November 1990

Report No. STAN-CS-90- 1342

Modeling Concurrency with Geometry

by

Vaughan Pratt

Department of Computer Science

Stanford University

Stanford, California 94305

To appear in POPL-91

Modeling Concurrency with Geometry

Vaughan Pratt †
Computer Science Department
St anford University
Stanford, CA 94305
pratt@cs.stanford.edu

Abstract

The phenomena of branching time and true or noninterleaving concurrency find their respective homes in automata and schedules. But these two models of computation are formally equivalent via Birkhoff duality, an equivalence we expound on here in tutorial detail. So why should these phenomena prefer one over the other? We identify dimension as the culprit: l-dimensional automata are skeletons permitting only interleaving concurrency, whereas ***true n-fold concurrency resides in transitions of dimension n.*** The truly concurrent automaton dual to a schedule is not a skeletal distributive lattice but a solid one! We introduce true nondeterminism and define it as monoidal homotopy; from this perspective nondeterminism in ordinary automata arises from forking and joining creating nontrivial homotopy. The automaton dual to a poset schedule is simply connected whereas that dual to an event structure schedule need not be, according to monoidal homotopy though not to group homotopy. We conclude with a formal definition of higher dimensional automaton as an n-complex or n-category, whose two essential axioms are associativity of concatenation within dimension aud an interchange principle between dimensions.

1 Background

A central problem in the semantics of imperative computation is the construction of convenient models of computation embodying the apparent aspects of both branching time and true or noninterleaving or causal concurrency.

Milner [Mil80] has made a convincing case for including the timing of nondeterministic choices in any comprehensive model of computation, distinguishing linear time (all choices made at the outset) from branching time (choices made on-the-fly to take into account the latest information). Park's not ion of bisimulat ion equivalence [Par81] has emerged as a particularly fine equivalence of labeled transition systems, though not so fine as to abandon the idempotence of choice. Two recent comprehensive yet complementary accounts are given by Bergstra and Klop [BK89] and their student van Glabbeek [Gla90], with the latter also addressing true concurrency in more detail.

The most natural computational setting for branching time would appear to be transition systems or state automata, where the branching is naturally represented by that of the automaton. These may be "unrolled" to yield synchronization trees so as not to represent other information. However Nielsen, Plotkin, and Winskel have defined event structures as an extension of schedules with a conflict relation to represent branching time, about which we will have more to say later.

Lamport [Lam86] and independently the present author argue against the interleaving model of concurrency on the ground that "exactly what is interleaved depends on which events of a process one takes to be atomic" [Pra86, p.37]. When a supposedly atomic action is "refined," which from a pragmatic viewpoint can be taken to mean "looked at more closely to reveal its substructure," the interleaving model can be seen not to have interleaved the subactions, having been forced to commit itself in advance to a particular level of granularity. This failure to interleave subactions is characteristic not of true concurrency but rather of mutual exclusion. Thus in an interleaving model the truly concurrent execution of two atomic events cannot be distinguished from the mutually exclusive execution of those events.

Castellano et al [CDP87] formalize this intuition with a simple example showing that interleaving semantics is not preserved under refinement (where refinement is defined as substitution of a complex behavior for an atomic

† This work was supported by the National Science Foundation under grant number CCR-88 1492 1.

behavior), but that partial order semantics is so preserved.

The natural computational setting for true concurrency is that of ***schedules*** or partially ordered sets of events. If branching time can be moved from automata to schedules [NPW81], can true concurrency go the other way?

Van Glabbeek and Goltz [vGG89] strengthen the result of Castellano et al in several ways. First they give a stronger example to show that step semantics (a form of concurrency intermediate between interleaving and partial orders, permitting multiple but synchronized actions at each clock tick) is, like interleaving semantics, not preserved under refinement. Then they extend the linear time result of [CDP87] to encompass branching time. Finally they point to ST-bisimulation, a delightfully simple extension of the notion of Petri net marking due to van Glabbeek and Vaandrager [GV87] that is preserved by refinement yet supports branching time, as an example of a semantics that models true concurrency and furthermore treats branching time, all without requiring partial orders.

Now ST-bisimulation comes within striking distance of the problem we just posed of moving true concurrency to the automaton model. However Petri nets are neither schedules nor automata, but a symmetric combination of both, in that Petri nets alternate the vertices of schedules, as transitions, with the vertices of automata, as places.

The main contribution of this paper is a geometric model of concurrency. This model completes the passage of true concurrency to automata by redefining the schedule aspects of a Petri net, namely its transitions represented in Petri nets as vertices, as higher dimensional transitions of an automaton! This also yields a model that is in our opinion mathematically more attractive than Petri nets, certainly those with unbounded place capacities.

There are two side contributions. One is a much needed tutorial on the duality of schedules and automata around which our introduction has revolved and which continues to play a role in the sequel. The other is the application of n-categories or n-complexes to concurrency theory, as a formal model of higher dimensional automaton, for which we contend they are more naturally adapted than other extant algebraic structures.'

[1] Among other recent applications of so-called Australian category theory to concurrency is an application of enriched categories to the abstract modeling of time a la Floyd-Warshall, presented to a category audience [CCMP89]. We regret not having had the opportunity to present this at POPL-88 or POPL-89.

2 A Geometric Model of Computation

Our thesis is that both branching time and true concurrency can be described together in a single geometric model. Branching time is represented literally by branchings of geometric objects, exactly as one would picture it. In particular we treat a transition system as a one-dimensional space consisting of edges (its transitions) meeting and branching at vertices (its states). True concurrency is represented by dimension: an n-dimensional cell (element of space) is used to represent the concurrent execution of n sequential processes, and its boundaries represent the starting or halting of some of those processes.

Our geometric view is closely related on the one hand to ST-bisimulation [vGG89], and on the other to Papadimitriou's geometrical model for database concurrency control [Pap86, chap.6].

To be cryptically succinct, we propose to extend interleaving concurrency to true concurrency by filling holes, and then to extend true concurrency to branching time by putting some of those holes back.

So what is a hole? Consider two automata, each having two states and one transition and accepting the respective regular sets a and b, each consisting of one unit-length string. With interleaving concurrency the concurrent execution of these two automata is their product, a square whose four sides represent four transitions, accepting the regular set $ab + ba$ as in Figure B' four pages hence.

This automaton contains a hole, namely the interior of the square. To formalize this, embed the automaton, treated as a graph, in the Euclidean (real) plane. For definiteness locate its states at the lattice points $(0,0)$, $(1,0)$, $(0,1)$, and $(1,1)$ (the corners of the unit square $[0,1]^2$; y is oriented negatively in 8'). Take the initial state to be the origin, and take the four transitions to be the four sides of this square. The hole is then the interior of the square.

