To appear in Math. Struct. in Comp. Science

Higher Dimensional Automata Revisited

VAUGHAN PRATT

Department of Computer Science, Stanford University,
Stanford, CA 94305-9045

Received February 2000

The dual of a true concurrency schedule appears to be a false concurrency automaton, a paradox we resolved in a previous paper by extending the latter to higher dimensions. This extension may be formalized via such discrete geometries as n-categories, simplicial complexes, cubical complexes, and Chu spaces. We advocate the last as having a clear notion of event, a well-defined process algebra uniformly extending that for event structures, and ease of extension beyond the basic before-during-after analysis.

1. The notion of higher dimensional automaton

A natural question for automata theory is how to represent two independent events a and b as an automaton without committing to their order of occurrence. The obvious traditional automaton

$$\begin{tikzcd}
00 \arrow[r, "a"] \arrow[d, "b"] & 10 \arrow[d, "b"] \\
01 \arrow[r, "a"] & 11
\end{tikzcd}$$

Figure 1

allows for the possibility of either order of occurrence. This automaton consists of four states and four transitions for a total of eight discrete elements. The two digits of each global state denote the local states of events a and b respectively, with 0 denoting unstarted and 1 done.

However this representation tacitly commits to the well-definedness of the order of occurrence, with its implication that a run of this automaton must choose one of the two available paths from 00 (both events unstarted) to 11 (both events done). There is a hidden assumption of excluded middle, or *mutual exclusion* as it is more usually called for concurrent processes.

The existence of this choice is reasonable for events that actually are mutually exclusive, such as two otherwise independent actions that are to be performed by an agent that can only do one thing at a time. But for events that are truly independent, such as a Mars rover picking up a rock in the same second that an operator in Houston is sending the rover a command, the communication delay of several minutes renders irrelevant their order of occurrence. Certainly from relativistic considerations, and arguably from engineering ones also, such a choice is not even absolutely defined, being observer-dependent.

This independence may be represented by abandoning automata altogether in favor of partially ordered schedules of events, with the above example represented as a set of two events unconstrained as to their order. This is the approach underlying the event structures of (NPW81) and the partially ordered multisets or pomsets of (Pra82).

But while it is convenient to abandon automata for this purpose, it does not follow that it is necessary. Higher dimensional automata permit this lack of commitment to choice to be represented in an extension of the automata-theoretic framework by representing the independent performance of n events as an n-dimensional entity in the automaton. In the example of Figure 1, the empty interior of the square is filled with a two-dimensional surface or 2-cell representing the independent occurrence of a and b, bringing to nine the number of discrete components of the automaton.

Had there been three events, we would have started with a 3-cube, namely an eight-state automaton having twelve edges, a total of 20 elements. Filling in the six faces represents the independent occurrence of any two out the three events, while filling in the interior of the cube represents the independent occurrence of all three events. All told we then end up with $8 + 12 + 6 + 1 = 27 = 3^3$ events.

For n independent events there are 3^n cells, of dimension from zero up to n, with each cell representing the "local" states of the n events, each of which may be either unstarted, active, or done. The possibility of an event being in the "active" state generalizes the 0-dimensional notion of state to higher dimensions. The dimension of such a generalized state is the number of events that are active in that state. In this view a transition in the usual sense becomes a state in which exactly one event is active. A state with more than one active event represents a notion of "joint transition" as a surface, volume, or higher-dimensional cell.

Returning to the original two-event process, the case of mutually exclusive occurrence of the two events may be obtained from the solid square by removing its interior. The requirement that a precede b (precedence) may be met by further removing the lower left state and attached edges. The requirement that not both a and b be performed (conflict) may be met by instead removing the lower right state and attached edges.

In general any process may be understood as the set A of all events it is capable of performing, typically infinite for processes that can loop or recurse, together with some subset of the set $\{\text{unstarted,active,done}\}^A$ constituting the possible states of the process. This is programming as sculpture: start from a sufficiently large cube and hew out the desired process by chiseling away the unwanted states. This point of view has been taken elsewhere in the higher-dimensional automata literature (Gou93; FGR98).

While this view is attractively simple conceptually, it is not by itself a practical way of specifying a concurrent process. An alternative approach is composition, in which complex processes are built from smaller ones with suitable operators, including intrinsically concurrent operators such as asynchronous parallel composition. Yet another approach is transformation, in which new processes are constructed from old by reshaping them appropriately.

These three activities, sculpture, composition, and transformation, are simultaneously compatible and complementary, and can therefore usefully be taken as a basis for concurrent programming. Very loosely speaking they correspond respectively to subalgebras, products, and homomorphisms, which play central and complementary roles in the algebraic approach to both logic and programming. Practicality aside, it is a good question (not addressed here) to what extent any two, or even one, of sculpture, composition, and transformation can fill in for the omitted approaches.

A premise of this paper is that sculpture on its own suffices at least for the abstract modeling of concurrent behavior, if not for its practical specification.

2. Origin of higher dimensional automata

2.1. *The short story*

The geometrical view of automata is implicit in A. Mazurkiewicz's algebraic notion of independence via partial monoids (Maz77; Maz84). It is made more explicit and put to practical use in C. Papadimitriou's model for database concurrency control (Pap86, chap.6), with however no accompanying formal notion of an automaton. Higher dimensional transitions make a brief appearance at the end of M. Shields' paper on deterministic asynchronous automata (Shi85). The explicit notion of higher dimensional automaton as an extension of traditional automata theory was introduced by the present author at POPL'91 (Pra91).

We arrived at the idea of "filling in the holes" of traditional automata by noticing a paradox implicit in the main theorem of (NPW81), the duality of prime event structures as event schedules and prime algebraic domains as state automata dual to schedules. While the event structures of that paper, interpreted as schedules, were by design a model of true concurrency, their dual prime algebraic domains were unmistakably automata of the "false concurrency" kind, with Figure 1 a case in point having $\{00, 01, 10, 11\}$ as its family of configurations. A perfect duality between true and false models of concurrency is paradoxical: for this distinction to be meaningful there must be something missing from the interpretation of prime algebraic domains as conventional automata.

One evident difference between schedules and automata is that with the former the passage from $a \leq b$ to $b \leq a$ can be accomplished by moving a and b smoothly past each other in time. We asked what was the dual of this smoothness for automata. Somehow the ab path must transform smoothly into the ba path. The only sensible way this could happen was via a surface connecting the two. Familiarity with the geometric representation of natural transformations of two functors, namely as 2-cells between those two functors viewed as two 1-cells having a common start and end, facilitated our arrival at this picture. From here it was an easy step to our formalization in that paper of higher dimensional automata in terms of higher dimensional cells in n-categories.

At question time after my presentation of (Pra91), Boris Trakhtenbrot asked from the front row, what is the relationship between HDA's and event structures?[†] Despite having addressed the issue in section 7 of (Pra91), focusing on persistence of conflict, I had no good answer to the question, a conflict that persisted for me until the solution presented here struck me several years later after gaining much experience with Chu spaces (Bar79, appendix).

The general problem here is to reconcile the respective motivating intuitions and supporting formalisms underlying the event-based and geometry-based approaches to concurrency. Event structures replace the traditional state-based view of computation by the view of $a||b$ as a set $\{a, b\}$ representing two events unconstrained as to order. In contrast the geometry-based approach retains the state-based view and models $a||b$ as the evident four-state "square" automaton accepting $ab + ba$ but with its square interior filled in as described above.

The question then becomes, can these be understood as simply the one notion of concurrent computation seen from two compatible perspectives, or is there some inconsistency preventing the reconcilation of these two models of concurrency?

We believe that Chu spaces are superior to event structures for organizing event-based computation. Chu spaces form a structurally attractive category (Bar79; LS91; Bar91; Bar99), with incidentally a wide range of other applications including concurrency (BG90; BGdP91; GP93; Gup94; VGP95), linear logic (dP89; Bar91), games (Bla95), and universal algebra (Pra93; Pra95).

