The Pomset Model of Parallel Processes: Unifying the Temporal and the Spatial

Vaughan Pratt
Stanford University
Stanford, CA 94305

1. Temporal Theory - Qualitative Scheduling

1.1. Background

The progenitor of our pomset model is Kahn's history-transformer model of nets [Kah74, KaM77], in which a process is viewed as a function from n-tuples of histories to histories, where a history is a sequence of values. Each connection or channel of the net is associated with such a history. Kahn's model, being functional rather than relational, only caters for determinate processes. This has the advantage of permitting a straightforward least-fixed-point analysis, and the disadvantage of excluding such basic processes as the merge process. The Brock-Ackerman anomaly [BA81] demonstrates the need for something more than histories in extending Kahn's model to treat nondeterminate processes. Brock and Ackerman show how to extend Kahn's model by inclusion of ordering information between events on different channels.

The pomset model was introduced by the present author [Pra82]. It is intended as a theoretical framework for the Brock-Ackerman extension of Kahn's model. However the topic of models of concurrency has been particularly active in recent years, with the inevitable consequence that any given viewpoint ages rapidly. Thus the present paper describes our original model but from a more current perspective.

The basic idea of the model remains unchanged: a process is a set of pomsets, networks are defined in terms of events as channel-data pairs, and real-time is modelled by extending partial orders to more general semirings (only hinted at near the end of the paper [Pra82]).

A major change is the symmetrization of input and output. Our channels are no longer directed; instead they are just places to share information. Communication is by consensus: information is shared simply when the communicating parties agree to share it. This both simplifies the model and makes it more useful and less oriented to any particular communication protocol, a goal of our original paper that was not as well met as now.

With this improvement to our model has come the ability to model bus communication, in which several processes all have both read and write access to a single channel. We can model quite complex bus protocols on even a single wire on which handshakes and data in both directions are represented simultaneously by appropriate combinations of say voltages and impedances. We also can model continuous as well as discrete systems. Indeed cooperating physical laws like $F = ma$ and $E = mv^2/2$ may be modelled as continuous communicating processes, permitting physical and information systems to be modelled not only in the same style but even in the same model. This opens up the possibility of proving correct a discrete flip-flop starting from the continuous equivalent-circuit model of its constituent transistors, an exercise we hope to engage in on some future occasion.

By far the most extensive work on pomsets to date is Jay Gischer's thesis [Gis84]. Although his thesis concentrates on axiomatizability questions for theories of pomsets under several different combinations of pomset operations, it also makes a number of other contributions to the subject of pomsets and their algebras. The material in this first section on pomsets is within the scope of Gischer's thesis, and we take the opportunity to advertise some of his results. The two following sections, on spatial processes and real time, are topics not addressed in the thesis. We adopt Gischer's notation throughout.

There are strong connections between our pomsets and Glynn Winskel's event structures [W84]. One difference is in the conflict relation that is a part of event structures but is absent from the pomset model.

1.2. Variety, Sequence, and Concurrency

In this section we build up to the pomset model by starting with what we feel are quite basic notions for any process-oriented model of computation. By "process" we have in mind either event-oriented or state-oriented computation, with a contrast being intended with say the applicative style of functional programming, where one emphasizes functions, their application, and their types. On the other hand we see less of a contrast with some models of imperative programs - we view the binary-relation-on-states model advocated by de Bakker [deB72] and used for dynamic logic as a special case of the current study.

A fundamental concept in a process-oriented model of computation is variety of behavior. A popular way to model variety is with sets: a program or process is modelled as the set of its possible behaviors.

If the program exhibits only stimulus-response behavior, as with the binary-relation-on-states model, each behavior may consist of a stimulus-response pair. In this case a program consisting of a set of such pairs may be considered a binary relation from stimuli to responses. If the program is determinate (same stimulus yields same response every time) and total (every stimulus yields a response), this relation will then be a function from stimuli to responses. Stimuli and responses may be either events or states - we will not attempt here to distinguish these two concepts.

A second fundamental concept is course of events, or thread of control, a concept that is important in the context of "ongoing" processes. The stimulus-response model admits only a limited notion of course of events, namely an initial event followed by a final event. A richer model would include intermediate events as well, generalizing behaviors from pairs of events to sequences of events. The concept of variety of behavior continues however to be modelled as a set of alternative such behaviors, making variety of behavior a notion more basic than, or at least independent of, course of events.

Whereas a set of pairs is a binary relation, a set of sequences is a language (possibly with infinite strings), and so languages constitute an appropriate model for ongoing processes. The language model is made particularly attractive by the extensive work done on formal languages over the past two decades in response to the needs of both programming and natural languages. The word "trace" is used by some authors to denote a sequence used in this application.

A third fundamental concept is concurrency: two events happening at the same time, or overlapping in time, or in the weakest sense of "concurrency," just having no specific temporal relationship to each other. One approach to modelling concurrency is to treat it in terms of variety in course of events, that is, as a derived concept utilizing both variety of behavior and course of events. A basic example of this is to represent the concurrent execution of two processes, modelled as languages L and M, as the language $L||M$ consisting of all interleavings or shuffles of strings one from each of L and M. This is the approach adopted in language or trace models.

