Linear process algebra

Vaughan Pratt

Stanford University, Stanford CA 94305-9045, USA
pratt@cs.stanford.edu

Abstract. A linear process is a system of events and states related by an inner product, on which are defined the behaviorally motivated operations of tensor product or orthocurrence, sum or concurrence, sequence, and choice. Linear process algebra or LPA is the theory of this framework. LPA resembles Girard's linear logic with the differences attributable to its focus on behavior instead of proof. As with MLL the multiplicative part can be construed via the Curry-Howard isomorphism as an enrichment of Boolean algebra. The additives cater for independent concurrency or parallel play. The traditional sequential operations of sequence and choice exploit process-specific state information catering for notions of transition and cancellation.[1]

Keywords: concurrency, event, state, duality, linear logic, Curry-Howard

1 Background

Computation itself, as distinct from its infrastructure (operating systems, programming languages, etc.) and applications (graphics, robotics, databases, etc.), has two main aspects, algorithmic and logical. The algorithmic aspect serves programmers by providing techniques for the design and analysis of programs. The logical aspect serves language designers, compiler writers, documentation writers, and program verification by proposing suitable concepts for the operations and constants of a language (abstract syntax), giving them meanings and names (semantics and concrete syntax), showing how to reason about them (logic), and studying their structure (abstract algebra).

Originally computation was performed on a single computer under the control of a central processing unit. Parallel and distributed computing emerged from the infrastructure with the advent of multiprocessors and networking, enhancing the applications at the expense of complicating both the algorithmic and formal aspects of computation. This paper focuses on the latter.

Formal methods divide broadly into logical and algebraic. On the logical side we find Amir Pnueli's temporal logic [45], which speaks of a single universal process from the point of view of a neutral observer. Pnueli has called this kind of specification *endogenous* to distinguish it from *exogenous* modal logics of programs such as dynamic logic [46, 32].

[1] More recent follow-up remarks and expansions on this paper may be found at http:boole.stanford.edu/pub/LPA.

In the algebraic approach process calculi provide one (but certainly not the only) framework. The most prominent of the early such calculi are Hoare's Communicating Sequential Processes (CSP) [33, 7], Milner's Calculus of Communicating Systems (CCS) [38], and Bergstra and Klop's Algebra of Communicating Processes (ACP) [5, 2, 6].

Each of these calculi imputes a certain nature to the process concept. Their differences raise the question of whether they are theories of essentially different entities or are merely refinements of a common core conception differing only in their emphases on secondary aspects.

A similar question arose millennia ago about the core of geometry, and was answered in turn by Euclidean geometry which postulated lines and circles as primitive concepts, cartesian geometry which expressed them as $ax + by + c = 0$ and $(x - x_0)^2 + (y - y_0)^2 = r^2$, and linear algebra as a source of more abstract spaces within which to conduct cartesian geometry and much else besides. Each in turn shed light on its predecessor. Yet all of them can be seen to be about Euler's notion of affine space as a common framework, with Euclidean geometry adding notions of metric and angle, linear algebra adding an arbitrary origin, and cartesian geometry doing both via the basis imputed by its coordinate frame.

Both space and computation can be understood either denotationally—what they are—or operationally—how to construct things. Euclid's account of Euclidean space was operational in that a good number of his postulates and propositions promised constructions in space rather than properties of space. Linear algebra on the other hand is founded on a denotational framework, with matrix inversion M^{-1} for example being defined denotationally as a solution in N to $MN = I$ before addressing questions of existence, uniqueness, and construction. The modern conception of Euclidean space is entirely denotational, making Euclid's operational treatment of the two-dimensional case of his eponymous space seem somehow nonmathematical to the modern reader, its trend-setting logical formulation notwithstanding.

Concurrency has likewise had both operational and denotational accounts. Among the former, perhaps the best known are Petri nets [42] and Plotkin's Structured Operational Semantics (SOS) [44]. The considerable popularity of both can be taken to mean that concurrency is best treated operationally, or that its denotational treatment is problematic, or both.

It cannot however be taken to mean that no one has tried. The denotational semantics of concurrency can be considered to have begun with the idea of sequential processes as sets of computation traces, by analogy with formal languages as sets of strings, with concurrency introduced via the shuffle operation [15, 68]. However this semantics captures neither branching (timing of nondeterminism) nor independence (determinism of nontiming), both of which can be seen as a deficiency not of the operations but of sets of traces themselves. In that model all decisions in a computation are reduced to the choice of a single trace, ignoring both the order in which the decisions were made and their timing relative to other events. Independence of a and b on the other hand is expressed

as the choice $ab+ba$, introducing both order and choice when neither are relevant to independence.

Branching time was first formalized denotationally by Milner's synchronization trees [38]. The implied distinction was formalized by Park in terms of a relation of bisimilarity [41] expressing lock-step equivalence. Bisimilarity is a more refined equivalence of processes than mere equality of two sets of traces, which is too coarse to distinguish $a(b+c)$ from $ab+ac$. Subsequently a hierarchy of congruences intermediate between trace equivalence and bisimilarity emerged: Figure 1 of [73] partially orders the 11 semantics by their relative positions in the linear time/branching time spectrum.

Independence, or "true concurrency," was formalized denotationally early on by Greif [28] in terms of events partially ordered by time as a semantics for Hewitt's actor framework, and later by Mazurkiewicz via traces quotiented by an equivalence relation of independence on the alphabet Σ extended to a congruence on the free monoid Σ^* [37]. Yet later Grabowski [27] and Pratt [48, 17] further abstracted Mazurkiewicz traces with a notion of partially ordered multiset or pomset. In the latter two models choice was expressed by defining a process to be a set of traces or pomsets.

All these semantics of independence were explicitly or implicitly based on the notion of multiset over an alphabet of symbols as a set labeled with symbols. Identifying unlabeled strings with ordinals, a string can be defined as a labeled ordinal, with the length of a string being the underlying ordinal. Pomsets generalize labeled ordinals to labeled posets.

Nielsen, Plotkin and Winskel's notion of event structure $(A, \leq, \#)$ [39, 76–78] formalizes independence as for pomsets, namely with a partial order $\leq$, with two differences. First it conflates the two-level set-of-pomsets approach to the one level used in treatments of branching time by expressing choice in terms of an irreflexive symmetric conflict relation $a\#b$ on events satisfying $a\#b \wedge b \leq c \Rightarrow a\#c$. Conflict creates the choice of which of the conflicting events *not* to perform, and is the only kind of choice expressible by event structures. Second the unlabeled event structures are taken to be the primary object of study, which is analogous to studying strings over a one-letter alphabet, i.e. ordinals, instead of general strings. Unlabeled event structures are already interesting enough in their own right without labels.

