Time and Information in Sequential and Concurrent Computation

Vaughan Pratt*
Stanford University

January 9, 2005

Abstract

Time can be understood as dual to information in extant models of both sequential and concurrent computation. The basis for this duality is phase space, coordinatized by time and information, whose axes are oriented respectively horizontally and vertically. We fit various basic phenomena of computation, and of behavior in general, to the phase space perspective. The extant two-dimensional logics of sequential behavior, the van Glabbeek map of branching time and true concurrency, event-state duality and schedule-automaton duality, and Chu spaces, all fit the phase space perspective well, in every case confirming our choice of orientation.

1 Introduction

Our recent research has emphasized a basic duality between time and information in concurrent computation. In this paper we return to our earlier work on sequential computation and point out that a very similar duality is present there also. Our main goal here will be to compare concurrent and sequential computation in terms of this duality.

First, we have previously analyzed a number of extant logics of sequential behavior under the general heading of two-dimensional logic [Pra90, Pra94c]. Here we fit the individual models of such logics to the information-time phase space framework of this paper.

Second, R. van Glabbeek has developed a comprehensive classification of the basic models of concurrent behavior, which we here dub the "van Glabbeek map." The key feature of this classification is that it plots degrees of concreteness along two roughly independent axes. One axis is associated with the passage from linear time to branching time, which adds to the basic model information about the timing of decisions, whether a given decision is made earlier or later relative to the events being decided between. Here we fit the whole van Glabbeek classification to the phase space framework.

*This work was supported by ONR under grant number N00014-92-J-1974

Third, sequential nonbranching computation can be understand as the alternation of events and states, like the alternation of moves of a game. We fit this simplistic linear notion of behavior to our phase-space framework and obtain the basic notions of *automaton* and *schedule*, as respectively the *imperative* and *declarative* views of programs, by projection of phase space trajectories onto respectively the information and time axes.

Fourth, the passage to concurrent and branching computation can be understood as the relaxing of the linear structure on each of the axes of phase space to something weaker. (In fact all structure can be dropped, and recovered later from the phase space itself.) In this way we arrive at the notion of Chu space as a basic notion of behavior. Chu spaces are mathematically universal in that they realize all relational structures, from which we infer that there is unlikely to be a more general basic notion of behavior.

2 Background

This section reviews some basic extant models of behavior. This will provide some of the necessary background, and hopefully will also serve to orient the reader's perspective to align it with ours.

2.1 Sequential Behavior

Two basic widely used models of sequential behavior are binary relations and formal languages.

The binary relation model begins with a set W of *states* or *possible worlds* and associates with each program or activity α a set a of pairs (u, v) of states. These pairs denote the *possible* state transitions of the program: the pair (u, v) indicates that whenever the program is started in state u, the possibility exists, at least when it starts, that it will eventually terminate, and that on that termination it will be in state v.

The total interpretation is that for every u such that there exists at least one v for which $(u, v) \in a$, the program is guaranteed to eventually reach some state v' such that $(u, v') \in a$. This is the natural interpretation when *identifying* the pairs with the possible computations or *runs* of α.

The more natural interpretation however is the partial one, whereby there also exists the possibility of failure to reach any final state. In this interpretation every pair (u, v) *reflects* a run (defined somehow, but other than as merely a pair of states) which starts in state u and terminates in state v. Here more than one run may give rise to a given pair, and, of greater concern, some runs may lack a final state (or conceivably a starting state depending on the exact definition of "run") and hence give rise to no pair.

There is no natural encoding of this possibility in a binary relation on ordinary states, but it may be accommodated by adding a fictitious "bottom" state $\perp$. The presence of the pair $(u, \perp)$ in a indicates the possibility that when started in state u, program a will fail to reach a final state. A further distinction

may be drawn here between two types of failure to terminate, namely blocking and divergence. One convenient basis for this distinction is whether the run is finite or infinite. When a nonterminating run is finite it is said to *block*, and when it is infinite it is said to *diverge*. This further distinction may be represented in a binary relation by replacing the single state $\perp$ with two states β and δ indicating a blocked run and a diverging run respectively. Then (u, β) and (u, δ) indicate the respective possibilities of blocking and diverging when started in state u. when started in state u. A natural restriction is that (β, v) is only permitted as a transition when $v = \beta$, and similarly (δ, v) requires $v = \delta$; this ensures that if a fails to terminate then so does $a; b$, and via the same size of run.

The formal language model begins with a set or *alphabet* Act (sometimes Σ) of *actions*, and interprets each program as a set of strings or *traces* $u \in \text{Act}^*$. Each such trace denotes one of the *possible* sequences of actions the program may perform.

Corresponding to the divergent state of the partial interpretation of binary relations, the possibility of infinite traces may be permitted, achieved by extending Act^* to Act^∞ defined as $\text{Act}^* \cup \text{Act}^\omega$. Just as a finite string of length n may be defined as a function from the initial segment $0, 1, \ldots, n-1$ of the natural numbers to Act, collectively forming Act^*, so may an infinite string may be defined as a function from the set ω of all natural numbers to Act, collectively forming Act^ω (following the usual convention by which A^B denotes the set of all functions from B to A).

2.2 Mutex Concurrency

The advantage of the formal language model over the binary relation model is that it admits various operations of *parallel composition*, none of which are meaningfully definable for binary relations. Robin Milner has with tongue in cheek referred to concurrency defined in this inherently sequential setting as "false concurrency." A less pejorative and more informative term would be *mutex* (for mutual exclusion) concurrency, expressing the idea that when a and b are atomic, their concurrent execution is no more than $ab + ba$, which excludes their actual concurrent execution.