There are two main reasons for pursuing Trakhtenbrot's question. First, both HDA's and event structures are attractive models of concurrency, raising the question of whether their respective intuitions about the nature of concurrency are compatible. Second, HDA's and event structures draw on complementary areas of mathematics: much of the HDA work since its introduction (GJ92; GC93; Gou93; Gun94; Gou95b; Gou95a; Gou96a; Gou96b; BJ96; Tak96) has drawn on methods in algebraic topology, in particular homotopy and homology, whereas event structures tend to depend more on methods from domain theory and logic, especially those involving duality, with some notable exceptions (vG91; SC96) that strive for a more elementary realization of geometry than either the ω-categories of our original formalization (Pra91) or the homological approaches. Given these reasons it is both intrinsically interesting in principle and important in practice for concurrent programming language design to understand how the one notion of concurrency can shift its emphasis in this way between these substantially different areas of mathematics.

What are some essential differences between HDA's and event structures?

Two basic differences involve conflict and duality. HDA's can express not only persistent but also transient conflict, whereas event structures cater better to schedule-automaton duality.

[†] More precisely, he asked about the relationship with event spaces (Pra92), but that distinction is a negligible one compared with the difference with HDA's.

Persistent conflict, notated $a\#b$ (in the binary case), means that at most one of a or b can happen. Transient conflict, as in mutual exclusion, means that a precludes b (again for the binary case) only temporarily, e.g. while a is executing. Whereas HDA's can express both kinds of conflict equally directly, event structures express only persistent conflict directly, and deal only clumsily with transient conflict. An unlabeled event structure with two events a and b can express their persistent conflict directly as $a\#b$, by design. To express a *mutex* b however, it is rendered as $ab + ba$, which instead of two unlabeled events requires four labeled events, two labeled a and two labeled b, with each of the two events of ab in persistent conflict with each of those of ba. HDA's do much better here, expressing the same concept simply by omitting the interior of the square.

On the other hand, event structures dualize easily to acyclic automata, or families of configurations as they are called in the event structure literature. Dualization is accomplished simply by taking the configurations to be those sets of events satisfying the constraints defining the event structure. Identifying sets of events with their characteristic functions to 2, dualization can be understood as "homming into" an object k (here the set 2), in general $\mathrm{Hom}(-, k)$ as the usual notion of representable contravariant functor. In contrast, HDA's as typically defined have no obvious dualization to higher dimensional schedules, or to any plausible notion of schedule.

Both transient conflict and schedule-automaton duality are too important to neglect. Transient conflicts of various kinds pervade the theory of concurrency control of databases (Pap86). And it is overly restrictive to limit the representation of computation to just one of automata as states connected by transitions or schedules as events separated by temporal constraints: the features of interest of a given computational process are seen more clearly sometimes as a schedule, sometimes as an automaton, just as physical interactions are seen best in terms of waves or particles depending on the circumstances, with no sharp boundary at the crossover.

Since both models have worthwhile advantages absent from the other, we cannot simply discard one. We therefore propose the following tightening of Trakhenbrot's question. Can the definitions of HDA's and event structures be adjusted in a way that would overcome the respective limitations of each relative to the other, yet without compromising in any way either the formal capabilities or underlying intuitions of either model?

The solution we propose is Chu spaces over 3. These extend Chu spaces over 2, which model event structures. The extension can be understood as extending event structures to higher dimensions by interpreting the new (middle) element of 3 as ongoing activity, a notion absent from ordinary event structures which has only the two-valued logic of before and after.

We treat this solution in detail in the latter part of the paper. In the following we examine in more detail how we arrived at a geometric model of concurrency.

2.2. *The details*

We now give a more detailed account of our reasoning up to this point, starting from the problem of giving a compositional semantics for concurrent processes. Such a semantics will consist of a class of mathematical objects representing processes, together with a family of operations for synthesizing larger processes from smaller ones, and a family of constants representing primitive processes.

For processes that are sequential, which we think of as a special case of concurrent processes, one possible representation of processes is as binary relations on a set W of worlds or states. A natural family of operations for this representation is that of regular expressions, with $+$ (choice) acting as union on binary relations, concatenation (sequence) acting as composition, Kleene star $*$ (iteration) acting as reflexive transitive closure, and the constant 0 denoting the empty binary relation. Another representation of processes is as sets of finite and infinite strings over an alphabet of actions, with the regular operations then interpreted as usual for formal languages.

Interleaving semantics provides a straightforward way to extend the latter to concurrent processes. A major advantage of interleaving semantics is that the representation of processes can remain unchanged. Instead one augments the operations with operations for combining concurrently executing processes. The most basic of these is the shuffle or interleaving operator, which given two sets of strings yields all strings obtainable by arbitrarily interleaving two strings one from each set.

However the idea of representing the joint execution of events a and b as the choice of a followed by b or b followed by a does not agree with intuition. When these two events are widely separated in space but very near in time, their relative order is normally either an irrelevant detail or entirely meaningless. One would like to say therefore that the events are *independent.*

One model capturing this independence is the ***Petri net*** (Pet62). Independent tokens of a net may fire at independent times.

A more algebraic model is Mazurkiewicz' notion of a ***trace*** (Maz77; Maz84) as the identification of strings differing only in the immaterial order of some of their symbols. Such an identification removes the element of choice from these ostensibly competing strings. (Mazurkiewicz provides only for pairwise mutual exclusion, so that one cannot remove the interior of the 3-cube without also removing a parallel pair of faces, but it should be possible to extend Mazurkiewicz traces to higher orders of independence by formulating a suitable notion of partial monoid based on higher dimensional automata.)

Yet another model is the partially ordered multiset or ***pomset*** (Gra81; Pra82; Pra84; Gis84; Gis88). This model starts from the idea of a string over an alphabet Σ as representing a sequential computation whose actions are drawn from Σ. Such a string may be defined as a linearly ordered multiset of symbols, with the order being that of the symbols in the string: $a < b$ when a appears earlier than b. This definition then permits concurrency to be introduced by generalizing to partially ordered multisets or pomsets. Two symbols of a pomset that are incomparable in the temporal order are then considered to be independent, meaning that their order of execution is undefined. As a generalization of the idea of a formal language as a set of strings, a ***pomset process*** is defined as a set of pomsets over a common alphabet Σ.

One approach to formalizing the notion of multiset over an alphabet Σ is as a labeled set A whose labels are drawn from Σ and whose elements constitute occurrences of elements of Σ in the multiset, with each element of Σ being permitted to occur multiple times. In this approach the set A is a partially ordered *set* or poset $(A, \leq)$ rather than a pomset. The labeling is then realized with a function $\lambda : A \to \Sigma$, with each $a \in A$ being understood as an occurrence of $\lambda(a)$, making a pomset a structure $(A, \leq, \Sigma, \lambda)$.

2.3. *Event structures*

Prime event structures (NPW81) enrich this partial-order model of computation with a symmetric irreflexive binary relation # denoting ***conflict***, subject to the requirement that if $a\#b$ and $b \leq c$ then $a\#c$. When events a and b stand in this relation it signifies the impossibility of both events happening: once one has happened, the other is forbidden to ever happen. By default event structures are unlabeled and consist just of a set A together with the two binary relations of order and conflict. *Labeled* event structures add to event structures a labeling alphabet Σ of actions and a labeling function $\lambda : A \to \Sigma$. The meaning of an event a with label $\lambda(a)$ is that a is an instance or occurrence of the action $\lambda(a)$. Labels permit an action to occur more than once in a process.

The notion of label allows us to distinguish events, transitions, and actions as follows. The primitive notion is that of *event*: the defining property of an event is that it is permitted to occur at most once. A *transition* is a located or situated event, that is, an event together with contextual information about the state of other events. In Figure 1 there is one event a associated with two transitions, namely one for which b is unstarted and the other for which b is done, and similarly one b event but two b transitions. An *action* is an event type or event label, such as "hit the nail on the head" or "add one to x," and may occur more than once.

With these distinctions drawn, cells of a higher dimensional automaton can be seen to be transitions. In the example of the 3-cube, namely three independent events with no constraints, each of the six 2-dimensional faces is a transition representing the independent occurrence of two of the three events, and is situated at one end of the third event, with its mate being a parallel face situated at the other end.

Actions for HDA's are as for ordinary event structures, namely event labels. The notion of an action is clearly important: for automata they are the symbols of the automaton's alphabet, without which the automaton is just a graph with no way of distinguishing behaviors that clearly should be different and with no reason to have more than one edge running from one state to another.

It is intuitively clear that action instances should be consistent with the event structure implicit in the cubical structure of an HDA, and hence that there should a well-defined notion of event. We return to this point later when we argue the preferability of Chu spaces over cubical sets for modeling HDA's.

