Arithmetic + Logic + Geometry = Concurrency

Vaughan Pratt*
Stanford University

February 28, 1992

0 Introduction

In defining concurrency one dichotomy that has arisen is between reductionism and primitivism. The reductionist viewpoint reduces concurrent behavior to nondeterministic sequential behavior by viewing it as interleaving. The primitive school treats concurrent behavior as a basic form of behavior in its own right, distinct from and indeed more general than sequential behavior.

A basic argument for reductionism is that nonsequential behavior is best understood in terms of sequential behavior because the latter is the kind we understand naturally and instinctively, or at least the kind with which we have by far the most experience. At the conscious level each of us appears to be a sequential agent. Our limbs, muscles, and cells may act concurrently, but our control of them is basically sequential and the necessary concurrent control seems to happen below the level of our conscious control, or it would be just as easy to twiddle our thumbs in opposite directions as in the same direction.

Primitivists object that reducing concurrent behavior to sequential makes the former harder rather than easier to understand. By analogy, organizing a TV image as the sequence of its scan lines in one dimension, the form in which it arrives at your receiver, makes it much harder to see. It is not so much a question of whether information is lost as the complexity of piecing the information back together. Even if reductionism is the only mode of thought presently available for the perception of concurrent behavior, it is such a poor way of thinking about it that we are better off learning to deal with concurrency in its own right than persisting with reductionism.

Reductionists counter that primitivism is not fully abstract, that is, draws unobservable distinctions. In the context of dataflow nets Kok [Kok87], Jonsson [Jon89], Rabinovich and Trakhtenbrot [RT88], Russell [Rus89], and others have shown that reductionism, unlike primitivism, is fully abstract.

*This work was supported by the National Science Foundation under grant number CCR-8814921.

The primitivist answer is that these results presume the observer to be sequential, and that the reductionists have shown only that *a sequential observer perceives a sequential world.* Plotkin and Pratt [PP88] have shown that a team of n sequential observers can resolve distinctions invisible to $n-1$ observers, whence for more than one observer *reductionism is unsound* (asserts observably false equalities).

Reductionism has emerged in response primarily to the needs of software engineers, making a number of assumptions valid in that domain but not in many other settings where concurrency theory could be fruitfully applied. Thus, with the interleaving of machine instructions in mind, it presumes fixed atomicity of events in order to define interleaving—once an event has been declared atomic there can be no subsequent "opening up" of the event to reveal subevents that might be interleaved with the subevents of some other previously atomic event. Furthermore reductionism is timing independent: no limits are placed on the relative speeds of parallel processors. And reductionism assumes discreteness, ruling out continuous processes.

This paper ties together three primitivist views of concurrency whose development the author has had some involvement with, namely the arithmetic of schedules, the logic of schedule-automaton duality, and the geometry of automata. Separately each of these views shed considerable light on concurrency. Our goal here is to bring these three views together coherently in the one place. The general picture is as follows.

$$\begin{array}{ccccc} 1 & & 2 & & 3 \\ \boxed{\begin{array}{c}\text{Arithmetic}\\ \text{of}\\ \text{Schedules}\end{array}} & \longleftarrow & \begin{array}{c}\text{Logic of}\\ \text{Schedule-Automaton}\\ \text{Duality}\end{array} & \longrightarrow & \boxed{\begin{array}{c}\text{Geometry}\\ \text{of}\\ \text{Automata}\end{array}} \end{array}$$

By a nice coincidence the numbering is both section number and $UF(1)$ (cardinality of the singly generated free algebra) in the respective equational categories. That is, 1 is the basic counting element of arithmetic, 2 (true and false) is (the dual of) the basic dualizing element of logic, and 3 (beginning, middle, and end) is the basic space-filling element of geometry.

A *schedule* is a set of necessary events of transition constrained by facts about pairs of events. An *automaton* is a set of possible states of information enabled by transitions between pairs of states.

This wording emphasizes the duality of event and state. Events are necessary in that an unconstrained schedule (no relations between events) requires that all events be performed. States are possible (the possible worlds of Kripke semantics) in that a disabled automaton (no transitions between states) permits one state to be occupied.

Here we limit the relations between events to those of temporal order ("binary distance") and conflict. In the absence of conflict this arithmetic has for its progenitor the Birkhoff arithmetic of partial orders [Bir42] as a basic programming language for schedules.

The Birkhoff-Stone duality [Bir33, Sto37] of posets as deterministic schedules and distributive lattices as deterministic automata then constitutes a logic in which the duality of true and false are replaced by the duality of doing and being. This duality interchanges events (as actions) and states (as facts or models), and dualizes the arithmetic by interchanging sum and product. Deterministic schedules extend to event structures via a notion of event conflict, introducing an element of choice or nondeterminism. Dually, deterministic automata extend to branching automata.

The study of ordered sets fixes a vertical orientation and conventionally locates truth at the top and falsehood at the bottom. For Birkhoff-Stone duality we shall analogously fix a horizontal orientation and conventionally locate schedules on the left and automata on the right.

There is some precedent for calling these the geometric and algebraic sides respectively, motivated by the case of Stone duality. However the meet-join mobility and the geometric interpretation of distributive lattices that we describe below show that these distinctions are not all that well correlated with Stone duality, in that we can find both geometric and algebraic elements on both sides simultaneously.

This confusion does not arise for necessity and possibility. When we put the sets with the necessary elements on one side, those with the possible elements automatically belong to the other side. The concrete interpretation of these sets as respectively schedules and automata parallels the concrete interpretation of the order-theoretic notions of 0 or bottom and 1 or top as the logical notions of false and true respectively.

Although we have limited ourselves here to time as a partial order, we have elsewhere studied a more general form of temporal organization, the generalized metric space [CCMP91]. The temporal order relation $x \leq y$ is generalized to a distance metric $d(x, y)$, not necessarily symmetric, representing a delay and satisfying a triangle inequality $d(x, y) \otimes d(y, z) \sqsubseteq d(x, z)$ (and its zeroary form $\top \sqsubseteq d(x, x)$ where $\otimes$ (with unit $\top$) associatively accumulates delays ("horizontally") and $\sqsubseteq$ orders delays by logical strength ("vertically"). This level of generality admits notions such as causality and real time. Partial orders are the case where distances are two-valued, i.e. a binary relation, where the triangle inequality becomes transitivity and its zeroary form reflexivity.