The basic such operation is independent or *asynchronous* parallel composition, realized by the *shuffle* or *interleaving* operation $a \| b$. We define $a \| b$ to be the set of all traces $u_1 v_1 \ldots u_n v_n$, $n \geq 0$, for which $u_1 \ldots u_n \in a$ and $v_1 \ldots v_n \in b$ and $u_i, v_i \in \text{Act}^*$. That is, the u_i's and v_i's are not actions but rather possibly empty finite traces of actions (otherwise this would be the *perfect* shuffle, with a going first). We imagine a scheduler first letting a run for a while (not just one step), then b, these two turns being called a *ply* in game parlance. The scheduler grants a total of n plies.

Whereas asynchronous parallel composition minimizes interaction, *synchronous* parallel composition maximizes it by defining $a \| b$ to be simply the intersection $a \cap b$, in the case that the programs have a common action set Act. This is the appropriate operation for the case when the two programs are run in lock step

performing the same actions.

Typical concurrent execution lies somewhere in between these two extremes. A natural blend of the two operations is obtained when a and b have their own action sets, respectively Act_a and Act_b. In this case $a \| b$ behaves synchronously on the common actions $\mathrm{Act}_a \cap \mathrm{Act}_b$ and asynchronously on the rest. Formally, $a \| b$ consists of those traces u for which $u \cap \mathrm{Act}_a^* \in a$ and $u \cap \mathrm{Act}_b^* \in b$.

2.3 True Concurrent Behavior

It is natural to regard sequential behavior as the special case of concurrent behavior in which exactly one agent is behaving. This has proved to be harder to formalize than it sounds. Our diagnosis is the the prior lack of appreciation for the independent roles of time and information in this generalization.

The crucial issue for both time and information is locality of each. In modelling a computation as the set of its possible traces, information resides in the choice of the trace while time is indicated by the position within the trace. In this model there is no connection between runs, whence we say that the information is global; creating connections localizes choice information by associating it with a particular point in a run. Dually there is complete connection within runs (i.e. they are linearly ordered), making the associated time global; relaxing linear orders to partial localizes time by dropping the premise that all pairs of events have a well-defined temporal order.

We note in passing that we may refine this model by equipping the set of traces with a real-valued probability measure, and by suitably timestamping the events of each run with a real-valued time relative to the start of the trace. This particular type of refinement does not change the global nature of either the time or the information.

The oldest model of concurrency, Petri nets [Pet62], made both time and information local, at a time when there was very little modeling of even sequential behavior in computer science.

Setting Petri nets to one side for the moment, the progression from sequential to concurrent behavior began with the basic step of replacing binary relations, as sets of state transitions (as used by Park and Hitchcock [HD73], DeBakker and deRoever [dBdR72], Pratt [Pra76], etc.), by processes as sets of traces [HL74]. This refinement permits the definition of concurrent composition as the interleaving or shuffle of traces. However this model is global as noted above with respect to both information and time.

The passage from sequential to concurrent behavior appears to have set off warning bells for some about the unsuitability of the extant sequential models for concurrent behavior. As is clear today there are two basic limitations of sets of traces. In the beginning this was much less clear, and each objector tended to focus on just one of these limitations. We now describe each of these types of objection.

2.4 Branching Time as Local Information

Milner [Mil80] was the first to spell out a notion of local information, which soon became known as branching time to distinguish it from the linear time of the trace model. Milner proposed synchronization trees as a model of behavior more concrete than sets of traces in that it fails to validate $ab+ac = a(b+c)$. A rigid synchronization tree is a rooted directed (away from the root) unoriented tree whose edges are labeled with actions from Act. Omitting "rigid" connotes a partial labeling, with the unlabeled edges understood equivalently as labeled with a "silent action" $\tau \notin$ Act. Whereas $ab+ac$ denotes a 4-edge tree branching at the root, i.e. having two a-labeled edges leaving the root, $a(b+c)$ denotes a 3-edge tree branching only after a.

While the passage from linear to branching time can be understood as generalizing a single linear order to a tree by permitting it to branch, this view misses the point that a general behavior is not a single trace but a set of traces. The passage is better understood as generalizing a set of linear orders to a synchronization tree by permitting similarly labeled initial segments to be identified. Such identifications localize choice information to the branch points of the resulting trees. With disconnected traces no information can be communicated about the timing of a choice: one may choose to understand all choices as being made at the beginning of time, or dually at the last possible moment, but no natural encoding of either a mixture of these or of intermediate timings has been proposed for sets of traces by themselves. The possibility of identifying initial segments of traces creates a degree of freedom that can be used to indicate when during the behavior a particular choice was made. Connecting only at the start means the choice is made at the start, connecting along some initial segment means that the choice is made at the end of that segment.

2.5 True Concurrency as Local Time

The dual of local information is local time, popularly called true concurrency in Europe starting around 1987. Local time was addressed early on by Mazurkiewicz [Maz77], who realized it in the form of a symmetric binary relation on the alphabet Act indicating *causal independence* of occurrences of those actions. This relation induces a congruence (with respect to Kleene's regular operations plus shuffle) on Act^*, and more generally on Act^∞, whereby two traces are congruent when they may be obtained from each other by interchanging adjacent occurrences of independent actions. Mazurkiewicz called the quotient of Act^* by this congruence a *partial monoid*; here partiality is understood to relax the ordering of traces, now understood to be partial in contrast to Act^*'s linear traces, rather than restricting the domain of concatenation as the term "partial monoid" might lead one to expect.

