Information, Processes and Games

Samson Abramsky
Oxford University Computing Laboratory

Contents

1 Introduction

Philosophers of science are concerned with explaining various aspects of science, and often, moreover, with viewing science as a kind of gold-mine of philosophical opportunity. The direction in both cases is *philosophy from science*. For a theoretical scientist, the primary inclination is often to see conceptual analysis as a preliminary to a more technical investigation, which may lead to a new theoretical development. In short: *science from philosophy*. This article is written mainly in the latter spirit, from the stand-point of Theoretical Computer Science, or perhaps more broadly "Theoretical Informatics": a — still largely putative — general science of information. That being said, we hope that our conceptual discussions may also provide some useful grist to the philosopher's mill.

1.1 Towards Information Dynamics

The best-known existing mathematical theories of information are (largely) *static* in nature. That is, they do not explicitly describe informatic processes and information flow, but rather certain *invariants* of these processes and flows. There is by now ample experience from Computer Science which indicates that it is fruitful, and eventually necessary, to develop fully-fledged dynamical theories. We shall try to map some steps in this direction.

We begin by reviewing some of the theories developed in Computer Science which form the background for our discussion. Then we consider a rather basic conceptual puzzle: **(how) does information increase in computation?** This will provide a context for discussing another important issue in theories of information: the distinction between *qualitative* and *quantitative* theories, and how they can be reconciled — or, more positively, combined. Our discussion here will still be at the level of *static* theories. We then go on to consider dynamic theories proper.

1.2 Matter and Method

This article is well outside the author's usual remit as a researcher. While it is clearly not a contribution to philosophy, it cannot be said to be the usual kind of conceptually-oriented overview of a scientific field which one might find in such a Handbook (and of which there are some fine examples in the present volume) either; not least for the reason that the scientific

field we are attempting to overview does not exist yet. Rather, the purpose of this article is to play some small part in causing this field to come into being.

What, then, is this nascent field? We would like to use the term *Information Dynamics*, which was proposed some time ago by Robin Milner, to suggest how the area of Theoretical Computer Science usually known as "Semantics" might emancipate itself from its traditional focus on interpreting the syntax of pre-existing programming languages, and become a more autonomous study of the fundamental structures of Informatics. I believe that the development of such a field would transform our scientific vision of Information, and give us a whole new set of tools for thinking about it. Hence its relevance for any attempt to develop a Philosophy of Information.

Rather than a developed field of Information Dynamics, with some consensus as to what its fundamental notions and methods are, what we have at present are some *partial exemplifications*; some theories which have been shown to work well over certain ranges of applications, and which exhibit both conceptual and mathematical depth. My approach to conveying the current state of the art, and indicating what seem to me the major objectives visible from where we stand now, is necessarily largely based on describing (some of) these current theories—emphasizing those that seem most promising to me. The obvious danger with this approach is that this article will appear to be a disjointed series of descriptions of various formalisms. We have probably not succeeded in avoiding this completely—despite the author's best efforts. But we regard the expository aspect of this article as important in itself. The theories we shall expound deserve to be known in wider circles than they presently are. And our discussions of Domain theory, Game semantics and Geometry of Interaction delve more into conceptual issues, while minimizing the level of technical detail, than other accounts of which I am aware.

To assist the reader in keeping their bearings, we mention some of the main themes which will thread through our discussion:

Information Increase in Computation We compute in order to gain information: but how is this possible, logically or thermodynamically? How can it be reconciled with the point of view of Information Theory? How does information increase appear in the various extant theories? This will be an important explicit theme in our discussion of background theories in Section 2, and particularly in Section 3. Obtaining a good account in the context of dynamic theories, as exemplified by those presented in Sections 4 and 5, is a key desideratum for future work.

Unifying Quantitative and Qualitative Theories of Information We mainly discuss this explicitly in Section 3, where we describe some remarkable recent progress which has been achieved by Keye Martin and Bob Coecke, in the setting of current static theories of information (Scott Domain Theory and Shannon Information Theory). A similar development in the setting of the dynamic theories described in Sections 4 and 5 is a major objective for future research.