But while unlabeled event structures can easily represent $a||b$, a and b acting independently, it is unclear how they can distinguish it from $ab + ba$, a and b acting in either order. Gaifman and Pratt [14] addressed this distinction with a notion of prosset $(E, \leq, <)$ with a reflexive weak and irreflexive strong partial order satisfying $a < b \;\Rightarrow\; a \leq b$, with a process defined as a set of prossets as for traces and pomsets. Both relations order events by time, with the difference being that only $a \leq b$ permits the simultaneous occurrence of a and b. Independence can be expressed with the requirement that if two prossets of a process differ only in that $a \leq b$ holds in one and $b \leq a$ in the other then the process also contains a prosset differing from those two in omitting both constraints, thereby expressing their independence. If however $a < b$ and $b < a$ hold in the respective prossets

then this is understood to mean $ab + ba$, the mutually exclusive execution of a and b in either order.

A quite different way of drawing this distinction is with the notion of a ***higher dimensional automaton*** [54] or HDA based on combinatorial geometry. This approach abandons events and reverts to states. However the conception of state is an n-dimensional one in which a process consists of combinatorial cells whose dimension gives the number of ongoing events in that cell. While the 0-dimensional cells correspond to the ordinary notion of state, the 1-dimensional cells look more like transitions, while the higher-dimensional cells express concurrency and have no counterpart in ordinary automata theory. This approach amounts to an automata-theoretic formalization of Papadimitriou's geometric treatment of concurrency control [40], as well as ST-bisimulation [18] and deterministic asynchronous automata [71]. Higher dimensional automata have since been studied by many authors [74, 26, 25, 19, 29, 21, 20, 22, 23, 70, 10, 72, 70, 12, 63], leading Eric Goubault to found a series of conferences on the geometry and topology of computation, GETCO, and a special issue of MSCS [24].

My own perspective on concurrency has evolved gradually over the past three decades [47–61, 31] by way of pomsets, prossets, event structures, higher dimensional automata, and finally Chu spaces. The last were so interesting in their own right as to distract me from process algebra in order to study their applications to mathematics for a few years before returning to their process algebra applications [63–65, 67]. [67] in particular made the observation that the usual two values 0 and 1 of Chu spaces, denoting event states of *ready* (not yet started) and *done*, could be extended with either or both of the intermediate value $\lrcorner$ of *transition* and the alternative value $\times$ of *cancelled*. These permit respectively higher-dimensional automata and what we now call cancellation automata to be represented as Chu spaces. In particular we show how van Glabbeek's example of a higher dimensional automaton not expressible as a Petri net [75, Fig.11] can be expressed instead as a pure cancellation automaton, one with no higher dimensional cells.

This paper serves the dual purposes of an up-to-date tutorial on the representation of processes as Chu spaces over $K = \{0, 1, \lrcorner, \times\}$, which we propose to call linear processes, and its relationship to these earlier models, along with an update on recent work. The Chu representation of concurrent processes is not as well known in concurrency circles as I feel it should be, despite having been in the literature for two decades [8, 9, 31, 30], justifying the tutorial part. The emphasis here is more on rationale, intuition, definitions, and perspective and less on a formal theorem-and-proof development of the theory; for more in-depth technical details see [62] for the relevant theory of Chu spaces and [67] for more on process algebra based on transition and cancellation.

Algebra traditionally starts with operations and laws forming a theory, e.g. the theory of commutative rings or of fields, leaving the values to emerge as the elements of models of the theory. Useful algebra however is often motivated by its intended or primary model, e.g. the ring of integers or the field of rationals, and for this reason it is preferable to begin with the intended values as motivation

for the operations, and to let both drive the laws rather than vice versa. This is the approach we follow here in starting with Chu spaces as the values, then explicitly defining operations on them, and lastly considering what form the laws should take and what they are.

2 Chu spaces as generalized linear algebra

The previous section noted the diverse notions of process, some based on events, such as pomsets and event structures, others on states, such as information systems, synchronization trees, and higher-dimensional automata. A common core compatible with both kinds needs somehow to cater for and reconcile both events and states. To unify such diversity would appear to call for a complex framework; in particular one would not expect the simplest conceivable framework meeting the brief desiderata of the previous sentence to be up to the job on its own. Nevertheless that is what we propose and study here.

We take for our core notion of (unlabeled) process simply a set of events, a set of states, and a binary relation between them, and nothing else. We write these as respectively A, X, and $r : A \times X \to K$, where K is a set (such as $\{0, 1\}$) making r a K-valued binary relation. A map $h : P \to Q$ transforming process $P = (A, r, X)$ into process $Q = (B, s, Y)$ is defined as an ***adjoint*** pair $(\hat{h}, \check{h})$ of functions $\hat{h} : A \to B$, $\check{h} : Y \to X$, meaning one that satisfies $s(\hat{h}(a), y) = r(a, \check{h}(y))$ for all $a \in A$ and $y \in Y$. The maps from P to Q are all and only those pairs of functions satisfying these conditions.

In general such a structure is called a Chu space over K, with the category of such and their maps being denoted by $\mathbf{Chu}(\mathbf{Set}, K)$. Chu spaces can be specialized to particular applications by a suitable choice of K. Before developing the process concept further, by way of background we first consider other areas whose objects can be organized as Chu spaces.

The paradigmatic example is linear algebra over a given field k. This is more than just an analogy: as pointed out by Y. Lafont [34, 35] it is the special case $K = |k|$, the set of elements of the field. Each vector space V is represented as the Chu space $\tilde{V} = (|V|, r, |V^*|)$ whose points are the vectors of V, whose states are its functionals or dual points, namely the linear transformations comprising the vectors of the vector space $V^* = k^V$ (treating k as a one-dimensional vector space over k) and $r : V \times V^* \to K$ is the inner product for V, satisfying $r(v, g) = g(v)$ for each vector $v \in V$ and functional $g \in V^*$. In mathematics inner product $r(v, g)$ is customarily written (v, g), in physics as $\langle g|v\rangle$ (bra $\langle g|$ and ket $|v\rangle$). It can be shown that each linear transformation $h : U \to V$ is represented uniquely as the pair $(h, \lambda g.gh) : \tilde{U} \to \tilde{V}$ constituting a morphism of the Chu spaces representing respectively U and V. That is, the category $\mathbf{Vct}_k$ of vector spaces over k and their linear transformations fully embeds in the category $\mathbf{Chu}(\mathbf{Set}, K)$, creating a bijection between homsets $\mathbf{Vct}_k(U, V)$ and $\mathbf{Chu}(\mathbf{Set}, K)(\tilde{U}, \tilde{V})$.

As another instance, Y. Lafont [34, 35] has further pointed out that the points and open sets of a topological space S can be treated by analogy with respectively

the vectors and functionals of a vector space, with r taken to be the two-valued relation of membership of a point in an open set. The counterpart of the one-dimensional space is the Sierpinski space with two points and three open sets, while each continuous function $h : S \to T$ is represented uniquely as the pair $(h, h^{-1}) : \tilde{S} \to \tilde{T}$ where h^{-1} is the inverse image function associated to h. We give a great many more such examples elsewhere [62, 66].