An alternative approach to modelling concurrency is to consider it a primitive notion in its own right, independent of both of the notions of variety of behavior and course of events. One argument for this goes as follows.

For two atomic (indivisible) processes L and M, modelled as languages each consisting of one unit-length string, their concurrent execution $L||M$ is simply $LM \cup ML$. For arbitrary languages however there is no such convenient expression for $L||M$ in terms of course of events (concatenation) and variety of behavior (union). If L and M are given as built up with concatenation and union (and possibly other operations, e.g. Kleene star) from atomic processes then one may derive an expression for $L||M$ in terms of the atoms and the other operations. However to require this decomposition into atoms represents a serious handicap for a model or logic of concurrent programs.

Consider for example the situation where a software engineer needs to verify the operation of a system containing two concurrently executing programs. He is provided with specifications of the behavior of the programs, but not to a sufficient level of detail that he can say what the interleavings of the atomic commands are. This situation can arise either because the atomic behavior is far too complex to be reckoned with, or because the supplier of the programs does not want the user depending on implementation details that go beyond the specifications, in order to preserve the supplier's flexibility in subsequently improving the software, or because the granularity of interleaving varies between applications.

If sets primitively model variety and strings primitively model sequentiality, what primitively models concurrency? We propose the pomset, or partially ordered multiset.

A string s of length n on alphabet Σ is commonly defined as a function $s:\{0,1,2,...,n-1\}\rightarrow\Sigma$. In order to lead up to pomsets we shall adopt a different definition: s is a totally ordered multiset of n elements drawn from Σ. To be a multiset means that the drawing is performed with replacement. If sets were used rather than multisets, the alphabet $\{0,1\}$ would yield only five strings, namely the empty string, 0, 1, 01, and 10. The distinction between multiset and set is whether sampling is with or without replacement. The distinction remains when the set or multiset is equipped with an order to make it a pomset or poset respectively.

Informally, a partially ordered multiset is a string with the requirement removed that its order be total. The phrase "partially ordered set" abbreviates to "poset", so by analogy we shall abbreviate "totally ordered multiset" to "tomset" and "partially ordered multiset" to "pomset." We now formalize the notion of pomset.

A *partial order* is a set $(V,<)$ where $<$ is an irreflexive transitive binary relation on a vertex set V. A *labelled partial order* or lpo is a 4-tuple $(V,\Sigma,\mu,<)$ where $(V,<)$ is a partial order and $\mu:V\rightarrow\Sigma$ labels the vertices of V with symbols from an alphabet Σ.

Two lpo's $[V,\Sigma,<,\mu]$ and $[V',\Sigma,<',\mu']$ are *isomorphic* when there exists a bijection $\tau:V\rightarrow V'$ such that for all u in V, $\mu(u) = \mu'(\tau(u))$, and for all u,v in V, $u<v$ just when $\tau(u) <' \tau(v)$.

A *pomset* $[V,\Sigma,<,\mu]$ is the isomorphism class of the lpo $(V,\Sigma,<,\mu)$.

The reason for defining pomsets only up to isomorphism is to suppress the identities of the elements of V, so that only the cardinality of V counts, leaving Σ as the only important set underlying a pomset, just as the alphabet is the only important set underlying a string. The motivation is the same as in graph theory, where the identity of graphs is normally defined only up to isomorphism. See the appendix for further discussion of isomorphism.

The carrier V may be considered the events of a behavior. The symbols of the alphabet Σ are the actions associated with those events. In the context of operating systems the distinction is that between job and program: the job is an *execution* of the program and the program *controls* the job. A job is a particular event, and the program controlling the job determines the action taking place. One job executes only one program, but one program may control many jobs, even concurrently. In the context of communication the distinction is that between message and contents: a message contains its contents and its contents appear in the message. One message has only one contents, but the same contents may appear in many messages. The word "event" is intended to be interpreted broadly to cover jobs, messages, experimental observations, etc.

Applications. We make no assumptions concerning the atomicity, temporal extent, or spatial extent of events. Events may be atomic, or may have a structure that can itself be represented as a pomset. An event may represent a point in time, or an interval of time. An event may represent the momentary occurrence of a voltage at a point in an integrated circuit, or the formation of the Rocky Mountains. Our emphasis is on keeping the model simple rather than forcing attributes of certain types of events on it. Applications are to be fitted to the model, not the model to any particular application.

Related structures. Multisets, sets, tomsets, strings, n-tuples, and posets may be considered particular kinds of pomsets. A multiset is a pomset with the empty order, that is, an unordered pomset. A poset (partially ordered set) is a pomset with an injective labelling. A set is a multiset that is also a poset (unordered, injective labelling). A tomset is at the other extreme from a multiset: it is totally ordered. A string is a finite tomset. An n-tuple is a string of length n. An ordered pair is a 2-tuple, written either ab or (a,b) depending on context.