Mazurkiewicz traces, as the elements of such a partial monoid have come to be called, associate the independence information with the action alphabet Act. A stronger notion of true concurrency associates it instead with the *action occurrences* or *events* of a particular behavior. Viewing traces as *linearly* ordered

sets of events in which each event is labeled with the action of which it is an occurrence, one relaxes this linear order to a partial order to arrive at what has been called a *partial word* [Gra81] and a *partially ordered multiset* [Pra82] or pomset, the term now used.

In a pomset with two occurrences of action a, an occurrence of b may be comparable with one of the a's and incomparable with the other, not possible with Mazurkiewicz traces as originally formulated (Mazurkiewicz subsequently extended his notion to multitraces to achieve this effect). If for example a is the action of writing a 1 into a memory and b that of reading 1, then a given occurrence of b should be comparable with the responsible occurrence of a, but not with subsequent occurrences of a, which having no causal relationship to that occurrence of b can be performed independently of it. This distinction must be expressible in order to distinguish a simple memory whose reads and writes are interleaved, with the read returning the most recently written value, from a more sophisticated one that permits simultaneous reading and writing with the expectation that the read will return a previously written value.

3 Two-Dimensional Logic

We recently [Pra94c] defined a general notion of two-dimensional logic, into which various logics of action could be fitted. The significance of two-dimensional logics for the present paper is that their disjunction and conjunction operations are the natural operations of accumulation for respectively information and time. We return to this point after a walk around the domain of two-dimensional logics to familiarize ourselves with the scope of such logics.

The following picture of the main two-dimensional logics should be understood as a cube with appendages.

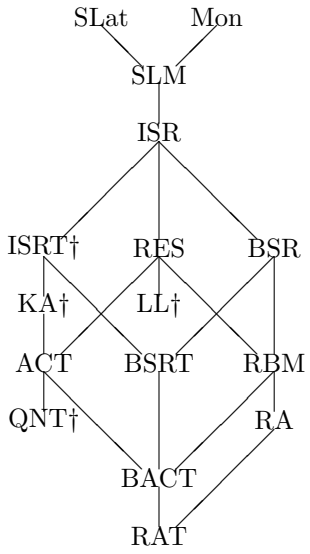

Figure 1. Roadmap of 2D Logics

Starting from the top of this diagram, a *semilattice* is a set A with an associative commutative idempotent binary operation $a+b$; **SLat** denotes the class thereof. We write $a \leq b$ for $a+b=b$, a binary relation that can be shown to be partially order A. Semilattices form the models of a basic logic of *disjunction*.

A *monoid* is a set A with an associative binary operation ab and a constant 1 as that operation's unit or neutral element, $a1 = a = 1a$; the associated class is **Mon**. We shall use general monoids to model *noncommutative conjunction*.

A set with both these operations and satisfying a two-sided distributivity principle $a(b+c) = ab + ac$ and $(a+b)c = ac + bc$ is called an *idempotent semiring*, forming the class **ISR**. This combines conjunction and disjunction in the one logic.

The three axes of the cube having **ISR** at its apex correspond to the following constraints on ISR's. The subclass **ISRT** of **ISR** consists of those ISR's in which for every element a there exists a least reflexive ($1 \leq a$) transitive ($aa \leq a$) element a^* such that $a \leq a^*$. Logics with such an operation cater to *iterated conjunction* or repetition.

The subclass **RES** of residuated ISR's consists of those ISR's in which for all elements a, b there exists a greatest element $a \rightarrow b$ such that $a(a \rightarrow b) \leq b$, and dually a greatest element $b \leftarrow a$ such that $b(b \leftarrow a) \leq a$; $a \rightarrow b$ is called the *right residual* of b by a, while $b \leftarrow a$ is dually called the *left residual* of b by a. Such logics add implication to the language. These are the Ajdukiewicz monoids [Ajd37], brought to greater prominence two decades later by Lambek [Lam58]. The term residuation was coined by Ward and Dilworth [WD39]. That the binary relations on a set formed a residuated ISR was first observed by De

Morgan as his "Theorem K" [DM60].

The subclass **BSR** of Boolean semirings has the property that the semilattice structure forms a Boolean algebra; for this it suffices for an antimonotone operation of negation $\neg a$ to exist and satisfy $\neg\neg a = a$. This yields classical logic with a second conjunction $\neg(\neg a + \neg b)$ distinct (in general) from the main conjunction ab; pure Boolean logic obtains when these operations coincide.

The remaining classes **ACT**, **BSRT**, and **RBM** combine these conditions pairwise, and **BACT** imposes all three conditions, see [Pra90, Pra94c] for details.

The class **QNT** of quantales consists of ISR's for which the semilattice is complete (has all suprema or joins $\Sigma_i a_i$ including the empty supremum 0 and infinite suprema). Residuals and transitive closures always exist in quantales, being expressible as respectively infinite infima and infinite suprema.

The Jonsson-Tarski class **RA** of *relation algebras* is obtained from **RBM** by identifying the left and right residuals of the Boolean complement 0' of 1 by any given value a: $a \rightarrow 0' = 0' \leftarrow a$. This class together with transitive closure, **RAT**, was the subject of Judith Ng's thesis [NT77, Ng84].