The term "linear" is also motivated by Girard's linear logic, LL, a substructural logic applicable to sequent-based proof theory. The "multiplicatives" of LL, namely perp $P^\perp$, tensor $P \otimes Q$, its De Morgan dual $P \parr Q = (P^\perp \otimes Q^\perp)^\perp$ and the multiplicative units 1 and $\perp$, constituting multiplicative linear logic MLL, make essentially the same connection with Boolean algebra via the Curry-Howard isomorphism as do their counterparts for linear process algebra. Furthermore the extension of MLL to MALL with the "additives" $P \oplus Q$ and $P\&Q$ of LL, as respectively direct sum (coproduct) and direct product as notated by Girard (we will write $P\&Q$ as the more customary $P \times Q$), also find application in both LL and LPA.

3 Linear processes

In this section we define processes as structures. Just as the Chu representation of linear algebra over a field k took $K = |k|$, and of topological spaces, $K = \{0, 1\}$, so do we define linear processes Chu spaces over the set $K = \{0, \lrcorner, 1, \times\}$. Organizing processes as Chu spaces makes events and states equally primary, paralleling Hamilton's reorganization of Newton-Langrange mechanics in 1837 by putting position and momentum on the same level.[2]

The elements of K constitute the four possible states an event can be in, namely *ready* 0 (i.e. not yet started), *transition* $\lrcorner$, *done* 1, and *cancelled* $\times$.[3] The intuitive meaning of transition is as in "Shh, the event is in progress", while that of cancellation is as in "Sorry but the event has been cancelled."

Thinking of these four as *local* states, we interpret the value of $r(a, x)$ as the local state of event a in state x. The latter is a *global* state or state vector in the sense that it can be interpreted extensionally via the function $X \to (A \to K)$ mapping each x in X to the function $\lambda a.r(a, x) : A \to K$, which we call the ***extension*** of x. Dually the extension of each event a is the function $\lambda x.r(a, x) : X \to K$. We can refer to a row of a matrix either intensionally by its index a or extensionally by the row itself, and dually for columns.

Following Barr's terminology [4] we call a Chu space ***extensional*** when if two columns have the same extension then they have the same intension; that is, there are no repeated extensional columns. Topological spaces can be understood as extensional Chu spaces by identifying their open sets with the extensions of states. Dually a Chu space is ***separated*** when two rows with the same extension

[2] Hamiltonian mechanics took 90 years to catch on in physics; hopefully event-state symmetry will not take that long!

[3] This is sufficient for a theory of ideal processes. In practice processes need to be abortable, which a fifth state, *aborted*, could address. We leave this to future work.

have the same intension; this corresponds to the notion of a T_0 topological space. A ***biextensional*** Chu space is one that is both extensional and separated, all rows and columns distinct. The above representation of vector spaces as Chu spaces is biextensional.

Viewed as event structures, Chu spaces are unlabeled. As with the theory of event structures we draw the distinction between events and actions. An event constitutes an instance of an action; it can happen only once, whereas an action can happen many times.

We take events to be more basic than actions on the ground that the order in which things happen in a process is an ordering of events, not of actions. This is not to say that actions are unimportant but that the structure of events independently of their labels is already of considerable interest in its own right.

A labeled process over an alphabet Λ of actions is a pair (P, λ) where $P = (A, r, X)$ is an unlabeled process and $\lambda : A \to \Lambda$ labels each event a with the action $\lambda(a)$ of which a is an instance.

4 Processes as transformable entities

The preceding section defined a process over a set K as a structure (A, r, X) where $r : A \times X \to K$. In this section we instead define processes analogously to how Zermelo-Fraenkel set theory, ZF, defines sets. Whereas all individuals of a model of ZF are sets, those of our theory are of two sorts, processes $P, Q, \ldots$ forming a class $\mathcal{P}$ and process transformations or maps $h : P \to Q$ forming a class $\mathcal{M}$. And whereas the language of ZF consists of a single binary relation $\in$ of membership on a homogeneous domain of sets, ours consists of one binary operation, three unary operations, and two constants. Besides permitting a more axiomatic definition, this perspective distinguishes the events of a process P from its states by exhibiting them as maps respectively to and from P[4] and makes the event-state schizophrenia of the elements of K more explicit by exhibiting them as states of the generic one-event process $\mathbf{1}$, which of course they are, as well as events of K, which of course they are.

The binary and unary operations are just those for a category, namely composition $m : \mathcal{M}^2 \to M$, source and target $s, t : \mathcal{M} \to \mathcal{P}$ and identity $i : \mathcal{P} \to \mathcal{M}$. Moreover they satisfy the usual laws, namely $s(i(P)) = t(i(P)) = P$, $m(k, h)$ or kh for short is defined just when $s(k) = t(h)$, $s(kh) = s(h)$, $t(kh) = t(k)$, $(kh)j = k(hj)$ (associativity), and $i(P)$ or 1_P for short is both a left and right identity for composition.

As usual a map $h : P \to Q$ is an isomorphism when it has an inverse $h^{-1} : Q \to P$ in the sense that both $h^{-1}h$ and hh^{-1} are identity maps, namely 1_P and

[4] Early on [31, 30] we took events and states to be respectively columns and rows for consistency with linear algebra, which traditionally identifies the points and functionals of a vector space with respectively columns (maps to the space) and rows (maps from it), but found this unnatural in thinking about processes and subsequently reversed the correspondence.

1_Q respectively. Processes are isomorphic when there is an isomorphism between them.

The only process-specific part of the language consists of two constant processes **1** and K, satisfying four axioms, A1-A4, the first two of which are elementary (first order).

Definition 1. *A process P is **rigid** when the only map $P \to P$ is the identity 1_P.*

Axiom A1 *The processes **1** and K are rigid.*

Definition 2. *An **event** is a map from **1**, while a **state** is a map to K.*

Events to P and states from P are considered to belong to P, as in "event (state) of P." Variables ranging over events and states are written $a, b, \ldots$ and $x, y, \ldots$ respectively. An ***ordinary*** map is one that is neither an event nor a state.

For convenience we denote by A_P and X_P the sets of respectively events and states of P. The following two propositions are routine.

Proposition 1. **1** *has one event, while K has one state.*

Proposition 2. *The events of K are precisely the states of **1**.* (So they have a dual identity as events and states.)

Definition 3. *The state of an event a of P in state x of P is the state xa of **1*** (and an event of K).

We distinguish states of processes from states of events by considering them respectively global and local states. By proposition 2 local states are events of K. Composition of states with events of a process is the axiomatic counterpart of r in the structural definition (A, r, X).

Given any map $h : P \to Q$ we define its ***left action*** $\hat{h}$ to be the function $\lambda a.ha$ mapping each event a of P to the event ha of Q, and its ***right action*** $\check{h}$ to be the function $\lambda y.yh$ mapping each state y of Q to the state yh of P. The types of these actions are respectively $\hat{h} : A_P \to A_Q$ and $\check{h} : X_Q \to X_P$.