In the case of a set (and hence also poset), the labels uniquely determine the elements of V. In this case it is sometimes convenient to identify V with the range of μ and to consider $<$ to partially order that range rather than V itself. Sets and posets may then be treated merely as subsets and posets of Σ, and V and μ dispensed with. The ordered sets Z and R of integers and reals respectively are particularly useful pomsets; they provide a basis for discrete and dense time. In the case of R time forms an ordered set of points, but without a metric for distance.

The (unique) empty or unit pomset, denoted ε, meets our definition of string and therefore must be our old friend the empty string. It is also a multiset, a set (the empty set), and a poset.

An atom, or atomic pomset, has just one element. Atoms also satisfy our definitions of string, multiset, set, and poset. The atoms are in one-to-one correspondence with Σ, and so just as in language theory we shall feel free to make the usual identification between atoms and symbols.

1.4. Algebras of Pomsets

Like most mathematical objects, pomsets do not make good hermits, but thrive when allowed to gather together into algebras. As in any society a certain degree of conformity is demanded of its members: an algebra of pomsets has a base consisting of an ordinal β, and an alphabet Σ consisting of an arbitrary set. Only $\beta\Sigma$-pomsets may join such an algebra, namely those having the form $[V,\Sigma,<,\mu]$ where V is an element of β. (As usual we identify each ordinal with the set of those ordinals less than it.)

The finite pomsets are those with base ω, the countable ones those with base ω_1 (the least uncountable ordinal, which consists of exactly the countable ordinals). The binary pomsets are those with alphabet $2 = \{0,1\}$, the finite decimal tomsets are decimal numerals.

Algebras come with operations. Among the more elementary operations are shuffle and concatenation. (Since no actual shuffling takes place one might prefer in place of shuffle a more neutral name; "co" for concurrency suggests itself, as in the pronunciation of a||b as "a co b.")

Definition. Let $p = [V,\Sigma,<,\mu]$ and $p' = [V',\Sigma,<',\mu']$ be pomsets with V and V' disjoint. Then their *shuffle* $p||p'$ is $[V \cup V',\Sigma,< \cup <',\mu \cup \mu']$ and their *concatenation* p.p', or just pp', is $[V \cup V',\Sigma,< \cup <' \cup (V \times V'),\mu \cup \mu']$.

No loss of generality is entailed by the assumption of disjointness since pomsets are only defined up to isomorphism. If we regard pomsets as graphs then shuffling merely juxtaposes them, placing them side by side to form one graph, while concatenation not only juxtaposes them but adds additional edges to the order, one from each element of V to each element of V'. It is easily verified that the ordering relation remains irreflexive and transitive even with the additional edges introduced by concatenation. Both operations generalize to more arguments; any *string* of pomsets may be concatenated, while any *multiset* of pomsets may be shuffled.

Both concatenation and shuffle are associative, and shuffle is commutative. Neither distributes over the other. The unit pomset ε serves as a two-sided identity for both concatenation and shuffle: $p\varepsilon = \varepsilon p = \varepsilon||p = p$. These laws completely axiomatize the equational theory of pomsets under concatenation and shuffle [Gis84].

Concatenation preserves strings, and indeed is just what is normally meant by concatenation of strings in formal language theory. Shuffle on the other hand does not preserve strings. Gischer defines a function from pomsets to languages that amounts to the completion of a pomset to a set of tomsets. This function, or functor if we were describing all this categorically, maps shuffle to the usual notion of language shuffle.

One might ask what is special about concatenation and shuffle as pomset operations. Here they form a natural introduction to a more general class of operations, the *pomset-definable* operations. Gischer [Gis84] defines both these and the notion of substitution or pomset homomorphism at the same time as follows, with an operation that we shall call expansion.

1.5. Expansion: Pomset-definable Operations and Substitution Homomorphisms

Let P,P' be algebras of pomsets with respective alphabets Σ,Σ', let $p = [V,\Sigma,<,\mu]$ be a pomset of P, and let $\alpha:\Sigma \rightarrow P'$ be a "Σ-tuple" of pomsets of P'. Informally the expansion of p via α transforms p into a pomset p' in P' by expanding each v in p into a whole pomset $p_v = \alpha(\mu(v))$, preserving order both within and between the p_v's.

More formally, let $p'' = [V',\Sigma,<'',\mu']$ be the shuffle (juxtaposition, or disjoint union) of the pomsets $p_v = \alpha(\mu(v))$ for v in V. Let $<^*$ partially order V' such that for each pair u',v' in V' drawn from p_u,p_v respectively, $u'<^*v'$ just when $u<v$ in p. Let $<'$ be the union of $<''$ and $<^*$. Then the *expansion of* p *via* α is just $p' = [V',\Sigma',<',\mu']$.

Thus in an expansion p' two elements of V' may be comparable for one of two reasons: they came from a common element v of V, in which case they were comparable in the pomset p_v that replaced v, or they came from different elements u and v which were comparable in p.