Thus far we understand only the arithmetic of this generalization of partial orders. We do not as yet know what form the automaton dual to a generalized metric space should take, neither the geometry of that form nor the logic of that duality. We would be delighted to see a generalization of the very attractive logic of event and state spaces to these generalized metric spaces. The present paper treats only the partial order model of time and the notion of conflict (of any arity).

1 Arithmetic of Schedules

1.1 Cardinal and Ordinal Arithmetic

The progenitor of the arithmetic of schedules is Birkhoff's generalized arithmetic of posets [Bir37, Bir42] as a common generalization of cardinal and ordinal arithmetic. The relevance to concurrency is that cardinals, as unordered sets, can be interpreted as concurrent schedules while ordinals and more generally chains constitute sequential schedules. Cardinal addition (disjoint union) $A+B$ of posets serves as asynchronous concurrence (do A and B independently) while ordinal addition (concatenation) $A;B$ is sequential composition (do A then B).

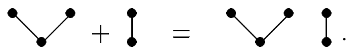
An algebra of processes simultaneously defines a parallel programming language and equips it with a semantics. Milner, Hoare, Bergstra and Klop, and others have defined various such algebras. While Birkhoff arithmetic only partially meets the needs of concurrent programming, namely what we think of as the "conjunctive" portion—concurrency without nondeterminism—it does have a sufficiently precise yet simple model theory and associated equational theory as to warrant admitting it as one component of the foundations of concurrency.

Birkhoff arithmetic is defined on partially ordered sets, considered to be defined only up to isomorphism, that is, $A \cong B$ implies $A = B$. This allows the isomorphism $A + B \cong B + A$ to be written as $A + B = B + A$, and $(A \times B) \times C \cong A \times (B \times C)$ as $(A \times B) \times C = A \times (B \times C)$. It also implies that there is just one unordered (discretely ordered) set of each cardinality, which we may then take to be the cardinals. Similarly when every subset of a poset X has a least element, that is, is well-ordered, we take X to be an ordinal.

The operations of addition, multiplication, and exponentiation each occur in two forms, cardinal and ordinal. Except for ordinal exponentiation, the operations of a given type preserve posets of that type, although they are defined on all posets.

In the following six binary operations we take the two arguments to be the posets $A = (X, \leq)$ and $B = (Y, \preceq)$.

The cardinal sum $A+B$ is simply the juxtaposition of A and B. For example

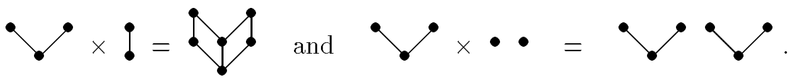


Formally, $A+B = (X+Y, \leq_0 \cup \preceq_1)$ where $X+Y$ denotes $(X \times \{0\}) \cup (Y \times \{1\})$, $\leq_0$ partially orders $X \times \{0\}$ to make $(X \times \{0\}, \leq_0)$ isomorphic to A via an isomorphism that pairs $(x, 0)$ with x, and $(Y \times \{1\}, \leq_1)$ is similarly isomorphic to B.

As an operation on schedules, cardinal sum is asynchronous concurrence.

The cardinal product $A \times B$ (Birkhoff writes AB) is $(X \times Y, \sqsubseteq)$, where

$(x, y) \sqsubseteq (x', y')$ just when $x \leq x'$ and $y \preceq y'$. For example



The cardinal product of schedules is *orthocurrence* or *interaction*, describing the flow of A through B, a symmetric form of interaction. Every event of A interacts with every event of B, each such interaction constituting an event of $A \times B$. For example the interaction of three trains passing through four stations contains twelve events, ordered in the shape of a 3×4 rectangle.

The cardinal exponentiation B^A consists of the monotone functions from A to B, ordered pointwise: $f \sqsubseteq g$ just when $f(x) \preceq g(x)$ for all x in A. For example

Birkhoff gives a number of laws for the cardinal operations, all of which however are inferrable as the consequences of the category **Pos** of partially ordered sets being a cartesian closed category with addition (coproducts $A + B$). For example addition and multiplication are associative and commutative, multiplication distributes over addition, and $A^{B+C} = A^B \times A^C$, $(A \times B)^C = A^C \times B^C$, $(A^B)^C = A^{B \times C}$, $A^1 = A$, and $1^A = 1$.

Posets are very obliging in this way, which is one thing that makes them so attractive for concurrent programming. The corresponding situation for the category of chains (linear orders) and their monotone functions is much less satisfactory. Neither product nor coproduct exist, that is, chains cannot be added and multiplied the way posets can. One can concatenate chains, an operation we are about to meet as ordinal sum, but this is a sequential operation on sequential objects and does not extend naturally to concurrency.

The ordinal sum $A; B$, notated $B \oplus A$ by Birkhoff who wrote $x > y$ in preference to $y < x$,, is $A + B$ with its order augmented with $X \times Y$, that is, with those pairs $(x, 0) \sqsubseteq (y, 1)$ for all $x \in X, y \in Y$. For example

As an operation on schedules ordinal sum is concatenation: do A then B.

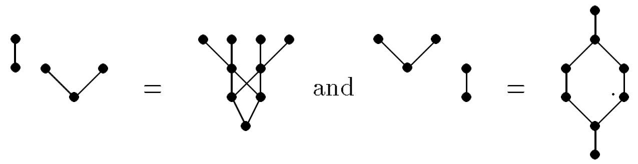
Ordinal sum is associative and has the same identity as cardinal sum, namely the empty poset 0 (which Birkhoff disallows on the grounds that 0^0 could be either 0 or 1 and that 1 would not be a quotient or "homonumber" of 0, both false by today's standards). However it is not commutative.

The ordinal or lexicographic product $A \circ B$ is $A \times B$ with its order augmented with those pairs $(x, y) \sqsubseteq (x', y')$ for which $x < x'$. For example

$A \circ B$ can be described as the substitution of a copy of B for each element of A. Unlike ordinal sum, ordinal product does not contribute any sequentiality of its own, but rather serves to nest iterations. If A and B are purely concurrent (discretely ordered) then $A \circ B = A \times B$. If they are purely sequential (chains) then $A \circ B$ iterates B A times. If A is sequential while B has some concurrency, that concurrency is confined to one iteration of the inner loop. However if B is sequential and A has some concurrency then the corresponding copies of the inner loop are run concurrently.

