow{\beta_i^2} Y_i^2, \quad \text{for} \quad i \in L^2 - S^2.$$

We have left the definition of δ for last since it is somewhat involved and must be unravelled in stages. We desire to find an expression for $\delta(q, (x, i))$ which captures the interactions of P^1 and P^2. Since communication between port automata is symmetric, we may assume that $i \in L^1 - S^1$. The complementary case may be done similarly.

Recall that state transitions in a port automaton are to occur as the result of communication with the environment. In a network of port automata, internal communications may take place in the interval between external communications. To determine what internal communications are possible between P^1 and P^2, we must know which of the connected ports are activated in a state $q \in Q$. We describe these in a set Z_q, the set of S^1-components of the set of simultaneously activated pairs of complementary ports. Let $q = (q^1, q^2)$.

$$Z_q = \{k \in S^1 : \exists x \in X_k^1 \text{ such that } \delta^1(q^1, (x, k)) \neq \varnothing, \text{ and}$$
$$\exists \bar{x} \in X_{c(k)}^2 \text{ such that } \delta^2(q^2, (\bar{x}, c(k))) \neq \varnothing\}.$$

The internal communications are responsible for state transitions in both P^1 and P^2, owing to the symmetric nature of the value exchanges involved. Let us observe what happens to P^1 and P^2 as a result of one internal communication via complementary ports k and $c(k)$, where $k \in Z_q$. We define a function $\hat{\delta} : Q \to 2^Q$ as follows:

$$\hat{\delta}(q) = \hat{\delta}(q^1, q^2)$$
$$= \{(\bar{q}^1, \bar{q}^2) : \bar{q}^1 \in \delta^1(q^1, \beta_{c(k)}^2(q^2)) \text{ and}$$
$$\bar{q}^2 \in \delta^2(q^2, \beta_k^1(q^1)), \text{ where } k \in Z_q\}.$$

We must not discount the possibility of multiple internal communications occurring between two external communications. Therefore, we extend $\hat{\delta}$ to its transitive closure $\hat{\delta}^*$ to include this possibility. For $j \geqslant 0$, $\hat{\delta}^j(q)$ is equivalent to the set of states reachable from q by j internal state transitions.

$$\hat{\delta}^0(q) = \{q\},$$
$$\hat{\delta}^j(q) = \hat{\delta}^{\#}(\hat{\delta}^{j-1}(q)), \qquad \text{where } (\ )^{\#} \text{ is extension by union,}$$
$$\hat{\delta}^*(q) = \bigcup_{j=0}^{\infty} \hat{\delta}^j(q).$$

Having represented by $\hat{\delta}^*(q)$ all states reachable from state q as a result of exclusively internal state transitions, we are now in a position to define the state transition directly effected by the input at port i. We first define this transition function for a single state $q \in Q$ as follows:

$$\tilde{\delta} : Q \times \bigsqcup_{i \in L} X_i \to 2^Q,$$

where

$$\tilde{\delta}(q, (x, i)) = \tilde{\delta}((q^1, q^2), (x, i))$$
$$= \{(\bar{q}^1, q^2) : \bar{q}^1 \in \delta^1(q^1, (x, i))\}.$$

Note that this definition implies that the port activation axiom which holds for P^1 and P^2 separately expands naturally to the port connection of P^1 and P^2.

By extending $\tilde{\delta}$ to the set of states obtained from internal transitions, we finally arrive at the completed definition of the transition function.

$$\delta(q, (x, i)) = \tilde{\delta}^{\#}(\hat{\delta}^*(q), (x, i)).$$

Thus, we have formalized our conjecture that a port connection of two port automata should again be a port automaton.

Machine Decomposition

We would like to show that the intuitive notion that any port automaton may be simulated by a port connection of deterministic port automata is indeed correct. However, the way in which we decompose a port automaton, though simple, is perhaps not obvious. That is, the port automata which comprise the port connection are not necessarily the deterministic port automata with which we are familiar, for example registers and semaphores.

Remember that arbitrary port automata needn't be deterministic. We must, somehow, incorporate nondeterminism into our simulation, and we do this by a port

connection of our deterministic automata. In fact, we shall find that almost any nondeterministic port automaton can be simulated by as few as two deterministic port automata connected appropriately.