Proposition 3. *For any map $h : P \to Q$, its left and right actions satisfy $x\hat{h}(a) = \check{h}(x)a$ for all $a \in A_P$ and $x \in X_Q$.*

Functions $A_P \to A_Q$ and $X_Q \to X_P$ satisfying this condition are said to be an ***adjoint pair*** of functions between P and Q.

Proof. The two sides expand to respectively $x(ha)$ and $(xh)a$, which are equal by associativity.

Definition 4. *Two maps $h, k : P \to Q$ with the same left and right actions are called **equivalent**.*

Axiom A2 (Extensionality) *Equivalent maps are equal.*

Thus far all definitions and propositions have been equivalent to elementary or first order ones; the second order constructs are inessential and can be translated into first order ones. By introducing two process operations $P \otimes Q$ and $P \multimap Q$ we could continue the ZF analogy with further elementary axioms. However a hybrid approach involving counterfactuals (whose explication in the absence of a self-contained foundation requires falling back on ZF) allows this transformation-based axiomatization to be completed much faster using just two more axioms, neither one elementary in the sense of first order logic but both elementary in the sense of being intuitively clear. We leave to another occasion the development of a purely first order theory of processes replacing A3 and A4.

The purpose of Axiom A3 is to ensure that all possible maps are present between the processes of the model. We state it as a counterfactual which contemplates the possibility of additional ordinary maps not already in whatever the model happens to be, and denies this possibility in a positive way. It is counterfactual in that it refers to a map that does not yet exist, a notion absent from first order logic. "Ordinary" is essential here since adding new events or states to a process would make it a different process.

Axiom A3 (No new maps) *Any new ordinary map is equivalent to an old one.*

A3 together with A2 implies that $\mathcal{M}$ has no proper extension. That is, $\mathcal{M}$ is maximal and the contemplated counterfactual is impossible.

The following establishes uniqueness up to isomorphism of such a maximal $\mathcal{M}$, namely the set of all pairs of actions.

Proposition 4. (Density) *For any two processes P, Q for which P is not $\mathbf{1}$ and Q is not K, every adjoint pair of functions between P and Q is the pair of actions of some map $g : P \to Q$.*

This notion of density is due to Gabriel and Ulmer [13] by analogy with density of the rationals in any extension thereof, namely any Archimedean field. Here it means, informally speaking, that all possible maps between two processes are present.

Proof. Suppose some adjoint pair between P and Q is realized by no map. Then a map $g : P \to Q$ with this pair as its actions can be adjoined, along with all required composites $hgf : P' \to Q'$ not already present where $f : P' \to P$ and $h : Q \to Q'$. No other maps than these need be adjoined (there is no chain reaction) since by associativity any map of the form $h'(hgf)f'$ is of the form $(h'h)g(ff')$ and hence is already adjoined. All compositions are uniquely determined by the actions of g and those of the f's and h's.

Since the possible pairs of actions between two processes form a set this maximum is well-defined.

Axiom A4 similarly depends on the counterfactual notion of extending $\mathcal{P}$ (whatever processes exist in the model at hand) with additional processes, subject to maintaining the preceding axioms. When a process is added, its events

and states and how they compose as maps of A_K are considered part of the addition, and these are thereafter left undisturbed by any further extensions of either $\mathcal{M}$ or $\mathcal{P}$. As part of this addition, all new morphisms needed to satisfy A3 are added. This is a weak denial of the impossibility of new processes in that it takes "new" to mean new up to isomorphism.

Axiom A4 (No new processes) *Any new process is isomorphic to an old one.*

As defined earlier P and Q are isomorphic when there exists an isomorphism $h : P \to Q$. We need A3 for this, without which processes that an external observer would judge isomorphic might nevertheless lack the requisite isomorphism witnessing their isomorphism.

In the following "equivalent" means in the sense of equivalent categories.

Proposition 5. (Completeness) *All models of these axioms are equivalent.*

Proof. For any two sets A, X and any function $r : A \times X \to A_K$ there exists a process P and bijections $\alpha : A \to A_P$, and $\omega : X \to X_P$, such that $r(a, x) = \omega(x)\alpha(a)$ (composition). It follows that every model of these axioms is equivalent to the category of Chu spaces over A_K, as defined in the preceding section, when its morphisms from (A, r, X) to (B, s, Y) are taken to be all adjoint pairs (f, g) of maps $f : A \to B$ and $g : Y \to X$, that is, maps satisfying $s(f(a), y) = r(a, g(y))$ for all $a \in A$ and $y \in Y$.

5 Linear Process Algebra

We turn now from linear process semantics, what processes are, to process algebra, how to name them compositionally and reason about them.

The operations of Linear Process Algebra, LPA, are orthocurrence $P \otimes Q$, concurrence $P||Q$, sequence $P;Q$ or just PQ, and choice $P+Q$. In defining operations we assume that all processes are extensional (states with equal extensions are equal, i.e. no repeated columns). The extensional collapse of a process P is the result of making P extensional by identifying states of P with equal extensions. Since some of the operations below do not necessarily preserve extensionality we enforce it by automatically collapsing extensionally the result of every operation.

Dual $P^\perp$. Duality makes the connection between processes viewed as a schedule of events and as an automaton comprised of states. Given $P = (A, r, X)$, $(A, r, X)^\perp$ is defined as (X, r', A) where $r'(x, a) = r(a, x)$, that is, transpose. Transpose also applies to process maps, with the transpose of $h : P \to Q$ being $h^\perp : Q^\perp \to P^\perp$, where h and $h^\perp$ have the same actions in the sense of Section 4 but with left and right simply interchanged.

Duality has no operational interpretation, but rather serves to convert an event-oriented process, meaning one whose events and states transform respectively covariantly and contravariantly, into a state-oriented process for which events and states transform respectively contravariantly and covariantly.

Orthocurrence $P \otimes Q$. This is the fundamental interaction operator of linear process algebra. Although it appeared in our work almost as early as did pomsets

[51, 53, 11, 31], it is at the heart of both Barr's category-theoretic notion of a $*$-autonomous category [3], and Girard's proof-theoretic notion of linear logic developed independently several years after Barr. [16].

Given two processes $P = (A, r, X)$ and $Q = (B, s, Y)$, their ***orthocurrence*** $P \otimes Q$ is the process $(A \times B, t, Z)$. Here Z is the set of all functions $z : A \times B \to K$ such that
(i) for each $b \in B$, $\lambda a.z(a, b)$ is a state of P, and
(ii) for each $a \in A$, $\lambda b.z(a, b)$ is a state of Q,
while $t : (A \times B) \times Z \to K$ is defined as $t((a, b), z) = z(a, b)$. Each z may be thought of as an $A \times B$ crossword whose rows are states of Q (the "across" dictionary) and whose columns are the states of P (the "down" dictionary).