Ordinal product is associative but not commutative, and has the same identity as cardinal product, namely the singleton. It distributes over both cardinal and ordinal sum, but only on the left in each case: $(A + B) \circ C = A \circ C + B \circ C$ and $(A; B) \circ C = A \circ C; B \circ C$.

The ordinal exponentiation ${}^{A}B$ is the set of all functions from A to B, *pre*ordered[1] so that $f \sqsubseteq g$ just when for all $x \in X$, either $f(x) \preceq g(x)$ or $f(x') \prec g(x')$ for some $x' > x$. (Intuition suggests this should be $x' < x$, but Birkhoff noticed that when A is implicitly reversed in this way the operation preserves chains.) In particular, if at x' we have $f(x') \prec g(x')$, then in deciding whether $f \sqsubseteq g$ by comparing f and g pointwise we may ignore the relationship between $f(x)$ and $g(x)$ for all $x < x'$. For example

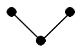


Both of these are easily computed using Birkhoff's two laws

$${}^{B+C}A = {}^{B}A \times {}^{C}A$$

$${}^{B;C}A = {}^{C}A \circ {}^{B}A$$

[1]A preorder is a reflexive transitive binary relation, lacking antisymmetry, a property that need not hold when the exponent A is infinite.

The first is computed as ${}^{1;1}A = {}^{1}A \circ {}^{1}A = A \circ A$ where $A =$  $= 1;(1+1)$.

The second is ${}^{1;(1+1)}2 = {}^{1+1}2 \circ {}^{1}2 = (2 \times 2) \circ 2$ where $2 = 1;1$ (ordinal 2).

As a programming connective ordinal exponentiation implements a generalized form of "iterated iteration." When A is a chain ${}^{A}B$ is the lexicographic product of B with itself $|A|$ times; thus if lexicographic product is nested iteration then we can now iterate the nesting process. When A is discrete ${}^{A}B$ is the cardinal product of B with itself $|A|$ times. The above formulas indicate how one may produce a mix of these products.

In this exposition of Birkhoff arithmetic we have adhered to Birkhoff's use of $=$ in place of $\cong$. This seemingly harmless convention is strongly reinforced by experience with the Hasse diagram representation of the elements of posets as anonymous dots.

Nevertheless this convention is in conflict with the constructive interpretation of $A+B \cong B+A$ as identifying a particular isomorphism between $A+B$ and $B+A$ determined by the proof. In the case when $A = B = 1$, the singleton, this isomorphism from $1+1$ to itself is, not unexpectedly, the one that interchanges the two elements of the cardinal $1+1$. In this constructive interpretation $A = B$ is reserved for the *identity* isomorphism on A, making $1+1 = 1+1$ an instance of $A = A$ but not of $A + B \cong B + A$.

2 Logic of Schedules and Automata

2.1 Motivation

Posets as we have understood them above are inherently deterministic: there is no free choice, indeed there is no choice at all. Their elements as events must all occur, constrained only in their order of occurrence. Adding poset arithmetic to the theory does not change this: its operations act on and yield deterministic behaviors.

When serialization of any kind is brought into play, nondeterminism enters so as to choose an order of serialization. For events that are unrelated, e.g. for reasons of distance, such nondeterminism is inessential. A relationship between the events, e.g. both use the same unsharable resource, may or may not make this nondeterminism essential, but this is a distinction we are unable to represent with plain posets. In any case we may quite appropriately refer to this kind of serialization-induced nondeterminism as *ephemeral*.

This situation arises with formal languages, without the complication of ephemeral nondeterminism. Strings are deterministic in that they represent the sequential occurrence or utterance of their symbols, all of which must occur in the prescribed order.

For formal languages the trick for introducing essential nondeterminism is to define a language to be a set of strings, from which one selects one for utterance.

All the nondeterminism is concentrated in the set structure: a single choice of a string is made from the whole set.

The same can be done for posets, by forming sets of them. This natural generalization of formal languages to partial strings, defined as posets labeled with symbols, was first expressed clearly and studied by Grabowski [Gra81]; subsequently the present author shortened "partial string" to "pomset."

The drawback of the set approach is that it moves the timing of all choices to the beginning of the execution. Many phenomena, such as partial correctness, do not depend on this timing. But the very act of making a choice does. In choosing between the strings ab and ac we have the option of deferring the choice until after the a. *This option is not representable with unadorned sets of strings.*

What is needed is a representation that permits nondeterminism to be spread throughout the string or computation. As such nondeterminism can be localized as in the regular expression $abc(d+e)fgh$ or widespread as in $abcd+efgh$, a distinction that disappers when we identify $abc(d+e)fgh$ with $abcdfgh+abcefgh$.

Nondeterminism then takes on more of the qualities of concurrency, which too can be localized or widespread, using the same two regular expressions but with their poset arithmetic interpretation where $abc(d+e)fgh$ is very different from $abcdfgh+abcefgh$.

This section develops an even earlier contribution of Birkhoff's than poset arithmetic, namely the duality of posets and distributive lattices. We will consider variations on this duality and home in on a variation, event and state spaces, and argue informally that among all these variations this particular one is optimal for the purpose of using the duality as a logic of concurrency.

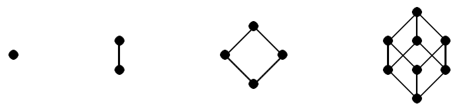
2.2 Nature of Birkhoff-Stone duality

The theme developed here is that of schedules X and their corresponding automata $A = 2^X$. The states of the automata are the functions from X to 2 (i.e. subsets of X) indicating for each event $x \in X$ whether or not it has been done yet.

The sets X with up to three elements are as follows.

$$\emptyset \qquad \bullet \qquad \bullet\;\bullet \qquad \bullet\;\bullet\;\bullet$$

Their power sets are the corresponding Boolean algebras.



It is clear that there is a 1-1 correspondence between sets and their power sets, since the former can be recovered as the atoms of the latter.

