March 1985

Report No. STAN-G-85-1048

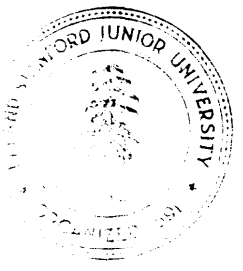
Some Constructions for Order-Theoretic Models of Concurrency

by

Vaughan Pratt

Department of Computer Science

Stanford University
Stanford, CA 94305



Some Constructions for Order-Theoretic Models of Concurrency

Vaughan Pratt
Stanford University
Stanford, CA 94305

Abstract

We give "tight" and "loose" constructions suitable for specifying processes represented as sets of pomsets (partially ordered multisets). The tight construction is suitable for specifying "primitive" processes; it introduces the dual notions of concurrence and orthocurrence. The loose construction specifies a process in terms of a net of communicating subprocesses; it introduces the notion of a utilization embedding a process in a net.

1. Introduction

1.1. An Application

In September 1983 STL (Standard Telecommunications Laboratories Ltd) and SERC (Science and Engineering Research Council) jointly sponsored a workshop on the analysis of concurrent systems [DH]. The workshop was organized around ten problems for solution by the attendees. Each problem informally described a concurrent system to be formally specified.

The first problem was probably the easiest. For whatever reasons, it also attracted the lion's share of the attention, The formal solutions presented varied in length from seven axioms of one line each (Koymans-de Roever) to two pages. The following paragraph reproduces verbatim the statement of the problem.

The "channel" between endpoints "a" and "b" can pass messages in both directions simultaneously, until it receives a "disconnect" message from one end, after which it neither delivers nor accepts messages at that end. It continues to deliver and accept messages at the other end until the "disconnect" message arrives, after which it can do nothing. The order of messages sent in a given direction is preserved.

Intuitively, the complexity of the presented solutions is out of proportion to the complexity of the concept of a two-way channel with disconnect. The "tight" construction of this paper yields a sufficiently short solution that we can conveniently preview it here:

$$(\Sigma^* \times TR \times \{C,D\}) \cap \tilde{} \blacklozenge \delta$$

In English this reads "(Message sequence, Transmitted then Received, on channels C and D) with no prior disconnect". It specifies essentially the same mathematical object that we required 13 lines of detail to specify in [Pr83] (our contribution to the workshop). A key construct here is the operation X of *orthocurrence* which provides the interpretation of the commas in the English.

1.2 The Significance of Pomsets

Strings arise naturally in modelling ongoing sequential computation, whether their elements correspond to states, state transitions, or message transmission-receipt events. Thinking of a string as a linearly ordered multiset, a mathematically natural generalization is to relax the linear order to a partial order. If the resulting objects are used for modelling computation, the meaning of incomparability between elements may be interpreted as concurrency of events.

We abbreviate partially ordered multiset to pomset.

The almost universal practice in modelling concurrency is to represent concurrency as 'choice of order in sequential computation. The two arguments supporting this approach are that partial orders are no more expressive than linear, and are not as convenient to work with,

We refute both these arguments. The first is correct for posets but not for pomsets, as demonstrated in [Gis] with the counterexample ab||ab, which in a linear model is indistinguishable from N(a,a,b,b) where N(1,2,3,4) schedules 1<3, 2<4, 1<4. The second is only true in that relatively little work has been done on partial orders for modelling computation. In fact the opposite seems to be the case - total orders turn out to be very inconvenient to work with for some basic concurrency applications, so much so that we speculate that this is one of the major reasons why the formal modelling of concurrency remains an arcane topic today.

In earlier papers [Pr82,Pr84] we showed how to derive event-order models as homomorphic images of real time models. For this to be possible the former must have a structure similar to that of the latter. Only partial orders on events meet this requirement.

In this paper we exhibit two dual operations and show various applications for them. We see these operations as being fundamental to concurrency. The only way we see to define them for a linearly ordered model is to define them for a partially ordered model and at the last step complete the partial orders to linear orders. The analog of this in astronomy would be to use Newtonian mechanics to derive the motions of the planets and at the last step convert all the results to epicycles.

1.3 Why Categories

Given "all" sets, ordered by inclusion, union and intersection are the operations of least upper and greatest lower bound respectively. This definition is popular, whether for its elegance, convenience, or algebraic flavor. It certainly makes it obvious that union and intersection are dual notions.