2.4. *Families of Configurations*

We now develop this informal notion of a cell more formally in terms of a notion of *state* of either a poset $(A, \leq)$ or an event structure $(A, \leq, \#)$. Their definition leads to the duality of schedules *qua* posets and automata *qua* sets of states.

A state of a poset is a subset $B \subseteq A$ such that if $b \in B$ and $a \leq b$ then $a \in B$. In the literature on ordered structures such a subset is standardly called an ***order ideal***, or sometimes a ***downset***, but we shall refer to it simply as a state. A state or configuration of an event structure is as for posets with the additional requirement that if $a\#b$ holds then no state may contain both a and b.

An ***event structure homomorphism*** is a function $f : A \to B$ between event sets A and B that is monotone with respect to the order and preserves conflict: if $a\#b$ then $f(a)\#f(b)$. (This is a different notion of morphism of event structures from that proposed by Winskel (Win84; Win88).) It should be clear from this definition that states can be understood as homomorphisms to the two-event event structure $0 < 1$, a chain (linearly ordered set) having the empty conflict relation. The latter event structure plays an analogous role to the two-point three-open-set Sierpinski space in topology, namely as a kind of truth-value object, allowing states of an event structure to be viewed as predicates on that structure. The idea of events as elements and states as predicates leads naturally to the Chu space view of processes we shall introduce later.

We now study the set of states of a poset. With a little thought one sees that a poset's state set is closed under arbitrary (including empty and infinite) union and arbitrary intersection. (The empty union is the empty subset of A while the empty intersection is the whole of A.) Furthermore every set of subsets of a set A enjoying these closure properties arises in a unique way as the states of some partial order $(A, \leq)$.[‡]

Less obvious is that both A and $\leq$ can be reconstructed, up to order isomorphism, knowing only the partial ordering of these subsets and not the elements of the subsets or the set from which those elements are drawn. More precisely, we are given a poset $(L, \leq)$ which is isomorphic, as a poset, to the set of states of $(A, \leq)$ ordered by inclusion. An alternative characterization of such a poset $(L, \leq)$, that is even more abstract by virtue of not mentioning $(A, \leq)$, uses the fact that $(L, \leq)$ must be a complete lattice: every subset $M \subseteq L$ has both a least upper bound or sup $\bigvee M$ and a greatest lower bound or inf $\bigwedge M$ in L. Furthermore the lattice is completely distributive: $a \wedge \bigvee_i b_i = \bigvee_i (a \wedge b_i)$ and $a \vee \bigwedge_i b_i = \bigwedge_i (a \vee b_i)$. So instead of merely a poset $(L, \leq)$ we may assume we start with such a complete and completely distributive lattice $(L, \bigvee, \bigwedge)$.

But these properties are still not enough. The unit interval $[0, 1]$ of reals, standardly ordered, is a complete completely distributive lattice under max and min, yet it is not isomorphic to the lattice of states of any poset since it does not contain a gap (ordered pair $a < c$ of elements with no third element $a < b < c$ between them). This counterexample therefore tells us we need to constrain L further.

To do so we identify the ***compact*** elements of a complete lattice L, defined as those elements $x \in L$ satisfying $\bigvee\{y < x\} < x$. We write $K(L)$ for the set of compact elements of L. We call the complete completely distributive[§] lattice L a ***profinite distributive lattice*** when every element $x \in L$ is representable as $x = \bigvee\{y \in K(L) | y \leq x\}$ (or equivalently, as the sup of some subset of $K(L)$).

[‡] To extract the order on A from the given set of subsets of A, take $a \leq b$ just when every subset containing b contains a. What is more work to show is that the order so defined gives rise to exactly this set of subsets of A as its states.

[§] A complete lattice is ***completely distributive*** when $x \wedge (\bigvee Y) = \bigvee_{y \in Y} (x \wedge y)$ for all $x \in L$ and $Y \subseteq L$.

Theorem 1. Every profinite distributive lattice $(L, \bigvee, \bigwedge)$ is isomorphic to the lattice of states of some poset $(A, \leq)$.

The proof of this well-known theorem is short enough to repeat here, see also (NPW81) for a similar argument.

Proof. Not surprisingly one takes A to be $K(L)$, ordered as in L. Now consider the map that takes $x \in L$ to $\{y \in K(L) | y \leq x\}$. This map is clearly monotone, and is injective because x can be recovered as $\bigvee\{y \in K(L) | y \leq x\}$, by the definition of profinite. If it is not surjective then it must be because there exist distinct order ideals Y, Z of A for which $\bigvee Y = \bigvee Z$ in L. Without loss of generality let $a \in Z - Y$. But then $a \leq \bigvee Z = \bigvee Y$ whence $a = a \wedge \bigvee Y = \bigvee_{y \in Y}(a \wedge y)$ by complete distributivity of L. Since a is not in the order ideal Y, we must have that for all $y \in Y$, $a \not\leq y$, so $a \wedge y < a$. But then by compactness of a we must have $\bigvee_{y \in Y}(a \wedge y) < a$, contradicting the above equality. Hence the map is surjective and so can be taken as the promised isomorphism. □

The converse of this theorem then constitutes a duality between the category **Pos** of posets and the category of profinite distributive lattices, called $\mathsf{StoneDLat}$ by Johnstone (Joh82) on account of having an alternative characterization in terms of Stone topology.

Theorem 2. Every poset $(A, \leq)$ is isomorphic to the poset of compact elements of some profinite distributive lattice.

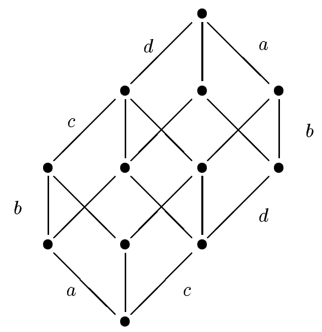
Proof. Take the lattice of states of $(A, \leq)$. The compact elements of this lattice are just the principal order ideals of the poset, those order ideals of the form $\{b \leq a\}$ for some $a \in A$, which are in an obvious bijection with A. Furthermore this bijection is monotone and hence an order isomorphism. □

3. Cubes

Our original definition of HDA (Pra91) was based on n-categories. These are discrete higher-dimensional structures whose n-cells have only two boundaries, source and target (domain and codomain), independently of n. Such n-cells have been called globs or sometimes globes. In contrast an n-simplex has $n+1$ boundaries (e.g. a triangle has 3 edges, a tetrahedron has 4 faces), while an n-cube has $2n$ boundaries (e.g. a square has 4 sides, a cube has 6 faces). Discussions with Rob van Glabbeek shortly after the appearance of (Pra91) convinced me that n-cubes were preferable to either n-globs or n-simplices as the basic n-cells representing the concurrent execution of n events, and also led to his note (vG91).

This leads to two natural questions: what is a cube, and how are concurrent processes defined in terms of them? The two approaches we see as most workable are cubical complexes and Chu spaces over 3.

A cubical complex is a set of n-cubes for various integers $n \geq 0$ along with their faces of all lower dimensions, some of which may be shared with other cubes. For example two 3-cubes may have a face in common representing the concurrent execution $a||b||(c; d)$ of event a, event b, and event sequence $c; d$, with the shared face parallel to the ab plane at the junction of c and d, as shown below.



Or one 2-cube may bend back to join its final vertex to its initial vertex, forming the iteration $(a||b)^*$: do a and b in parallel and when both are finished do them both again and so on.

A Chu space over 3 is a set A of events together with a subset X of the "solid" cube 3^A having 2^A as its 0-cells. The dimension of the Chu space is that of the largest cube in X, which may be finite even though A is infinite.

An advantage of cubical complexes is that iteration can be represented directly in the form of cycles in the complex. In this way a finite cubical complex can represent potentially infinite behavior. The drawback of Chu spaces here is that they do not admit cycles. They can however represent any behavior representable by a cubical complex by expanding or "unfolding" cycles to their infinite acyclic counterparts. A similar distinction arises between finite state automata and formal languages: the latter can be understood as the unfoldings of the former, more precisely the paths in those unfoldings.

An advantage of Chu spaces is that they make explicit both the events and the states of the process, attaching equal importance to them. Cubical sets do not specify their associated events, and only in sufficiently simple cases is it unambiguous what events coordinatize each cube.

We regard the advantages of cyclic structure in cubical complexes as not offsetting the disadvantage of not knowing the events of a process.