For example let $\Sigma = 2 = \{0,1\}$, let C = 01, S = 0||1 be two-element pomsets (taking 0 and 1 as atoms as explained above), and let <C,C> and <S,S> be pairs (2-tuples, i.e. functions with domain 2). Then the expansion of one of C or S via one of <C,C> or <S,S> leads to one of four pomsets: C and <C,C> yield the string 0101, C and <S,S> yield (0||1)(0||1), S and <C,C> yield (01)||(01), and S and <S,S> yield 0||1||0||1.

The expansion of C via a pair (p,q) is just pq, the concatenation of the pair. Thus the pomset C defines the binary operation of pomset concatenation. Similarly the expansion of S via (p,q) is p||q, so the pomset S defines the binary shuffle operation. More generally, any pomset with alphabet $n = \{0,1,2,...,n-1\}$ defines some n-ary operation in an algebra of pomsets on some (other) alphabet Σ' mapping n-tuples of pomsets to pomsets. When the defining pomset is infinite the significance of such operations is not clear, so it is natural to consider the pomset-definable operations to be restricted to those defined by finite pomsets.

If we regard a Σ-tuple α as a function from atomic pomsets to pomsets, then the pomset function mapping p to the expansion of p via α may be regarded as the natural extension of α to pomsets. This situation is very common in algebra. Suppose we are given some set of free generators, say the variables in some term language, or the symbols of an alphabet used to form strings. Then a function f from the generator set to some algebra has a unique extension f^+ to a homorphism from the algebra that the generators generate. In the case of variables occurring in terms, such a homomorphism is called a substitution. In the case of symbols in strings it is the notion of string homomorphism encountered in language theory. (There also exist language homomorphisms, which further extend string homomorphisms to homomorphisms on sets of strings.) As with terms and strings, pomsets are freely generated by their atomic constituents.

It is natural to identify the homomorphism itself with the generator-mapping function that it extends, since the extension always exists and is unique. Thus we may consider a Σ-tuple to define, or more simply to be, a pomset homomorphism. We shall follow the terminology that goes with terms and call such homomorphisms *substitutions.*

To summarize: expansion takes one pomset from an algebra P with alphabet Σ and a Σ-tuple of pomsets from an algebra P' with alphabet Σ', and yields a pomset from P'. Fixing the one pomset determines a Σ-ary operation on the algebra P'; when Σ is the ordinal n and P is finite, this operation is an n-ary pomset-definable operation. Fixing the Σ-tuple of pomsets yields a function f that in turn determines a substitution f^+ from P to P'.

The notion of tomset is a potentially useful generalization of the notion of string. The concatenation of two tomsets is defined and a tomset, even when the tomsets are uncountable.

The operations of concatenation and shuffle are obviously of considerable interest. One may ask whether any of the other operations are of interest, or even whether any of them are not already expressible directly in terms of concatenation and shuffle. The latter question is answered positively in Gischer's thesis - concatenation and shuffle do not constitute a basis for all the finitary pomset-definable operations. Gischer proves the much stronger result that those operations have no finite basis - no finite set of operations forms a basis. This is in contrast say to the set of finitary Boolean operations, for which a single operation may serve as a basis.

One operation that is of some interest is the quaternary operation N(p,q,r,s). This is the operation defined by the poset {0,1,2,3} with $0<2$, $1<2$, and $1<3$. This operation is not expressible using shuffle and concatenation.

Substitutions are of interest wherever there is structure. In the purely temporal theory of this section, as opposed to the spatial theory below, the structure is in the events. What may be an atomic event from one perspective may be revealed as a more complex event closer up. A substitution may expand a point into a hive of scheduled activity.

We will also find substitutions useful in developing the spatial theory, where they allow us to describe the effect on behavior of connecting a component into a system.

1.7. Discrete vs. Continuous Pomsets

Functions may map reals to integers or integers to reals or reals to reals just as readily as integers to integers. By the same token we may have pomsets with either the base or the alphabet or both being a continuum.

In the pomset model time appears only as an order; there is no measure. For a discrete order (every element but the last has a well-defined successor) one may use the successor relation to infer a measure: each element follows its predecessor one unit later. But in a dense order there is no such obvious measure. Thus if we take (V,<) to be the set of reals with its standard order, we have an unmeasured time dimension in which there is no way to detect the speeding up or slowing down of time, either locally or globally.

1.8. Example: Modelling the Two-way Channel with Disconnect

At an STL/SERC workshop on the analysis of concurrent systems, held in Cambridge, UK, in August 1983, the participants were asked to specify each of ten information systems. The first system, a two-way channel with disconnect, was probably the simplest; it also got the lion's share of the participants' attention. It was presented thus. "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."

Here is a solution to this problem within the pomset framework. The solution emphasizes formality at the expense of succinctness; we have an approach to achieving both at once that we shall describe elsewhere.