To "fill the hole," take the interior of the square, a surface, as the ninth component of this product automaton. The four vertices, ***qua*** states, and the four edges, ***qua*** transitions, comprise the other eight ***cells*** of what we shall refer to as a cell ***complex.***

These nine cells represent the nine possible "states" of the concurrent automaton. Four states are completely idle (states in the usual sense), four have one constituent automaton active, and one has both constituents active. This last may be described as a two-dimensional or "superficial" transition. It provides a notion of "joint transi tion" of two processes.

Van Glabbeek and Vaandrager [GV87] have introduced ST-bisimulation, in which not only places (*Stellen*) of Petri nets but transitions (*Transitionen*, whence ST) are marked in a state (an assignment of tokens to vertices). (It is extraordinary that this simple and intuitively clear extension of the notion of Petri net state has not been proposed before!) The relationship with ST-bisimulation may be easily seen by considering the corresponding example for Petri nets. As usual a one-transition automaton becomes a Petri net simply by converting its transition from an edge to a vertex and adding two edges to represent respectively the pre-event or input arc and postevent or output arc, each incident on the one transition. The concurrence of two such nets is "smaller" than that of the corresponding automata, being just their juxtaposition, having for its vertices not the Cartesian product of the vertices of the underlying nets as was the case for the automata but rather their disjoint union.

If we mark the initial place of each of the two components of this juxtaposition and then play the usual "token game ," we obtain four possible markings of places, corresponding to the four vertices of the square automaton. If however we extend the token game as in ST-bisimulation to permit transitions to be marked, we obtain an additional four markings each involving one transition (the four edges of the square), together with a single marking involving both (the interior of the square), completing the promised correspondence.

There are three trivial generalizations we can make immediately. First we may increase the number of automata executing concurrently. With three automata we obtain a cube in the obvious way, with four a 4-cube, etc. We refer to the d-dimensional elements of such a complex as d-cells, with O-cells or points corresponding to the old notion of state, l-cells or edges the old notion of transition, and the higher-dimensional cells constituting a new notion of concurrent transition.

Second, we may let the i-th automaton, for $0 \leq i < d$, run for m_i transitions, provided its graph forms a chain so that it accepts just one string, of length m_i. Our unit cube then expands to a larger complex of size $\prod_i m_i$.

Third, we may label transitions. If we associate alphabet Σ_i with the i-th sequential automaton for $0 \leq i < d$, we may label its edges in the standard way for automata. To extend this to higher dimensions we require for each subset of this set of d alphabets the alphabet formed as the product of that subset, yielding 2^d alphabets. Each n-cell is then labeled with an n-tuple of labels from the appropriate alphabet, namely that corresponding to the subset of automata whose activity is represented by the cell. All completely quiescent states (O-cells) are labeled with the unique 0-tuple, indicating the absence of activity. l-cells are labeled with 1-tuples, as in an ordinary automaton. 2-cells are labeled with pairs (a, b) indicating the joint execution of transitions a and b, and so on for higher dimensions.

Less trivial is the next generalization, which imposes order constraints. For any two transitions of different automata we may require that one not start until the other has finished, a precedence constraint. We shall call such a collection of constraints a ***schedule,*** the term used in (inter alia) the Macintosh world for PERT charts or pomsets.

With this generalization the second trivial generalization now becomes redundant, since we can achieve the effect of a sequential automaton having m transitions by using m concurrent one-transition automataconstrained to execute in a specified sequential order. To simplify the model we therefore withdraw generalization two. This restricts our basic automata to single-transition automata, more conventionally called *events.* We may now describe each precedence constraint as holding between such events i and j, without having to further specify a particular transition within each event. These constraints, which we write as $i < j$, then amount to a partial ordering of the set (or multiset in the labeled case) of d events.

We saw already that the square (2-cube) had nine cells. More generally the d-cube has 3^d cells. The number three arises as the possible states of an event: initial, transition, final, or 0, T, 1 for short. Each cell can then be identified as a d-tuple over $\{0, T, 1\} = 3$. ***The standard interpretation of $i < j$*** is ***to exclude all cells*** save ***those whose j-th event*** is ***0 or whose i-th event is 1.***

In place of $\{0, T, 1\}$ we may take the unit interval [0, 1] on the real line, whose interior (0, 1) corresponds to T alone and whose endpoints are the correspondingly named elements of $\{0, T, 1)$. Then the d-cube becomes the unit cube $[0, 1]^d$ in d-dimensional Euclidean space $\mathbf{R}^d$, having a continuum of points rather than just 3^d. Now one quite reasonable interpretation of $i < j$ is to restrict to the polyhedral subspace of the cube consisting of those points whose i-th coordinate is greater than their j-th coordinate. This corresponds to permitting events i and j to run concurrently but without letting j get ahead of i. For any partial ordering of events this subspace is convex and can be easily shown to have volume $k/d!$ where k is the number of linearizations of the partial order, being the (essentially) disjoint union of k tetrahedra one per linearization.

However this is not the proper analog in $\mathbf{R}^d$ of subspaces of 3^d, since it cuts faces into tetrahedra. The appropriate real-valued analog further restricts the subspace to those points such that either the j-th coordinate is 0 or the i-th coordinate is 1. But this is exactly how we expressed the condition for 3^d. In fact $i < j$ was interpreted without mentioning T at all, being expressed solely in terms of the initial and final states of an event, regardless of what structure we impart to its interior. Unlike the interpretation of $i < j$ in the previous paragraph, the subspace of $\mathbf{R}^d$ given by the standard interpretation is not convex. However, as will be seen to be important shortly in our approach to nondeterminism, it contains no holes.

3 Schedule-Automaton Duality

With this last generalization we have passed from unscheduled to scheduled activity, the latter being the essence of the pomset model [Gra81, Pra82]. From a mathematical perspective we have passed (at least in the finite case) from the Birkhoff-Stone duality between sets (of events, as points of a finite discrete topological space) and Boolean algebras (of states) to the much richer Birkhoff duality between posets (still of events) and distributive lattices (still of states).

The Birkhoff duality is becoming better known in the CS community of late. However there remains considerable confusion in both the mathematical and CS literature over the difference between Birkhoff duality and Stone duality, perpetrated in our opinion by an unwarranted enthusiasm for topological methods to the exclusion of combinatorial. It is very helpful to see these distinctions clearly when working with our geometrical model of concurrency. Hence we give here an overview of this duality and its application to concurrency.

Sequential computers alternately work during a transition and then rest up at a state. This scenario is conventionally rendered as a graph whose vertices represent either transitions or states. If transitions then we have a PERT chart or ***schedule,*** and the edges of the graph denote precedence relations, possibly labeled with durations indicating bounds on the time from one transition to the next. If states then we have a machine or *automaton* whose edges are transitions from state to state, possibly labeled with attributes of the transition. But which picture is the right one?

The Petri net answer is neither: transitions and states should be granted equal rights by both being vertices. These then serve as respectively the conjunctive and disjunctive elements of an intriguing logic of concurrency.