Informally, the argument proceeds as follows. Let us say that we wish to simulate an arbitrary port automaton $P = (L, Q, \tau, \delta, \beta, X, Y)$. We will define a deterministic port automaton P^1, which is very similar to P in that it possesses P's port label set plus one additional port α through which communications pass to the other deterministic automaton P^2, and P^1's state set essentially "includes" P's state set plus an extra set of states to simulate the nondeterminism. This latter clause will be clarified in a moment. P^2 is a simple automaton consisting of one state and one port $c(\alpha)$, (where c is the port connection map). The state transition map simply takes the single state to itself.

$$
\begin{aligned}
P^2 \text{-----} \; & L^2 = \{c(\alpha)\}, \\
& Q^2 = \tau^2, \\
& X^2_{c(\alpha)} = 1 = Y^2_{c(\alpha)}, \\
& \delta^2(\tau^2, (\#, c(\alpha))) = \tau^2, \\
& \beta^2(\tau^2) = \#.
\end{aligned}
$$

The key idea in the design of P^1 is that it is the exchange with P^2 that will enable the deterministic P^1 to simulate the nondeterministic P. We shall enumerate possible next states, and then let the number of exchanges between P^1 and P^2 determine the new state of P^1, which when followed by an external transition, corresponds to the new state of P, following the identical external transition. Thus far, we have

$$
\begin{aligned}
P^1 \text{-----} \; & L^1 = L \cup \{\alpha\}, \\
& Q^1 = \tilde{Q} \cup Z \text{ (explained later on)}, \\
& X^1_i = X_i, \qquad Y^1_i = Y_i, \qquad \text{for} \quad i \in L, \\
& X^1_\alpha = 1 = Y^1_\alpha.
\end{aligned}
$$

We now begin to construct, in stages, $Q^1 = \tilde{Q} \cup Z$. For $q \in Q$, $x \in X_i$, $i \in L$, define $S = \delta(q, (x, i)) = \{q_1, q_2, \ldots\}$, a possibly infinite set. First, let us describe $\tilde{Q}$. For each $q \in Q$, we define $\tilde{q} \in \tilde{Q}$ such that if port i is activated in q, then port i is not activated in $\tilde{q}$. However, port α is activated in $\tilde{q}$ to allow communication between P^1 and P^2, in order to simulate the states contained in the set S. We will return to $\tilde{Q}$ later on.

We now proceed to define Z, the state set associated with the internal communications between P^1 and P^2. We decompose Z as follows,

$$
Z = \bigcup_{\tilde{q} \in \tilde{Q}} Z_{\tilde{q}}, \qquad \text{where } Z_{\tilde{q}} \text{ is the set of states generated by internal transitions starting from state } \tilde{q}.
$$

Note that since the internal communications are discrete, then the maximum number of states generated by internal communications beginning in state $\tilde{q} \in \tilde{Q}$ is at most countably infinite, the maximum being achieved when a different state transition occurs as the result of each internal communication. Thus, $|Z_{\tilde{q}}| \leqslant \infty$. In fact, we define the cardinality of each $Z_{\tilde{q}}$ to be equal to N, where N is the maximum of the cardinalities of the sets $\delta(q, (x, i))$ over all $q \in Q$, $x \in X_i$, and $i \in L$. This particular definition is made, since we shall see that each state in $S = \delta(q, (x, i))$ corresponds to a state in $Z_{\tilde{q}}$. We will, therefore, require that for all $q \in Q$, $x \in X_i$, and $i \in L$, $\delta(q, (x, i))$ must be at most countably infinite.

Thus, we may enumerate the elements of $Z_{\tilde{q}} = \{z_1, z_2, \ldots\}$, where

$$\delta^1(\tilde{q}, (\not\#, \alpha)) = z_1,$$
$$\delta^1(z_j, (\not\#, \alpha)) = z_{j+1}, \qquad \text{for} \quad 1 \leqslant j < N.$$

This explicitly shows that communication of P^1 with P^2 generates in P^1 the state set $Z_{\tilde{q}}$ from which the sets corresponding to S for each $x \in X_i$ and $i \in L$ will be created.