The desired channel is the set of all its possible behaviors. Each such behavior is a pomset which is constructed, in a way specified below, from a structure (V,<,m,port,contents,erase) satisfying conditions 1-8 below. V is a finite set of events (either transmissions or receipts of messages), < is a partial ordering of V indicating necessary temporal precedences, m:V→V is a function giving, for each transmission or receipt, its matching receipt or transmission, port:V→{0,1} is a function giving for each event v the port (0 or 1) at which v occurs, erase is a predicate holding for those events to be erased by the construction, and contents:V→M maps each event to its contents, drawn from the set M of possible message contents, among which is the message D for disconnect. We write u<>v to denote comparability of u and v, namely u<v or v<u.

For all u and v:

1. u = m(m(u))	m is a pairing function
2. u<>m(u)	transmission-receipt pairs linearly ordered
3. (port(u) = port(v)) → u=v or u<>v	ports linearly ordered
4. port(u) + port(m(u)) = 1	matching spans ports
5. u<m(u) & u<v & v<m(v) → m(u)<m(v)	channel is order preserving
6. contents(u) = contents(m(u))	channel is noiseless
7. u<v & erase(u) → erase(v)	erasure is suffix-closed
8. contents(u)=D & u<v → erase(v)	D forces erasure of all subsequent events

Events come in matched pairs u,m(u) (1), with one preceding the other (2). Whichever comes first is the transmission event, the other is then the matching receipt. The set of events at each port is linearly ordered in time (3). Transmission and receipt occur at opposite ports (4). Messages are received in the order transmitted (5). Message contents are received as transmitted (no noise on the channel) (6). The predicate "erase" specifying which events did not really happen is suffix-closed: any event following a non-happening is itself a non-happening (7). Nothing happens after transmission of a disconnect message (including the matching receipt of that disconnect message) (8). Note that this does not preclude two nonerased concurrent transmissions of disconnect messages from the two ports.

Now from each structure satisfying these conditions construct a pomset $(H,\Sigma,<',\mu)$ where H is that subset of V for which erase does not hold. <' is the restriction of < to H, $\Sigma = (2\times 2)\times M$ (where 2 = {0,1}), and $\mu(u)$ = ((port(u),m(u)<u),contents(u)) where m(u)<u is 0 or 1 depending on whether that predicate fails or holds respectively. Then the process consisting of all possible such pomsets is the

desired two-way channel with disconnect.

The label ((p,t),x) on each event in each such pomset indicates the port p at which that event occurred, the type of event - 0 is transmission, 1 is receipt - and the message x received or transmitted.

It will be noted that the function m that pairs up events is absent from the label. The idea is that this information, though visible during our construction of the process, should not be visible in the finished process on the ground that it is not an "observable" of that process. We can see messages being transmitted and others being received, but the problem did not require that we be able to keep track of the connection between transmissions and receipts. Accordingly the connection is deleted from the final specification.

The construction of the pomsets may be viewed as their implementation, and the connection between transmissions and receipts as an implementation detail. The implementation style of specifying things, where a structure is built up and then partially discarded, is widely used in mathematics. Consider for example the construction of the integers as the quotient of sets of pairs (a,b) of natural numbers with the equivalence relation $(a,b) \equiv (a+c,a+c)$, interpreting each equivalence class [(a,b)] as the integer $a-b$; the rationals may be constructed similarly as pairs (a,b), reduced modulo the relation $(a,b) \equiv (ac,bc)$.

2. Spatial Theory - Communication in Nets

2.1. Projection and Net Behaviors

We have viewed a process behavior thus far as a collection of events distributed only in time, with the distribution being determined by the order. The term "endogenous," applied by A. Pnueli to distinguish temporal logic from an "exogenous" logic like dynamic logic, seems to be equally applicable to this purely temporal notion of process. In an endogenous model the universe is viewed as a single process. An exogenous model has distinct processes each with its own identity independent of other processes, yet able to coexist and communicate with other processes. We now wish to be more exogenous, that is, to distinguish between independent communicating processes.

Definition. A *translation* is a function $t:\Sigma\rightarrow\Sigma'$ between two alphabets.

As an example of translation, suppose we are given a module with two channels (ports) 0 and 1 on each of which may appear values from a set D, and suppose we wish to use this module in a context having connections 2,3,4 by attaching 0 to 3 and 1 to 4. Then the event (0,d) denoting the appearance of value $d\in D$ on channel 0 is translated to the event (3,d) in the 2,3,4 context. Similarly an event (1,d) is translated to (4,d). In this example Σ is $2\times D$ and Σ' is $\{2,3,4\}\times D$, and translation affects channel names but not data values.

As another example we may have for Σ the real interval [0,5] and for Σ' the set {0,X,1} with X denoting invalid, and a translation mapping all elements of the interval [0,2] to 0, the (open) interval (2,3) to X, and [3,5] to 1. This translation would correspond to the interpretation of analog signals, perhaps voltages, as digital signals in a three-valued system. The obvious application is in using an analog module in a digital context.

The above two examples can be combined into a single example which translates both the channels and the data values, translating (c,d) to (f(c),g(d)) where f and g are the respective translations of those examples.

Definition. Given a translation $t:\Sigma \rightarrow \Sigma'$, the *projection induced by* t is the substitution $t^{-+}:\Sigma'\dagger \rightarrow \Sigma\dagger$ mapping pomsets on Σ' to pomsets on Σ.