The advantages of cyclic structure do not as one might first think reside in the finite representability of infinite behavior. Much reasonable infinite behavior is finitely representable in terms of Chu spaces as the least solution to a finite set of finitely presented recursive equations, or as the Kleene star (iteration) of a finite or finitely presented Chu space. Cycles are merely an intuitively appealing representation of iteration.

However we do not have a counterpart for higher dimensional automata of Kleene's theorem that every ordinary automaton is trace-equivalent (though not bisimilar) to a regular set. Thus cycles for HDA's may well have an advantage in greater expressive power.

One can however question the practical benefits of such expressivity. Cycles permit arbitrarily knotted and otherwise topologically convoluted structures, to a much greater degree with HDA's than with ordinary finite automata. We are unable to defend cubical sets on the ground that they offer cyclic structures that can be readily understood by code maintainers yet that cannot be represented algebraically. We therefore take the position that little is lost by limiting programmers to those infinite behaviors that are representable algebraically via recursion and iteration.

We had originally intended to treat cubical complexes anyway, in particular the presentation of the category of cubical sets in terms of an algebraic theory defined as the category of finite bipointed sets. However some of our early readers have suggested we get to the point and treat what we really believe in, which we now do.

4. Chu spaces

In the cubical set approach to higher dimensional automata an automaton is a (possibly infinite) set of cubes of various dimensions. In the Chu space approach one starts instead with a single cube of very large, possibly infinite, dimension and "sculpts" the desired process by removing unwanted faces. The axes of the starting cube constitute the events, initially a discrete or unstructured set. The removal of states has the effect of structuring the event set. For example sculpting renders two events equivalent, or synchronized, after all states distinguishing them have been removed.

The sculpture way of looking at Chu spaces does not reveal the intrinsic symmetry of events and states. An alternative presentation that brings out the symmetry better is as a matrix whose rows and columns are indexed by events and states respectively, and whose entries are drawn from the set $3 = \{0, 1, 2\}$. The columns of this matrix constitute the selected faces of the cube.

Chu spaces have several advantages over cubical sets for formalizing higher dimensional automata. First, the transition from ordinary event structures to higher dimensional event structures is made simply by allowing one additional truth value. Second, events are well-defined; with cubical sets one sometimes cannot tell whether an axis of one cube is associated with the same event as an axis of another. Third, the duality of events and states implicit in event structures extends without complication to the higher dimensional case. And fourth, the full gamut of both the process algebra and the linear logic operations as defined on two-valued Chu spaces, when suitably stated, carries over to the three-valued case with no change to the wordings of the definitions.

A little more formally, a Chu space is simply a matrix over a set Σ, that is, a rectangular array whose entries are drawn from Σ, for example $\boxed{\begin{smallmatrix}011\\001\end{smallmatrix}}$. The smallest useful alphabet is $2 = \{0, 1\}$, which is appropriate for representing ordinary event structures, whose events may be either not done or done. The alphabet of principal interest for higher dimensional automata is $3 = \{0, 1, 2\}$ for unstarted, active, and done. We will consider even larger Σ's later on. We now give the formal definition.

Definition 3. A Chu space $\mathcal{A} = (A, r, X)$ over a set Σ, called the ***alphabet***, consists of a set A of ***events*** constituting the ***carrier***, a set X of ***states*** constituting the ***cocarrier***, and a function $r : A \times X \to \Sigma$ constituting the ***matrix***. ∎

We distinguish between an event a as an index of a row of the matrix and the row so indexed, which we define as the function $r_a : X \to \Sigma$ satisfying $r_a(x) = r(a,x)$. The meaning of a row is that it represents an event as an information-dependent value, namely one that varies from state to state. When the representation is ***faithful***, meaning that distinct events are represented by distinct rows, the Chu space is said to be ***separable***. We think of events as intensional and their representation as rows as extensional.

Similarly each state x indexes a column $r^x(a)$, making the type of a column $A \to \Sigma$. The meaning of a column is that it represents a state as a time-dependent value, one that varies from event to event. When this representation is faithful the Chu space is called ***extensional***, and ***biextensional*** when in addition it is separable.

Thus the same matrix has two dual interpretations, namely as two sets A and X each of whose elements is interpreted in terms of its interactions with all the elements of the other set.

Definition 4. Given two Chu spaces $\mathcal{A} = (A, r, X)$ and $\mathcal{B} = (B, s, Y)$, a ***Chu transform*** from $\mathcal{A}$ to $\mathcal{B}$ is a pair (f, g) consisting of functions $f : A \to B$ and $g : Y \to X$ such that $s(f(a), y) = r(a, g(y))$ for all a in A and y in Y, the *adjointness condition.* ∎

Chu transforms compose via $(f', g') \circ (f, g) = (f' \circ f, g \circ g')$. Well-definedness and associativity of this composition are readily verified. The category $\mathsf{Chu}(\mathsf{Set}, \Sigma)$ has as objects all Chu spaces over Σ and as morphisms the Chu transforms between them. Chu transforms play the role for Chu spaces that linear transformations play for vector spaces, continuous functions for topological spaces, etc.

4.1. *From event structures to Chu spaces*

An event structure with set A of events and family X of configurations may be represented as the Chu space $(A, \in, X)$, thinking of the membership predicate $a \in x$ as a function $A \times X \to 2$ (so the alphabet is $2 = \{0, 1\}$).

The event structure homomorphisms between two event structures can be shown to coincide with the continuous functions between their respective representing Chu spaces. This gives a strong sense in which the information in an event structure has been captured by its representing Chu space.

4.2. *HDA's via Chu spaces*

The Chu notion of higher dimensional automaton takes it to be a Chu space over the three letter alphabet $\{0, 1, 2\}$, or $\{0, \frac{1}{2}, 1\}$ as some authors prefer. The three letters represent the three states an event can be in, namely *before*, *during*, and *after*. Following the convention by which the events and states of a Chu space index respectively the rows and columns of that space's matrix, each state x of a Chu space specifies the states of the events a in that state, namely $r(a, x)$. We view x as a global state and $r(a, x)$ as the local state or "microstate" of event a in state x.

The column $r^x(a)$ defined as $\lambda x.r(a, x)$ is a function from A to 3, i.e. an element of 3^A. The set of all columns of a Chu space thus constitute a subset of 3^A.

Its intended interpretation is as the totality of possible (permitted) state vectors. That is, elements of 3^A not appearing as columns of the Chu space are understood as being disallowed.

4.3. *Process Algebra of HDA's*

In this section we develop a four-operation process algebra of higher dimensional automata based on their representation as Chu spaces. The beauty of this process algebra is that the passage from ordinary automata to higher dimensional automata is accomplished simply by increasing Σ from $2 = \{0, 1\}$ to $3 = \{0, 1, 2\}$, entailing no modification at all for two of the definitions, and at most attention to minor details for the other two. Other operations are also definable, such as synchronous concurrence, restriction, and (with strong caveats) recursion as shown by Gordon Plotkin. The four we treat here however will suffice to illustrate the general idea of process algebra operations interpreted over Chu spaces.

The operations we define are, in order (leaving the more difficult ones to the end), concurrence (noninteracting parallel composition), orthocurrence (not standard for process algebra, but we believe important nonetheless), choice, and sequential composition. The definitions are given independently of the structure of Σ as far as possible. Concurrence and orthocurrence make no assumptions about the structure of Σ. Choice assumes that Σ contains an element 0 as the initial state of an event. Sequence assumes both an initial and a final element of Σ.

Readers wishing to experiment with the Chu spaces that can be so built, and whose web browser has Java enabled, can visit the site `http://boole.stanford.edu/live`, *Chu Spaces Live.* This site offers a calculator for Chu spaces accompanied by a tutorial with many exercises. For ordinary event structures set K (synonymous with Σ) to 2, for higher-dimensional automata set K to 3, for the two Anger-Rodriguez examples we treat later, set K to respectively 4 and 6. The notation used there is that of linear logic, with $\otimes$ rendered in ASCII as $*$ and $\invamp$ as $\#$.

Concurrence

The asynchronous parallel composition or ***concurrence*** $\mathcal{A} \| \mathcal{B}$ (linear logic: $\mathcal{A} + \mathcal{B}$) of two processes $\mathcal{A} = (A, r, X), \mathcal{B} = (B, s, Y)$ is defined relatively straightforwardly as $(A + B, t, X \times Y)$ where $t(a, (x, y)) = r(a, x)$ and $t(b, (x, y)) = s(b, y)$. Its operational meaning is the process in which both $\mathcal{A}$ and $\mathcal{B}$ can happen independently of each other.