Another commonly encountered dual pair arc disjoint union and Cartesian product. These too can be described as the operations of least upper and greatest lower bound, if we generalize from inclusions to all functions between sets and define "least upper bound" and "greatest lower bound" appropriately so that they continue to make sense, and are willing to settle for uniqueness up to isomorphism.

A category is a collection of objects such as these sets together with some morphisms between them such as inclusions or functions, viewed at the right distance for seeing the commonality of these definitions. At this distance least upper and greatest lower bounds on a set X of objects are coproducts and products rcspectivcly, while the upper and lower bounds, whether or not least or greatest, are the vertices of cones from or to X. (A minimal account of cones and products is provided for convenience as an appendix.)

In our case WC want to define two basic operations of concurrency for pomsets that turn out to be coproducts and products in the category of pomsets with morphisms all pomset homomorphisms. Both the duality of these operations and their affinity with the important cases above arc made clearly visible with a categorical treatment.

Winskcl [W84a] describes a morphism of Petri nets, having in mind just these sorts of applications: defining coproducts and products. This is then cxtcndcd [W84b] to a number of morphisms appropriate to various models of concurrency, whose relationships to each other arc expressed via adjunctions. With luck there will turn out to be either a straightforward or an

interesting connection between his morphisms and ours.

2. Definitions

2.1 Pomsets

The setting is order-theoretic models of concurrency. We take the central concept to be the pomset or partially ordered multiset. Informally, a pomset is to a string as a partial order is to a total order; a string is not a totally ordered set but rather a totally ordered multiset. Pomsets are intended to model ***behaviors*** (synonyms: trace, computations) of some process. The process itself is identified with the set of all its possible behaviors.

Pomsets may be defined in elementary language as labelled partial orders up to isomorphism. In the context of dealing with pomsets as objects of a category the obvious way (in hindsight) is to treat them as functors from a category of events to a category of actions. To keep the material reasonably accessible we do both here. More detailed elementary treatments may be found elsewhere [Gis,Pr84].

A *labelled* ***partial order*** (lpo) is a 4-tuple $(V, \Sigma, \mu, \leq)$ consisting of

'(i) a set V of ***events,***

(ii) an ***alphabet*** Σ of ***actions,*** e.g. the arrival of integer 3 at port Q, the transition of pin 13 of IC-7 to 4.5 volts, or the disappearance of the 14.3 MHz component of a signal,

(iii) a labelling function $\mu: E \rightarrow \Sigma$ 'assigning actions to events, each labelled event representing an ***occurrence*** of the action labelling it, with the same action possibly having multiple occurrences, and '

(iv) a partial order $\leq$ on E, with the intended interpretation of $e \leq f$ being that e precedes f in time.

A *pomset* (partially ordered multiset) is then the isomorphism class of an lpo, denoted $[V, \Sigma, \mu, \leq]$. By taking lpo's up to isomorphism we confer on pomsets a degree of abstractness equivalent to that enjoyed by each of strings (regarded as finite linearly ordered labelled sets up to

isomorphism), ordinals (regarded as well-brdered sets up to isomorphism), and cardinals (regarded as sets up to isomorphism).

(Cryptic remark: the reason isomorphism appears in our definition of pomsets but not in the usual definitions of string, ordinal, or cardinal, is that partial orders, unlike well-ordered sets, have nontrivial automorphisms. This complicates finding a canonical representative of the isomorphism class.)

No assumption is made about atomicity of events; in fact our tight construction of processes depends on events having structure.

2.2 Types of Pomsets

It will be convenient to regard all of the following structures as kinds of pomsets.

Multiset	Pomset with a minimal (i.e. empty) order
Tomset	Pomset with a maximal (i.e. total) order
String	Finite tomset
Poset	Pomset with an injective labelling
Set	Poset that is also a multiset
Atom	A singleton pomset (both a set and a string)
Unit	The empty pomsct (hence the empty string, and empty set)

A ***process*** is a set of pomscts. Close analogs are binary relations, which are sets of pairs, and languages, which are sets of strings. In all cases the set structure can bc regarded as modclling variety of one or another kind of behavior. A ***language*** is a process all of whose behaviors are strings. The language of a process is the set of strings in that process.