In more detail, the inverse of an arbitrary function $t:\Sigma \rightarrow \Sigma'$ is not $t^{-}:\Sigma' \rightarrow \Sigma$ but rather $t^{-}:\Sigma' \rightarrow 2^{\Sigma}$, a function mapping symbols to sets of symbols. Any symbol not in the range of t will be mapped by t^{-} to the empty set; we can think of this as being "projected out" by t^{-}. Any symbol in Σ' hit only once by t will be mapped by t^{-} to exactly one symbol in Σ; we can think of this as a coordinate transform, or renaming, from Σ' back to Σ. Any symbol in Σ' hit more than once by t will be mapped to a set of two or more symbols in Σ; we can think of those symbols as needed for labelling simultaneous events in Σ that are mapped by t to a single event in Σ'. (Consider a translation which attaches two pins of an integrated circuit to the same printed circuit board wire. A single event on that wire can happen only if the corresponding events on each of those two pins can happen simultaneously.)

Since a set of symbols is also a pomset of symbols with the empty order, t^{-} is therefore a function mapping symbols to pomsets. In the temporal section we saw how such a function had a unique extension to a substitution, so let us form that extension, t^{-+}, to yield a mapping from pomsets to pomsets. We call this a projection because a major use of it is to project out some of the events of a pomset. However as noted just above it may also duplicate some events, so its function really goes beyond the normal notion of projection. We have not thought of a better word than projection to describe this action.

The action of a projection on a process P is the set of pomsets resulting from applying the projection to each of the pomsets of P, a process, so projection maps not only pomsets to pomsets but processes to processes.

A net N of processes P_i each with associated translation t_i embedding it in that net is the set N of those behaviors p such that, for all i, $t_i^{-+}(p) \in P_i$.

Such a net includes all its internal behaviors. Where a translation t defines an embedding in the net of its external connections, the external behavior of the net N is just $t^{-+}(N)$, the projection of N induced by t. The effect of this projection is to hide the internal behavior and provide the appropriate external port names for the externally visible portion of the behavior.

At this point the whole procedure probably looks quite mysterious. Let us dispel some of the mystery by showing how it works in the familiar context of composition of binary relations.

2.2. Example: Composition of Binary Relations

Let us begin with a familiar operation, the composition MN of two binary relations M and N on a set A. Using the representation of binary relations called for by our approach, we shall exhibit their composition as a projection of the set of behaviors projecting to behaviors of M and N.

The usual way this can be done for binary relations is to have three projections $\mu(a,b,c) = (a,b)$, $\nu(a,b,c) = (b,c)$, and $\kappa(a,b,c) = (a,c)$. Then the net behavior MN is $K = \{(a,b,c) | \mu(a,b,c) \in M \ \& \ \nu(a,b,c) \in N\}$, the set of all behaviors whose projections under μ and ν are in M and N respectively. The composition itself is then the projection $\kappa(K)$.

To fit this into our scheme we shall show how to represent μ,ν,κ as inverses of translations. We take $\Sigma = \{0,1\} \times A$ and $\Sigma' = \{0,1,2\} \times A$ for the domain and codomain of all three translations. We shall regard the pair (a,b) as the ordered multiset $(0,a) < (1,b)$, and the triple (a,b,c) as $(0,a) < (1,b) < (2,c)$. The appropriate translations are then $m(c,x) = (c,x)$, $n(c,x) = (c+1,x)$, and $k(c,x) = (-c,x) \pmod 3$. (That is, $m(0,x)=(0,x)$, $m(1,x)=(1,x)$, $n(0,x)=(1,x)$, $n(1,x)=(2,x)$, $k(0,x)=(0,x)$, $k(1,x)=(2,x)$.)

Now $m^{-}(0,x) = \{(0,x)\}$, $m^{-}(1) = \{(1,x)\}$, and $m^{-}(2) = \{\}$. Hence the behaviors mapped by m^{-+} to behaviors of M will be just those pomsets having an event (0,a), an event (1,b), and any number of events (2,c) for various values of c, with the order arbitrary except that $(0,a) < (1,b)$, and with $(a,b) \in M$. Of these, the ones mapped by m^{-+} to behaviors of N will have one event (1,b) and one event (2,c), where $(b,c) \in N$ and $(1,b) < (2,c)$, but with no constraints on events of the form (1,a). But this then limits the possible behaviors of the net to just $(0,a) < (1,b) < (2,c)$ where $(a,b) \in M$ and $(b,c) \in N$.

The most noteworthy difference from the standard construction $\{(a,b,c) | \mu(a,b,c) \in M \ \& \ \nu(a,b,c) \in N\}$ is that our construction does not assume at the outset that the result will consist only of triples. Instead the construction "discovers" this for itself.

It should now be clear how the projection k^{-+} discards the middle element of each triple to yield the desired composition.