Looked at from the event structure side of the duality, concurrence simply juxtaposes two event structures. On the state side, the product of states is formed, which corresponds to the standard parallel composition for finite state automata.

Writing 1 for the discrete singleton Chu space $\boxed{01}$ over 2, this definition of concurrence makes $1 \| 1$ the Chu space $\boxed{\begin{smallmatrix}0011\\0101\end{smallmatrix}}$, the discrete doubleton. Switching to Chu spaces over 3 but continuing to write the discrete singleton as 1, we have that 1 is $\boxed{012}$, and $1 \| 1$ is $\begin{smallmatrix}a\\b\end{smallmatrix}\boxed{\begin{smallmatrix}000111222\\012012012\end{smallmatrix}}$, again the discrete doubleton but this time over 3.

These two versions of the discrete doubleton may be depicted as follows.

On the left is the prime algebraic domain of states of a two-event discrete event structure, with the initial state 00 drawn at the bottom (so these automata are rotated 135° counterclockwise from the orientation adopted at the start of the paper). On the right is the corresponding higher dimensional automaton.

Each of these automata can be constructed using the Chu calculator when first visited by setting K to 2 or 3 and then clicking on $+$. (This relies on the inputs to the operation initially being the discrete singleton 1.)

Orthocurrence

The tensor product or ***orthocurrence*** (Pra85; Pra86; CCMP91) $\mathcal{A} \otimes \mathcal{B}$ of two processes $\mathcal{A} = (A, r, X), \mathcal{B} = (B, s, Y)$ is defined as $(A \times B, t, Z)$ where Z is the set of Chu transforms from $\mathcal{A}$ to the transpose $\mathcal{B}^{\perp} = (Y, r^{\smile}, B)$ of $\mathcal{B}$ ($r^{\smile}(y, b) = r(b, y)$). and $t((a, b), (f, g)) = s(b, f(a))$.

This definition does not at first sight seem symmetric in $\mathcal{A}$ and $\mathcal{B}$. However the definition makes $\mathcal{B} \otimes \mathcal{A} = (B \times A, t', Z')$ where Z' is the set of Chu transforms (g, f) from $\mathcal{B}$ to $\mathcal{A}^{\perp}$ and $t'((b, a), (g, f)) = r(a, g(b))$. By adjointness, $r(a, g(b)) = s(b, f(a))$, so $t'((b, a), (g, f)) = t((a, b), (f, g))$. Thus up to bijection $\mathcal{A} \otimes \mathcal{B}$ and $\mathcal{B} \otimes \mathcal{A}$ have the same carrier, cocarrier, and matrix, that is, they are isomorphic as Chu spaces.

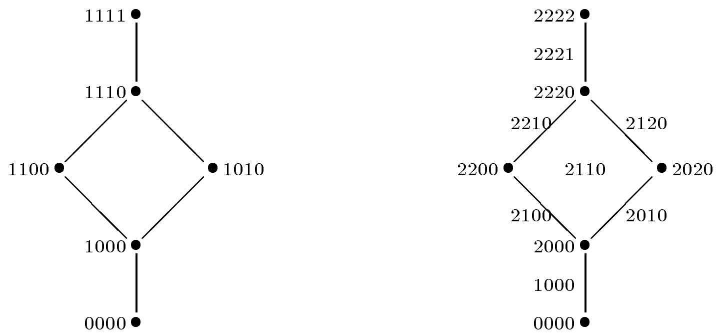
The intended operational meaning of orthocurrence is "flow-through," as with the flow of a system $\mathcal{A}$ of trains through a system $\mathcal{B}$ of stations, a river flowing along its bed, a system of signals flowing through a circuit made up of gates, or two particle systems in collision.

In addition to this operational interpretation, orthocurrence is intimately linked to the transformational structure of Chu spaces via the definition of the "internal homfunctor" $\mathcal{A} \multimap \mathcal{B}$, namely the Chu space whose points are the Chu transforms from $\mathcal{A}$ to $\mathcal{B}$. This can be defined either as $(\mathcal{A} \otimes \mathcal{B}^{\perp})^{\perp}$ or $(\mathcal{B} \otimes \mathcal{A}^{\perp})^{\perp}$, these definitions being isomorphic though not identical. Using $\mathcal{A}^{\perp\perp} = \mathcal{A}$ (transposition is an involution), the first definition makes $\mathcal{A} \otimes \mathcal{B}$ equal to $(\mathcal{A} \multimap \mathcal{B}^{\perp})^{\perp}$ while the second makes it $(\mathcal{B} \multimap \mathcal{A}^{\perp})^{\perp}$.

These standard relationships from linear logic can be related back to the operational interpretation as follows. A state (f, g) of $\mathcal{A} \otimes \mathcal{B}$ can be understood either as a variable state $f(a)$ of $\mathcal{B}$ where $a \in A$ is the parameter of variation, or as a variable state $g(b)$ of $\mathcal{A}$ as b varies over B. The requirement that $\mathcal{B}$ "looks the way it should" from each point (event) of $\mathcal{A}$, *and* dually $\mathcal{A}$ looks sane from each $\mathcal{B}$ vantage point, constitutes a joint constraint on the overall sanity of each state of $\mathcal{A} \otimes \mathcal{B}$.

For each choice of state (f, g), the function $t((a, b), (f, g))$, now with only a and b left to vary, may be thought of as an $A \times B$ crossword puzzle whose rows and columns must both spell out sensible words: the a-th row must represent a state of $\mathcal{B}$ while the b-th column must represent a state of $\mathcal{A}$. The following illustrates this.

A basic example of orthocurrence consists of two trains a followed by b travelling through two stations c followed by d. The two trains are represented by the Chu space $\begin{array}{c} a \\ b \end{array}\boxed{\begin{matrix} 011 \\ 001 \end{matrix}}$ and the two stations by $\begin{array}{c} c \\ d \end{array}\boxed{\begin{matrix} 011 \\ 001 \end{matrix}}$. Their tensor product is $\begin{array}{c} (a,s) \\ (a,t) \\ (b,s) \\ (b,t) \end{array}\boxed{\begin{matrix} 011111 \\ 000111 \\ 001011 \\ 000001 \end{matrix}}$, consisting of the four possible train-station pairs: train a at station b, etc. Ordering the six states (columns) by inclusion yields the poset shown on the left.



On the right is its higher-dimensional counterpart. As with concurrence, orthocurrence can be calculated with the Chu calculator. Set K to 2 or 3 as desired, then concatenate 1 with itself to form $\boxed{\begin{matrix} 011 \\ 001 \end{matrix}}$ when $K = 2$ and $\boxed{\begin{matrix} 01222 \\ 00012 \end{matrix}}$ when $K = 3$. In either case call that space p, then form $p \otimes p$. This will produce a 4×6 Chu space when $K = 2$, namely $\boxed{\begin{matrix} 011111 \\ 000111 \\ 001011 \\ 000001 \end{matrix}}$, and a 4×13 Chu space when $K = 3$, namely $\boxed{\begin{matrix} 0122222222222 \\ 0000011122222 \\ 0001201201222 \\ 0000000000012 \end{matrix}}$.

In the former case the 6 states constitute the six possible solutions to the 2×2 crossword puzzle for which the legal words both down and across are 00, 10, and 11, namely $\boxed{\begin{matrix} 00 \\ 00 \end{matrix}}$, $\boxed{\begin{matrix} 10 \\ 00 \end{matrix}}$, $\boxed{\begin{matrix} 10 \\ 10 \end{matrix}}$, $\boxed{\begin{matrix} 11 \\ 00 \end{matrix}}$, $\boxed{\begin{matrix} 11 \\ 10 \end{matrix}}$, and $\boxed{\begin{matrix} 11 \\ 11 \end{matrix}}$. Reading out a solution row by row yields the four elements of the corresponding column of the above 4×6 Chu space.

In the latter case the 13 states correspond to J.R. Allen's 13 primitive relationships between two intervals (All84). These are the 13 solutions to the 2×2 puzzle whose legal words both down and across are 00, 10, 20, 21, and 22. The event of a train arriving at a station corresponds to one of the four possible coincidences of endpoints of two intervals as they slide past one another. Later on we consider extensions of Allen's interval algebra.