Besides interaction, othocurrence $P \otimes Q$ can also be understood as dual to the *observation* $P \multimap Q^\perp$ of states of process Q (events of $Q^\perp$) from vantage points of process P, or by symmetry the observation $Q \multimap P^\perp$, giving a sense in which observation is as symmetric as orthocurrence [64].

Concurrence $P||Q$. This is the independent or noninteracting parallel behavior of P and Q. Given two processes $P = (A, r, X)$ and $Q = (B, s, Y)$, their ***concurrence*** $P||Q$ is the process $(A + B, t, X \times Y)$. Here $A + B$ denotes the disjoint or marked union of A and B, where the marking indicates for each event of $A + B$ whether it came from A or B (e.g. by defining $A + B = A \times \{1\} \cup B \times \{2\}$) while $t(a, (x, y)) = r(a, x)$ and $t(b, (x, y)) = s(b, y)$ where a and b denote events of $A + B$ coming from respectively A and B.

Initial states. The initial local or event state is 0 or *ready*. The initial global state is the all-zero state vector: all events are in their ready state 0.

Final states. The two final local states are 1 and $\times$. A state is final just when all its events are in a final event state.

This definition is facilitated by the cancel state. Without it the event structure literature has struggled with the concept of final state, termination, and sequence. One might suppose that a final state could be defined simply as one with no successor state. This however fails to represent a process that may choose nondeterministically to halt or continue. For example the process $a + \emptyset$ that chooses to do either a or nothing has a state in which a is ready, but that state cannot be final because it has 1 as a successor state. While solutions have been proposed, none are as simple as merely allowing a to enter the cancelled state to indicate termination.

Cancelled states. A state is cancelled when all its events are cancelled. The typical application is to the definition of choice $P + Q$, whose first step is to cancel one of P or Q simultaneously with starting the other.

Sequence PQ. Given two processes $P = (A, r, X)$ and $Q = (B, s, Y)$, their ***sequence*** PQ is defined as for $P||Q$ but with only those states (x, y) of $X \times Y$ for which either y is initial or x is final. When y is initial we consider P to be happening when in state (x, y), while when x is final we consider Q to be happening.

If Q has no initial state (usually not the case in practice) then P cannot run and is understood to be in one of its final states. If P has no final state then Q

cannot run. If P has n final states x_i then PQ has n copies (x_i, y) of each state y of Q.

Choice $P + Q$. Given two processes $P = (A, r, X)$ and $Q = (B, s, Y)$ with both A and B nonempty, their ***choice*** $P + Q$ is the process $(A + B, t, \{*\} + X' + Y')$ where X' and Y' are X and Y less their respective initial states, $t(a, *) = t(b, *) = 0$ (making $*$ the initial state of $P + Q$), $t(a, x) = r(a, x)$, $t(b, y) = s(b, y)$, and $t(a, y)$ and $t(b, x)$ are the cancelled state ($\times$ if available, otherwise 0). Operationally, $P + Q$ begins in the initial state (all events of both P and Q ready) and then simultaneously cancels all the events of one of P or Q and begins the other.

Constants $\emptyset$ *and* $\mathbf{1}$. The process $\emptyset$ is $(0, !, 1)$, consisting of no events and one state. It is the unit for both concurrence and sequence, that is, $a||\emptyset = a\emptyset = \emptyset a = a$. However $\emptyset$ is not the unit for choice because $P + \emptyset$ creates a state in which all events of P are cancelled.

The process $\mathbf{1}$ is as defined in Section 4. As a Chu space it can be taken to be $(\{*\}, s, K)$ where $s(*, k) = k$. Up to isomorphism $\mathbf{1}$ is the unit for orthocurrence: when (A, r, X) "flows through" $\mathbf{1}$, $A \times \{*\}$ is isomorphic to A and the states of $A \otimes \mathbf{1}$ are in bijection with those of A.

6 The Curry-Howard correspondence with Boolean algebra

The core of linear process algebra is orthocurrence $P \otimes Q$ as interacting concurrency, which we distinguish from concurrence $P||Q$ as noninteracting concurrency, parallel play as kindergarten teachers call it. Orthocurrence together with the involution $P^\perp$ share essential features with Boolean conjunction $x \wedge y$ and complement $\neg y$.

The connection is made via the Curry-Howard correspondence[5] between logical values and mathematical objects. In this case the logical values may be taken to be 0 and 1 while the objects are linear processes. Conjunction and negation of the former correspond respectively to orthocurrence and dual of the latter.

Many of the logical laws involving terms built with these two logical operations have their counterpart as natural isomorphisms between functors built from these two process operations. In particular associativity and commutativity of conjunction carry over, the latter as a symmetry $P \otimes Q \cong Q \otimes P$, but idempotence has no counterpart. Double negation $\neg\neg x = x$ does carry over, with $P^{\perp\perp}$ being not only isomorphic but equal to P. And just as $x \vee y$ is definable by De Morgan's law as $\neg(\neg x \wedge \neg y)$, so is Girard's *par* operation $P \parr Q$ definable as $(P^\perp \otimes Q^\perp)^\perp$. The meaning of $P^\perp$ in LPA is P viewed as an automaton consisting of (covariantly transforming) states instead of as a schedule consisting of events, while the meaning of $P \parr Q$ is just the automaton counterpart of orthocurrence.

[5] This is commonly called the Curry-Howard isomorphism but since it is not technically an isomorphism we prefer to call it a correspondence.

7 Example terms

We now consider the behavior of the operations on atomic processes, showing how they compose in certain cases to produce larger terms. Table 1 lists a dozen terms and the linear processes they denote.

a	a 0⌟1	$a+\emptyset$	a 0⌟1×
ab	a 0⌟111 b 000⌟1	$a+b$	a 0⌟1×× b 0××⌟1
$a\|\|b$	a 0⌟10⌟10⌟1 b 000⌟⌟⌟111	$ab+ba$	a 0⌟1010⌟1 b 000⌟⌟111
$a(b+c)$	a 0⌟11111 b 000⌟1×× c 000××⌟1	$ab+ac$	a 0⌟111⌟111 b 000⌟1××× c 0×××00⌟1
$(b+c)a$	a 000⌟100⌟1 b 0⌟111×××× c 0××××⌟111	$ba+ca$	a 000⌟100⌟1 b 0⌟111×××× c 0××××⌟111
$ab\otimes cd$	ac 0⌟11111111111 ad 00000⌟⌟⌟11111 bc 000⌟10⌟10⌟111 bd 00000000000⌟1	$(a+b)\otimes(c+d)$	ac 0⌟⌟11×××× ad 0××××⌟⌟11 bc 0××××⌟1⌟1 bd 0⌟1⌟1××××

Table 1. Example LPA expressions

Cancellation distinguishes $a+\emptyset$ from a by adjoining a fourth state to a allowing it to be cancelled. Hence a has only one final state while $a+\emptyset$ has two.