There are two basic models of two-dimensional logic that pervade computer science, formal languages as sets of strings, and binary relations as sets of pairs. For both, $a + b$ is union. For languages, ab is concatenation, while for binary relations it is composition or relative product. For a fixed set X, the algebra of all binary relations on X, i.e. all subsets of X^2, under $a + b$ and ab, belongs to both **QNT** and **RAT**. Likewise so does the algebra of all languages on X treated as an alphabet, i.e. all subsets of the set X^* of all finite strings on X.

Sometimes one does not want all languages or all binary relations but only certain ones, which collectively then may not form an RAT or a quantale but may nevertheless belong to one of the larger classes in the above classification. In addition other structures closed under operations $a + b$ and ab suitably interpreted may satisfy enough conditions to locate them collectively in one or another of these classes.

Now let us turn to the connection with information and time. Our intuitions as to how each of these accumulate are nicely captured by the two basic operations of two-dimensional logic.

We understand information from an information theoretic perspective. Here information is measured not by number of facts but by variety of choices. When there is only one option no information can be conveyed by indicating that it is the option. Two options permit one bit, four two bits, etc. The operation $a+b$ is taken to denote accumulation of choices. The order in which two choices are presented does not matter, expressed by the commutativity of $a + b$. Being presented with the same choice twice does not increase our options, expressed by idempotence $a + a = a$.

We think of ab as the *temporal* conjunction *a and then b*, as in "open the door and then walk through it." This operation accumulates not facts but rather events. Like $a + b$ it is associative. However it is not commutative, witness walking through the door and then opening it. Nor is it idempotent, witness the action of paying a dollar, which when repeated becomes the action

of paying two dollars. This in contrast to *factual* conjunction, where two facts may be learned in either order (commutativity), and having learned a fact one learns nothing new by learning it again (idempotence).

An ordered monoid in ISR may be understood as a monoidal category, and as such, as a one-object 2-category. A 2-category has two compositions, conventionally called horizontal (composing the 1-cells) and vertical (composing the 2-cells). These compositions correspond to respectively the monoid conjunction and the semilattice disjunction. 2-cells are then naturally drawn with the orientation as typified by the following diagram illustrating the interchange law for 2-categories.

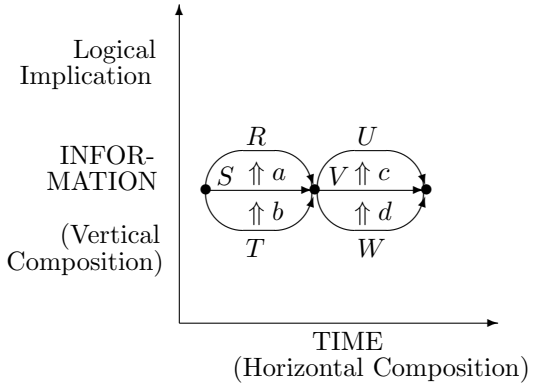

Figure 2. 2D Logic

This is the first of what will be four situations all fitting the same basic information-time phase diagram. In all four cases the conventions that have emerged favor orienting information vertically and time horizontally.

4 The van Glabbeek Map

In the previous section the individual logics were two-dimensional, forming a landscape that was not itself two-dimensional. We pass now to the landscape of concurrency models studied in the thesis of our colleague R. van Glabbeek. Here the landscape itself is a two-dimensional logic. Van Glabbeek's original landscape, which he chose to orient as here, contained some 36 models of concurrency; here we have cut it down for simplicity to an incompletely filled 3×3 array of a mere half dozen models.

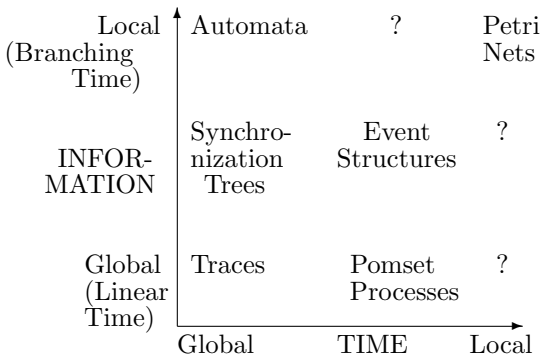

Figure 3. A Van Glabbeek Map.

In the previous section the two axes measured respectively increasing number of options (no choices at the bottom, many choices at the top) and increasing time (early on the left, late on the right). Here the axes measure *expressiveness*. The information axis measures the precision with which the timing of a branch may be specified. At the bottom no information is conveyed as to when a decision is made. At the synchronization tree level $ab + ac$ can be distinguished from $a(b + c)$, which we associate with respectively early and late commitment to the choice of b or c. At neither of these levels can it be specified when the information represented by these choices is subsequently forgotten; it would seem that all choices are remembered forever. At the automaton level we can distinguish between $ac + bc$ and $(a + b)c$ as respectively long and short memory. The former remembers the choice longer until after c, while the latter forgets it before c, a distinction drawn by conventional (finite) state automata as well as by Boudol's flow event structures [Bou90].

The vertical axis therefore indicates the passage from global choice, where choice commitment and forgetting has no specific location in the computation, to local choice, that is, localized to specific points in the computation.

The time axis measures the precision with which we may distinguish the truly concurrent execution $a\|b$ of a and b from the "false concurrency" or mutual exclusion $ab + ba$. Traces fail to draw this distinction, for which Grabowski [Gra81] and Pratt [Pra82] introduced pomsets, as labeled *partial* orders generalizing the notion of trace as a labeled *linear* order. Indicating time by a single global clock leads naturally to the trace model, in which all pairs of events have a well-defined order. When time can only be measured by local clocks, this linear order can no longer be guaranteed and it becomes natural to understand time as only partially ordering events.