Choice

The ***choice*** $\mathcal{A} \sqcup \mathcal{B}$ of processes $\mathcal{A} = (A, r, X), \mathcal{B} = (B, s, Y)$ is defined as $(A + B, t, X + Y)$ where $t(a, x) = r(a, x)$, $t(b, y) = s(b, y)$, and $t(a, y) = t(b, x) = 0$. That is, the events of $\mathcal{A} \sqcup \mathcal{B}$ are formed as the disjoint union of those of $\mathcal{A}$ and $\mathcal{B}$. Its states partition into two kinds: those assigning 0 to all events of $\mathcal{B}$, corresponding to choosing to execute $\mathcal{A}$, any state of which is permitted in $\mathcal{A} \sqcup \mathcal{B}$, and vice versa: choose $\mathcal{B}$ and set all events of $\mathcal{A}$ to 0.

This should raise no questions when $\mathcal{A}$ and $\mathcal{B}$ each have a zero state (all events 0). If however $\mathcal{B}$ say lacks a zero state, choosing $\mathcal{A}$ would seem to force $\mathcal{B}$ into a disallowed zero state. To resolve this we view the choice of $\mathcal{A}$ as forcing $\mathcal{B}$ into the zero state whether or not it is allowed.

A plausible alternative definition of choice is to take the dual point of view of processes as state spaces rather than event spaces and take choice to be sum in this dual view. This is the same as defining choice to be product in the normal event-oriented view, namely $(A \times B, t, X + Y)$ where $t((a,b),x) = r(a,x)$, $t((a,b),y) = s(b,y)$.

One difficulty with this definition is that the choice of a process having at least one event over a process with no events yields a process with no events, which is clearly not satisfactory. One might resolve this difficulty by postulating an additional dummy event for every process but then all the other operations would need to cater for the dummy event, complicating the process algebra.

Sequence

We begin by defining sequential composition or ***sequence*** $\mathcal{A};\mathcal{B}$ for an enriched notion of process, namely one furnished with subsets I and F of X containing respectively the initial and final states of the process. Then $\mathcal{A};\mathcal{B} = (A + B, r, Z)$ where $Z \subseteq X \times Y$ consists of those states (x, y) such that either $x \in F_A$ or $y \in I_B$.

The resulting process $\mathcal{A};\mathcal{B}$ should be similarly enriched, which we arrange by defining $I_{A;B} = Z \cap (I_A \times Y)$ and $F_{A;B} = Z \cap (X \times F_B)$. That is, (x, y) is initial in $\mathcal{A};\mathcal{B}$ when x is initial in $\mathcal{A}$, and is final in $\mathcal{A};\mathcal{B}$ when y is final in $\mathcal{B}$.

Provided we specify I and F for all primitive processes, and also by induction for all compound processes (with sequential composition as our first example), sequential composition is then defined on all processes namable with this process algebra. This raises two issues. First, there is the question of whether different names for the same process (as an unenriched Chu space) assign the same I and F. Second, sequential composition remains undefined for processes having no name.

Both issues can be resolved simultaneously by having a rule for determining I and F from the unenriched Chu space. For Chu spaces over 2 in their role as a generalization of ordinary event structures, one choice is to take F to consist of all maximal states, those which do not allow any further events to occur, and dually I to consist of all minimal states (which for ordinary processes will be just the empty state). One limitation of this approach is that it does not distinguish deadlock states, which might not be maximal yet which cannot make progress.

For Chu spaces over 3 however, adopting the same criterion raises questions that don't arise for 2. Consider for example a maximal state having an active event. Maximality of the state means that the active event cannot terminate. It would seem unreasonable to allow $\mathcal{B}$ to start while $\mathcal{A}$ still has active events. Thus one must answer the question, may F contain a state in which an event is active? With dual motivation the same question must be answered for I.

As another example, suppose $\mathcal{A}$ consists of one event a, which is permitted to be either unstarted or done but never active. Are the two states of a to be considered part of a run in which the active state of a is skipped, or simply two alternative states of a stationary process, one having no permitted state transitions? In the former case only the done state would be final, but in the latter both states could well be considered final.

Yet another example would be a maximal chain that is disconnected, which can happen for example when there are two consecutive cells where only one event changes state, namely from 0 to 2 without passing through 1, and furthermore no other connected chain properly extends the chain that runs up to that 0 point. When such a discontinuous chain arises we may interpret the discontinuity as resulting from a deadlock state. This suggests requiring that final states be connected. But disconnected is not a reliable indicator of deadlock, which can occur even when there is no extension of that chain that starts strictly above the point of deadlock.

It seems to us that circumstances should be allowed to dictate these answers: different needs may have to be met in different ways.

5. Beyond three-valued time

The essential difference between ordinary event structures and their higher dimensional counterparts is their logic. The former in effect have only two truth values, corresponding to "before" and "after" as the two times for events. Higher dimensional automata can be understood as introducing "during" or "middle," making them a sort of intuitionistic logic for concurrency. In this section we consider other states an event may be in besides merely before, during, and after, extending the temporal logic of higher dimensional automata beyond three values.

5.1. *Quantales*

Definition 5. A *quantale* (Mul86; Ros90; AV93) is a complete join-semilattice Q (and hence a complete lattice) with an associative binary operation $x \otimes y$, its *multiplication*, that distributes over arbitrary sups on both sides ($x \otimes \bigvee_i y_i = \bigvee_i (x \otimes y_i)$ and $(\bigvee_i x_i) \otimes y = \bigvee_i (x_i \otimes y)$, including the empty sup $x \otimes 0 = 0 = 0 \otimes x$). A quantale *with unit* contains an element 1 satisfying $1 \otimes x = x = x \otimes 1$. A quantale is called *commutative* when it satisfies $x \otimes y = y \otimes x$ for all x, y, and *idempotent* when it satisfies $x \otimes x = x$ for all x. ∎

Besides the trivial singleton quantale, the basic quantale is the two-element chain $0 < 1$, a semilattice, with meet as its multiplication and hence 1 as its unit. It is both commutative and idempotent. We associate this quantale with ordinary event structures. As another example, there is just one commutative idempotent quantale $0 < 1 < 2$ having 1 as its unit, which we have elsewhere called $3'$ (CCMP91), and which we associate with higher dimensional automata.

Every quantale admits a unique implication, residual, or "distance measure" $x \to y$, defined as the maximal z satisfying $x \otimes z \leq y$ (and a second dually defined implication $y \leftarrow x$ when not commutative, but our quantales will all be commutative). Implication for the quantale $3'$ can be straightforwardly calculated as satisfying $0 \to x = 2 = x \to 2$ for all x, $1 \to 1 = 1$, and otherwise $x \to y = 0$.

We now generalize the notion of poset (as used e.g. in specifying event structures). A

Q-***schedule*** $\mathcal{A} = (A, \to)$ is a set A of events together with a Q-valued distance measure $a \to b$ on A satisfying the *triangle inequality* $(a \to b) \otimes (b \to c) \leq a \to c$, and also $1 \leq a \to a$ when Q has a unit 1. A ***homomorphism*** of Q-schedules $\mathcal{A} = (A, \to)$, $\mathcal{B} = (B, \to')$ is a function $f : A \to B$ such that for all $a, b \in A$, $a \to b \;\leq\; f(a) \to' f(b)$, that is, an expanding (distance-nondecreasing) function.

For $Q = 2$, the unique two-element quantale, a Q-schedules is simply a preordered set $(A, \leq)$ where $\leq$ is a reflexive transitive binary relation (but need not be antisymmetric), while a homomorphism of such is a monotone function. For $Q = 3'$, our 3-element quantale for HDA's, a Q-schedule is a generalization of posets to what Gaifman and Pratt called a "prosset" (GP87), namely a structure $(A, \leq, <)$ with two transitive binary relations, one reflexive and the other irreflexive, such that $a < b$ implies $a \leq b$. The latter restriction is equivalent to having a single binary relation having three truth values, corresponding to "no restriction," $\leq$, and $<$. The multiplication $x \otimes y$ of $3'$ works as expected: $\leq \otimes <$, $< \otimes \leq$, and $< \otimes <$ are all $<$ while $\leq \otimes \leq$ is $\leq$.