Sequence ab and choice $a+b$ each have five states. These are performed sequentially for ab, while in $a+b$ they form two branches each with three states, the initial state of which is common to both branches.

Processes $a||b$ and $ab+ba$ are almost identical, the one difference being that ⌟⌟ is a state of the former but not of the latter. Thus $a||b$ has a two-dimensional state while $ab+ba$ does not, but otherwise has the same states of dimension 0 and 1 as $a||b$.

Process $a(b+c)$ does not cancel either b or c until after a is done. At that point the state is 100, which can be viewed as the initial state of $b+c$. One of b or c is then cancelled while simultaneously the other gets under way in the transition state, and then is done, for a total of 7 states. Process $ab+ac$ on the other hand cancels one of b or c as soon as a enters its transition state, after which it behaves like an ordinary sequence. There are thus two branches with 5 states each, but with the initial state shared so that there are only 9 rather than 10 states. Hence $a(b+c)$ and $ab+ac$ are distinct.

Processes $(b+c)a$ and $ba+ca$ however are the same: both begin by cancelling one of b or c while beginning the other. This gives them both two branches, branching at the root with each branch having 5 states as for $ab+ac$.

The orthocurrence $ab \otimes cd$ involves no cancellation. However there is one instance of concurrency, namely ad with bc, where ac is done and bd is ready.

One example of this situation is a train schedule involving two sequential trains a then b passing through two stations c then d. The pair ac is the event of train a arriving at station c. When ac is in transition the train is standing at the station; passing to done corresponds to the train having left the station. The only opportunity for concurrency here is when the first train is standing at the second station while the second train is at the first station.

Another example of $ab \otimes cd$ is Allen's 13 configurations of a pair of intervals sliding past each other [1], with a and b denoting the endpoints of one interval and c and d those of the other. Allen's 13 configurations correspond in the evident way to the 13 states shown in Table 1. In particular aligning the two intervals at both ends corresponds to the one two-dimensional state $1 \llcorner \llcorner 0$. Rodriguez and Anger [69] have studied extensions of Allen's configurations to handle richer notions of time such as relativistic time, accomplished by suitably enlarging the local event set K. With one extension they characterize as branching time the 13 states extend to 29, with their relativistic one it extends to 82 states, see [67] for further details.

8 Laws

As indicated in Section 6, the Curry-Howard counterpart of many (but not all) of the equations of Boolean algebra are natural isomorphisms between terms involving orthocurrence (*tensor*), duality (*perp*), and dual orthocurrency (*par*). There are in addition laws governing concurrence, sequence, and choice, but already just those governing the first three mentioned operations raise interesting questions.

Perhaps the most important question is, what is the Curry-Howard counterpart of logical truth in the LPA setting? One might suppose that the same question would arise for linear logic and therefore serve as a guide. However Girard has taken the position that truth is merely that which proof establishes, as opposed to being definable independently. Since LPA deals with processes rather than proofs, this view would appear to make no sense for LPA.

Yet there is one point of commonality: the classical notion of truth does arguably not make sense for LPA. The usual conception of truth for a proposition, at least for Boolean logic, entails a binary decision. Such a decision is necessarily centralized, with data from sensors being collected at one point to arrive at a binary determination.

The Curry-Howard counterpart of a proposition is a process. Processes need not be local as they can be distributed over an arbitrarily large area or volume. This is not consistent with logical decision-making as a centralized notion as decisions must be made locally if they are to be timely. This in turn implies that decision itself should be a concurrent notion.

We therefore propose to dispense with the traditional notion of propositions as special entities that are true or false and simply define a proposition to be a

process. We take reasoning to be an extension of behavior that introduces events of a propositional or judgmental nature above and beyond the ordinary events of behavior. There can be many of these running concurrently in a distributed fashion, with no requirement of global coordination at any point.

The benefit of this approach is that reasoning can be absorbed into the framework without making special provision for it as a distinct notion. Reasoning becomes simply a kind of parallel behavior.

We take reasoning to be the transformation of one process into another, in its most general sense. Transformation acts on terms, and terms are realized by functors. When all functors are covariant a transformation can be defined simply as a natural transformation. In the presence of contravariant functors, logic of this kind is more delicate. Dinatural transformations have been proposed for this [36]; however Chapter 6 of [62] points out difficulties with dinaturality that are overcome using binary logical transformations [43].

When the processes being so transformed represent behavioral rather than propositional information, transformations can be regarded as serving simply to establish program equivalence. Processes incorporating distributed propositional information may convey more nuanced decision-oriented information, much in the manner of couriers carrying information between locations and multiple local headquarters planning for their immediate neighborhood. In short, very much how reasoning is carried out in the real world, namely by many individuals, organizations, and machines, in a distributed fashion.

This conception of distributed reasoning is not at all well worked out here, and we hope to sharpen these ideas more satisfactorily in due course.

9 Beyond Petri nets

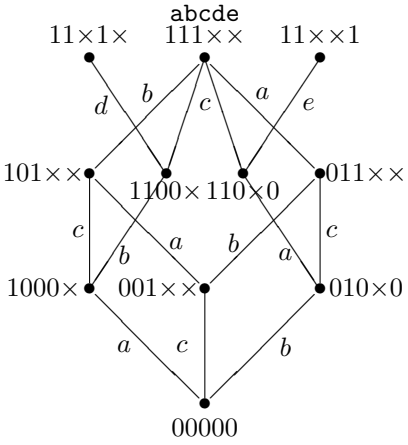
In [75, Fig.11] van Glabbeek gives an example of a higher-dimensional automaton expressing a process that is not expressible as a Petri net. The following story relocates van Glabbeek's process to a more rural setting than the race across the conference podium that enlivened Rob's presentation of this example at EXPRESS'04.

Alphonse and Gaston are walking abreast along a path when they come to a gate that seems stuck half-open, obliging them to pass through it in single file. Neither one wishing to be the first to emerge, they try to push the gate open as they pass through in an attempt to emerge together. The three possible outcomes, c, d, and e, are that they succeed (c), or that they fail and one of Alphonse (d) or Gaston (e) emerges first. That is, exactly one of the events c, d, and e must occur. Taking a and b to be the events of respectively Alphonse and Gaston entering the opening (which could happen either before, after, or instead of succeeding in opening the gate), represent this scenario suitably.

Van Glabbeek proposed the following 11-state automaton, which however he interpreted as a higher-dimensional automaton with 31 states when the 15 one-dimensional and 5 two-dimensional states are counted. Our depiction here labels the 11 states to exhibit it as a pure cancellation automaton, one with no

higher-dimensional cells. We give it in both forms, matrix and visual (its Hasse diagram).

a	0	1	0	0	1	1	1	0	1	1	1
b	0	0	0	1	0	1	1	1	1	1	1
c	0	0	1	0	1	0	0	1	×	1	×
d	0	0	×	×	×	0	×	×	1	×	×
e	0	×	×	0	×	×	0	×	×	×	1



The success of this representation depends on the observation that as soon as either party has entered the opening at least one of d or e may be immediately cancelled. Without cancellation, the state of both a and b being in the opening with the gate still stuck would be 110 regardless of their single-file order. With cancellation, exactly one of d or e is cancelled in that situation, refining 110 into the two states 110×0 and 1100×, thereby creating the natural five-state automaton for $ab+ba$ which is what the stuck gate obliges, thereby splitting the "ground floor" $c = 0$ of what would otherwise be the abc cube. The top floor, $c = 1$, gate open, represents $a||b$.