Now one would naturally assume that if A is the above eight-element Boolean algebra, then 2^A would be a 256-element Boolean algebra with 8 atoms. But surprisingly 2^A turns out to be isomorphic to the 3-element set of which A is the power set. The reason for this apparent violation of Cantor's theorem is that A and 2 are not sets but Boolean algebras, and 2^A is defined to be not arbitrary functions between the underlying sets of A and B but rather homomorphisms, that is, structure-preserving functions.

We did not notice this when forming automata because we were exponentiating plain sets, whose homomorphisms are just functions. In the other direction the structure to be preserved consists of *all* the meets and joins (we follow Davey and Priestley [DP90] in using meet and join synonymously with inf and sup) of a complete Boolean algebra. For our eight-element Boolean algebra there are just three such maps, namely those taking two of the atoms to 0 (the lower element of 2) and the third to 1. No two of these three maps are comparable pointwise, hence collectively they form a discrete or unordered set.

We now extend this duality off in two different directions, webs and posets, and then reunite them as event structures.

A *web* [Gir87] is an undirected graph without self-loops, equivalently an irreflexive symmetric binary relation. We think of the edges as expressing a conflict[2] between its endpoints. Of the webs with at most three elements, the four discrete or conflict-free ones appear above. The remaining four are as follows.

Their dual automata realize the notion of conflict by excluding those subsets containing conflicting pairs, leaving just the "coherent" subsets; Girard [Gir87] has called these *coherence spaces*. The above four webs have the following duals.

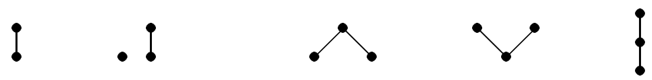
The idea of the power set of X as 2^X can be continued here by requiring that web maps take at least one end of every conflict edge to 0. One way for this to "happen automatically" is to treat webs as just directed graphs that happen to be irreflexive and undirected in the sense that all edges come in matched pairs (x, y), (y, x). (A directed graph is a binary relation, namely an underlying set X and a subset of X^2. A graph map is an edge-preserving function between the underlying sets.) Take 2 to be the relation $R(x, y) \equiv x = 0 \vee y = 0$, that is, the graph of all edges on $\{0, 1\}$ except the self-loop at 1. Then for any web X the

[2]Girard assigns the opposite or dual interpretation: edges denote compatibility or *coherence*. By choosing to show the conflicts instead we don't have to go back and redraw the four conflict-free sets we started with.

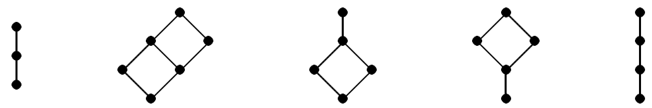
graph maps from X to 2, ordered pointwise as before, constitute the coherence space of X.

We know of no way of defining the maps between coherence spaces so that the set of coherence space maps from A to 2 comes out to be the set of points of the web, unlike the situation with sets, and unlike Birkhoff duality, which comes next. To get back the web we therefore make do with picking out the atoms of the coherence space as the elements of the web and taking the edges of the web to consist of those ordered pairs of atoms of the coherence space that lack a join.

We now come to Birkhoff duality. The notion of partially ordered set should need no introduction. We have already listed those with the discrete or identity order with up to three elements, here are the remaining partially ordered sets up to that size.



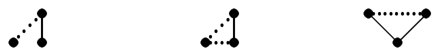
Their dual automata require that if $x \leq y$ and y is in the set then so is x. Such subsets are called *order ideals*. The order ideals of a poset, ordered by inclusion, form a profinite (complete, completely distributive, and algebraic) distributive lattice, including top and bottom. The above five posets have the following duals.



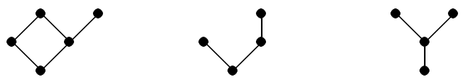
Again we dualize in each direction by exponentiating to base 2 the converse or order dual $X^{\smallsmile}$ of each poset X to yield the distributive lattice $2^{X^{\smallsmile}}$ of all order ideals of X. This time the maps on the poset side are monotone functions (the morphisms of the category **Pos** of posets) while the maps on the distributive lattice side are lattice homomorphisms, preserving all meets and joins including the empty meet or top and empty join or bottom. The poset maps to 2 correspond to the order ideals of the poset, the correspondence being that $f^{-1}(0)$ is the order ideal corresponding to f, and every order ideal clearly determines that f. In the other direction, certain elements in the distributive lattice are special, namely those with exactly one edge entering them from below, at least in the finite case—in general they are the elements not representable as the join of the set of elements strictly less than them, which we call the *completely prime* elements.

Very conveniently, the same exponentiation trick that picked out the atoms in the set/Boolean algebra duality generalizes to this situation, and picks out exactly the prime elements. Just as each Boolean algebra map sent exactly one atom to 1, with every atom determining exactly one such function, each complete distributive lattice map turns out to send exactly one prime to 1, again with every prime determining exactly one such function. Among the order ideals making up the dual of a poset, the primes can be shown to be exactly the *principal* order ideals, those of the form $\downarrow x = \{y \leq x \mid y \in X\}$ for some $x \in X$, which are evidently ordered in the dual exactly as their parent x's are ordered in X. Those order ideals that aren't principal can be represented as the union of the principal order ideals generated by each of their maximal elements. This establishes a 1-1 correspondence between the elements of the posets, the principal order ideals, and the primes of the dual space. Thus the primes, in the order they occur in the automaton, form a poset isomorphic to the schedule we started with. (Lawvere has called this the Dedekind-Yoneda embedding of a poset X in its ideal completion $2^{X^{\smile}}$.)

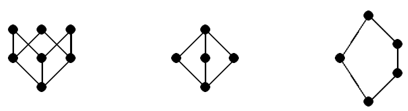
Event structures [NPW81], more precisely prime coherent event structures, are the common generalization of webs and posets obtained by admitting both a binary conflict relation $\#$ and a partial order relation $\leq$. Those prime coherent event structures with up to three elements that are either webs or posets are all listed above. The remaining three that are neither webs nor posets are:



Their automata consist of the coherent order ideals, giving structures called *prime algebraic coherent posets* [NPW81]. Coherence once again removes the convenience of exponentiation.