As already remarked, every quantale Q comes with its own distance metric. This allows Q to be understood schizophrenically both as a quantale and as a Q-schedule. A ***state*** of a Q-schedule $\mathcal{A}$ is a homomorphism from $\mathcal{A}$ to Q understood in this way as a Q-schedule. For the two-element quantale a state can be seen to be simply an order filter or "upset." While this is the order dual of the notion of state for conflict-free event structures, this discrepancy is due merely to interpreting the distance $a \to b$ in a Q-schedule as the strength of constraint on whether b may precede a rather than vice versa.

The distance metric of $3'$ is given by the table $\boxed{\begin{smallmatrix}2&2&2\\0&1&2\\0&0&2\end{smallmatrix}}$, whose rows and columns are indexed by 0,1,2 (before,during,after) respectively. Whereas each of 0 and 2 is distance 2 from itself, 1 is at distance only 1 from itself. The effect of this is that when $b \leq a$ in a $3'$-schedule, a and b can both be in state 1 (during), but not when $b < a$.

Why quantales? Although Heyting algebras are more general than Boolean algebras, quantales are more general still, making quantale-based logics more broadly applicable than those based on Heyting or Boolean algebras. This is especially important for logics of time where $x \otimes y$ denotes the accumulated delay of the consecutive delays x and y, which might be truth values, integers, reals, etc., and $x \leq y$ denotes the relative logical strengths of such delays. Furthermore quantales have enough structure to support many useful constructions. These two considerations taken together make quantales very useful, a theme that we have expanded on in some detail elsewhere (CCMP91). One might call this general subject algebraic temporal logic.

5.2. *Generalizations of Allen's interval algebra*

F. Anger and R. Rodriguez (AR91; RA93b; RA93a) have generalized Allen's 13 primitive relationships or states to richer situations motivated by relativistic considerations. In Allen's interval algebra a pair of endpoints one from each interval can only stand in three relationships to each other: before, at, or after. The first Anger-Rodriguez extension adds a further relationship catering for the situation where the endpoints are so far apart in space that it is hard to be sure about the "at" case: the best one can say in this situation is that neither endpoint is in a light cone of the other, so one settles for "near."

The principal difference between "at" and "near" is that, whereas it is not possible for both endpoints of one interval to be simultaneously *at* one endpoint of the other, they can both be *near* that endpoint, in the sense that neither of them are in the light cone of that endpoint.

When interval endpoints are replaced by trains and stations, the corresponding distinction between at and near is that of stop and through trains. The stop train stops *at* the station while the through train only gets "near" it, from say the point of view of a passenger desiring to alight. A slow-moving observer may make its observations at a rate sufficient to guarantee that it never sees two stop trains simultaneously in the same station, but if two express trains rush through sufficiently close together the observer's window of observation may be wide enough to give it the impression that both trains were in the station at the same time. This provides a second more down-to-earth application for the Anger-Rodriguez refinement of Allen's interval algebra besides relativity.

Anger and Rodriguez calculate that there are 29 configurations of this refinement. These can be seen to correspond to the 29 states of the tensor product of the two-event eight-state Chu space $\boxed{\begin{smallmatrix}01232333\\00002213\end{smallmatrix}}$ with itself, with $\Sigma = 4$, as may be verified with the Chu calculator. Here 0, 1, 2, and 3, as the possible values of an interval endpoint in a given state, denote respectively before, at, near, and after, from the perspective of an observer watching that endpoint go by. The eight states are readily verified to be exactly the allowed combinations of states of two consecutive endpoints (or trains) as seen by an observer. The absence of a 11 state expresses the illegality of both endpoints of the interval being at an observer. However state 22 is legal, corresponding to both endpoints being near the observer. The significance of the other states, both those present and those absent, should be clear.

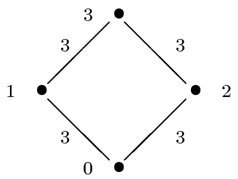
Orthocurrence computes the global behavior resulting from the observations of the two endpoints of one interval by each of the endpoints of the other interval. In doing so it extends the 13-cell HDA to a structure that is not obviously an HDA (since Σ is now 4 instead of 3), but which can nevertheless be viewed as an HDA. 22 of the 29 resulting cells can be described as all possible ways of changing 1 to 2 in the original 13 cells. In particular the 6 vertices contain no 1's and so remain unchanged. The 6 edges contain one 1 and hence are duplicated to become 12 edges. And the one surface contained two 1's, namely 3110, and hence is quadruplicated to become the four surfaces 3110, 3120, 3210, and 3220, bringing the total to 22. The remaining 7 cells are 2200, 2020, 2220, 2222, 3222, 3322, and 3232. All but the last 7 have all their faces; for example the surface 3210 has all four edges 3010, 3310, 3200, and 3230, but the surface 2200 has only edges 2000 and 3200, lacking 2300 and 0200.

On the face of it a Chu space over 4 is something beyond an HDA. However at and near clearly have something in common not shared with before and after, namely proximity to the station and a sense of activity (in both cases the observer on the station sees a train in action). Thus it is reasonable to take the dimension of a cell to be the number of 1's and 2's in it. For example the two three-dimensional cells are 2220 and 3222 while the only four-dimensional cell is 2222. This conflation allows us to view Chu spaces over 4 as cubical structures and hence higher dimensional automata.

Whereas the 13-cell HDA had trivial homotopy, the 29-cell is multiply connected and contains holes. This indicates the emergence of choice in the structure. For example the parallel edges 1000 and 2000 represent the choice of the first train stopping at the first station or rushing through. They are homotopically distinct because there is no connecting surface between them through which the 1000 path may be deformed to become the 2000 path. Similarly the four surfaces 3110, 3120, 3210, and 3220 represent four choices no two of which can be deformed into each other.

The question arises as to the appropriate quantale structure to impose on $\{0, 1, 2, 3\}$. We take it to be the commutative quantale satisfying $0 \otimes x = 0$, $x \otimes x = x$, and $x \otimes y = 3$ elsewhere, making the multiplication table $\boxed{\begin{matrix}0000\\0133\\0323\\0333\end{matrix}}$. The corresponding table for the residual $x \to y$ is $\boxed{\begin{matrix}3333\\0103\\0023\\0003\end{matrix}}$.

The nonzero entries of the latter table constitute the following automaton.



(By transitivity there is a transition from 0 to 3 labeled 3, not shown.) The self-loops at 0, 1, 2, and 3 are labeled respectively 3, 1, 2, and 3. In this framework an Allen interval is modeled as a schedule with two events a, b, corresponding to the initial and final endpoints of the interval, such that the distances, in the sense of (CCMP91), are given by the table $\begin{array}{cc} & a\,b \\ a & \multirow{2}{*}{$\boxed{\begin{matrix}30\\23\end{matrix}}$} \\ b & \end{array}$. A state of this schedule is as before a Q-homomorphism to Q. The 2 on the self-loop at state 2 of the automaton allows both a and b to be in state 2 simultaneously, since the distance between a and b is 0 or 2, both of which are less or equal to the 2 on the self-loop at 2, satisfying the expanding condition for a Q-homomorphism. The 1 on the self-loop at state 1 prevents a and b from being in state 1 simultaneously because the distance from b to a is 2 which is incomparable with the 1 on the self-loop. The 0 from a to b in the schedule forces a to precede b. In this way the eight states of the above Chu space representing an Allen interval are arrived at as all the homomorphims from the interval to Q.

An unsatisfactory aspect of this quantale is that it lacks a unit. And indeed any lower bound on $a \to a$ must be a lower bound on both $1 \to 1 = 1$ and $2 \to 2 = 2$, and hence 0, i.e. no bound at all. Question: does there exist a four-element quantale Q with unit such that Q as a Q-schedule has the desired effect of preventing simultaneity in state 1 while allowing it in state 2?

This generalization of the Allen algebra admits a further refinement dealing with the passage from "before" to "near". Light cones being well-defined, there must be an instant at which the observed endpoint makes this passage, call it "enter" (actually the endpoint is exiting from the "before" half of the light cone of the observer but entering the "near" region). Dually there is a passage out of the "near" region into the "after" light cone, call it "exit." No such refinement is necessary for the passage from "before" to "at" since "enter", "at", and "exit" all coincide, this being the apex of the light cone.