The five two-dimensional cells that naturally suggest themselves, namely two instances of $a||c$ and $b||c$ and one of $a||b$ (so five out of the six faces of the abc cube), could certainly be added, making this a full-blown higher-dimensional cancellation automaton with $11 + 15 + 5 = 31$ cells. However when $a||b$ is understood by default to include all possible higher-dimensional cells, that is, no mutual exclusion, it is not necessary.

Van Glabbeek's example raises an important distinction that we have glossed over so far. Table 2 makes the distinction between $a||b$ and $ab+ba$ one of mutual exclusion: the latter forbids the state ⌟⌟. The role of events d and e here is to record the primacy of respectively a and b in the choice implied by $ab + ba$. In the absence of c the above automaton simplifies to that on the left below. Alternatively we can be more faithful to our disjoint-union definition of $P + Q$

than thus far and obtain an isomorphic automaton in terms of marked copies of just a and b as on the right.

a	0 1 1 0 1
b	0 0 1 1 1
d	0 1 1 $\times$ $\times$
e	0 $\times$ $\times$ 1 1

a	0 1 1 $\times$ $\times$
b	0 0 1 $\times$ $\times$
a'	0 $\times$ $\times$ 1 1
b'	0 $\times$ $\times$ 0 1

References

1. J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.
2. J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge University Press, 1990.
3. M. Barr. *$*$-Autonomous categories*, volume 752 of *Lecture Notes in Mathematics*. Springer-Verlag, 1979.
4. M. Barr. $*$-Autonomous categories and linear logic. *Math Structures in Comp. Sci.*, 1(2):159–178, 1991.
5. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.
6. J.A. Bergstra, A. Ponse, and S.A. Smolka (editors). *Handbook of Process Algebra*. Elsevier (North-Holland), 2000.
7. S.D. Brookes, C.A.R. Hoare, and A.D. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.
8. C. Brown and D. Gurr. A categorical linear framework for Petri nets. In J. Mitchell, editor, *Logic in Computer Science*, pages 208–218. IEEE Computer Society, June 1990.
9. C. Brown, D. Gurr, and V. de Paiva. A linear specification language for Petri nets. Technical Report DAIMI PB-363, Computer Science Department, Aarhus University, October 1991.
10. R. Buckland and M. Johnson. Echidna: A system for manipulating explicit choice higher dimensional automata. In *AMAST'96: Fifth Int. Conf. on Algebraic Methodology and Software Technology*, Munich, 1996.
11. R.T Casley, R.F. Crew, J. Meseguer, and V.R. Pratt. Temporal structures. *Math. Structures in Comp. Sci.*, 1(2):179–213, July 1991.
12. L. Fajstrup, E. Goubault, and M. Raussen. Detecting deadlocks in concurrent systems. In *Proc. of CONCUR'98*, volume 1466 of *Lecture Notes in Computer Science*, pages 332–347. Springer-Verlag, 1998.
13. P. Gabriel and F. Ulmer. *Lokal präsentierbare Kategorien*, volume 221 of *Lecture Notes in Mathematics*. Springer-Verlag, 1971.
14. H. Gaifman and V.R. Pratt. Partial order models of concurrency and the computation of functions. In *Proc. 2nd Annual IEEE Symp. on Logic in Computer Science*, pages 72–85, Ithaca, NY, June 1987.
15. S. Ginsburg and E.H. Spanier. Mappings of languages by two-tape devices. *Journal of the ACM*, 12:423–434, 1965.
16. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
17. J.L. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61:199–224, 1988.

18. R.J. van Glabbeek and F.W. Vaandrager. Petri net models for algebraic theories of concurrency. In *Proc. PARLE, II, LNCS 259*, pages 224–242. Springer-Verlag, 1987.
19. E. Goubault. Homology of higher-dimensional automata. In *Proc. of CONCUR'93*, volume 630 of *Lecture Notes in Computer Science*, pages 254–268, Stonybrook, New York, August 1993. Springer-Verlag.
20. E. Goubault. *The Geometry of Concurrency.* PhD thesis, École Normale Supérieure, 1995.
21. E. Goubault. Schedulers as abstract interpretations of hda. In *Proc. of PEPM'95*, La Jolla, June 1995. ACM Press.
22. E. Goubault. Durations for truly-concurrent actions. In *Proceedings of ESOP'96*, pages 173–187. Springer-Verlag, 1996.
23. E. Goubault. A semantic view on distributed computability and complexity. In *Proceedings of the 3rd Theory and Formal Methods Section Workshop.* Imperial College Press, 1996.
24. E. Goubault. (ed.) geometry and concurrency. *Mathematical Structures in Computer Science, special issue*, 10(4):409–573 (7 papers), August 2000.
25. E. Goubault and R. Cridlig. Semantics and analysis of Linda-based languages. In *Proc. 3rd Int. Workshop on Static Analysis*, volume 724 of *Lecture Notes in Computer Science*, pages 72–86, Padova, 1993. Springer-Verlag.
26. E. Goubault and T.P. Jensen. Homology of higher dimensional automata. In *Proc. of CONCUR'92*, volume 630 of *Lecture Notes in Computer Science*, pages 254–268, Stonybrook, New York, August 1992. Springer-Verlag.
27. J. Grabowski. On partial languages. *Fundamenta Informaticae*, IV.2:427–498, 1981.
28. I. Greif. *Semantics of Communicating Parallel Processes.* PhD thesis, Project MAC report TR-154, MIT, 1975.
29. J. Gunawardena. Homotopy and concurrency. *EATCS Bulletin 54*, pages 184–193, October 1994.
30. V. Gupta. *Chu Spaces: A Model of Concurrency.* PhD thesis, Stanford University, September 1994. Tech. Report, available as `http://boole.stanford.edu/pub/gupthes.pdf`.
31. V. Gupta and V.R. Pratt. Gates accept concurrent behavior. In *Proc. 34th Ann. IEEE Symp. on Foundations of Comp. Sci.*, pages 62–71, November 1993.
32. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* MIT Press, Boston, 2000.
33. C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–672, August 1978.
34. Y. Lafont. The linear abstract machine. *TCS*, 59:157–180, 1988.
35. Y. Lafont and T. Streicher. Games semantics for linear logic. In *Proc. 6th Annual IEEE Symp. on Logic in Computer Science*, pages 43–49, Amsterdam, July 1991.
36. J. Lambek and P. Scott. *Introduction to Higher-Order Categorical Logic.* Cambridge University Press, 1986.
37. A. Mazurkiewicz. Concurrent program schemes and their interpretations. Technical Report DAIMI Report PB-78, Aarhus University, Aarhus, 1977.
38. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, 1980.
39. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures, and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.
40. C. Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Press, 1986.