We write $\Sigma\dagger$ for the set of all finite pomscts with alphabet Σ, by analogy with Σ^* for the set of all strings with alphabet Σ, which happen to bc all strings in $\Sigma\dagger$.

Implicit in this two-layered notion of process (sets of pomscts of events) is a two-sided distributivity axiom; that order may bc distributed over choice, i.c. concatenation over union. This axiom is appropriate for partial correctness but not always for termination and deadlock. WC impose this axiom on our notion of processes for now in the interest of understanding the basic pomset model better, with a minimum of distracting clctail. Note that this is a limitation only of our

notion of processes, not of the pomsets themselves.

2.3 Pomsct I homomorphisms and the Category POM

We now wish to define homomorphism between pomsets, to facilitate the definitions of concurrcnce (disjoint union) and orthocurrence (direct product).

We take a pomset homomorphism to be a consistently-relabelling monotonic function between the underlying sets of the two pomsets. Consistent relabelling means that if two events have the same label so do their images under the homomorphism. Monotonicity is defined in the usual way with respect to the partial order.

The category POM consists of ail pomsets together with all their homomorphisms.

We call coproduct and product (see appendix) in this category respectively ***concurrence*** and ***orthocurrence***. Concurrence takes the disjoint union of pomsets while orthocurrcnce takes their direct product. Events accumulate under concurrence but combine orthogonally under orthocurrence, as will become clearer below. We write p||q for the concurrence of p and q (p + q is a plausible alternative notation), and p×q for their orthocurrence.

If the string ab, a linearly ordered multiset and hence pomset, is thought of as two events a followed by b, then the concurrence of ab and cd contains four events, a followed by b, together with c followed by d, with no additional temporal relationships between ab and cd.

The orthocurrence of ab with cd contains four events, ⟨a,c⟩, ⟨a,d⟩, ⟨b,c⟩, and ⟨b,d⟩, with ⟨a,c⟩ ≤ ⟨a,d⟩ ≤ ⟨ b,d⟩ and ⟨a,c⟩ ≤ ⟨b,c⟩ ≤ ⟨b,d⟩.

If WC write 2 for the multiset (0.0) and 00 for the result of linearly ordering 2, 2 and 00 both being pomsets, then p×2 = p||p (ignoring the extra 0 on each label) whereas p×00 = p.p. This second operation is concatenation, in this case of p with itself. Like concurrcncc, the concatenation p.q is an "upper bound" on (the vertex of a cone from, see appendix) p and q. This makes it clear that concatenation is not the dual of concurrence but rather is a bound on {p,q} superior to p||q (since concurrcncc is the least upper bound) - indeed concatenation lies on the opposite side of concurrcncc from orthocurrcnce.

Bctwccn the concurrcncc and the concatenation of p and q lies a complete lattice of intermediate

upper bounds on p and q, each consisting of p||q with some additional ordering information running from p to q, an arbitrary subset of the concatenation ordering in p.q. We will find some of these intermediate bounds of importance below.

All these intermediate bounds are representable as quotients of a concurrence, the quotients modifying only the order and otherwise constituting label-preserving bijections. We may call quotients of this form ***serializations.***

3. The Tight Construction

3.1 The Cons Process

By way of motivation consider the Cons process. This process has two input ports A and B and one output port 0. The first value output from 0 is the first value input from A; thereafter all outputs on 0 are taken in order from B. Further inputs on A' are not accepted and therefore do not appear at all in the behaviors of Cons.