The duality theory answer is both: for at least a certain class $\mathcal{S}$ of schedules and class $\mathcal{A}$ of automata they depict the same scene because $\mathcal{S}$ and $\mathcal{A}$ are equivalent. Not isomorphic, which would imply a l-l correspondence between the ***elements*** of $\mathcal{S}$ and A, but equivalent in the sense of a l-l correspondence between the ***isomorphism*** classes of $\mathcal{S}$ and A. (This is exactly the category theoretic notion of equivalence.)

Here is a simple special case of this equivalence.

A finite schedule S that is just an unordered set of n jobs (transitions) to be done in parallel is very easy to compile into an automaton. The automaton is just the power set 2^S drawn in the standard way with the empty set at bottom as the start state and S at top as the final state, and edges between just those sets differing by exactly one job, with that job labeling that edge. For the automaton to be in state Y means that the set of jobs done thus far is Y. This is the ***Hasse diagram*** of its lattice, i.e. the smallest DAG whose transitive closure is the inclusion order $\subseteq$ between subsets. This lattice is of course a Boolean algebra, meaning a distributive lattice with a complement operation.

It is helpful to regard the states of the automaton as 2" bit vectors of length ***n,*** with bit x being 1 in state Y just when job x has run by the time that state is reached. Then the set operations Y $\cup$ Z and Y $\cap$ Z are bit vector operations such that in each bit position they are just $\vee$ and A.

Decompilation is traditionally harder than compilation, but in this case decompilation is just as easy as compilation. Given such an automaton A, form its power automaton 2^A, consisting of certain sets of states of ***A.***

Now if an automaton were simply a set of states, 2^A would mean the power set of ***A.*** But that would be huge-and fortunately wrong. The right way to form the power widget 2^W of a widget W is to take all widget maps from W to 2. This works provided there's a sensible way to construe 2 as a widget. The automaton A is a lattice, and luckily the poset 2 is schizophrenic enough to be also the lattice $0 \leq 1$ of truth values. Hence we take for 2^A all lattice maps (homomorphisms) from $A = 2^S$ to 2, meaning functions that preserve the lattice structure, i.e. $f(\emptyset) = 0$, j(S) = 1, $f(Y \cup Z) = f(Y) \vee f(Z)$, and $f(Y \cap$ Z) = j(Y) A j(Z). It can be easily seen that preserving this much implies preserving complement as well, so these are also Boolean algebra maps.

So what maps does this give us? One function that satisfies the desired conditions above is the predicate f_x that tells of each state whether job x has run yet. But looking at A as 2^n bit vectors, this is just the function that watches bit x. As long as bit x works reliably, when Y $\cup$ Z is formed bit x will appear to be computing $Y_x \vee Z_x$, and dually for $\cap$. But this is what the above conditions require. So these functions are lattice maps. They are in fact the n projections of 2^X onto 2, 2^X being the product of n copies of 2. That they are different can be seen by their behavior on singleton states. Hence these maps form a set isomorphic to X, that is, having cardinality $|X|$. That it is only isomorphic and not equal is why the duality is only an equivalence and not an isomorphism!

And these are all the lattice maps there are from A to 2. For j can't be 0 on every singleton or it would make j(S) = 0, S being finite. But j can't be 1 on two or more singletons since that would make j(0) = 1. So j must be 1 on exactly one singleton {x}. But now $f(\{x\}$ n Y) = j({x}) $\wedge$ j(Y) = j(Y) whence j(Y) = 1 exactly when x $\in Y$, making j = f_x.

But the theorem is that decompilation back to schedules works for an isomorphism class of automata, so we aren't allowed to refer to states as singletons per se. However this is no problem: we can spot the singletons by context as being just those states immediately following the start state, namely the atoms of the Boolean algebra. So we revise the above argument to work for all automata isomorphic to 2^S by substituting "atom" for "singleton."

So we now have a l-l correspondence between all isomorphism classes of finite sets and some set of isomorphism classes of finite Boolean algebras. But the reader well knows (though it is some work to prove) that every finite Boolean algebra is isomorphic to the power set of some finite set, and so we have the promised equivalence.

But this very special case of Birkhoff-Stone duality is as boring as the natural numbers, since the finite sets fall into isomorphism classes according solely to their cardinality n, one class per number, and correspondingly there is just one isomorphism class of Boolean algebras of cardinality 2".

So the picture so far is that if schedules are just unordered sets of jobs, there is, up to isomorphism, just one schedule of each size n and one matching automaton of size 2^n.

This duality can now be spiced up in two essentially orthogonal ways, a combinatorial one due to G. Birkhoff [Bir35] and a topological one due to M. Stone [Sto36]. Remarkably, that these ways were orthogonal passed unnoticed until pointed out by H. Priestley in 1970 [Pri70].

Keeping everything finite, Birkhoff duality generalizes the discrete schedules to partially ordered schedules, and generalizes the automata to distributive lattices. On the other hand, keeping the automata Boolean, Stone duality generalizes everything to the infinite case. In order to allow every Boolean algebra to be viewed as the compilation of some schedule Stone generalizes schedules of jobs to schedules of sets of jobs called (nowadays) Stone spaces. Instead of running individual jobs one must now run sets of jobs at a time, and only in those combinations that are permitted. The automaton produced by compiling such a schedule will now be a proper Boolean subalgebra of the power set of the set of all jobs in the schedule, due to the schedule restrictions eliminating some states, e.g. those containing only finitely many jobs.

Stone did extend his Boolean duality to distributive lattices [Sto37], but purely topologically rather than via the more natural blend of order and topology devised by Priestley. As Rota put it, "Stone's representation theorem of 1936 for distributive lattices closely imitated his representation theorem for Boolean algebras, and as a consequence turned out to be too contrived." [Rot73]

Priestley simply equips the schedule with both Birkhoff's partial order and Stone's topology to make it a partially ordered Stone space, a set bearing two structures, that of a partial order and that of a topology, Stone's in this case. The partial order deals with the absence of complementation, generalizing Boolean algebras to distributive lattices, and dually on the schedule side, sets to posets. The Stone topology, which resides on the schedule side, caters to the phenomenon whereby a countably infinite Boolean algebra, and by the same token a distributive lattice, can have a countably infinite subalgebra not isomorphic to its parent, by forbidding those combinations of jobs (i.e. subsets of X) corresponding to missing elements of said subalgebra.

Thus Birkhoff's and Stone's halves of this beautiful duality theory are essentially independent. Both halves have potential computational significance. Birkhoff duality is relevant to scheduling, the main thrust of this paper. Stone duality is relevant to continuous situations, e.g. parallel solutions to stock-cutting problems where the regions can get arbitrarily small without ever becoming points. It may also prove fruitful in reasoning about large systems where the number of jobs makes it uneconomical or infeasible for a scheduler to deal with individual jobs, forcing it to batch them, the chief difficulty here being that of translating the logic of infinity down to large but finite numbers.