The horizontal axis therefore indicates the passage from global time to local time.

Petri nets achieve more locality in these two coordinates than any competing model. They accomplish this by permitting in-degree and out-degree greater than one at both places (state-holders) and transitions (event-holders). The branching, both in and out, is disjunctive at places and conjunctive at transitions. Branching at places increases the expressiveness of Petri nets in the information domain while branching at transitions increases expressiveness in the time domain.

An ordinary sequential automaton may be understood as a Petri net with no branching at transitions, either in or out, only at places. In the "token game" standardly used to equip Petri nets with a semantics, an automaton can neither create nor destroy tokens. Sequential computation can then be understood as the trajectory of a single token through the net.

A pomset process, as a set of pomsets analogous to a trace process as a set of traces, may be understood as a Petri net with no branching at places.

5 Linear Automata and Schedules

We turn now to a point of view we developed recently [Pra92] as a simple setting in which time and information were dual. There we gave up all branching in both the time and information domains to reduce this duality to its simplest form. The essence of this duality reduced to the self-duality of suitable categories of chains, which we illustrated there with the category of finite chains with bottom and their monotone functions; certain other categories of chains are also self-dual.

Persisting with our two-dimensional phase space, we return to our original understanding of the horizontal axis as indicating increasing time. We take the vertical axis however to be *decreasing* options, whence we associate upwards motion with learning as the process of narrowing the possible alternatives. For simplicity we suppose that both axes order their respective domains linearly.

We now view computation, and more generally any behavior, as alternately changing state and passing time, like a chess player making a move and then awaiting her opponent's move. Thus behavior may be depicted as a monotonically increasing trajectory as follows.

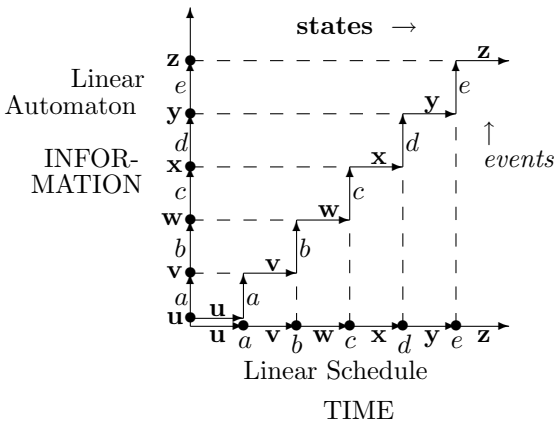

Figure 4. A Linear Phase Space.

The horizontal moves are *states*, in which time passes while information remains fixed. We use $u - z$ to denote states, which we suppose are drawn from a set X of states.

The vertical moves are *events*, which holds time fixed (events are ephemeral) while changing information. We use $a - e$ to denote events, drawn from an event set A.

A trajectory in phase space constitutes a neutral view of behavior. We take sides by projecting the trajectory onto one or the other axis. Projecting it onto the information axis sends the states to points while preserving the spatial extent of the events. The resulting graph can then be seen to be an *automaton*, of the straightline kind associated with a completely nonbranching behavior.

Dually, projecting the trajectory onto the time axis sends events to points and states to lines, recognizable as a *schedule*, again linearly ordered as with the automaton.

Because the trajectory is simply a staircase, only the number of steps matters. Since this can be recovered from either the automaton or the schedule (taking suitable care at the endpoints not to lose information), either one suffices as our view of this admittedly very simple behavior.

We have represented the trajectory as a line, but it may equivalently be understood as a partition of phase space into an upper and lower part, representable by a function from phase space to $\{0, 1\}$. We may then recover the trajectory as the common boundary of the two blocks of this partition of phase space. This point of view leads naturally into the Chu space generalization of linear computation. This generalization will retain the property that projection onto either axis loses no information. That is, we can program general Chu spaces with either schedules or automata with equal precision of specification.

6 Chu Spaces

6.1 Concept

A Chu space is a structure of the kind considered in the previous section, less the assumption of linearity for the two axes. Chu spaces are particularly simple in that their axes are postulated with *no* particular structure, that is, they are taken to be pure sets X of states (the vertical axis) and A of events (the horizontal axis). In place of a trajectory of alternating states and events through phase space, we associate with each pair (x, a) a value which may be interpreting as either the *truth* of a in x (an information-theoretic view), or the *time* of a relative to x (when in state x, how long ago did a happen?).

The simplest nontrivial Chu spaces have binary entries. A 1 at location (x, a) may read as either that a is true in possible world x (the modal logic viewpoint), or that in state x, a has already happened (the dynamic viewpoint). There is no identifiable notion of "a happening during x," in each state a either has or has not happened.

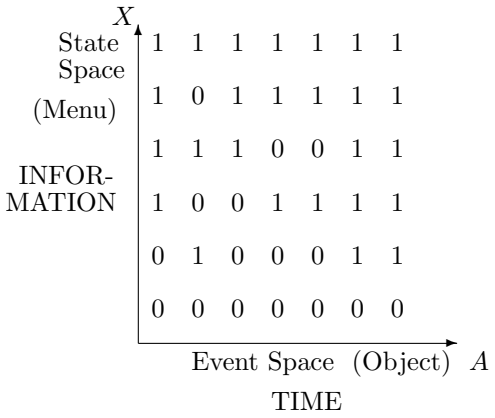

Figure 5. A Chu Space.