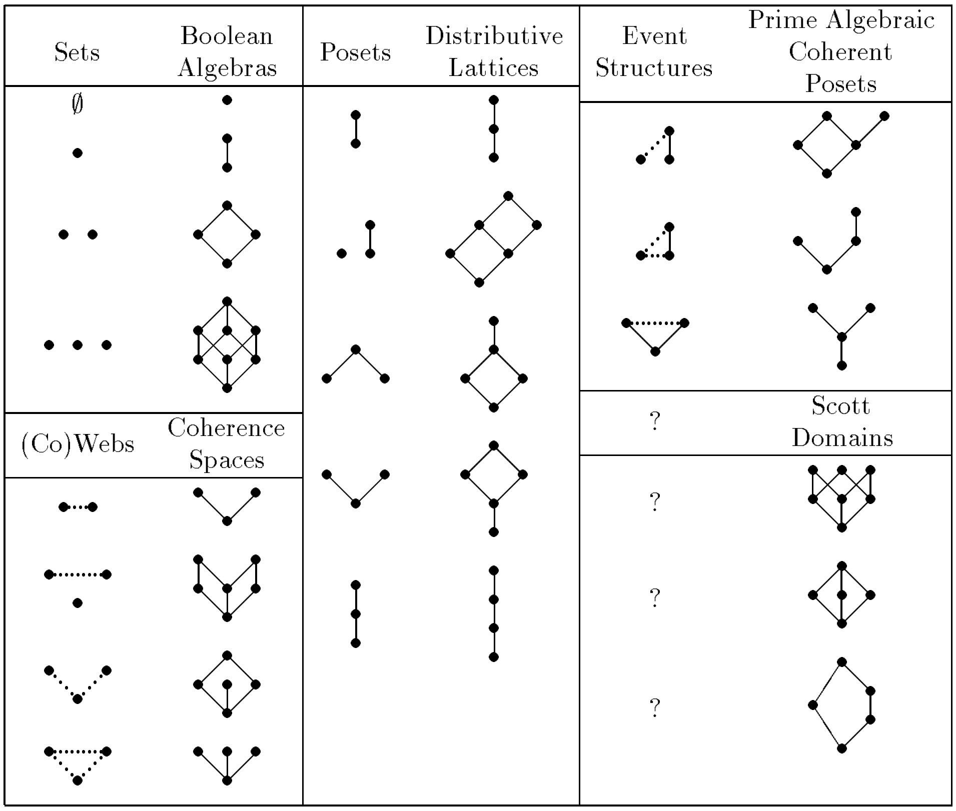


Now it happens that all automata we have seen so far have been Scott domains. A Scott domain is a CPO (complete partial order, including a bottom element) that is algebraic and bounded-complete [DP90, p61]. Here are three Scott domains that are not the dual of any prime coherent event structure.



The first is however the dual of a prime (but not coherent) event structure, one where n-way conflicts can exist without necessarily implying $n-1$-way conflicts even for $n \geq 3$. (Coherence means that conflicts are binary.) The question then arises, are these the "power set" of anything, in a reasonably natural way?

There is an unexpectedly beautiful answer to this question, in the form of the event spaces and their dual state spaces. Let us first collect together all the above diagrams for convenient reference.



2.3 Mobility of Bounds

There is a conservation principle applying to the collective meets and joins of the two sides of each instance of this duality, with the Birkhoff duality of posets vs. distributive lattices with top and bottom being one such instance, that of posets with top and bottom vs. plain distributive lattices being another, and that of posets with all joins vs. posets with all meets being yet another.

Very loosely speaking every meet and join shows up on exactly one side of the instance, with their distribution determining the instance. But this is the only constraint; it confers a certain degree of individuality on the meets and joins but permits travel rights to the individuals, allowing them to be moved to the opposite side of the duality to create a different instance of Birkhoff-Stone duality.

But these are *syntactic* meets and joins. This is in contrast to the semantic meet of a subset Y of a poset, which is the greatest lower bound on Y. A syntactic meet is an operation yielding a semantic meet. If a semantic meet exists but is not named by an operation then it is not a syntactic meet.[3]

In this regime one dualizes by forming $2^{X^{\smile}}$.

The duality assumes posets on both sides. (Sets on one side are then a subcase.) In the finite case we have four operations, meet (1 and $\wedge$) and join (0 and $\vee$) of arity 0 and 2 in each case. These can be moved around independently in 16 ways. The symmetries of two sides of the duality, and of meet vs. join, reduce these to six essentially different dualities. With all operations on the right we have posets on the left and bounded (1 and 0) distributive lattices on the right. If we move 0 to the left we have posets with 0 vs. distributive lattices with 1. Moving the 1 left as well then gives bounded posets and distributive lattices.

When we put both joins, 0 and $\vee$, on the left we have join semilattices with 0 on the left and meet semilattices with 1 on the right. If we then move 0 to the right we have join semilattices on the left and bounded meet semilattices on the right. Finally if we then move 1 to the left we have what we shall call finite event spaces on the left and finite state spaces on the right. These last are what we shall be most interested in here.

In case it proves useful to someone we define here a numbering system for these arrangements. The four bits of a binary number specify the arrangement. Reading from low to high order (right to left, weights 1,2,4,8) the bits are for 0, 1, $\vee$, and $\wedge$. A 1 denotes the left side of the duality. Thus Birkhoff duality is duality 0. Dualities 5 and 10 have complete semilattices on each side. Duality 6 is that of event and state spaces.

When we pass to infinite posets on each side a similar question arises as to the distribution of the infinite meets and joins. These can be treated more or less independently by cardinality (e.g. countable vs. uncountable), but we shall not so separate them here, instead keeping all the infinite meets on the same side, and similarly for all the infinite joins. The notation extends with two additional bits, weight 16 denoting infinite join and weight 32 denoting infinite meet. Since we will always want to allow for the possibility of infinite dualities we will use only the 6-bit duality numbering.

The informal notion of infinite meets and joins is expressed formally as the *Stone topology*. In the case of posets and distributive lattices, putting the Stone topology on the left, duality 48, yields what Stone described obscurely [Sto37] but which was much later described more naturally by Priestley [Pri70] as ordered Stone spaces. In the special case where the order is discrete (but still duality 48) we have Stone spaces on the left and Boolean algebras on the right.