Let us now turn to the final phase of our definition, that is, the effect of the external communication through port i. We can enumerate the elements of $S = \delta(q, (x, i)) = \{q_1, q_2, \ldots\}$. We then define the value of the transition function for the specified external communication to be

$$\delta^1(z_j, (x, i)) = \tilde{q}_j \in \tilde{Q}, \qquad \text{if} \quad q_j \in \delta(q, (x, i)).$$

Note that for $k = |\delta(q, (x, i))| \leqslant j \leqslant N$, we define

$$\delta^1(z_j, (x, i)) = \tilde{q}_k.$$

To recapitulate, we see that our set of next states $\{\tilde{q}_1, \tilde{q}_2, \ldots\}$ for $\delta^1(\tilde{q}, (x, i))$ is generated from an intermediate state set of next states, $Z_{\tilde{q}}$, which in turn is generated by communication with P^2. We have completed our definition of the state set and the transition map for P^1; (the output map β^1 is defined in the obvious way.)

This concludes the proof of the following theorem:

THEOREM. *Let P be a port automaton with the following restrictions: τ is single-valued, and $|\delta(q, (x, i))|$ is countable for all $q \in Q$, $x \in X_i$, and $i \in L$. Then the response of P is identical to the response of $P^1 \|_c P^2$, for a port connection c of a deterministic port automaton P^1 and a one-state deterministic automaton P^2.*

We proceed to note a few consequences of the above construction.

1. The set of reachable states of P is (under the isomorphism "$\tilde{\ }$") contained in the set of reachable states of $P^1 \|_c P^2$. However, there is not necessarily any correlation between the unreachable states of P and those of $P^1 \|_c P^2$.

2. Note that although the transition from q to $q_j \in \delta(q, (x, i))$ may require only one time unit, the similar transition from $\tilde{q}$ to $\tilde{q}_j$, where $\tilde{q}_j = \delta^1(z_j, (x, i))$, requires at

least j more time units, since P^1 must pass through j intervening states $z_1, \ldots, z_j$ before reaching $\tilde{q}_j$. (Note that if $|Z_{\tilde{q}}| = M < \infty$, then $P^1 \|_c P^2$ will require at most M more time units at each transition than P. Thus, if, for an input of length n, P requires $t(n)$ time units, then $P^1 \|_c P^2$ would require at most $\tilde{t}(n) = (M+1)\,t(n)$ time units.)

3. If $|Z_{\tilde{q}}|$ is infinite for $\tilde{q} \in \tilde{Q}$, then the possibility of a nonterminating sequence of transitions between P^1 and P^2 exists. As a result, no future external communication may take place. This can be remedied by requiring an input to be present at a port in a finite but not necessarily bounded length of time following activation. This does not change the definition of a port connection, but rather is a restriction which may be necessary to add to a particular process description in order for the process to behave in the manner desired. Deadlock may not occur in a network of processes where this constraint is imposed upon the processes. (Note that this measure is necessary only in the situation in which P's inputs behave properly and $Z_{\tilde{q}}$ is not finite. In this case, we must ensure that P^1 also behaves properly, and so restrict its time from activation to communication to be finite for each port.)

This concludes our formal discussion of process decomposition.

Examples

We now return briefly to the port automaton, illustrated in Fig. 1, and consider the consequences of connecting together m of these automata in a cycle. The port connection maps for each such automaton P^i, $1 \leqslant i \leqslant m$, are as follows:

For $i = 2$ to $m - 1$:

$$S^{i-1} = \{\gamma_{i-1}\}, \qquad T^{i-1} = \{\psi_{i-1}\},$$
$$S^i = \{\gamma_i\}, \qquad T^i = \{\psi_i\},$$
$$S^{i+1} = \{\gamma_{i+1}\}, \qquad T^{i+1} = \{\psi_{i+1}\}.$$
$$c^1 : T^{i-1} \to S^i.$$
$$c^2 : T^i \to S^{i+1}.$$

(Note: P^1 and P^m are connected.)

Each of these processes has no terminating state, and will attempt to execute forever. However, if an input at any port of any of the m processes is not received within a finite length of time, then a deadlock situation will result, as described in the following scenario.

Suppose that for some $1 \leqslant i \leqslant m$, process P^i does not receive an external input that it needs in order to continue execution. Then it will be unable to send any more messages through port ψ in order to enable the next process. Process P^{i+1} may at present be executing, but it will soon receive no new enable signal, and then it too will stop execution. Eventually, all processes (except P^i) will be in a state corresponding to τ, that is, each will be waiting for an input through port γ (provided

of course that all inputs from the external ports have been received). The similarity of the processes and their cyclic arrangement fosters this type of deadlock.

In the above discussion, we spoke loosely about the notions of deadlock and nontermination; we will now attempt to solidify these notions. Consider a network of port automata.