The typical behavior of this process can be diagrammed as in Figure **1.**

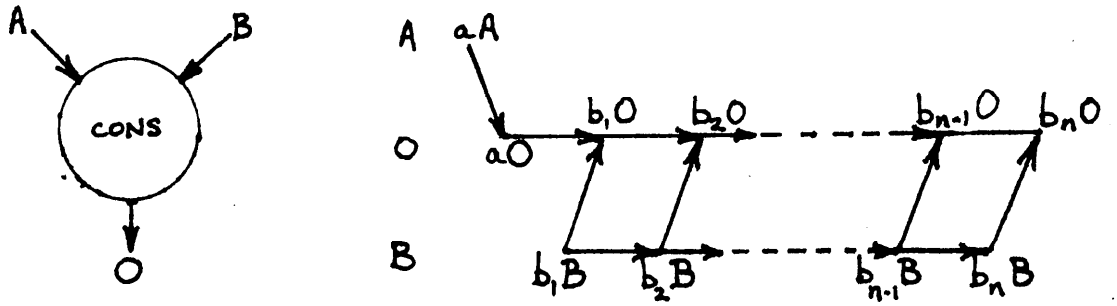


Figure 1. The Cons Process and its Typical Behavior

We may represent this process as a sum of products. There are two products. corresponding to two channels, from A to 0 and from B to 0 respectively. The AO channel passes only o n e datum, the BO chan nel is an ordinary order-preserving channel. The sum is constrained so as to force the one 0 output from A (and hence all of the AO channel) to precede all the 0 outputs from B; thus it is intermediate between concurrence and concatenation. We adopt the ad hoc notation A.0 for this operation: p A.0 q indicates partial concatenation in which the A component of p prcccdcs the 0 component of q. Omission of either subscript denotes no restriction, so p.q means concatenation. When p and q are strings p.q is string concatenation; more generally it means that every element of p precedes every element of q.

Now consider the individual products. A channel is a set of behaviors, each which in turn is a sequence of message transitions. Such a sequence has two orthogonal components: a string σ of messages, and a string AO or BO consisting of a transmission action A or B followed by a receipt action 0 and constituting a "transition schema." The orthocurrence of σ with AO yields the pomset of Figure 2; BO is similar.

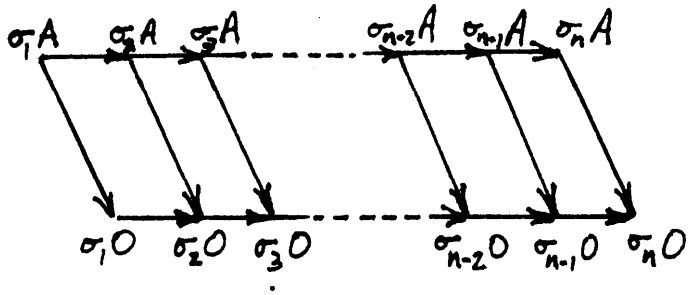


Figure 2. Orthocurrence of a Message String with the Transition Schema

Letting Σ^* denote the set of all message sequences, the set of all possible behaviors for the BO channel is then $\Sigma^* \times BO$ (with orthoccurrence being distributed over union).

We may now define the Cons process:

$$Cons = CXAO \cdot_O \Sigma^* \times BO.$$

3.2 Related Processes

The dual of Cons is CarCdr, a process with one input I and two outputs A and B, which feeds its first input to A and the rest to B. Like Cons CarCdr is a sum of products. We leave it to the reader to interpret:

$$CarCdr = \Sigma \times IA \cdot_I \Sigma^* \times IB.$$

The duality of Cons and CarCdr is made quite clear from the formulas.

A somewhat different example is given by Merge, which has inputs A and B and output 0 and merges its two stream inputs arbitrarily. Similarly to Cons, Merge is a sum of two products, in this case $\Sigma^* \times AO$ and $\Sigma^* \times BO$. The sum however differs from that for Cons in that instead of specifying a particular order we impose the weaker requirement that the 0 channel merely be linearly ordered. Again adopting an ad hoc notation we write $[p]_A$ to indicate the set of all

augmentations of the order of p in which the A component of p is linearly ordered, with omission of the subscript again denoting no restriction. This yields:

$$\text{Merge} = [\Sigma^* \times AO \parallel \Sigma^* \times BO]_O$$

We might call the dual of Merge Spray. Spray has input I and outputs A and B, and sends each of its inputs to an arbitrarily chosen output. It is given by

$$\text{Spray} = [\Sigma^* \times IA \parallel \Sigma^* \times IB]_I$$

A functional process F which computes the function f repeatedly has one input I and one output 0, and for each input σ outputs $f(\sigma)$. It may be defined as

$$F = \varphi(\Sigma^* \times IO)$$

where φ is a pomset morphism which replaces each label $\langle \sigma, O \rangle$ with $\langle f(\sigma), O \rangle$ and leaves the rest of the pomset unchanged.