Had we removed the order in defining binary relations, we would have hit a small snag. Net behaviors would still have only three events, (0,a), (1,b), (2,c), this time with (0,a) and (1,b) being incomparable and similarly for (1,b) and (2,c). However the ordering between (0,a) and (2,c) would be unconstrained. Hence for each triple (0,a), (1,b), (2,c) we would have *three* pomsets, one for each of the possible order relationships between (0,a) and (2,c): incomparable, $(0,a) < (2,c)$, and $(2,c) < (0,a)$. We avoided this variety by forcing $(0,a) < (1,b) < (2,c)$, which by transitivity of $<$ forces $(0,a) < (2,c)$.

An alternative and acceptable approach would have been to consider the set of all three pomsets representing all possible orderings of a pair to be an acceptable encoding of that pair. Then we would have pomsets coming in threes in M and N as well as in the projection of their intersection, and in 13's for the net behaviors (1 completely unordered, 6 with one element incomparable to the other two, 6 linearly ordered).

It is fair to ask what influence the choice of Σ' had on this example. What if it had been taken to be {0,1,2,3}×A instead of {0,1,2}×A? Would the fourth channel have confused matters? The answer is that the net behaviors would contain, in addition to the three events $(0,a)<(1,b)<(2,c)$, a cloud of events (3,d) related arbitrarily by < to these three and to each other. However the final projection would project out not only (1,b) but the whole of this cloud. Thus although {0,1,2,3}×A is a less elegant choice of Σ' than {0,1,2}×A, in the end it does not affect the outcome. The physical interpretation of all this is that the fourth channel is a loose wire whose behavior is unknown but irrelevant.

Note that nowhere in our notions of translation and projection do we make any assumptions about either the structure of the network or the type of constituent. This method of describing the behavior of a network of processes works equally well for any processes connected in any fashion.

3. Real-Time Theory - Quantitative Scheduling

3.1. Semirings

So far our notion of temporal relationship has been qualitative, namely whether one event precedes another. We would now like to extend the theory to deal with a richer notion of temporal relationship. One obvious notion is that of time as a number: by how many femtoseconds or teracenturies did one event precede another. Other notions of time, or even only marginally timelike relationships between events, may also suggest themselves.

Our thesis is that the appropriate algebraic structure for supplying the elements of a temporal relationship is the semiring. Semirings cater for parallel and serial composition of relationships in a suitably general way, providing one binary operation for each of these two concepts.

A semiring is an algebra (A,+,.,0) such that (A,+,0) is a monoid (+ is associative and has 0 as left and right identity) and . is associative and distributes over +, and has 0 as left and right annihilator ($a.0 = 0.a = 0$). The + operation caters for parallel composition of relationships and the . operation for their serial composition.

Up to now we have built into our theory the assumption of a particular semiring that we shall call the Boolean semiring. It consists of two values 0 and 1, with + interpreted as disjunction and . as conjunction. The idea is that every pair of events is related by either 0, meaning no temporal order, or 1, meaning that the first precedes the second. The operation + deals with variety of behavior: given several sources of information about whether one event precedes another, if any is a 1 then the upshot is a 1, otherwise it is 0. The operation . deals with course of events: if a precedes b *and* b precedes c then a precedes c. This is transitivity, a conjunctive concept.

A semiring that we shall call the *taxidriver's semiring* consists of the nonnegative reals with + interpreted as max, . as addition, and 0 as the number 0. The numbers can be thought of as the cost of getting from one event to another. When there are competing costs, the highest is always chosen. The cost of getting from a via b to c is the sum of the costs of getting from a to b and from b to c. The discrete version of this semiring substitutes the natural numbers for the nonnegative reals.

The *taxi passenger's semiring* is similar but includes infinity in the algebra, interprets . as min (the passenger prefers the cheapest route), and 0 as infinity (the least identity for min). To satisfy the semiring identity $a.0 = 0$ we need to take the product of numeric 0 with semiring 0 (here infinity) to be infinity. This semiring too has a discrete version, consisting of the natural numbers and infinity.

The taxidriver's semiring is useful in dealing with times between events that may not be reduced, e.g. for preventing events from interfering with each, meeting specifications for integrated circuits, etc. Dually the taxi passenger's semiring is good for times that may not be exceeded, e.g. for preventing timeouts, or establishing upper bounds on the running time of processes.

3.3. General Theory

We may now generalize our theory of processes from the Boolean semiring to arbitrary semirings. We begin with the idea that a binary relation from A to B is an $A \times B$ Boolean matrix. The interior and exterior operations of matrix multiplication are respectively conjunction and disjunction; that is, the dot product of two vectors is formed as the disjunction of pairwise conjunctions, and the matrix product MN of matrices M and N is the matrix of dot products whose ik-th entry is the dot product of the i-th row of M with the k-th column of N.

A binary relation M from A to A is a partial order when it is irreflexive (leading diagonal of M is all zeroes) and transitive ($M^2 \leq M$).

Now these interior and exterior operations associated with binary relations are respectively the . and + operations of the Boolean semiring. If we substitute for that semiring any other semiring, our definitions of irreflexive and transitive need not be changed since they are expressed in terms of semiring operations independently of any particular semiring such as the Boolean semiring.