6.2 Example: Sets of Traces

It will be helpful to see how traditional formal languages as sets of strings fit in here. To begin with, these Chu spaces are to formal languages as the real plane is to the Mona Lisa: only the canvas is represented, the paint is omitted. For languages this means that we regard an *occurrence* of a symbol in a string as a point somewhere along the string, and the symbol of which it is an occurrence as a mere label on that point indicating the "color" it has been painted, our palette being the alphabet Σ. Our theory neglects these labels or colors and treats the underlying geometry of the behavior of automata rather than the full notion

of behavior corresponding to a finished work of art. In this respect it is just like Hilbert space as it arises in quantum mechanics, which makes no attempt to label either its dimensions or its points with anything concrete, this being left to operators for extracting physical quantities from wavefunctions defined as points of Hilbert space.

A formal language is then understood as just a set of anonymous finite strings, the only feature of which that now matters is its length. The set A of events is taken to be the set of all occurrences of symbols in strings, equivalently the disjoint union of the strings, together with one additional point called ∞ (this point is added for the sake of a nice duality property of formal languages that does not obtain in general for arbitrary Chu spaces). If two distinct strings of the same language have the same length, they remain distinguished in A. In particular if language L consists of two strings both of length three, e.g. *abc* and *aba*, A will consist of seven points, namely the six occurrences of symbols in L together with ∞.

We take the set X of states to be A, reflecting the above-mentioned peculiarity of formal languages that they are self-dual. We define $x \models a$ to be 1 (true) just when x and a are in the same string and x follows a (including the case when x *is* a). Otherwise it is 0, the case when x and a come from different strings, or from the same string but with x strictly preceding a, or when either is ∞.

The underlying geometry of L can be recovered from this construction of $(X, A, \models)$ by defining an equivalence relation on A which makes two points equivalent just when one precedes or equals the other. The strings are then the equivalence classes, and ∞ is the element that is not related to any element, even itself. The restriction of $\models$ to a string linearly orders that string, allowing us to reconstruct the position of each occurrence in its containing string. This construction works even for languages with infinitely many strings, including the duality property, which however does not hold in general in the presence of infinitely long strings. This completes the account of how a language consisting of finite strings can be represented as a Chu space.

6.3 Schedules and Automaton

We observed that in the linear case no information was lost in the projection of a trajectory onto either axis. With suitable care in the definition of projection we may accomplish the same for *biextensional* Chu spaces, namely those having no repeated rows or columns, via a process we have previously described [GP93], which we sketch briefly here.

We project onto the information axis to yield an automaton in two steps. First close the rows of the Chu space under arbitrary union and intersection (OR and AND of the rows as bit-vectors), by adding new rows as needed. Now draw the poset of all resulting rows ordered by inclusion, and color the elements black or white according to whether they were originally present or not. This poset turns out to be a profinite distributive lattice, that is, a complete distributive lattice (one having all meets and joins including the empty and infinite ones)

whose maximal chains are nowhere dense: between any two distinct points of the chain lie two distinct points with no other point between them. We recover A as those elements of this lattice not the join of the lattice elements strictly below them, X as the set of black points of the lattice, and $x \models a$ as the relation $a \leq x$. The theorem [GP93] is that recovered space is isomorphic to the originally projected space.

This construction works equally well when "row" and "column" are interchanged everywhere, dealing immediately with the case of projection onto the time axis, yielding a schedule.

That these objects are recognizable automata and schedules respectively (with loops and disjunctive confluences unfolded) can be appreciated from the examples in [Gup93, GP93, Pra93b, Gup94, Pra94a].

6.4 Universality of Chu Spaces

The categories $\mathbf{Str}_\kappa$ of κ-ary relational structures and their homomorphisms where κ is any ordinal are universal categories for mathematics to the extent that they *realize* many familiar categories: groups, lattices, and Boolean algebras when $\kappa = 3$, rings, fields, and categories when $\kappa = 4$, etc. We say that a concrete category D *realizes* a concrete category C when there exists a functor $F : C \to D$ that is full and faithful and which commutes with the respective underlying-set functors of C and D.

Elsewhere [Pra93b, pp.153-4] we proved that the self-dual category $\mathbf{Chu}_{2^\kappa}$ of Chu spaces over the power set of $\kappa = \{0, 1, \ldots, \kappa - 1\}$ realizes $\mathbf{Str}_\kappa$. Relational structures being universal, this makes Chu spaces at least as universal. The embedding being "sparse," there are many other Chu spaces besides those representing relational structures; in particular those Chu spaces dual to some relational structure may prove as useful as that structure itself. We have more recently streamlined our original argument, and it is presently available by anonymous ftp as /pub/uni.tex.Z from boole.stanford.edu.

When a model of behavior at the same time forms a universal category for mathematics, it is reasonable to infer that a more general model is not possible. The one flaw in this argument is in the amount of structure preserved by the functor doing the modeling. For the functor to destroy essential structure constitutes an "incompleteness" in the universality of the model, in turn weakening the claim to being a universal model of computation. This situation could be resolved either way: Chu spaces could turn out to capture all the structure that matters, or the category of Chu spaces over K as a whole may turn out to lack essential structure that can be recovered only by giving up the appealing self-duality of the category, or its completeness or cocompleteness, etc. This issue most certainly deserves to be pursued further.