DEFINITION. We say that a computation of a network is in a *deadlocked* state if, for each pair of connected ports, at most one port of the pair is activated, and for each activated external port, the expected input value will take an infinite amount of time to arrive.

DEFINITION. The computation of a network is said to be *nonterminating* if, in every state of the composite automaton, there exists at least one activated external port. A computation is said to have achieved *termination* if all external ports of all automata in the network are inactivated.

Note that at the level of the component automata of the network, no state transitions may occur in any automaton in a deadlock situation. However, it is possible that some of the automata in the network are themselves port connections of other port automata, and at that level, they may undergo various internal state transitions as the result of internal communications. These internal transitions are, however, invisible to the rest of the network, since they do not change the activation of any of the ports at the network level.

In addition to deadlock and nontermination, we also define lockout which may arise as a side-effect of a port connection of automata. An instance of this occurs in our next example.

DEFINITION. We say that a nonterminating computation of a network is *locked out* of a set R of reachable states if, after attaining a state q in a finite length of time, the set of states R is reachable from q, but the computation will take an infinite amount of time to reach any state in R.

The last example, a critical section problem for two processes, is adapted from Ladner [9] to fit our process conventions. Each of the processes, realized as a port automaton, may be in one of the following four states: "dormant," "trying," "critical section," and "end" (abbreviated "d," "t," "c," and "e," respectively). Each process has two ports, α and γ, where the status of the process (namely whether or not it is in its critical section) is sent through α, and the status of the other process is received through γ. (We shall adopt the convention that "2" indicates that a process is in its critical section and that "1" indicates otherwise. At each state, the process will attempt to inform the other process of its present state, while concurrently it will be expecting to receive the present state of the other process.

A process may only change state as a result of receiving the state of the other process. The process begins in the "dormant" state and may then proceed to the "trying" state. From the "trying" state, the process may advance to its "critical section" only if the other process is not in its critical section. From the "critical

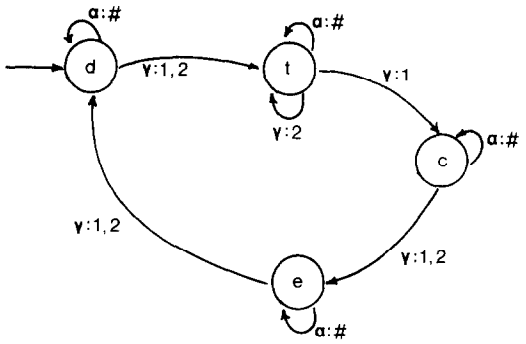

FIGURE 6

section," the process proceeds to the "end" state, and from "end," back to "dormant," indicating with this state change that the other process is now free to enter its critical section. Figure 6 pictures the state graph for one such automaton. (At each state, both ports α and γ are activated, where the output sets for each are $\alpha : \{1, 2\}$ and $\gamma : \{\neq\}$. This labelling will be deleted from each node, since it is the same for all nodes of the state graph.)

We now connect the two processes, P^1 and P^2, through ports α_1 and γ_2, and ports γ_1 and α_2, and allow these processes to communicate in any state, even when one of them is in its critical section. (Notice that after the port connections, the network no longer has any external ports, and is thus incapable of any visible state transitions. To avoid this problem, we add an external port η to each process P^i, which functions as a signal to the environment that process P^i has completed its critical section; the output set of η is simply $\{\neq\}$. Port η will only be activated in the "end" state of process P^i.)

The state graph for the specified port connection of P^1 and P^2 is pictured in Fig. 7. (Normally, we only depict the external communications of a network in a state diagram. However, in order to facilitate the reader's comprehension of the example, we explicitly show all nontrivial state transitions, which, in this case, result from internal communications only. Since all ports are activated in all states, with the exception of the "end" state of each process, then the list of activated and inactivated ports and their output sets will be dropped from the node labellings. Each node will be labelled with a pair of states, each element of the pair representing the state of one of the port automata, P^1 and P^2, subscripted accordingly.)

We notice that we do indeed have mutual exclusion from the critical section. However, in every state, there is the potential for lockout, since one of the processes may continually send its status to the other process, thus remaining in the same state while permitting the other process to advance. Since at each state, the probability of receiving an input at α or at γ is 1/2, and since each of the processes executes forever, then the probability of a lockout goes to zero in the limit.

This concludes our discussion of process interaction.