Thus we may generalize the structure $[V,\Sigma,<,\mu]$ to $[V,\Sigma,S,M,\mu]$ where S is a semiring (no longer necessarily the Boolean semiring $\{0,1\}$) and M is an irreflexive transitive matrix over S. Such a structure is no longer a pomset; one might call it a measured multiset, where M provides the measurements between elements of the multiset. A process then becomes a set of measured multisets.

The notion of substitution does not generalize smoothly. Suppose we have events $e<f<g$ and we map f to $f1<f2$ and then extend that map to take $e<f<g$ to $e<f1<f2<g$. Now if $<$ is replaced by some measure of the delay between events, we have a problem relating $e<f<g$ and $e<f1<f2<g$. The problem is that whatever delay there is between f1 and f2 does not appear in $e<f<g$, where the event f itself is treated as having no delay of its own. Thus if there were at least a 2-microsecond delay between e and f and at least a 3-microsecond delay between f and g, then there would be a 5-microsecond delay between e and g. But this assumes that f itself involves no delay, which may contradict the expansion of f to $f1<f2$, in which there may be a nonzero delay.

This problem does not arise however for the special case of "length-preserving" homomorphisms, ones that map atoms only to sets, that is, pomsets with the empty order. This special case is all that is used in defining the notion of projection, which therefore remains unchanged when more general semirings are used. Thus the spatial theory does extend gracefully to general semirings.

However we do not see how to integrate this semiring approach with general substitutions. Some adjustment is needed to our temporal model to cater for this. We would be interested in hearing reasonable solutions.

3.4. Applications

A typical requirement in designing hardware is to achieve minimum delay times between certain events. For this the taxidriver's semiring is appropriate. Whenever there are two separate delay requirements for the same pair of events, the larger is taken, corresponding to semiring $+$ being numeric max. Whenever there must be a delay of at least m between events e1 and e2, and a delay of at least n between e2 and e3, there must be a delay of at least $m+n$ between e1 and e3, corresponding to semiring being numeric addition.

4. Appendix

Since strings (finite tomsets) may be defined very simply as a function from $\{0,1,2,\ldots,n-1\}$, it is natural to ask why pomsets should not have an equally simple definition. The basic obstacle, as we shall see, is that partial orders have nontrivial automorphisms.

In the case of a pomset that is a string of length n, V has n elements and the order $<$ is total. In this case the set $n = \{0,1,\ldots,n-1\}$ with its standard order may serve as a canonical representative of the isomorphism class of (V,<). That is, the function $\mu{:}n \rightarrow \Sigma$ is for our purposes equivalent to the tomset $[n,\Sigma,<,\mu]$. This establishes the connection between our isomorphism-based definition of tomsets and the simpler definition as a function.

More generally, any "womset" (well-ordered multiset) may be defined as a function from the appropriate ordinal to Σ. For the usual notion of an infinite string $s_0s_1s_2$... the appropriate ordinal is ω. An ordinal serves as a canonical representative of an isomorphism class of well ordered sets.

A well-ordered set (V,<) has no non-trivial automorphisms (isomorphisms from the set to itself). Another way to say this is that each element of a well-ordered set (V,<) uniquely determines an element of the corresponding ordinal. Hence an ordinal may be used as a representative of an isomorphism class of a well-ordered set without contributing any additional information not already present in the class.

In general however, ordered sets, whether ordered totally or partially, may have nontrivial automorphisms. For example the function $x+5$ is an automorphism of the structure (Z,<) of integers with their standard order. In general therefore the function $\mu{:}Z \rightarrow \Sigma$ and its composition with $x+5$ will be distinct functions mapping the integers to Σ, yet they will both be representatives of the isomorphism class $[Z,\Sigma,<,\mu]$ with no way of telling which is the canonical representative. Hence such functions overspecify the classes they represent.

Therefore, rather than attempt to base the theory of pomsets on canonical representatives of isomorphism classes we just base it on the classes themselves.

5. Bibliography

[BA81] Brock, J.D. and W.B. Ackerman, Scenarios: A Model of Non-Determinate Computation. In LNCS 107: Formalization of Programming Concepts, J. Diaz and I. Ramos, Eds., Springer-Verlag, New York, 1981, 252-259.

[deB72] de Bakker, J.W., and W.P. de Roever, A calculus for recursive program schemes, in **Automata, Languages and Programming**, (ed. Nivat), 167-196, North Holland, 1972.

[Gis84] Gischer, J., **Partial Orders and the Axiomatic Theory of Shuffle**, Ph.D. Thesis, Computer Science Dept., Stanford University, Dec. 1984.

[Kah74] Kahn, G., The Semantics of a Simple Language for Parallel Programming, IFIP 74, North-Holland, Amsterdam, 1974.

[KaM77] Kahn, G. and D.B. MacQueen, Coroutines and Networks of Parallel Processes, IFIP 77, 993-

998, North-Holland, Amsterdam, 1977.

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

[W84] Winskel, G., Categories of Models for Concurrency, this volume.