For our geometry-of-concurrency purposes however we currently have no application for Stone duality. In this paper we focus on schedules structured by a partial order, and hence on Birkhoff duality, a finite and pleasantly combinatorial phenomenon.[2]

Finite posets are far less boring than finite sets. Whereas the number of isomorphism classes of sets of each cardinality goes 1, 1, 1, 1, 1, 1, 1, . . . the corresponding sequence for posets goes $1, 2, 5, 16, 63, 318, 2045, \ldots$. So when you are scheduling 7 jobs, there are 2045 different ways to schedule them, one of which is the discrete order, no precedence constraints, at one extreme and another of which is the linear order (this is only up to isomorphism, or we would have $7!$ linear orders) at the other.

Despite this enormously richer software library (and this is before we've labeled the vertices to say what each job does, being only up to isomorphism), the story about schedules and automata remains almost completely unchanged! To compile S form $A = 2^S$. To decompile A back again form S = 2^A.

It will become clear shortly that these definitions as they stand are time reversing in both directions. We couldn't see this before because sets and Boolean algebras are both isomorphic to their duals. We fix this by reversing the input to each exponentiation. Thus

[2]Some say topology should become pointless, others that it always was. Poincaré, the father of combinatorial topology, said of Cantor's *Mengenlehre* that it was a disease from which later generations would regard themselves as having recovered. It is a sporting bet that we will recover sooner from the internal combustion engine.

we actually compile with $2^{S^{op}}$ and decompile with $2^{A^{op}}$, where X^{op} denotes the order dual of X.

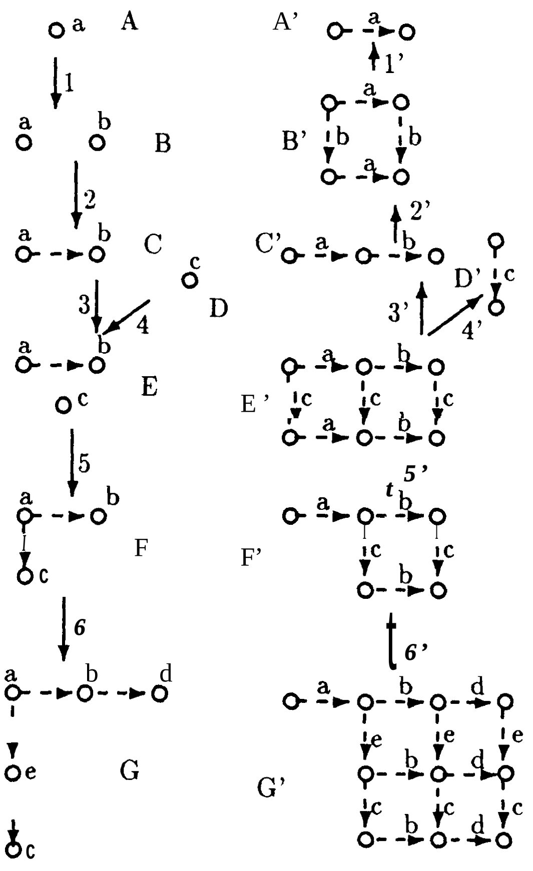
Remarkably, nothing about our reasoning for unordered schedules needs be changed until we get to the matter of whether there are any other maps besides the f_x's. The arguments involving singletons are no longer sound, because there may now be fewer singletons, in the extreme case only one if some job has to go first. But we achieve the same effect by taking, for each x, the state $\{y|y \leq x\}$ (a so-called principal order ideal because it is generated by one element, x), clearly in A. This state is the earliest moment at which x could have run. To make this abstract (remember, we claim only an equivalence, not an isomorphism), the analog of "atom" is now obtained by noticing that these states are exactly those that aren't the union, or rather join since this is going to be for lattices, of two smaller states, i.e. they are ***join-irreducible.*** The states we've picked can't be so represented because x has to be in one of them and there is no smaller such state. Those we left out are not principal and hence have at least two maximal elements, in which case they can be represented as the union of the principal order ideals generated by each of their maximal elements.

We now observe that our f_x's are all distinct because each first becomes true at its own join-irreducible state. Furthermore when we compare the f_x's coordinatewise by inclusion we find $f_x \subseteq f_y$ (which we can read as "if x has happened then y has happened") just when $y \leq x$ in S. So this ordering of the f_x's coincides with the original order on S, showing that 2^A is isomorphic to the original poset S.

This equivalence of the classes of finite posets and finite distributive lattices is actually a contravariant equivalence of categories. This means that the correspondence between posets and lattices extends to their maps, with "contravariant" meaning that corresponding arrows go in opposite directions, i.e. the poset map $f : p \rightarrow q$ corresponds to a lattice map $f' : q' \rightarrow p'$ between the corresponding lattices.

The adjacent figure contains two diagrams, one in the category of finite posets, on the left, the other on the right in the category of finite distributive lattices. Each diagram has seven objects A through G, primed on the right. The six poset maps on the left, numbered 1 through 6, go down while the corresponding lattice maps on the right go up.

Object A is a one-job schedule while A' *is* the corresponding one-transition automaton. The map $1 : A \rightarrow B$ is an inclusion which adds job b while the corresponding map $1' : B' \rightarrow A'$ is a projection, projecting out b. $2: B \rightarrow$ C is an augmentation of the discrete order in B to a linear order in C. This constraints knocks out the bottom left corner of B', represented by an inclusion $2' : C' \rightarrow B'$. C' is the top and right edges of B', straightened out. Now a new element c is added in on the left; the relevant piece of the diagram is a coproduct E whose inclusions are maps 3 and 4. The dual of a coproduct is a product; $3' : E' \rightarrow$ C' projects out c while $4' : E' \rightarrow D'$ projects out a and b. Now 5 augments with $a \leq$ c, again knocking out a corner. Finally 6 adds two new elements, d after b and e spliced in between a and c.



One may well imagine that distributive lattices would be harder to visualize than Boolean algebras. This need not be so. The key to visualizing a distributive lattice $2^{S^{op}}$ is to interpret the precedence constraints of S in $2^{S^{op}}$. What each constraint $x \leq y$ does is to delete all states violating that constraint, namely those containing y but not x. The remaining states retain their original positions, preserving the geometry of the cube. Thus a finite distributive lattice is simply an eroded cube.

One immediate application of dualities of this kind is that it maps colimits to limits in passing from schedules to automata. This follows from the manner in which duality is obtained as the exponentiation D^x, for a suitable dualizing object D, in the category of posets, yielding for example $D^{x+y} = D^x \times D^y$. Casley et al [CCMP89] outline a language for concurrency a number of whose operations are describable as colimits in a category of schedules; these are therefore carried to the corresponding limits in the dual situation. The language is defined for schedules having various notions of time, such as real-valued time[3], set-valued time, and causal-accidental time, with posets having merely truth-valued (0 and 1) time.