For this extension Anger and Rodriguez find 82 configurations. Once again these may be presented as the states of the tensor product of a certain Chu space with itself. We work with Chu spaces over 6, with 0, 1, 2, 3, 4, and 5 as shorthand for respectively before, enter, near, at, exit, and after. A little inspection will convince the reader that the two-event 15-state Chu space describing this situation is $\boxed{\begin{smallmatrix}012345245245555\\000000111222345\end{smallmatrix}}$. Deleting those columns containing either 1 (entry) or 4 (exit) changes it back to the 8-state space (after suitable renaming).

Visualizing this 82-state automaton as a higher-dimensional automaton is not as daunting as it looks. Take each of 1, 2, 3, and 4 as contributing 1 to the dimension, these being regions of activity, and 0 and 5 contributing 0 as before, being quiescent regions. The binary choice we had before, of being either at or near, remains binary but the near alternative is refined to the sequence enter;near;exit, with the expected combinatorial blowup leading to 82 states. The simple edge 2000 now turns into the three-edge chain 1000;2000;4000, and similarly for all other states containing 2's.

On the other hand we confess to some pessimism as to the existence of a suitable 6-element quantale as the appropriate structure for its alphabet. Notions of this complexity may be beyond the reach of the quantalic point of view.

There is something dissatisfying about assigning the *enter* (1) and *exit* (4) states the same dimension as that of the *near* state. But since they refer to activity of an event, they should not have zero dimension either. Intuitively they are the glue attaching the *near* state to its endpoints; as such they might be considered infinitesimal vectors tangent to the *near* edge at its endpoints. More generally it would seem reasonable to associate qualitatively different geometric elements of a higher dimensional automaton with distinct elements of Σ.

That orthocurrence exactly describes the above complex situations, while at the same time interpreting linear logic's tensor operation compatibly with its process algebra interpretation, indicates to us that it should viewed as among the central operations of process algebra, with concurrence as its closest sibling and choice and sequence nearby.

What is particularly appealing here is that the one definition of orthocurrence correctly predicts all three of the 13-element, 29-element, and 82-element variants of the Allen interval algebra. Moreover it created insightful geometric structure for them, allowing them all to be viewed as components of what is basically a diamond with a spike at each end.

This suggests that orthocurrence, originally introduced just for pomsets, is a robust notion that not only extends gracefully to higher dimensional automata but that can be relied on to accurately describe a wide range of types of interacting (flow-through) situations, and to furnish them with insightful structure into the bargain. That it also accurately models linear logic's tensor product, to the point of full completeness of at least the multiplicative fragment MLL (DHPP99), makes its general applicability all the more plausible.

References

J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

F. D. Anger and R. V. Rodriguez. Time, tense, and relativity revisited. In B. Bouchon-Meunier, R. R. Yager, and L. A. Zadeh, editors, *Uncertainty in Knowledge Bases: Proc. of the 3rd International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU'90*, pages 286–295. Springer, Berlin, Heidelberg, 1991.

Samson Abramsky and Steven Vickers. Quantales, observational logic and process semantics. *Math. Struct. in Computer Science*, 3:161–227, 1993.

M. Barr. *$*$-Autonomous categories*, volume 752 of *Lecture Notes in Mathematics*. Springer-Verlag, 1979.

M. Barr. $*$-Autonomous categories and linear logic. *Math Structures in Comp. Sci.*, 1(2):159–178, 1991.

M. Barr. $*$-autonomous categories: once more around the track. *Theory and Applications of Categories*, 6:5–24, 1999.

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.

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.

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.

Andreas Blass. A category arising in linear logic, complexity theory, and set theory. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 61–81, Ithaca, NY, June 1995. Cambridge University Press.

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.

H. Devarajan, D. Hughes, G. Plotkin, and V. Pratt. Full completeness of the multiplicative linear logic of chu spaces. In *Proc. 14th Annual IEEE Symp. on Logic in Computer Science*, pages 234–243, Trento, Italy, July 1999.

V. de Paiva. A dialectica-like model of linear logic. In *Proc. Conf. on Category Theory and Computer Science*, volume 389 of *Lecture Notes in Computer Science*, pages 341–356, Manchester, September 1989. Springer-Verlag.

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.

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.

J.L. Gischer. *Partial Orders and the Axiomatic Theory of Shuffle*. PhD thesis, Computer Science Dept., Stanford University, December 1984.

J.L. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61:199–224, 1988.

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.

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.

E. Goubault. *The Geometry of Concurrency.* PhD thesis, École Normale Supérieure, 1995.

E. Goubault. Schedulers as abstract interpretations of hda. In *Proc. of PEPM'95*, La Jolla, June 1995. ACM Press.

E. Goubault. Durations for truly-concurrent actions. In *Proceedings of ESOP'96*, number 1058, pages 173–187. Springer-Verlag, 1996.

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.

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.

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.

J. Grabowski. On partial languages. *Fundamenta Informaticae*, IV.2:427–498, 1981.

J. Gunawardena. Homotopy and concurrency. *EATCS Bulletin 54*, pages 184–193, October 1994.

V. Gupta. *Chu Spaces: A Model of Concurrency.* PhD thesis, Stanford University, September 1994. Tech. Report, available as ftp://boole.stanford.edu/pub/gupthes.ps.Z.

P.T. Johnstone. *Stone Spaces.* Cambridge University Press, 1982.

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.

A. Mazurkiewicz. Concurrent program schemas and their interpretation. In *Proc. Aarhus Workshop on Verification of Parallel Programs*, 1977.

A. Mazurkiewicz. Traces, histories, graphs: Instances of a process monoid. In *Proc. Conf. on Mathematical Foundations of Computer Science*, volume 176 of *Lecture Notes in Computer Science*. Springer-Verlag, 1984.

C.J. Mulvey. &. In *Second Topology Conference*, Rendiconti del Circolo Matematico di Palermo, ser.2, supplement no. 12, pages 99–104, 1986.

M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures, and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.

C. Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Press, 1986.

C.A. Petri. Fundamentals of a theory of asynchronous information flow. In *Proc. IFIP Congress 62*, pages 386–390, Munich, 1962. North-Holland, Amsterdam.

V.R. Pratt. On the composition of processes. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Programming Languages*, January 1982.

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.

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.

V.R. Pratt. Modeling concurrency with partial orders. *Int. J. of Parallel Programming*, 15(1):33–71, February 1986.

V.R. Pratt. Modeling concurrency with geometry. In *Proc. 18th Ann. ACM Symposium on Principles of Programming Languages*, pages 311–322, January 1991.

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.

V.R. Pratt. The second calculus of binary relations. In *Proceedings of MFCS'93*, volume 711 of *Lecture Notes in Computer Science*, pages 142–155, Gdańsk, Poland, 1993. Springer-Verlag.

V.R. Pratt. The Stone gamut: A coordinatization of mathematics. In *Logic in Computer Science*, pages 444–454. IEEE Computer Society, June 1995.

R. V. Rodriguez and F. D. Anger. An analysis of the temporal relations of intervals on relativistic space-time. In B. Bouchon-Meunier, L. Valverde, and R. R. Yager, editors, *IPMU'92: Advanced Methods in Artificial Intelligence - Proc. of the 4th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 139–148. Springer, Berlin, Heidelberg, 1993.

R. V. Rodriguez and F. D. Anger. Constraint propagation + relativistic time = more reliable concurrent programs. In *Proc. of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE-93*, pages 236–239, Edinburgh, Scotland, 1993.

K.I. Rosenthal. *Quantales and their applications.* Longman Scientific and Technical, 1990.

V. Sassone and G. L. Cattani. Higher-dimensional transition systems. In *Proceedings of LICS'96*, 1996.

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.

Y. Takayama. Extraction of concurrent processes from higher-dimensional automata. In *Proceedings of CAAP'96*, pages 72–85, 1996.

R. van Glabbeek. Bisimulations for higher dimensional automata. Manuscript available electronically as `http://theory.stanford.edu/~rvg/hda`, June 1991.

R. Van Glabbeek and G. Plotkin. Configuration structures. In *Logic in Computer Science*, pages 199–209. IEEE Computer Society, June 1995.

G. Winskel. A new definition of morphism on Petri nets. In *Lecture Notes in Computer Science*, volume 166. Springer-Verlag, 1984.

G. Winskel. A category of labelled Petri nets and compositional proof system. In *Proc. 3rd Annual Symposium on Logic in Computer Science*, Edinburgh, 1988. Computer Society Press.