6.5 Duality

The duality of time and information is reflected for Chu spaces in a variety of ways. Mathematically it is represented by how Chu spaces transform. A Chu

transform from $(A, X, \models)$ to $(A', X', \models')$ consists of a pair (f, g) of functions $f : A \to A'$, $g : X' \to X$, satisfying $g(x) \models a \;=\; x \models f(a)$ for all $a \in A$ and $x \in X'$. These have the evident composition as $(f, g)(f', g') = (ff', g'g)$ with evident identies and thus constitute the morphisms of the category $\mathbf{Chu}(\mathbf{Set}, 2)$ of Chu spaces over the two-element set $2 = \{0, 1\}$. Chu spaces have an equally evident duality obtained by transposition, which has the side effect of reversing the direction of the Chu transforms. As noticed by Barr [Bar91] and further developed by several authors [dP89, BG90, LS91, BGdP91, Pra93b], Chu spaces provide a straightforward interpretation of full linear logic; the fact that **Set** (as well as **Pos** and other even larger cartesian closed categories) is comonadic (cotripleable) in **Chu** gives a straightforward interpretation of Girard's "bang" operation $!A$ as the functor of that comonad.

For those who are put off by the transformational view of duality, it is possible to appreciate the duality of Chu spaces in entirely noncategorical terms.

The interpretation of each row of a Chu space as one of the *possible* "paintings" of the underlying set A of points of the space is in agreement with the basic theme of this paper, that the vertical axis of the information-time phase diagram denotes a *disjunctive* space, and thereby constitute a *menu*. *The basic feature of a menu is that its entries are selected one at a time.*

Likewise each column of a Chu space as the "identifier" of a point makes the horizontal axis conjunctive: this point is identified in this way *and* that point is identified in that way and so on. The points of the space coexist, and thereby constitute an *object*. *The basic feature of an object is that its points coexist.*

With regard to orientation, menus normally list their selections vertically, while the elements of an object such as a Lisp list or an APL array are usually listed horizontally.

A set (of points), as a set transforming by functions, is a pure body or *object*. An antiset (of states), as a set transforming by antifunctions (the converse of a function), is a pure mind or *menu*. A Chu space $(A, X, \models)$, as a binary relation $\models$ from an antiset X to a set A, denotes a more or less productive blend of mind and body. The extremal Chu spaces are $(A, 2^A, \ni)$ and $(2^X, X, \in)$, denoting respectively the object or pure body A and the menu or pure mind X. The productive interaction of mind and body obtains for the squarer spaces in between.

The idea that a menu is mental is reinforced by the observation that the category of sets and antifunctions is equivalent to the category of complete atomic Boolean algebras and their complete homomorphisms. Boolean logic is the purest possible logic in that it has the maximum possible equations for its signature: adding just one more equation to the equational theory of Boolean algebras leads to inconsistency in the sense that $x = y$ can be proved for all terms x and y.

Menus and objects are analogous to your moves and your opponent's moves. The former are your opportunities, one of which you must select from, the latter are your risks, all of which must be guarded against since any one might happen.

Computer scientists are chided by some mathematicians for their preference for logic over algebra. Yet this preference is more deeply rooted than generally

appreciated. Algebra transforms via *functions*, which up to isomorphism serve to *identify* (not all functions are injective) and *adjoin* (not all functions are surjective). Dually, *antifunctions* serve respectively to *copy* and *delete*. But in computer science the natural operations are copy and delete, which are easily implemented. Identification of existing elements and adjoining new elements are more sophisticated operations that each require some effort to implement.

This bias makes antifunctions, alias complete Boolean homomorphisms but more naturally understood as copy and delete operations, the natural side of the set-antiset duality for computer scientists to gravitate towards. Mathematicians on the other hand are above mere copying and deleting of individual points and prefer to operate on whole spaces. This moves them "up one exponential," and their concern is therefore not surprisingly with the dual space.

We close by mentioning two striking similarities with quantum mechanics. First, the rows and columns of a Chu space interfere in essentially the same way that conjugate variables of a quantum mechanical system interfere to limit their joint precision according to Heisenberg's inequality $\Delta p.\Delta q \geq \hbar$ expressing the celebrated uncertainty principle of quantum mechanics. Second, residuation of Chu spaces, from which the associated state transition matrix (which makes no reference to events) may be recovered, amounts to their inner product when binary relations are understood as the points of a sort of "logician's Hilbert space." Furthermore the progression from automata to Chu spaces recapitulates the progression from Langrangian mechanics retaining even some of the essential properties of the Legendre transform by which this passage is standardly accomplished. Some of these connections are developed in [Pra93a, Pra94b], although the details of this second point of contact have only become clear to us since then.

These connections get considerably closer to the essence of quantum mechanics than does quantum logic [BvN36], which abstracts away from complementarity to capture just the underlying projective geometry of quantum mechanics. Quantum logic is to Chu spaces as nonconstructive methods of mathematics are to constructive, e.g. deciding whether one can get somewhere vs. exhibiting a route. Nonconstructivity confines itself to the "pure mind" extreme of mathematics, in the neighborhood of Boolean algebras and antisets. The full gamut of mathematics, from pure body to pure mind, can only be spanned constructively.

I believe that with due care computer science and physics can be made to look more like each other, and like mathematics, than they presently appear. Such connections would then broaden the role of constructivity to make it equally essential for all three subjects. For computer science at least, the least surprised by this will be the humble programmer, who would have great difficulty imagining that much good could come of purely nonconstructive software that indicated the end but not the means.