We take for our concluding example the two-way channel with disconnect described in our opening remarks, which was

$$(\Sigma^* \times TR \times \{C,D\}) \cap \tilde{} \blacklozenge \delta$$

This has some features in common with the previous examples. However there are a couple of new ideas requiring explication here. We have remarked earlier that orthocurrence of p with a set such as {C,D} produces essentially the concurrence of p with itself, except that the events are marked with one of C or D according to which copy of p they are in. Hence this construction yields two concurrently operating one-way channels C and D. Each action in this system is a triple of the form $\langle$msg,type,channel$\rangle$ where type is 'I' or R and channel is C or D, with $\langle$type,channel$\rangle$ providing the location of the action and msg its value.

The part about disconnection breaks newer ground. It is the one example in which temporal logic [Pn] is used. δ is a predicate on messages, and hence on actions and hence on events, defining which messages say to disconnect. $\blacklozenge$ (a solid diamond, Koymans-de Roever [DH]) is a temporal modality, an existential predicate transformer yielding predicates on events. $\bullet$ projects out or smears its argument forwards in time. In our case it smears δ forwards so that it holds for

any event f for which there exists an event e<f satisfying 6(e); hence $\blacklozenge\delta$ asserts that a disconnect request has happened in the past. We wish to forbid or eliminate all such events, so we change the sign to yield $\tilde{}\blacklozenge\delta$.

If we regard $\tilde{}\blacklozenge\delta$ as the set of all processes all of whose events meet this condition then intersection with it forbids such events, an operation of a logical character. If we regard it as an operation on pomsets that shrinks them by removing unwanted events then application of it to each behavior in the process eliminates such events, an operation of an algebraic character. Either approach will work; in the formula above we used the former,

There is one detail in which the process named by this expression differs from the process named in [Pr83], namely that in the latter the events at each "endpoint" of the two-way channel were linearly ordered. The informal specification does not raise this as an issue, so the process named in our present solution is adequate. On the other hand one might object that our two channels are completely independent of each other. The linearization introduces at least a small degree of coupling. We may bring it into coincidence with the process named in our previous solution by use of the linearization operator $[p]_A$ introduced above. The two endpoints may then be incorporated into the solution by considering them as the names "a" for {TC,RD} (Transmit on C and Receive on D) and "b" for {TD,RC} respectively. This leads to our final solution:

$$[[\Sigma^* \times TR \times \{C,D\}]_{\{TC,RD\}}]_{\{TD,RC\}} \cap \tilde{}\blacklozenge\delta$$

Thcsc varied examples suggest that the notions of concurrence and orthocurrence, and associated serializations, may prove to be useful tools for the specification of processes.

4. The Loose Construction

In this section WC give a construction of proccsscs in terms of networks of constituent processes. In this manner of combining processes the constituent processes may be recognized merely by intercepting the flow of data along network channels. When wires implement channels the coupling can bc adjusted merely by adjusting the connections of the wires. This seems to be a looser kind of coupling of processes than that of the last section, where the components of an assembled process were not identifiable mcrcly by watching data flowing on channels, and where the coupling was of a more logical than physical character.

The general idea here is to define the notions of translation, projection, and utilization, each in

terms of its predecessor, with translation being a particularly simple-minded notion. Then a net process may be defined as the intersection of utilizations of its constituent processes, and a process implemented by a net may be defined as a projection of a net process.

The main difference between this account of composition and that of [Pr84] is the explicit notion of a utilization as an operation determined solely by an insertion, with utilizations combining via intersection.

4.1 Substitutions

A pomset algebra is a collection of pomsets closed under the pomset-definable operations [Gis], described below. A *substitution* is an endomorphism of a pomset algebra that replaces each event e of a pomset by a pomset p, such that if $e \leq f$ ($f \leq e$) before the substitution then after the substitution $e' \leq f$ ($f \leq e'$) for every event e' in p, and such that the same pomset is substituted for events with the same label.

A substitution is determined by its behavior on atoms (singleton pomsets). As such it has the character of a homomorphism on a free algebra, which is determined by its behavior on generators. Pomset substitutions are the natural generalization of string homomorphisms.