Information Dynamics: Logic and Geometry We introduce Game Semantics and Geometry of Interaction on Sections 4 and 5 as substantial partial exemplications of Information Dynamics. They have strong connections to both Logic and Geometry, and form a promising new bridge between these two fields. While we shall not be able to do full justice to these topics, we hope at least to raise the reader's awareness of these developments, and to provide pointers into the literature.

The Power of Copying, and Logical Emergence This is mainly developed in Section 5, in the context of Geometry of Interaction-type models.

One theme which we have, regretfully, omitted is that of the emerging connections with Physics, in particular with **Quantum Information and Computation**. Here there is already much to say (see e.g. [6, 7, 8]). We have not included this material simply for lack of the appropriate physical resources of space, time and energy.

2 Some Background Theories

Following our previous discussion, we can classify theories of information along two axes: as static or dynamic, and as qualitative or quantitative. We list some examples in the following table.

	Static	Dynamic
Qualitative	Domain theory, Dynamic Logic	Process Algebra
Quantitative	Shannon Information theory	

Shannon Information theory is discussed in detail in another Chapter of this Handbook. In this Section, we shall give brief overviews of the other three theories listed above, which have all been developed within Computer Science—Domain Theory and Dynamic Logic originating in the 1970's, and Process Algebra in the 1980's.

We shall devote rather more space to Domain Theory than to the other two theories, for the following reasons:

- Domain Theory is much more integrally and explicitly a theory of information than Dynamic Logic or Process Algebra, and will figure significantly in our subsequent discussions.
- The other theories will receive some coverage elsewhere in this Handbook, notably in the Chapter by Baltag and Moss.

2.1 Domain Theory

Domain Theory was introduced by Dana Scott *c.* 1970 [74] as a mathematical foundation for the denotational semantics of programming languages which had been pioneered by Christopher Strachey. A *domain* is a partially ordered structure $(D, \sqsubseteq)$. The best intuitive reading of elements of D is as *information states*. We pass immediately to some illustrative examples.

2.1.1 Examples of Domains

Flat Domains Given a set X, we can form a domain $X_\perp$ by adjoining an element $\perp \notin X$, and defining an order by

$$x \sqsubseteq y \iff x = \perp \vee x = y.$$

Frequently used examples : $\mathbb{N}_\perp$, $\mathbb{B}_\perp$, $\mathbb{O} = \mathbf{1}_\perp$. Here $\mathbb{N} = \{0, 1, 2, \ldots\}$, the set of *natural numbers*; $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\}$, the set of *booleans*; and $\mathbf{1} = \{*\}$, an (arbitrary) one-element set.

We can use such flat domains to model computations in terms of very simple *processes of information increase*. Thus a (possibly non-terminating) natural number computation can

be modelled in $\mathbb{N}_\perp$ in the following sense. Initially, no output has been produced. This "zero information state" is represented by the bottom element $\perp$. If the computation terminates, a natural number n is produced. Thus we obtain the "process"

$$\perp \sqsubseteq n.$$

The case where no output is ever produced is captured by the "stationary process" $\perp$, which we can view more "dynamically" as

$$\perp \sqsubseteq \perp \sqsubseteq \cdots$$

Streams Now consider the scenario where we have an unbounded or potentially infinite tape (much as in a Turing machine), on successive squares of which symbols from some finite alphabet Σ can be printed. This computational scenario is naturally modelled by the domain Σ^∞, the set of finite and infinite sequences of elements of Σ. This is ordered by *prefix*: $x \sqsubseteq y$ if $x = y$, or x is finite, and for some (finite or infinite) sequence z, $xz = y$. Example:

$$\langle 0 \rangle \sqsubseteq \langle 0, 0 \rangle \sqsubseteq \langle 0, 0, 0 \rangle \sqsubseteq \cdots \sqsubseteq 0^\omega$$

where 0^ω is the infinite sequence of 0's.

This example shows the ability of domain theory to model infinite computations as *limits* of processes of information increase, where at each stage in the process the information state is finite.