References

[Ajd37] K. Ajdukewicz. Die syntaktische konnexität. *Studia Philosophica*, I:1–27, 1937. English translation in S. McCall, Polish Logic 1920-1939, Clarendon Press, Oxford, 1967.

[Bar91] M. Barr. $*$-Autonomous categories and linear logic. *Math Structures in Comp. Sci.*, 1(2), 1991.

[BG90] C. Brown and D. Gurr. A categorical linear framework for Petri nets. In J. Mitchell, editor, *Logic in Computer Science*, pages 208–218. IEEE Computer Society, June 1990.

[BGdP91] C. Brown, D. Gurr, and V. de Paiva. A linear specification language for Petri nets. Technical Report DAIMI PB-363, Computer Science Department, Aarhus University, October 1991.

[Bou90] G. Boudol. Computations of distributed systems, part 1: flow event structures and flow nets, 1990. Report INRIA Sophia Antipolis, in preparation.

[BvN36] G. Birkhoff and J. von Neumann. The logic of quantum mechanics. *Annals of Mathematics*, 37:823–843, 1936.

[dBdR72] J.W. de Bakker and W.P. de Roever. A calculus for recursive program schemes. In M. Nivat, editor, *Automata, Languages and Programming*, pages 167–196. North Holland, 1972.

[DM60] A. De Morgan. On the syllogism, no. IV, and on the logic of relations. *Trans. Cambridge Phil. Soc.*, 10:331–358, 1860.

[dP89] V. de Paiva. A dialectica-like model of linear logic. In *Proc. Conf. on Category Theory and Computer Science, LNCS 389*, pages 341–356, Manchester, September 1989. Springer-Verlag.

[GP93] V. Gupta and V.R. Pratt. Gates accept concurrent behavior. In *Proc. 34th Ann. IEEE Symp. on Foundations of Comp. Sci.*, pages 62–71, November 1993.

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

[Gup93] V. Gupta. Concurrent kripke structures. In *Proceedings of the North American Process Algebra Workshop, Cornell CS-TR-93-1369*, August 1993.

[Gup94] V. Gupta. *Chu Spaces: A Model of Concurrency.* PhD thesis, Stanford University, September 1994. Tech. Report, available as ftp://boole.stanford.edu/pub/gupthes.ps.Z.

[HD73] P. Hitchcock and Park D. Induction rules and termination proofs. In M. Nivat, editor, *Automata, Languages and Programming*. North-Holland, 1973.

[HL74] C.A.R. Hoare and P.E. Lauer. Consistent and complementary formal theories of the semantics of programming languages. *Acta Informatica*, 3:135–153, 1974.

[Lam58] J. Lambek. The mathematics of sentence structure. *American Math. Monthly*, 65(3):154–170, 1958.

[LS91] Y. Lafont and T. Streicher. Games semantics for linear logic. In *Proc. 6th Annual IEEE Symp. on Logic in Computer Science*, pages 43–49, Amsterdam, July 1991.

[Maz77] A. Mazurkiewicz. Concurrent program schemas and their interpretation. In *Proc. Aarhus Workshop on Verification of Parallel Programs*, 1977.

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

[Ng84] K.C. Ng. *Relation Algebras with Transitive Closure*. PhD thesis, University of California, Berkeley, 1984. 157+iv pp.

[NT77] K.C. Ng and A. Tarski. Relation algebras with transitive closure, Abstract 742-02-09. *Notices Amer. Math. Soc.*, 24:A29–A30, 1977.

[Pet62] C.A. Petri. Fundamentals of a theory of asynchronous information flow. In *Proc. IFIP Congress 62*, pages 386–390, Munich, 1962. North-Holland, Amsterdam.

[Pra76] V.R. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proc. 17th Ann. IEEE Symp. on Foundations of Comp. Sci.*, pages 109–121, October 1976.

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

[Pra90] V.R. Pratt. Action logic and pure induction. In J. van Eijck, editor, *Logics in AI: European Workshop JELIA '90, LNCS 478*, pages 97–120, Amsterdam, NL, September 1990. Springer-Verlag.

[Pra92] V.R. Pratt. The duality of time and information. In *Proc. of CONCUR'92, LNCS 630*, pages 237–253, Stonybrook, New York, August 1992. Springer-Verlag.

[Pra93a] V.R. Pratt. Linear logic for generalized quantum mechanics. In *Proc. Workshop on Physics and Computation (PhysComp'92)*, pages 166–180, Dallas, 1993. IEEE.

[Pra93b] V.R. Pratt. The second calculus of binary relations. In *Proceedings of MFCS'93*, pages 142–155, Gdańsk, Poland, 1993. Springer-Verlag.

[Pra94a] V. Pratt. Chu spaces: complementarity and uncertainty in rational mechanics. Technical report, TEMPUS Summer School, Budapest, July 1994. Manuscript available as pub/bud.tex.Z by anonymous FTP from Boole.Stanford.EDU.

[Pra94b] V.R. Pratt. Chu spaces: Automata with quantum aspects. In *Proc. Workshop on Physics and Computation (PhysComp'94)*, Dallas, 1994. IEEE.

[Pra94c] V.R. Pratt. A roadmap of some two-dimensional logics. In J. Van Eijck and A. Visser, editors, *Logic and Information Flow (Amsterdam 1992)*, pages 149–162, Cambridge, MA, 1994. MIT Press.

[WD39] M. Ward and R.P. Dilworth. Residuated lattices. *Trans. AMS*, 45:335–354, 1939.