4 Event Structures.

The next generalization takes us to the event structure model [NPW81].[4] Here we specify a symmetric irreflexive binary relation i#j on events, whose meaning is that not both of i and j may run. In the presence of scheduling we require that i#j and $j < k$ imply $i\#k$, that is, conflict is a ***persistent*** condition. An event structure is then such a set $(V, <, \#)$, V being the set of events.

The effect of the scheduling order < was to erode the corresponding automaton by deleting certain cells, converting it from a cube or Boolean algebra to a distributive lattice. (Every finite distributive lattice arises in this way.) The effect of conflict # is to further erode the automaton, this time only from the top down (a consequence of the persistence of conflict), deleting just those cells such that for some i#j neither the i-th nor j-th component of that cell is 0, corresponding to states in which both i and j are either running or have halted'.

In these generalizations the filled-in holes of a given distributive lattice have not themselves added any information, since they can be reconstructed from the distributive lattice by first passing to its dual poset, taking this to be a scheduling of events, and constructing the filled-in lattice from the poset by the procedure given above, as a subspace of either $\mathbf{3}^d$ or $\mathbf{R}^d$ as desired. However event structures currently provide the largest known class of schedules to date that we are aware of for which this holds.

5 Monoidal Homotopy

We now define a preliminary notion of monoidal homotopy and use it as the basis for a definition of "true nondeterminism." It is expected that a more formal notion of monoidal homotopy will be able to be based on the n-complex model of higher dimensional automata presented in the concluding section.

A finite distributive lattice is both ***confluent*** and ***simply-connected*** in the following sense. Given any two (ascending) paths in the lattice (defined as just the vertices they pass through, the edges can be inferred), the lattice property uniquely determines two vertices, respectively the meet and join of the union of those paths, in turn determining (not uniquely) extensions of the paths to those points to give them a common beginning and end, this being confluence. And the distributivity property uniquely determines the appropriate filling-in of the "holes" in the lattice, yielding a simply-connected space through which the two paths may be continuously deformed into each other in the intuitively obvious way, provided we regard this space as Euclidean, i.e. a subset of $\mathbf{R}^d$ rather than 3^d. By suitably defining the discrete analog of continuous deformation we may achieve the same effect in $\mathbf{3}^d$, addressed below, but for now our intuition with Euclidean space will suffice.

In topological language, any two paths with common endpoints in such a simply-connected space are automatically ***homotopic,*** an equivalence relation on paths [Bro88, Whi49, Whi78]. They become nonhomotopic when a hole appears somewhere in the space between them to inhibit their deformation into one another.

Homotopy is ordinarily studied for spaces the movements in which form a group under composition, where homotopy is inhibited only by holes. The typical irreversibility of computation however calls for a monoid. This in turn calls for a generalization of the notion of homotopy from grouphomotopy to monoid-homotopy, leading to other ways of inhibiting homotopy. For now we settle for the following stopgap notion, defined only for our present setting of distributive lattices.

Given two ascending paths p, q in a distributive lattice L having common endpoints, and given a subspace X of the Euclidean fill-in of this lattice, we say that p and q are ***group-homotopic*** with respect to X when they are homotopic in X in the standard sense of topology. Given two ascending paths p, q not necessarily having common endpoints, we say that p and q are ***monoid-homotopic*** with respect to X when they extend to paths that are group-homotopic with respect to X.

This generalizes homotopy by permitting paths without common endpoints, the application of which will be apparent shortly. Henceforth by "homotopic" we shall mean monoid-homotopic.

We think of decisions involving choices between paths as essential just when the paths are not homotopic, and hence of filled-in distributive lattices as deterministic or choiceless automata. Nondeterminism is the condition in which there exist paths not homotopic to each other; we think of the choice between nonhomotopic paths as an essential decision.

Conflict introduces nondeterminism into distributive lattices, by selectively destroying confluence in the forward (upward) direction, by making "notches" in the "top" of an automaton. Two paths on opposite sides of such a notch are committed to their respective sides and cannot be extended upward to group-homotopic paths,

[3] Infinitesimally fine interleaving looks like true concurrency until one notices it is taking time L_1 or x + y. Truly concurrent real time requires only L_∞ or $\max(x, y)$, a theme developed in future work.

[4] As Girard [Gir87] has noticed to good effect in developing the notion of coherent space, this generalization can be made independently of the preceding one.

Without scheduling these are Girard's coherent spaces, or rather coherent algebras if we adhere to the duality theorists' naming convention for such dualities.

there being no join to extend to; hence they are not homotopic in our broader sense. (Since the effect of persistence is to limit notching to the top, making such an automaton an order ideal of the distributive lattice it was obtained from, it follows that for conflict it would make no substantial difference if we restricted the definition of monoid-homotopy just to paths with common beginnings, since any two paths can be extended to have a common beginning.)

6 Benefits of a Geometric Model

The biggest advantage for us of this geometric perspective is that it makes the duality of schedules and automata more convincing. If by leaving two events unordered we have supposedly represented their true concurrency, how do we then explain the automaton dual to this discrete poset, namely the four-state four-transition square already discussed? The latter is *exactly* the automaton one would expect the interleaving model to produce!

One answer to this puzzle is to say that such a square *always* indicates true concurrency, and to rep resent ij + ji (namely the mutual exclusion of i and j) by not letting the ij and ji branches rejoin when done.

Our answer is instead to fill in the square to indicate true concurrency and simply leave it hollow to indicate mutual exclusion.

We see this approach as having the following benefits.

(i) Naturality. Lines, surfaces, and volumes, attached to each other in possibly branching ways, are familiar and easily visualized concepts, requiring relatively little mathematical sophistication to at least begin to appreciate.

(ii) Flexibility. A greater variety of situations may now be represented. For example we now can choose whether ij and ji are to rejoin immediately, a little later, or never. For reason (iii) (complexity) we will in general prefer the first, but we may on occasion find it meaningful to "rip upwards" a little or a long way.

(iii) Complexity. An m x n rectangle contains only mn squares but $\binom{m+n}{m}$ paths (between adjacent lattice points) from start to end. Even if we associate information with every square (e.g. by forbidding some), the geometric representation of this information is of polynomial size in the sides whereas the interleaving representation, in terms of the paths, is of exponential size.

(iv) Continuity. Tearing a picture into little strips is an inherently discontinuous operation. Decomposing a d-dimensional space as a set of paths through it is somewhat akin to identifying a TV picture with its one-dimensional representation as a sequence of scan lines. A TV serviceman who never saw a screen but only the scan lines laid end to end may find this perspective natural, but certainly not a viewer.

We programmers tend to be more like servicemen than viewers. Years of thinking about individual instruction sequences have conditioned us to believe that the sequence is the correct object of study. If instead we were to assemble many such sequences, properly aligned, into a single picture, the resulting shapes that would materialize would make it much easier to reason about our processes.