When we add the Stone topology to the bounded distributive lattices instead we still have the same posets on the left but now we have so-called *profinite* dis-

[3] A feature of syntactic meets and joins that we shall not avail ourselves of here is that they continue to specify their result equally well in a preordered set (no asymmetry condition), unlike semantic meets which cease to be well-defined as operations. A syntactic meet of Y in a preordered set is that member of the equivalence class of greatest lower bounds on Y specified by the meet operation.

tributive lattices on the right. This is duality 0, in this sense the most basic one. The special case of the discrete order on the left, yields profinite Boolean algebras on the right, better known as complete atomic Boolean algebras. A CABA is isomorphic to the power set of its atoms. In a profinite distributive lattice the notion of atom is generalized to that of *complete prime*, most conveniently defined as an element not the join of the set of elements strictly below it. (This is more constructive than but yet equivalent to saying it is not the join of any set of elements strictly below it). This is easily seen to be just the atoms in the case of a CABA. The functions in 2^A are in correspondence with the complete primes of distributive lattice A, and even have the same order.

If the only meets and joins on the left are the zeroary ones, duality 3, the schedules become bounded posets while the automata become top-and-bottom-less profinite distributive lattices, enabling geometry by permitting for example the 2x2 square as a distributive lattice (in fact a Boolean algebra) to be embedded in any square of such a lattice.

When all the joins are on the left and all the meets on the right we have *complete join semilattices* on the left and *complete meet semilattices* on the right, duality 21. Interchanging 0 and 1, duality 22, yields event spaces on the left and state spaces on the right.

This duality was identified and named by the author [Pra91a]. It constitutes a unification, extension, and tidying up of several notions floating around in the literature. The arrangement balances the nonempty meets and joins so as to achieve a sort of impedance matching. Most importantly it properly subsumes the Nielsen-Plotkin-Winskel event structure model. It also generalizes Scott domains, which subsume prime algebraic coherent posets, which in turn subsume both profinite distributive lattices and coherence spaces. It is also a nondegenerate model of full linear logic, with the comonad structure being that defined by its underlying posets, a cartesian closed category. Yet it has the above simple definition (via the location of meets and joins), and like complete semilattices, each side of the duality comprises an equational category in the sense of being monadic in **Set**. On the schedule side the signature consists of the nonempty joins and the empty meet or top, while the equations are just those defining complete join semilattices modified to move the constant normally at the bottom to the top.

We now define event spaces and state spaces.

An event space is a poset $(X, \leq)$ with a greatest element ∞ and for every nonempty subset Y a sup or join $\bigvee Y$. Figure 1 [Pra91a] enumerates the event spaces with up to 4 events.

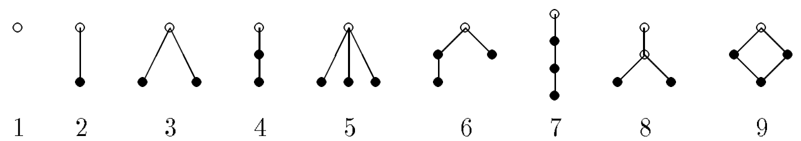


Figure 1. Event Spaces with at most 4 events.

The intended interpretation is as follows. $x \le y$ means event x precedes event y. The event $\bigvee Y$ is that of completing all of Y; it follows every event of Y and equals the last or maximal event of Y just when such exists. ∞ denotes the unperformed or overcommitted event and permits conflict to be expressed equationally as $\bigvee Y = \infty$, as in Figures 1.3, 1.5, 1.6, and 1.9. The interpretation of conflict is that at most a proper subset of Y may be performed.

The black events are the basic ones, defined as those events x satisfying neither $x = \infty$ nor $x = \bigvee\{y < x\}$. They correspond to the basis vectors (however chosen) of a vector space, or the atoms of a coherence space.

Like linear transformations of vector spaces, but unlike coherence space maps [Gir87], an event map is simply a homomorphism in the usual sense of algebra, that is, a function between event spaces satisfying $f(\infty) = \infty$ and $f(\bigvee Y) = \bigvee(fY)$.[4] These objects and their maps form the category **Ev**.

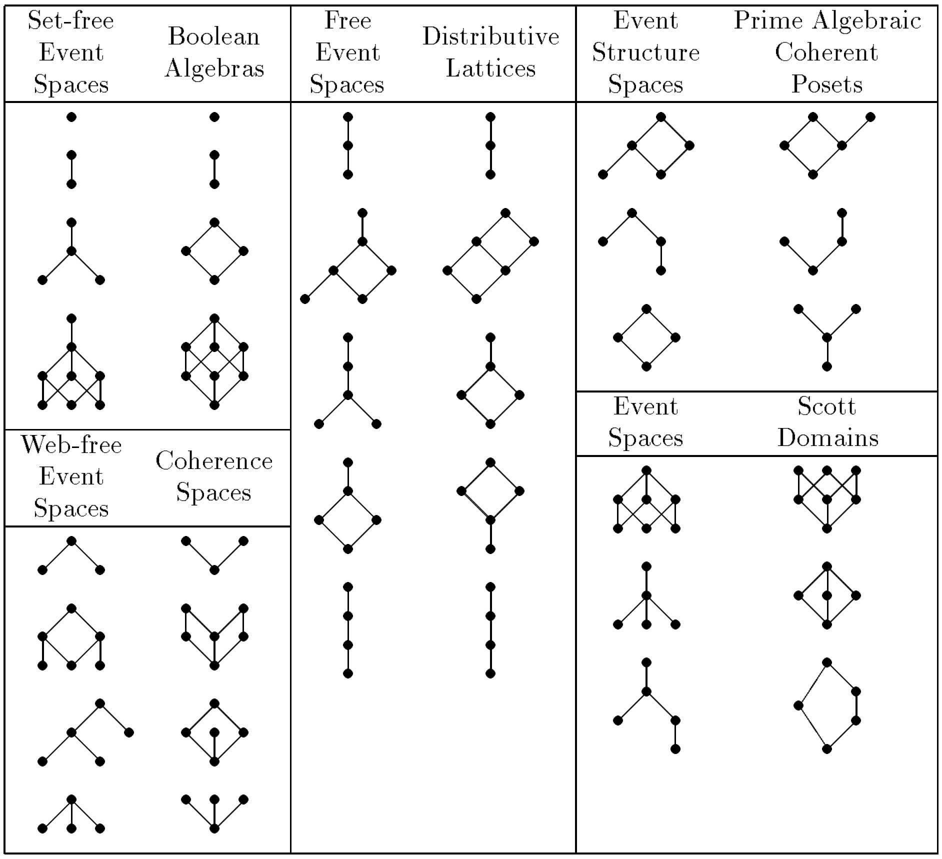
A state space is the order dual of an event space, with $\bigwedge$ and q_0 in place of $\bigvee$ and ∞. The state $\bigwedge Y$ is that of knowledge of the disjunction of the states in Y. The state q_0 is that of ignorance and permits dilemma, the dual of conflict, to be expressed equationally as $\bigwedge Y = q_0$, as in the order dual of Figure 1.3. A state map is a homomorphism of state spaces, satisfying $f(q_0) = q_0$ and $f(\bigwedge Y) = \bigwedge(fY)$. State spaces and their maps form the category **St**.