The Interval Domain Now suppose our computational scenario is that we are computing a real number in the unit interval $[0, 1]$. Clearly we can only compute to finite precision in finite time (and with finite resources), so we are forced to consider a scenario of approximation. The appropriate domain here is $\mathbb{I}[0, 1]$, consisting of all closed non-empty intervals $[a, b]$ where $0 \leq a \leq b \leq 1$. We read an interval $[a, b]$ as expressing our current state of information about the real $r \in [0, 1]$ we are computing, namely that $a \leq r \leq b$. The ordering is by reverse inclusion of intervals, or equivalently by

$$[a, b] \sqsubseteq [c, d] \iff a \leq c \ \wedge \ d \leq b.$$

This corresponds to *refinement* of our information state to a more accurate determination of the location of the ideal element r. Note that the case $[r, r]$ is allowed, for any $r \in [0, 1]$. In fact, this embeds the unit interval into the interval domain as the set of *maximal elements* of $\mathbb{I}[0, 1]$. Note that for any real number $r \in [0, 1]$, there is a process of information increase

$$[0, 1] \sqsubseteq [a_1, b_1] \sqsubseteq [a_2, b_2] \sqsubseteq \cdots$$

where $a_{n+1} = a_n$ and $b_{n+1} = b_n/2$ if r is in the left half-interval of $[a_n, b_n]$, and $a_{n+1} = a_n/2$ and $b_{n+1} = b_n$ if r is in the right half-interval. Clearly r is the supremum of the a_n and the infimum of the b_n. Thus every real can be computed as the limit of a process of information increase where at each finite stage of the process the interval has rational end-points, and hence represents a finite information state.

Partial Functions A somewhat more abstract example is provided by the set $\mathbf{Pfn}(X, Y)$ of partial functions from X to Y, ordered by inclusion. To see how this can be used in computational modelling, consider the recursive definition of the factorial function:

$$\mathsf{fact}(n) = n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1.$$

$$\mathsf{fact}(n) = \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times \mathsf{fact}(n-1).$$

We can understand this recursive definition as specifying a process of information increase over the domain $\mathbf{Pfn}(\mathbb{N}, \mathbb{N})$. Initially, we are at the zero information state (least element of the domain) $\varnothing$; we know nothing about which ordered pairs are in the graph of the function being defined recursively. Inspection of the base case of the recursion (where $n = 0$) allows us to deduce that the pair $(0, 1)$ is in the graph of the function. Once we know this, we can infer that in the case $n = 1$,

$$\mathsf{fact}(1) = 1 \times \mathsf{fact}(0) = 1 \times 1 = 1.$$

Thus the process of information increase proceeds as follows:

$$\varnothing \subseteq \{(0,1)\} \subseteq \{(0,1),(1,1)\} \subseteq \cdots$$

We can see inductively that the n'th term in this sequence will give the values of factorial on the arguments from 0 to $n-1$; and the *least upper bound* of this increasing sequence, given simply by its union, will be the factorial function.

2.1.2 Technical Issues

These examples serve to motivate a number of additional *axioms for domains*. There is in fact no unique axiom system for domains. We shall mention the most fundamental forms of such axioms.

Completeness As we have seen, an essential point of Domain Theory is to allow the description of infinite computations or computational objects as *limits* of processes of information increase. A corresponding property of completeness of domains is required, to ensure that a well-defined unique limit exists for every such process. Such limits are expressed as *least upper bounds* in order-theoretic terms. The idea is that for a process

$$d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \cdots$$

the limit should contain *all* the information produced at any stage of the process; and *only* the information produced by some stage of the process. The first point implies that the limit should be an upper bound; the second, that it should be the *least* upper bound.

Which class of increasing sets should be regarded as processes of information increase? The most basic class, which has figured in all our examples to date, is that of *increasing sequences*, or "ω-chains" in the usual technical parlance. The axiom requiring completeness for all such chains, which picks out the class of "ω-complete partial orders", is often used in Domain Theory. Sometimes completeness for a larger class of sets, the *directed sets*, is used. This reflects technical issues akin to the distinction in Topology between sequential completeness and completeness for nets or ultrafilters, and we shall not pursue this here.