(v) Algebraic structure. Geometry is an "algebraically mature" subject, and we are optimistic that some of this algebra will rub off on our geometrically based model of computation. Thus far we have in fact elicited some algebraic structure, but mainly from that of event structures via duality. We are presently working on replacing this convenient but limited derivation with an elementary development of homology based on monoids instead of groups to yield what we feel will be a more comprehensive and convincing algebrization. To be more precise, we base it not exactly on monoids but rather on their generalization to n-categories [Ehr63] following roughly the lines of Street [Str87], as more ap propriately expressing the dimensional aspect.

7 Extensions of Event Structures

Despite the fact that event structures were introduced more than a decade ago [Win80], there have been no further generalizations of event structures where the duality has been maintained. Yet there are many phenomena of computation each describable with a suitable extension of this model, some of which we give now. Thus far we have been able to make these extensions only on the algebra or automaton side of the duality, where the higher-dimensional cells are described explicitly. The role of the cells in making these extensions may once again be made inessential, albeit almost certainly still convenient, if a suitable extension of the duality can be found, a problem we return to later.

Although event structures themselves reside on the topology side of the duality, the completeness of the duality up to this point permits us to extend starting from either side. All extensions considered here will be to the algebra or automaton side, where the nature of the higher-dimensional cells, of dimension two and up, is clear.

***Mutual* Exclusion.** The mutual exclusion of events i and j is the condition that i and j not overlap. In ST-bisimulation terms it is the naturally expressed condition that in no state are transitions i and j both marked. The corresponding requirement for the geometric model is of course that no cell exist whose i-th and j-th components are both T (for the discrete or $\mathbf{3}^d$ model) or both in the open interval $(0, 1)$ (for the Euclidean model).

As defined here mutual exclusion is not persistent. We may have $i \# j$ and $j < k$ without $i \# k$ since k may not need the resource that i and j are presumably competing for. After i and j are done, in whichever order, the two branches of the computation may rejoin, corresponding to the rest of the computation not remembering the order.

A persistent version of the mutual exclusion of i and j can be defined using two copies of each of i and j, arranged as ij + ji, with each of the two events of ij in conflict with each of the two events of ji. Persistence of conflict means that the commitment to one of ij or ji is permanent; two paths making this decision differently cannot subsequently rejoin.

This brings us to the question, to persist or not to persist?

One advantage of persistence is that it permits our geometrical model to be dispensed with. However to the extent that geometry offers the several benefits we claimed earlier, all of which are compromised more or less seriously by insisting on persistence, this is at the same time a serious disadvantage of persistence.

For example persistence destroys the complexity advantage of geometry. If the computation keeps splitting into alternatives without ever rejoining, its size grows exponentially. Rejoining permits a much smaller model of a given computation when there is much branching. (The dauntingly many alternative parallel universes of science fiction may similarly be dramatically reduced in number, say to around 10^{120} as a match for the number of particles in any one such universe.)

It is also particularly distressing to see a local mutual exclusion requirement, which we can represent with just a small hole, be blown up into a giant tear in the fabric of the computation simply due to the requirement of persistence.

For these and similar reasons we prefer not to insist on persistence, and to permit the expression of nondeterminism not only by absence of confluence (the status quo) but also by the presence of holes even where confluence is possible (our proposed extension to the status quo).

Communication. Consider a rectangle representing two processes X and Y running in parallel. A communication from X to Y consists of a transmission T by X and a receipt R by Y. These two events together determine a point in the rectangle, which we may think of as the communication. Taking this point as the origin, divide the rectangle into four quadrants. Thinking for the moment in terms of computations being one-dimensional paths, a particular computation path passes through three of these, avoiding the quadrant in which the message would have been received before it was transmitted. The middle quadrant the path passes through, diagonally opposite the excluded quadrant, represents the period during which the message is in transit.

The effect of many communications is to erode the upper left (early X and late Y) and lower right (late X and early Y) regions of the rectangle, creating two jagged boundaries each representing the places where one process must wait for messages from the other. The boundaries are suggestive of teeth, suggesting the whimsical term "jaws of communication." We refer to the space between the jaws as the communication corridor. In the interior of the corridor there is no waiting for messages; computation may flow unimpeded.

As the rate of exchange of messages increases the corridor narrows (the jaws close). If this corridor were a tube with liquid flowing through it, we would expect such narrowing to impede the flow. In fact we see just such an effect, attributable to the high cost of communication.

But we need not think of a computation as a specific path through the corridor, we can broaden it to reflect our ignorance of its exact position, just as we would do for the trajectory of a car on a highway, the accuracy of which depends on the application. In the limit we may think of the whole corridor as the computation. If it contains no holes or failures of confluence (i.e. persistent conflicts) we regard it as a deterministic computation. Otherwise it is nondeterministic, with the essence of the nondeterminism residing in homotopy classes (the equivalence classes of paths induced by homotopy).

The concern of branching time is then with the details of just where the homotopy classes paste together, since pasted regions represent regions preceding (or following) where the classes diverge (or converge). Synchronization trees are the one-dimensional case of this where words are pasted together along a common initial segment, a one-dimensional pasting. In general pasting can take place not just along curves but across surfaces and through volumes etc. The pasting need not necessarily be the maximal such, otherwise branching time would convey no information not already in linear time. At a fork earlier decisions correspond to earlier suspension of pasting, and dually at a join.

8 Directed Cell Complexes and n-Dimensional Automata

Thus far the discussion of geometry has relied on an informal intuition about geometry which might be characterized as pastings of fragments of Euclidean space oriented somehow to represent the irreversibility of computation. This intuition is formalized in the following notion of higher dimensional automaton.

A sequential automaton or transition system is a graph[6] whose vertices denote states and whose edges denote transitions. One vertex q_0 is distinguished as the start state and a set F of vertices constitutes the final states. The edges are labeled with elements of a set Σ. The linear-time meaning of this automaton is defined in terms of paths in the graph, each determining an element of Σ^*. The automaton ***accepts*** those elements of Σ^* that are determined by some path from q_0 to a state in F. Two such automata are linear-time-equivalent or trace-equivalent when they accept the same subset of Σ^*.

In geometric terms such an automaton is a one-dimensional cell complex whose one-dimensional cells or l-cells are its transitions and whose O-cells are its states. We would like a model whose definition reduces in the l-dimensional case to the above combinatorial notion of directed graph, and which captures the geometric intuitions of the foregoing discussions.

Algebraic topology offers a range of models to select from, such as CW-complexes and simplicial complexes. But as far as we have been able to tell, all the extant notions of cell complex in algebraic topology assume reversible geometry too early in their development, and depend too heavily on structural properties of groups, to permit their easy adaptation to the irreversible case. We would be delighted to have a pointer to a counterexample. The following notion of higher dimensional automaton takes its inspiration not from algebraic topology but instead from the geometry of n-categories [Ehr63].