The complement a^- of an event space a is the result of moving the top element to the bottom, a state space.

Every "power set" in the above table is a Scott domain, and *a fortiori* a state space. Complementation gives us a method of working backwards from every state space to the event space of which it is the "power set." That event space is also the power set of its complementary state space.

We can now redraw our table as follows.

[4] Omitting ∞ does not change the notion of event space since $\bigvee X = \infty$, but it does change the notion of event map since preservation of ∞ does not follow from preservation of joins.



The connection between event spaces and event structures (and hence webs and posets) is made as follows. The basic elements of an event space are the elements of the corresponding event structure. Their order is as in the event space. The pair $\{x, y\}$ is in the conflict relation just when $\bigvee\{x, y\} = \infty$.

To pass in the other direction, an event structure is filled out to an event space by putting in all nonempty nonconflicting order ideals explicitly to represent "concurrently compound" events. In addition the ∞ element is added at the top. We call such an event space an event structure (or event-structure-free) space. In the absence of conflict we call it a *free* (or poset-free) event space, while in the absence of order we call it a *web-free* event space, and in the absence of both a *set-free* event space. All these notions of Q-free are associated with a forgetful functor $U : \mathbf{Ev} \to Q$ and its left adjoint $F : Q \to \mathbf{Ev}$ taking each object q of Q to the Q-free event space *on* q. When Q is omitted it defaults to $Q = \mathbf{Pos}$, the left side of the Birkhoff duality, a cartesian closed category. We take Girard's $!X$ [Gir87] to be $FU(X)$ for this choice of Q.

2.4 Linear Logic as Concurrent Programming Language

We return now to poset arithmetic, but redone for event spaces.

Ordinary posets show up here as *free* event spaces. The cardinal product or orthocurrence of posets becomes tensor product of the corresponding free event spaces, $F(X \times Y) = F(X) \otimes F(Y)$. This motivates the definition that *tensor product $X \otimes Y$ of event spaces is orthocurrence.* (Even though it appeared to be product before, and was originally identified as such by the author [Pra86], it was later determined to really be tensor product [CCMP91]. The two operations coincide on posets, whence the original error in identification.)

Sum (coproduct) of posets becomes sum of the free event spaces they generate: $F(X+Y) = F(X)+F(Y)$ (left adjoints preserve colimits). This motivates the definition that *sum $X + Y$ of event spaces is asynchronous concurrence.*

This raises the question, what does product of event spaces mean if not orthocurrence? This is where the symmetry of event and state spaces starts to pay off. *Product $X \times Y$ of event spaces is now guarded choice.* For example the guarded choice of two copies of Figure 1.2 is Figure 1.9, of which the bottom element is the guard.

The operations on state spaces with the corresponding meanings are the respective duals of these: tensor sum for orthocurrence, product for asynchronous concurrence, and sum for choice.

Besides providing a programming language this also provides a complete model of linear logic, except that the "additive" constants $\top$ and 0 (in Girard's notation) degenerate to the same event space, namely that containing only ∞, while the "multiplicative" constants 1 and $\perp$ degenerate to the two-event space, Figure 1.2.

The match between linear logic and programming is by no means perfect. On the one hand the other linear logic connectives are of more use for a logic of programs than as actual control structures. On the other hand there remain various programming operations not provided by linear logic. All three are best defined on state spaces; their definitions on event spaces are then just the De Morgan duals of these definitions.

These operations include blind choice, concatenation, and synchronous concurrence [Win88]. Blind choice forms the disjoint union of the state spaces and then identifies their q_0's.

Concatenation $A; B$ substitutes one copy of B for each *maximal* state of A. This will be recognized as just lexicographic product with the substitution limited to maximal states.

Further discussion of the above may be found in [Pra91a].

Synchronous concurrence $A||B$ is the set of partial partitions of the disjoint union $A + B$ whose respective restrictions to A and B are discrete partitions of principal order ideals of A and B respectively and whose nonsingleton blocks, called *synchronizations*, contain only complete primes of A and B. (A partial partition of a set X is a pairwise-disjoint set of subsets of X. The partial partitions of X are intended to be ordered by inclusion. Note that its union need not equal X, unlike a partition.) This will be described in more detail in a forthcoming paper. One may obtain some idea of the notion by relating the 5 partial partitions of a 2-element set to the synchronous concurrence of two one-transition automata, which contains the 4 states of the asychronous concurrence plus one more state representing the synchronization, branching off from the rest at the initial state.

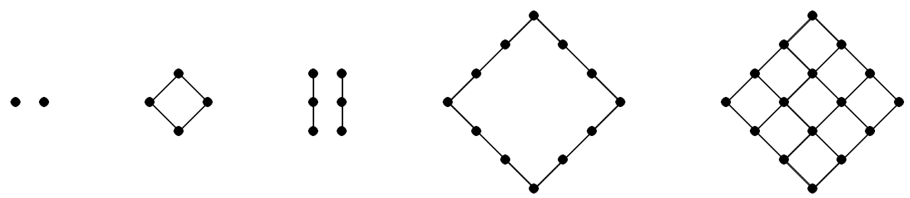
3 Geometric Automata

3.1 Motivation

The progenitor of the geometry of automata is the Poincare-Liapounov-Birkhoff geometry of dynamics [AS89, Bir27]. More recent and relevant to computation is Papadimitriou's geometric model of concurrency control [Pap86]. Shields [Shi85] refers briefly to geometrical automata, but the first detailed treatment of the notion is that of the present author [Pra91b].