41. D. Park. Concurrency and automata on infinite sequences. In *Proc. Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.
42. C.A. Petri. Fundamentals of a theory of asynchronous information flow. In *Proc. IFIP Congress 62*, pages 386–390, Munich, 1962. North-Holland, Amsterdam.
43. G.D. Plotkin. Lambda definability in the full type hierarchy. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–373. Academic Press, 1980.
44. G.D. Plotkin. A structural approach to operational semantics. Technical Report Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, 1981. Reprinted with corrections in J. Log. Algebr. Program. 60-61: 17-139 (2004).
45. A. Pnueli. The temporal logic of programs. In *18th IEEE Symposium on Foundations of Computer Science*, pages 46–57, October 1977.
46. V.R. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proc. 17th Ann. IEEE Symp. on Foundations of Comp. Sci.*, pages 109–121, October 1976.
47. V.R. Pratt. Process logic. In *Proc. 6th Ann. ACM Symposium on Principles of Programming Languages*, pages 93–100, San Antonio, January 1979.
48. V.R. Pratt. On the composition of processes. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Programming Languages*, January 1982.
49. V.R. Pratt. Position statement. Circulated at the Panel on Mathematics of Parallel Processes, chair A.R.G. Milner, IFIP-83, September 1983.
50. V.R. Pratt. The pomset model of parallel processes: Unifying the temporal and the spatial. In *Proc. CMU/SERC Workshop on Analysis of Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 180–196, Pittsburgh, 1984. Springer-Verlag.
51. V.R. Pratt. Some constructions for order-theoretic models of concurrency. In *Proc. Conf. on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 269–283, Brooklyn, 1985. Springer-Verlag.
52. V.R. Pratt. Two-way channel with disconnect. In *The Analysis of Concurrent Systems: Proceedings of a Tutorial and Workshop*, volume 207 of *Lecture Notes in Computer Science*, pages 110–111. Springer-Verlag, 1985.
53. V.R. Pratt. Modeling concurrency with partial orders. *Int. J. of Parallel Programming*, 15(1):33–71, February 1986.
54. V.R. Pratt. Modeling concurrency with geometry. In *Proc. 18th Ann. ACM Symposium on Principles of Programming Languages*, pages 311–322, January 1991.
55. V.R. Pratt. Arithmetic + logic + geometry=concurrency. In *Proc. First Latin American Symposium on Theoretical Informatics*, volume 583 of *Lecture Notes in Computer Science*, pages 430–447, São Paulo, Brazil, April 1992. Springer-Verlag.
56. V.R. Pratt. The duality of time and information. In *Proc. of CONCUR'92*, volume 630 of *Lecture Notes in Computer Science*, pages 237–253, Stonybrook, New York, August 1992. Springer-Verlag.
57. V.R. Pratt. Event spaces and their linear logic. In *AMAST'91: Algebraic Methodology and Software Technology*, Workshops in Computing, pages 1–23, Iowa City, 1992. Springer-Verlag.
58. V.R. Pratt. Chu spaces: complementarity and uncertainty in rational mechanics. Technical report, TEMPUS Summer School, Budapest, July 1994. Manuscript available as `http://boole.stanford.edu/pub/bud.pdf`.
59. V.R. Pratt. Time and information in sequential and concurrent computation. In *Proc. Theory and Practice of Parallel Programming (TPPP'94)*, volume 907 of

Lecture Notes in Computer Science, pages 1–24, Sendai, Japan, November 1994. Springer-Verlag.
60. V.R. Pratt. Chu spaces and their interpretation as concurrent objects. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 392–405. Springer-Verlag, 1995.
61. V.R. Pratt. Types as processes, via Chu spaces. In *Electronic Notes in Theoretical Computer Science*, volume 7, Santa Margherita, 1997. URL: http://www.elsevier.nl/locate/entcs/volume7.html, 21 pages.
62. V.R. Pratt. Chu spaces: Notes for school on category theory and applications. Technical report, University of Coimbra, Coimbra, Portugal, July 1999. Manuscript available as `http://boole.stanford.edu/pub/coimbra.pdf`.
63. V.R. Pratt. Higher dimensional automata revisited. *Math. Structures in Comp. Sci.*, 10:525–548, 2000.
64. V.R. Pratt. Orthocurrence as both interaction and observation. In *Proc. Workshop on Spatial and Temporal Reasoning (ed. R. Rodriguez and F. Anger), IJCAI'01*, Seattle, August 2001.
65. V.R. Pratt. Event-state duality: the enriched case. In *Proc. CONCUR'02*, Brno, August 2002.
66. V.R. Pratt. Chu spaces as a semantic bridge between linear logic and mathematics. *Theoretical Computer Science*, 294(3):439–471, February 2003. Selected papers from Linear Logic'96, Tokyo.
67. V.R. Pratt. Transition and cancellation in concurrency and branching time. *Math. Structures in Comp. Sci., special issue on the difference between sequentiality and concurrency*, 13(4):485–529, August 2003.
68. W. Riddle. *The Modeling and Analysis of Supervisory Systems*. PhD thesis, Computer Science Dept., Stanford University, March 1972. 174 pp.
69. R. V. Rodriguez and F. D. Anger. Branching time via Chu spaces. In *Proc. Workshop on Spatial and Temporal Reasoning (ed. R. Rodriguez and F. Anger), IJCAI'01*, Seattle, August 2001.
70. V. Sassone and G. L. Cattani. Higher-dimensional transition systems. In *Proceedings of LICS'96*, 1996.
71. M. Shields. Deterministic asynchronous automata. In E.J. Neuhold and G. Chroust, editors, *Formal Models in Programming*. Elsevier Science Publishers, B.V. (North Holland), 1985.
72. Y. Takayama. Extraction of concurrent processes from higher-dimensional automata. In *Proceedings of CAAP'96*, pages 72–85, 1996.
73. R. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, Vrije Universiteit te Amsterdam, May 1990.
74. R. van Glabbeek. Bisimulations for higher dimensional automata. Manuscript available as `http://theory.stanford.edu/~rvg/hda`, June 1991.
75. R. van Glabbeek. On the expressiveness of higher dimensional automata. *Theoretical Computer Science*, 356(3):169–194, 2006.
76. G. Winskel. *Events in Computation*. PhD thesis, Dept. of Computer Science, University of Edinburgh, 1980.
77. G. Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986*, volume 255 of *Lecture Notes in Computer Science*, Bad-Honnef, September 1986. Springer-Verlag.
78. G. Winskel. An introduction to event structures. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, REX'88*, volume 354 of *Lecture Notes in Computer Science*, Noordwijkerhout, June 1988. Springer-Verlag.