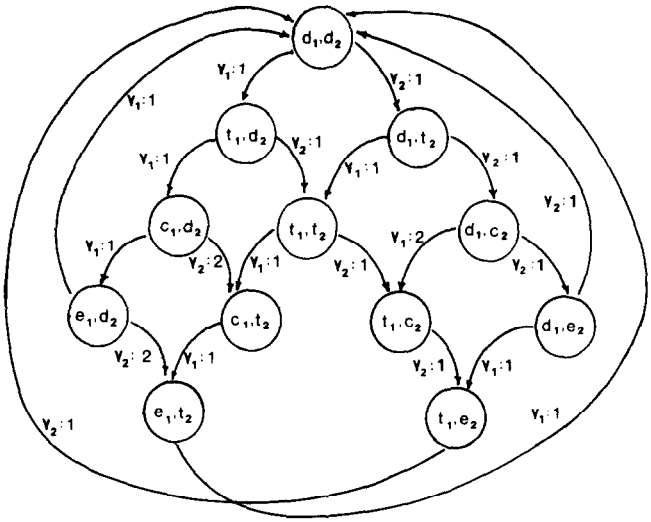

FIGURE 7

CONCLUSIONS

In approaching concurrent computation from the automata-theoretic perspective, we have attempted to present a relatively simple and straightforward model of communicating processes, in the form of port automata. We have observed that our intuitive conceptions of process interconnection and decomposition are indeed formalizable in this framework. We have found that port automata lend themselves naturally to a state graph representation, a visual aid in understanding process interaction. We have observed that, using an automata-theoretic approach to concurrency, we may define an algebraic notion of process without requiring ordered sets.

REFERENCES

1. M. A. ARBIB AND E. G. MANES, Fuzzy machines in a category, *Bull. Austral. Math. Soc.* **13**, No. 2 (1975), 169–210.
2. M. A. ARBIB AND E. G. MANES, The greatest fixpoint approach to data types, *in* "Proceedings of the 3rd Workshop Meeting on "Categorical and Algebraic Methods in Computer Science and System Theory," November 3–7, 1980, Dortmund, West Germany.
3. P. BRINCH–HANSEN, Distributed processes: A concurrent programming concept, *Comm. ACM* **21**, No. 11 (1978), 934–941.
4. E. W. DIJKSTRA, The structure of "THE" multiprogramming system. *Comm. ACM*, **11**, No. 5 (1968), 341–346.
5. E. W. DIJKSTRA, Co-operating sequential processes, *in* "Programming Languages" (F. Genuys, (Ed.), pp. 43–112, Academic Press, New York, 1968.

6. R. H. CAMPBELL AND A. N. HABERMANN, The specification of process synchronization by path expressions, *in* "Operating Systems" (E. Gelenbe and C. Kaiser, Eds.), pp. 89–102, International Symposium, Rocquencourt,, 1974, LNCS, No. 16, Springer-Verlag, Berlin, 1974.
7. C. HEWITT, Viewing control structures as patterns of passing messages, *Artificial Intelligence* **8**, No. 3 (1979), 323–364.
8. C. A. R. HOARE, Communicating sequential processes, *Comm. ACM* **21**, No. 8 (1978), 666–677.
9. R. E. LADNER, The complexity of problems in systems of communicating sequential processes. *J. Comput. System Sci.* **21** (1980), 179–194.
10. G. MILNE AND R. MILNER, Concurrent processes and their syntax. *J. Assoc. Comput. Mach.* **26**, No. 2 (1979), 302–321.
11. R. MILNER, "A Calculus of Communicating Systems," Lecture Notes in Computer Science No. 92, Springer-Verlag, Berlin, 1980.
12. R. MILNER, "A Calculus of Communicating Systems" (G. Goos and J. Hartmanis, Eds.), pp. 33–40, Lecture Notes in Computer Science No. 92, Springer-Verlag, Berlin, 1980.
13. C. A. PETRI, Concepts of net theory, in "Mathematical Foundations of Concepts of Computer Science," pp. 137–146, High Tatras, Sept. 1973.
14. G. D. PLOTKIN, Apowerdomain construction, *SIAM J. Comput.* **5**, No. 3 (Sept. 1976), 452–487.
15. A. PNUELI, "The temporal Semantic of Concurrent Programs," Lecture Notes in Computer Science No. 70, Springer-Verlag, 1979.
16. M. B. SMYTH, Power domains, *J. Comput. System Sct.* **16** (1976), 23–36.