The algebraic structure of strings is straightforward: they combine under the single binary operation of concatenation. It can be shown [Gis] that there is no corresponding finite set of operations for building all finite pomsets from atoms. However the pomsets themselves can be used to define an infinite set of operations using which all pomsets may be built from atoms, the *pomset-definable operations* [Gis]. Such an algebra is free, freely generated by its atoms, and substitution is a homomorphism on it. That is the sense in which substitution is a morphism of pomset algebras.

4.2 Translations

We shall refer to a function $f: \Sigma \rightarrow \Sigma'$ between two alphabets Σ and Σ' as a *translation.* This extends in the usual way to a length-preserving homomorphism of strings. In the same way it also extends to a size-and-structure preserving homomorphism of pomsets, as introduced in [Gis] and recounted below in the section on substitutions.

43 Insertion

Consider an alphabet of the form CXD, where C is to be thought of as denoting a set of channels or locations and D a set of data or values. In this context an insertion. is a data-preserving translation, one that affects only the location.

The example that gives rise to the name "insertion" is the insertion of a component (say an integrated circuit) into a net (say a printed circuit board). Each pin of the IC connects to some wire on the PC board. We assume two asymmetries: every pin connects to some wire, but not conversely, and a wire may short (connect) two pins but a pin may not short two wires (certainly true for socketed IC's, and more generally we may define "wire" to make it always true). It follows that the connection relation between pins and wires is actually a connection function mapping pins to wires. Not being injective corresponds to shorting pins, and not being surjective corresponds to some wires not being connected to any pin. (On the other hand, if insertion were a relation it would not compromise the development.)

An insertion is then a translation mapping (c,d) to (f(c),d) where f is a connection function.

4.4 Projection

. **A** projection is the extension to a substitution of the inverse of a translation. The name is motivated by the case when the translation is an insertion, as will become apparent.

The inverse of a translation $t: \Sigma \to \Sigma'$ is $t^- : \Sigma' \to 2^\Sigma$, defined as usual as $t^-(\sigma') = \{\sigma | t(\sigma) = a'\}$. In this context we take a set of actions to be a pomset that is a poset with the empty order; thus the inverse maps actions, i.e. atoms, to pornsets. This inverse t^- then extends homomorphically, as described above for substitutions, to a substitution $t^- +$ from pomsets to pomsets.

The inverse of an insertion amounts to a coordinate transformation for actions, going from the net coordinates (wires) to the component coordinates (pins). In addition to renaming locations it takes care of whether a particular net action is relevant to this component, deleting it if not, and also whether it must occur simultaneously on more than one channel of the component, duplicating it if necessary. Duplicated events arc unordcrcd, which is as close as we can come in this model to saying that they occur simultaneously.

For example consider a componenthaving channels C and D and a net having channels E and F.

Insert the component into the net via the connection mapping C and D to E, thus connecting C and D together. Let the data domain be {0,1}. Then the inverse insertion maps action (E,1) to the set {(C,1),(D,1)} (meaning that event (E,1) can only occur in the net when events (C,1) and (D,1) both occur in the component) and action (F,1) to the empty set (meaning that this action is irrelevant to the component). More generally, if in a pomset (net behavior), (E,1) precedes (E,O) precedes (F,1), then the projection of that pomset onto the component has four events (C,1), (D,1), (C,0), (D,0), with each of the first two preceding each of the second two.

4.5 Utilization

A utilization is the union of the pointwise extension (to a function on processes) of the inverse of a projection. Let us take this one step at a time. Begin with $\pi: \Sigma' \dagger \to \Sigma \dagger$, a projection from pomsets to pomsets Its inverse $\pi^- : \pi: \Sigma \dagger \to 2^{\Sigma \dagger}$ maps the pomset p to the set of those pomsets whose projection under π is p; we interpret that set as a process. For a process P, n-(P) is then a set $\{\pi^-(p) | p \in P\}$ of processes, the pointwise extension of π^- to a function from processes to sets of processes. Finally we write $\pi^- \cup (P)$ for $\cup\{\pi^-(p) | p \in P\}$, the union of that set.

The intended application for utilization is to embed a process's behavior into a net in such a way that, on the one hand the behavior contains information only about that process's behavior in terms of the topology of that net, yet on the other the behavior of the whole net reduces to the intersection of the utilizations of its components, as explained in the next section.