Ordinarily we define an automaton as an edge-labeled graph, and define its operation in terms of the paths in that graph and the operation of path concatenation. However we could just as well start with the paths and dispense with the underlying graph. When passing from discrete to continuous automata, whose every state transition can be decomposed as a concatenation of shorter transitions, this is in fact necessary since they have no suitable underlying graph.

For qualitatively different reasons we shall similarly not start from the underlying graph when passing from one-dimensional to higher-dimensional automata. The problem is not that suitable underlying graphs ***cannot*** exist, as with continuous automata, but that a suitable notion of n-graph has proved elusive, and only halfway-decent notions have emerged to date. The most successful of these would appear to be M. Johnson's notion of pasting diagram [Joh87]. However it seems to us that his definition presently requires both simplification and generalization in order to constitute a workable notion of n-graph. Hence as an interim measure we define an n-complex to be an n-category, with the eventual goal of redefining it so that it refers to the underlying n-graph when that notion is fully operational.

Proceeding top-down, we first define the notion of concurrent automaton in terms of that of complex, provide a motivational interlude, then define complex.

An n-automaton A = (Q, Σ, δ, S, T) consists of n-complexes Q and Σ, an n-map 6 : Q $\rightarrow \Sigma$, and subsets S, T of (the underlying set of) Q. The m-language ***accepted*** by A is the subset of Σ consisting of those $\delta(x)$ for x in Q such that $s_m(x) \in$ S and $t_m(x) \in T$ where s_m , t_m are the m-th boundary operators of Q.

Before defining complexes let us touch ground momentarily. Our 5-tuple definition parallels the traditional definition of an automaton as (Q, Σ, 6, q_0, F) [HU79], consisting of state set Q, symbol set Σ, transition function 6, start state q_0, and final state set F, along with the usual definition of accepted language. With the following adjustments the traditional definition matches ours in the case n = 1, m = 0.

View the transitions defined by traditional 6 as a pair (E, δ) where E is a set of unlabeled edges and 6 : $E \rightarrow \Sigma$ labels them. Rename Q to V for vertices, move E in with V to form a graph *(V, E)* and recycle Q as Q = *(V, E)*. ***Now*** revamp Σ as a l-graph with one vertex and with edge set old Σ. Interpret 6 : Q $\rightarrow \Sigma$ as the obvious graph map. Finally replace (Q, Σ, 6) by (Q^*, Σ^*, δ^*) where Q^* is the set of paths in Q, Σ^* is as always the free monoid on Σ, and δ^* : $Q^* \rightarrow \Sigma^*$ is the corresponding extension of 6 : Q $\rightarrow \Sigma$ from paths of length 1 to all paths. Take S = $\{q_0\}$ and T = F. This gives us the desired l-automaton (Q^* , Σ^*, δ^*, S, T). We then rename Q, Σ, δ one more time so as to dispense with the $*$'s. This is the translation of a standard automaton into our framework.

We now continue with our top-down definitions. Following Street [Str87], we define an n-complex in terms of l-complexes and 2-complexes.

An ***n-complex,*** or small n-category, is a set bearing the structure of n l-complexes $C_0, \ldots , C_{n-1}$, such that for all $i < j$, (C_i, *Cj)* is a 2-complex.

It remains to define i-complex for $i \leq 2$. In the following the s, t, $*$ terminology is taken from Street [Str87]; we have taken some liberties with the wording of his definition but not its content. Note that this definition of a l-complex ***is as*** a ***homogeneous*** category, namely one where the object-morphism distinction is not made; the objects can be recovered as any of either the range or fixpoints of either s or t, or as the identities of $*$. Homogeneity simplifies the extension to n-categories.

A ***l-complex*** C = (P, s, t, $*$) consists of a set P of (abstract) paths, two boundary operations s, t : $P \rightarrow P$, and a binary operation $*$: $P^{\underline{2}} \rightarrow P$ of path concatenation. The domain $P^{\underline{2}} \subseteq P^2$ of $*$ is the set of ***consecutive*** pairs (x, y) in P^2, namely those for which tx = sy. Furthermore the following conditions must be satisfied.

(i) $s(P)$ = $t(P) \overset{def}{=}$ ***Po,*** constituting the ***U-cells*** of C,

[6] We allow multiple edges from one vertex to another. In some circles the term multigraph is used to distinguish these from the kind where $E \subseteq V^2$.

while $P_1 = P$, the *1-cells* of C. (So $P_0 \subseteq P_1 = P$.)

(ii) $x \in P_0$ implies $sx = x = tx$. (Hence s and t are idempotent, and $st = t = tt$, $ss = s = ts$.)

(iii) For all $(x, y) \in P^{\underline{2}}$, $s(x * y) = s(x)$ and $t(x * y) = t(y)$.

(iv) (Identities.) $x \in P_0$ implies for all (x, y) in $P^{\underline{2}}$, $x * y = y$, and for all (y, x) in $P^{\underline{2}}$, $y * x = y$.

(v) (Associativity.) For all (x, y) and (y, z) in $P^{\underline{2}}$, $x * (y * z) = (x * y) * z$.

A *2-complex* $C = (P, s_0, t_0, *_0, s_1, t_1, *_1)$ is a pair (C_0, C_1) of 1-complexes $C_0 = (P, s_0, t_0, *_0)$, $C_1 = (P, s_1, t_1, *_1)$ on a common set P such that

(i) $P_0 \subseteq P_1$ where $P_i = s_i(P)$, the set of i-cells. $P_2 = P$ (everything is a 2-cell). (Hence $s_1 t_1 = t_1$ but $s_0 t_1 = s_0$, and $P_0 \subseteq P_1 \subseteq P_2 = P$.)

(ii) (Interchange.) $(w *_1 x) *_0 (y *_1 z) = (w *_0 y) *_1 (x *_0 z)$ when all terms are defined.

Referring to our definition of n-complex $(C_0, \ldots, C_{n-1})$ then reveals it in more detail to be a structure $C = (P, s_0, t_0, *_0, \ldots, s_{n-1}, t_{n-1}, *_{n-1})$ every pair (C_i, C_j) of which for $i < j$ is a 2-complex. Evidently $P_0 \subseteq P_1 \subseteq \ldots \subseteq P_n = P$.

An *n-map* or *n-functor* of n-complexes C, C' is a homomorphism; equivalently, a function $f : P \rightarrow P'$ between their underlying sets such that $f : C_i \rightarrow C'_i$ is a functor for $0 \leq i < n$ (i.e. on the 1-complex at each dimension).

It is intuitively clear that associativity is the essential axiom of concatenation in one dimensional irreversible geometry. The straightforward extension of this to two dimensions is that a matrix can be formed as a column of rows, and that vertical composition of columns is associative, and dually it can be formed as a row of columns, with horizontal composition of rows being associative.