Least Elements All our examples have had a least element: $\perp$ for flat domains, the empty stream for Σ^∞, the unit interval $[0,1]$ for $\mathbb{I}[0,1]$, and the empty set for $\mathbf{Pfn}(X,Y)$. This provides a zero information point, and hence a canonical starting point for processes of information increase. Mathematically, least elements are essential for the least fixed point theorem which we shall encounter shortly. There are schemes for Domain Theory in which domains (or "pre-domains") are not required to have least elements in general, but they always enter the theory at crucial points, sometimes through a general operation of adjoining a least element to a predomain to form a domain ("lifting").

Approximation The intuition developed through our examples for how general elements of the domain can be approximated by others, which may in particular be of finite character, is captured formally by requiring domains to be *algebraic* or *continuous*. We shall not develop these notions here, but will simply note for our examples:

- For flat domains such as $\mathbb{N}_\perp$, we can regard all elements as of finite character.
- Every stream in Σ^∞ can be realized as the least upper bound of an increasing sequence of finite streams.
- Every real in $[0,1]$, and more generally every interval in $\mathbb{I}[0,1]$, can be realized as the least upper bound of an increasing sequence of intervals with rational end-points.
- Every partial function in $\mathbf{Pfn}(X,Y)$, and in particular every total function from X to Y, where X and Y are countable, can be realized as the least upper bound of an increasing sequence of finite partial functions. (The case where X or Y are uncountable is a typical example where we would naturally resort to general directed sets rather than sequences.)

2.1.3 Conceptual Issues

Why Partial Orders? Having developed some examples and intuitions, we now re-examine the basic concept of domains as partial orders $(D, \sqsubseteq)$. If we think of the elements of D as information states, the way we articulate this structure is *qualitative* in character. That is, we don't ask *how much* information a given state contains, but rather a relational question: does one state convey *more* information than another? We read $d \sqsubseteq e$ as "e conveys at least as much information as d". If we consider the partial order axioms with this reading:

Reflexivity	$x \sqsubseteq x$					
Transitivity	$x \sqsubseteq y$	$\wedge$	$y \sqsubseteq z$	$\Longrightarrow$	$x \sqsubseteq z$	
Anti-Symmetry	$x \sqsubseteq y$	$\wedge$	$y \sqsubseteq x$	$\Longrightarrow$	$x = y.$	

then Reflexivity is clear; and Transitivity also very natural. Anti-Symmetry can be seen as embodying an important *Principle of Extensionality*: if two states convey the *same* information, they are regarded as *equal*.

States of What? We have been using the term "information state" to convey the intuition for what the elements of a domain represent. In fact, there is a certain creative ambiguity lurking here, between two interpretations of what these are states *of*:

- We may think of states of a *computational system* in itself, characterized in terms of the information they contain, as an "intrinsic" or "objective" property of the system, independently of any observer.

- We may implicitly introduce an *observer* of the system, and understand the information content of a system in terms of the observer's state of information about it.

In the first reading, we think of the partial elements of the domain in a more ontological way, as necessary extensions to our universe of discourse to represent the range of possible outputs of computational systems which may run for ever, and may fail to terminate or to produce information beyond some finite stage of the computation. In the second reading, we are thinking more epistemologically.

In fact, both readings are useful—and are widely used. It is very common to slip without explicit mention from one to the other—nor, for the technical purposes of the theory, does this seem to do any harm. Mathematically, this distinction can be related to the duality between *points* and *properties*, in the sense of Stone-type dualities: the duality between the points of a topological space, and its basic "observable properties"—the open sets. The particular feature of domains which allows this creative ambiguity between points and properties to be used so freely without incurring any significant conceptual confusions or overheads is that *basic points and basic properties (or observations) are essentially the same things.* We explain this in terms of an example. Consider a finite steam s in Σ^∞. On the one hand, this can be viewed as a point, *i.e.* as an element of the domain — which may be produced by some system which computes the elements of s in finite time, and then continues to run forever without producing any more output. On the other hand, we may view this finite stream s as a property: the property satisfied by any system with output stream t such that $s \sqsubseteq t$. It is a *finitely observable property*, since we can tell whether a system satisfies it after only a finite time spent observing the system. Whether we take Σ^∞ as the space of points X generated as limits of increasing sequences of finite streams, or as the "logic" (or open-set lattice) L of properties generated by the basic observations given by finite streams, we get the same thing: the topology of X will be L, and the space of points generated (as completely prime filters) over L will be X. This is Stone duality. An extensive development of Stone duality for Domain Theory has been given in [1].