4.6 Composition

Given a family Pi of processes, each on an alphabet $C_i \times D$, and their associated connections $c_i: C_i \to C$ into a common net having connection set C, we may now express the process P consisting of the set of all possible behaviors of the net via the following formula.

$$P = \Gamma I_i (c_i \times I_D)^- + - \cup (P_i).$$

Each function $c_i \times I_D$ is the insertion induced by c_i, I_D being the identity function on D. The superscripts then promote each insertion to a utilization. Finally the product of the resulting processes is formed, the product operator being ordinary intersection for the case of processes as sets of behaviors.

This rule is very far-reaching. It can model composition of binary relations, composition of

processes via arbitrary nets, connection via a bus or ethernet where messages may be broadcast by any process attached to the bus or ethernet to any other such process, and analog circuits such as a net of resistors. The rule handles these diverse systems by being defined independently of whether behaviors are discrete orders (as associated with conventional models of computation) or dense orders (as associated with analog circuitry), and also independently of "direction" of flow of data on channels.

In the case of resistive nets, Ohm's law shows up as a local property of the constituent resistors and Thevenin's Theorem (including the special cases in which series resistances sum and parallel resistances sum harmonically) shows up as a global property of the net. This is all accomplished using resistor behaviors each consisting just of two events (e,i,p) giving the voltage e and current i at each of two ports p = 0 and 1, with the sum over the ports of the i's vanishing and the sum of the e/i's being the resistance; utilization then does all the remaining work of predicting the behavior of an arbitrary net of resistors. We hope to get into more depth with analog processes in future papers.

4.7 Evolution of the Utilization-Intersection Approach

Our approach to process composition evolved in small steps from Gilles Kahn's original rule for composing processes. The steps are:

1. [K,KM] Define a component to be a continuous function from n-tuples of sequences to sequences. Continuity is defined relative to the prefix order on sequences. Define a net to be a system of equations each of the form x = f(y,...,z) where each variable x,y,z denotes a connection in the net and each equation a component. Express the behavior of the net as the least solution (under prefix ordering) to this system.

2. [BA] Instead of a total order on events within each channel, have a partial order on all events, permitting between-channel ordering as well. This is to extend Kahn composition to deal with nondeterministic processes, which Brock and Ackerman showed could not work merely by using relations on sequences instead of functions.

3. [Pr82] Introduce the pbmset as the underlying abstraction for Brock-Ackerman semantics. Separate the Brock-Ackerman composition rule into two steps: composition of component behaviors to form the maximal net behavior whose restrictions to those components are those components, then restriction of that behavior to the external ports of the net, itself modelled as a

component of the net. Take all solutions, not just the least,

4. [Pr84] Recast the above using only standard constructions: inverse, homomorphic extension, pointwise extension, union. Remove the distinction between input and output ports. Remove all finiteness and discreteness requirements to admit analog circuits.

5. [this paper] Introduce the concept of a utilization. Separate the rule for forming net behavior into utilizations followed by intersection.

Appendix - Pure Categories

The many categorically innocent readers seeking relief in a comprehensive text [ML] have the dual burden of extracting just the relevant material and massaging it to fit our viewpoint. We address this here by listing all the definitions we need for this and the next paper, slanted appropriately. This is probably misguided; your indulgence is begged.

An edge-labelled graph, or *multigraph*, is one with a set of edges between any two vertices, as opposed to zero or one edges. A *free **category*** is an algebra consisting of all paths of some directed multigraph. This algebra has only one operation, path concatenation g;f. Paths determine their endpoints, whence to each object corresponds a distinct empty path and g;f is only defined if g starts where f ends. By convention vertices are called ***objects,*** paths ***morphisms,*** loops ***endomorph&u,*** concatenation *composition*, the empty path at b the ***identify*** lb, and the two' endpoints of a path its ***domain*** and *codomain* rcspcctively.

A ***category*** is a quotient (homomorphic image) of a free category. (This just means that some paths are collapsed together by an equivalence relation that is furthermore a congruence with respect to concatenation,) A *discrete* category has only identity morphisms. Two morphisms $f:a\to b$, $g:b\to a$ satisfying $g:f = 1$, and $f:g = 1_b$ arc called ***isomorphisms*** and a and b are said to bc ***isomorphic.*** An *automorphism* is both an isomorphism and an endomorphism. A ***preorder*** is a category with at most one morphism between any two objects (in each direction). A *clique* is a maximal preorder. A *poset* is a preorder with only trivial (identity) isornorphisms. (Contrast: cliques and posets are prcorders with respectively a maximal and a minimal set of isomorphisms.)