But there is more to it than that. The same matrix can be assembled row by row or column by column. While this has the same flavor as associativity, it is not formal associativity in the sense that vertical composition of columns or horizontal composition of rows is associative. We therefore require a separate axiom from associativity. This is the function of the interchange law in irreversible geometry.

An example of a 2-complex is a polygonal decomposition of a simply connected region of the Euclidean plane $\mathbf{R}^2$. The edges of the decomposition form a directed acyclic graph having a single source vertex and a single sink vertex, as hence do the edges of any of the polygons. This partitions every boundary of a polygon x into halves $s_1 x$ and $t_1 x$, start and terminal, both leading from a common source $s_0 x$ to a common sink $t_0 x$. If two polygons share an edge then that edge must belong to the start of one and the terminal of the other. It may be verified that any two such polygons x, y for which $t_i x = s_i y$ for i 0 or 1 together form a polygon meeting these conditions. (Under these conditions the two halves of the boundary of a polygon may touch repeatedly, but if the whole region is 2-connected-no cut point-they can never cross.) Hence the set of all simply connected polygons that can be assembled from the elementary polygons of the decomposition constitutes a 2-complex under the compositions $x *_0 y$ and $x *_1 y$.

The interchange law is evidently sound in this model. What is less obvious is that it is complete in the sense that any two ways of assembling a polygon from elementary polygons (which will take the form of two terms in the language consisting of constants naming polygons and the two compositions) can be proved to give the same polygon using just the two associativity axioms (one for each composition) and the interchange axiom under the standard rules of equational logic. This was claimed by Kelly and Street [KS74], and was more recently extended by John Power to all n-complexes (to appear). The idea of the proof for $n = 2$ is that if $w *_1 x$ and $y *_1 z$ denote the same polygon x, with source $S = s_0 x$ and sink $T = t_0 x$, then each term partitions that polygon via a cut from S to T. If these two cuts do not cross each other then they bound a region $w \cap z$ or $x \cap y$ with source S and sink T leaving two outside regions of the polygon with that source and sink, and one then proves equality of those regions separately by induction, completing with one application of associativity for $*_1$. Otherwise several island regions are produced, and interchange can then be used to represent each of these regions as one region from S to T by extending it out to S and T with 1-cells, and then concluding with applications of associativity of $*_1$.

This shows that n-complexes capture the essence of ordinary geometry of cells, though without its reversibility, ruled out here by ***directing*** all edges and surfaces. (This is in contrast to orienting them, where a direction is specified but each edge has an inverse.) As the earlier sections argue informally, it is just such irreversible cellular geometry that is needed for a geometric model of concurrency.

This completes the definition of higher dimensional automaton and the acceptance criterion most closely corresponding to language acceptance. This induces a congruence on the class of such automata. We leave the further study of this model and associated congruence and the pursuit of other acceptance criteria and congruences to subsequent communications.

References

[Bir35] G. Birkhoff. On the combination of subalgebras. *Proc. Cambridge Phil. Soc*, 29:441-464, 1935.

[BK89] J.A. Bergstra and J.W. Klop. Process theory based on bisimulation semantics.

In *Proc. REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 50-122, Noordwijkerhout, The Netherlands, 1989. Springer-Verlag.

[Bro88] R. Brown. *Topology: A geometric account of general topology, homotopy types and the fundamental groupoid.* Halsted Press, New York, 1988.

[CCMP89] R.T Casley, R.F. Crew, J. Meseguer, and V.R. Pratt. Temporal structures. In *Proc. Conf. on Category Theory and Computer Science, LNCS*, Manchester, September 1989. Springer-Verlag. Revised version to appear in Math. Structures in Comp. Sci., 1:1.

[CDP87] L. Castellano, G. De Michelis, and L. Pomello. Concurrency vs interleaving: an instructive example. *Bulletin of the EATCS*, 31:12-15, February 1987.

[Ehr63] C. Ehresmann. Categories structurees. *Ann. Sci. Ecole Norm. Sup.*, 80:349-425, 1963.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1-102, 1987.

[Gla90] R.J. van Glabbeek. *Comparative concurrency semantics and refinement of actions.* PhD thesis, Vrije Univ., Amsterdam, 1990.

[Gra81] J. Grabowski. On partial languages. *Fundamenta Informaticae*, IV.2:427-498, 1981.

[GV87] 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.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[Joh87] M. Johnson. *Pasting Diagrams in n-Categories with Applications to Coherence Theorems and Categories of Paths.* PhD thesis, Dept. of Pure Mathematics, Sydney University, October 1987.

[KS74] G.M. Kelly and R. Street. Review of the elements of 2-categories. In *LNM* 4.20. Springer-Verlag, 1974.

[Lam86] L. Lamport. On interprocess communication. *Distributed Computing*, 1:77-101, 1986.

[Mil80] R. Milner. *Calculus of Communicating Behavior, LNCS* 92. Springer-Verlag, 1980.

[NPW81] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures, and domains, part I. *Theoretical Computer* Science, 13, 1981.

[Pap86] C. Papadimitriou. *The Theory of Database Control.* Computer Science Press, 1986.

[Par8 1] D. Park. Concurrency and automata on infinite sequences. In *Proc. Theoretical Computer* Science, *LNCS 104*, pages 167-183. Springer-Verlag, 198 1.

[Pra82] V.R. Pratt. On the composition of processes. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Programming Languages*, January 1982.

[Pra86] V.R. Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33-71, February 1986.

[Pri70] H.A. Priestley. Representation of distributive lattices. *Bull. London Math. Soc.*, 2:186-190, 1970.

[Rot73] G.-C. Rota. The valuation ring of a distributive lattice. In *Proc. Univ. of Houston Lattice Theory Conf.* Dept. of Math., Univ. of Houston, 1973.

[Sto36] M. Stone. The theory of representations for Boolean algebras. *Trans. Amer. Math. Soc.*, 40:37-111, 1936.

[Sto37] M. Stone. Topological representations of distributive lattices and brouwerian logics. *Časopis Pěst. Math.*, 67:1-25, 1937.

[Str87] R. Street. The algebra of oriented simplexes. *Journal of Pure and Applied Algebra*, 49:283-335, 1987.

[vGG89] R. van Glabbeek and U. Goltz. Partial order semantics for refinement of actions-neither necessary nor always sufficient but appropriate when used with care. *Bulletin of the EATCS*, 38: 154-163, June 1989.

[Whi49] J.H.C Whitehead. Combinatorial homotopy I. *Bull. Amer. Math. Soc.*, 55:213-245, 1949.

[Whi78] G. W Whitehead. *Elements of Homotopy Theory.* Springer-Verlag, 1978.

[Win80] G. Winskel. *Events in Computation.* PhD thesis, Dept. of Computer Science, University of Edinburgh, 1980.