In fact, we would argue that it is hard to avoid the epistemic stance entirely. For example, the plausibility of something as basic as the Anti-Symmetry axiom is much greater if we think in terms of an observer. Much of the conceptual power of Domain Theory comes from the idea that it articulates how we can approximate infinite ideal objects by processes which use only finite resources at each finite stage.

Static or Dynamic? Another subtle underlying issue which is not usually made explicit is that Domain Theory is *a static theory resting on dynamic intuitions.* Indeed, we have motivated the theory in terms of certain *processes of information increase.* Processes happen in time; thus time is present implicitly in Domain Theory. This underlying temporality can be developed more explicitly within the Domain Theoretic framework:

- One can add axioms to the basic ones for domains to pick out those domains which are *concrete*, in the sense that we can understand information increase in terms of a temporal flow of events. Now the ordering is not simply one of information content, but involves an idea of *causality*, so that some events *must* temporally precede others. This leads to notions of *event structures*, which have been applied to the study of concurrent processes. Very similar structures have shown up recently in Theoretical Physics, in the Causal Sets approach to quantum gravity.

- In some remarkable recent work, Domain Theoretic tools are used to characterize globally hyperbolic space-time manifolds in terms of their causal orderings.

However, it should be said that most of the applications of Domain Theory in denotational semantics are carried out at a much higher level of abstraction, where temporality appears only in the most residual form. This arises from the fact that computations or programs are modelled in the Domain-Theoretic denotational framework essentially as *functions* from inputs to outputs.

2.1.4 Continuous Functions

We now consider the appropriate notion of function between domains. Let D, E be ω-complete partial orders. A function $f : D \to E$ is *monotonic* if, for all $x, y \in D$:

$$x \sqsubseteq y \implies f(x) \sqsubseteq f(y).$$

It is *continuous* if it is monotonic, and for all ω-chains $(x_n)_{n\in\omega}$ in D:

$$f(\bigsqcup_{n\in\omega} x_n) = \bigsqcup_{n\in\omega} f(x_n).$$

Examples We consider a number of examples of functions $f : \Sigma^\infty \to \mathbb{B}_\perp$, where $\Sigma = \{0, 1\}$.

1. $f(x) = \mathsf{tt}$ if x contains a 1, $f(x) = \perp$ otherwise.
2. $f(x) = \mathsf{tt}$ if x contains a 1, $f(0^\infty) = \mathsf{ff}$, $f(x) = \perp$ otherwise.
3. $f(x) = \mathsf{tt}$ if x contains a 1, $f(x) = \mathsf{ff}$ otherwise.

Of these: (1) is continuous, (2) is monotonic but not continuous, and (3) is not monotonic.

As these examples indicate, the conceptual basis for monotonicity is that *the information in Domain Theory is positive; negative information is not regarded as stable observable information.* That is, if we are at some information state s, then for all we know, s may still increase to t, where $s \sqsubseteq t$. Thus we can only use information which is stable under every possible information increase in a computable process. This idea is very much akin to the use of partial orders in Kripke semantics for Intuitionistic Logic, in particular in connection with the interpretation of negation in that semantics. The continuity condition, on the other hand, reflects the fact that a computational process will only have access to a finite amount of information at each finite stage of the computation. If we are provided with an infinite input, then any information we produce as output at any finite stage can only depend on some finite observation we have made of the input.

Note by the way how this discussion is permeated with the epistemic stance. Continuous functions produce *points* as outputs on the basis of *observations* they make of their inputs. Thus the duality between these two points of view plays a basic rôle in our very *understanding* of continuous functions. (Mathematically, this duality appears in the guise of the compact-open topology for function spaces). This point can (and often is) glossed over in Domain Theory by virtue of the coincidence of finite points and finite properties which we have already discussed.