A *functor* is a homomorphism bctwccn two categories (c.g. the aforementioned quotient). A *natural transformation* τ of functors $S,T:C\to B$ is a function τ which maps each object c in C to a morphism $\tau c:Sc\to Tc$ in B such that the disjoint union of S(C) and T(C) together with the

morphisms of $\tau(C)$ form a category (i.e. the set of all those morphisms is closed under the composition defined on those morphisms in C). Natural transformations compose, according to $(\tau;\sigma)c = \tau c;\sigma c$ to yield a natural transformation (!). Hence the collection of all functors between two categories itself forms a category whose morphisms are the natural transformations of those functors.

An object *is initial (final)* in a category C when there is exactly one morphism in C from it to (to it from) each object in C. Initial (final) objects are isomorphic (!).

Given a functor $F:C \rightarrow B$, define C^+ to be C augmented with a final object (and hence with the additional morphisms required to make it final). A *cone from* F is any functor $F^+:C^+ \rightarrow B$ extending F; its ***vertex*** is the image under F^+ of the final object of C^+. The ***category of cones from*** F has as objects those cones and as morphisms those natural transformations of cones which assign 1_{Fc} to each object c of C.

A ***colimit*** of F is the vertex of an initial object of the category of cones from F. A ***coproduct*** is a colimit of a functor with a discrete domain. The respective dual notions are *cone to* F, *limit,* and ***product.***

Acknowledgments

I thank Jose Meseguer for helpful discussions and hints.

Bibliography

[B] Brauer, W., Net Theory and Applications, Springer-Verlag LNCS 84, 1980.

[BA] Brock, J.D. and W.B. Ackerman, Scenarios: A Model of Non-Determinate Computation, In LNCS 107: Formalization of Programming Concepts, J. Diaz and 1. Ramos, Eds., Springer-Verlag, New York, 1981, 252-259.

[Dl-l] Denvir, T., W. Harwood, M. Jackson, and M. Ray, **The Analysis of Concurrent Systems,** Proceedings of a Tutorial and Workshop, Cambridge University, Sept. 1983, LNCS, Springer-Verlag, to appear.

[Gis] Gischer, J., Partial Orders **and the Axiomatic Theory of Shuffle,** Ph.D. Thesis, Computer

Science Dept. Stanford University, Dec. 1984.

[H] Hoare, C.A.R., Communicating Sequential Processes, CACM, 21, 8, 666-672, August, 1978,

[K] Kahn, G., The Semantics of a Simple Language for Parallel Programming, IFIP 74, North-Holland, Amsterdam, 1974.

[KM] Kahn, G. and D.B. MacQueen, Coroutines and Networks of Parallel Processes, IFIP 77, 993-998, North-Holland, Amsterdam, 1977.

[ML] Mac Lane, S., Categories for the Working Mathematician, Springer-Verlag, NY, 1971.

[M] Milner, R., A Calculus of Communicating Behavior, Springer-Verlag LNCS 92, 1980.

[Pn] Pnueli, A., The Temporal Logic of Programs, 18th IEEE Symposium on Foundations of Computer Science, 46-57. Oct. 1977.

[Pr82] Pratt, V.R., On the Composition of Processes, Proceedings of the Ninth Annual ACM Symposium on Principles of Programming Languages, Jan. 1982.

[Pr83] Pratt, V.R., Two-Way Channel with Disconnect, in [DH], section 3.1.3.

[Pr84] Pratt, V.R., The Pomsct Model of Parallel Processes: Unifying the. Temporal and the Spatial, Proc. CM U/SERC Workshop on Logics of Programs, to appear in Springer Lecture Notes in Computer Science series, Pittsburgh, 1984.

[W84a] Winskel, G., A New Definition of Morphism on Petri Nets, Springer Lecture Notes in Computer Science, 166, 1984.

[W84b] Winskel, G., Categories of Models for Concurrency, Technical Report no. 58, University of Cambridge, England, undated (rec'd Dec. 1984).