Consider the one-event schedule a and its dual one-transition automaton. Suppose that by looking more closely at a we find that it is really a sequence $a_0a_1a_2$. The automaton dual to this refinement of a then has three consecutive transitions, as shown.

Now consider the two-event concurrent schedule $a+b$ on the left below and its dual 4-transition automaton, ostensibly representing $ab+ba$. If it really did represent $ab+ba$ then a similar refinement of a and b as shown should yield the 12-transition automaton (the large empty square). But that is not the automaton dual to $a_0a_1a_2+b_0b_1b_2$, which instead fills in the interior of the square with an additional 12 transitions.



We would omit the inner 12 transitions only if there were a requirement that a and b be mutually exclusive. This suggests a more detailed look at the automaton dual to $a+b$. When we really do mean that a and b are mutually exclusive, $mutex(a,b)$, meaning that their execution may not overlap, then the interpretation of the automaton as a hollow square is appropriate. But in general we should not make this assumption; instead we should regard the square as having an interior.

This is not to say that mutual exclusion is to be forbidden altogether, but rather that the default case should be no mutual exclusion. It may indeed be appropriate in certain situations to postulate $atomic(a) \wedge atomic(b) \rightarrow mutex(a,b)$. In doing so however we take the point of view that atomicity is a property of actions expressed by a predicate holding of those actions that are atomic, rather than a property inhering in all actions that are denoted by constants.

So what is the interior of a square? Must it be a continuum? An edge in a graph need not be a continuum, merely an element of the set of edges. In general we have a set of vertices called 0-cells, a set of edges called 1-cells, a set of (interiors of) squares called 2-cells, etc.

An edge e has a start vertex $s(e)$ and a final vertex $f(e)$ which together constitute its boundary. Likewise a square q has a boundary consisting of four edges we shall denote $ts(q), st(q), tf(q), ft(q)$.

These six operations are completely axiomatized by the equations $s \circ ts = s \circ st$, $f \circ ts = s \circ ft$, $f \circ st = s \circ tf$, and $f \circ ft = f \circ tf$. To decide such an equation, evaluate each composite $x \circ y$ by substituting the letter x for the occurrence of t in the string y of length 2; an equation holds just when both sides evaluate to the same string. For example $f \circ ts$ evaluates to fs as does $s \circ ft$.

This generalizes to cubes of all finite dimensions as follows. Regard every string u of length n over the alphabet s, t, f as a boundary operation on the n-cube, yielding a subcube of dimension equal to the number of t's in F. The composition $v \circ u$ is defined just when the length of v equals the number of $t's$ in u, and is then equal to the result of substituting v (preserving order of letters) for the occurrences of t in u. The resulting equational theory then consists of all equations holding between defined composites when evaluated according to this rule.

It will be noticed that this structure is a category whose objects are the natural numbers and whose morphisms are the strings, with the morphisms from m to n being the strings of length m containing n t's. The identity morphism at n is the string of n t's.

This is as much as we have time to say for now on geometric automata. Many more details may be found in [Pra91b], which includes a definition of higher-dimensional automaton based on the n-categorical notion of n-cell, generalizing 2-categorical 2-cells which in turn generalize 1-categorical 1-cells, i.e. category morphisms.

4 Acknowledgments

Thanks to Bill Lawvere for telling me about bipointed sets and their dual cubes, which gave me the idea of meet-join conservation and led within a few hours to event spaces. Thanks also to Rob van Glabbeek for help with geometric automata, and to Michael Barr for help with monads beyond any imaginable call of duty. The figure macros are a simple extension to Latex pictures and can be extracted from the beginning of this paper, available by anonymous ftp as boole.stanford.edu:/pub/algecon.tex.

References

[AS89] R.H. Abraham and C.D. Shaw. *Dynamics–The Geometry of Behavior.* Aerial Press, Inc., Santa Cruz, CA, 1989.

[Bir27] G.D. Birkhoff. *Dynamical Systems*, volume 9 of *Colloquium Publications.* American Math. Soc., 1927.

[Bir33] G. Birkhoff. On the combination of subalgebras. *Proc. Cambridge Phil. Soc*, 29:441–464, 1933.

[Bir37] G. Birkhoff. An extended arithmetic. *Duke Mathematical Journal*, 3(2), June 1937.

[Bir42] G. Birkhoff. Generalized arithmetic. *Duke Mathematical Journal*, 9(2), June 1942.

[CCMP91] 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.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

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

[Jon89] B. Jonsson. A fully abstract trace model for dataflow networks. In *Proc. 16th Annual ACM Symposium on Principles of Programming Languages*, pages 155–165, January 1989.

[Kok87] J. Kok. A fully abstract semantics for data flow nets. In *Proc. PARLE, II, LNCS 259*, pages 351–368. Springer-Verlag, 1987.

[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.

[PP88] G.D. Plotkin and V.R. Pratt. Teams can see pomsets. Manuscript available as pub/pp.tex by anonymous FTP from Boole.Stanford.EDU, October 1988.

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

[Pra91a] V.R. Pratt. Event spaces and their linear logic. In *Proc. Second International Conference on Algebraic Methodology and Software Technology*, Workshops in Computing, Iowa City, 1991. Springer-Verlag, to appear.

[Pra91b] V.R. Pratt. Modeling concurrency with geometry. In *Proc. 18th Ann. ACM Symposium on Principles of Programming Languages*, pages 311–322, January 1991.

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

[RT88] A. Rabinovich and B.A. Trakhtenbrot. Nets of processes and data flow. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, REX'88, LNCS 354*, pages 574–602, Noordwijkerhout, June 1988. Springer-Verlag.

[Rus89] J. Russel. Full abstraction for nondeterministic dataflow networks. In *Proc. 30th IEEE Symposium on Foundations of Computer Science*, October 1989.

[Shi85] 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.

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

[Win88] 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.