2.1.5 The Fixpoint Theorem

We now consider a simple but powerful and very widely applicable theorem, which is one of the main pillars of Domain Theory, since by virtue of this result it provides a general setting in which recursive definitions can be understood.

Theorem 2.1 (The Fixpoint Theorem) *Let D be an ω-cpo with a least element, and $f : D \to D$ a continuous function. Then f has a least fixed point $\mathsf{lfp}(f)$. Moreover, $\mathsf{lfp}(f)$ is defined explicitly by:*

$$\mathsf{lfp}(f) = \bigsqcup_{n\in\omega} f^n(\bot). \tag{1}$$

We give the proof, since it is elementary, and exhibits very nicely how the basic axiomatic structure of Domains is used.

Proof Note that $f^n(\bot)$ is defined inductively by:

$$f^0(\bot) = \bot, \qquad f^{k+1}(\bot) = f(f^k(\bot)).$$

We show firstly that this sequence is indeed an ω-chain . More precisely, we show for all $k \in \mathbb{N}$ that $f^k(\bot) \sqsubseteq f^{k+1}(\bot)$. For $k = 0$, this is just $\bot \sqsubseteq f(\bot)$. For the inductive case, assume that $f^k(\bot) \sqsubseteq f^{k+1}(\bot)$. Then by monotonicity of f, $f(f^k(\bot)) \sqsubseteq f(f^{k+1}(\bot))$, *i.e.* $f^{k+1}(\bot) \sqsubseteq f^{k+2}(\bot)$, as required.

Next we show that (1) does yield a fixpoint. This is a simple calculation using the continuity of f:

$$f(\bigsqcup_{n\in\omega} f^n(\bot)) = \bigsqcup_{n\in\omega} f^{n+1}(\bot) = \bigsqcup_{n\in\omega} f^n(\bot).$$

The last step uses the (easily verified) fact that removing the first element of an ω-chain does not change its least upper bound.

Finally, suppose that a is a fixpoint of f. Then we show by induction that, for all k, $f^k(\bot) \sqsubseteq a$. The basis is just $\bot \sqsubseteq a$. For the inductive step, assume $f^k(\bot) \sqsubseteq a$. Then by monotonicity of f,

$$f^{k+1}(\bot) = f(f^k(\bot)) \sqsubseteq f(a) = a.$$

Thus a is an upper bound of $(f^n(\bot) \mid n \in \omega)$, and hence $\bigsqcup_{n\in\omega} f^n(\bot) \sqsubseteq a$. □

Factorial revisited We now reconstrue the definition of the factorial function we considered previously as a function on *domains*:

$$F : \mathbf{Pfn}(\mathbb{N}, \mathbb{N}) \longrightarrow \mathbf{Pfn}(\mathbb{N}, \mathbb{N}),$$

defined by

$$F(f)(n) = \mathbf{if}\ n = 0\ \mathbf{then}\ 1\ \mathbf{else}\ n \times f(n-1).$$

We can check that F *is continuous*. Hence we can apply the fixpoint theorem to F, and conclude that it has a least fixpoint $\mathsf{lfp}(f)$, defined explicitly by (1). Now we can make the (explicit, non-circular) definition:

$$\mathsf{fact} = \mathsf{lfp}(F).$$

One can check that this definition yields exactly the expected definition of factorial. In fact, the increasing sequence constructed in forming the least fixpoint according to (1) is exactly the one we described concretely in our previous discussion of the factorial.

Thus in particular the processes of information increase we have been emphasizing are involved directly in the construction underpinning the Fixpoint Theorem.

2.1.6 Further Developments in Domain Theory

This is of course just the beginning of an extensive subject. We mention a few principal further features of Domain Theory:

Function Spaces A key point of the theory is that, given domains D and E, $[D \longrightarrow E]$, the set of continuous functions from D to E, will again be a domain, with the following *pointwise ordering*:

$$f \sqsubseteq g \iff \forall x \in D.\, f(x) \sqsubseteq_E g(x).$$

Moreover, operations such as function application and currying or lambda-abstraction are continuous. This means that we can form models of typed λ-calculi and higher-order computation within Domain Theory, which is of central importance for the denotational semantics of programming languages. Of course, such domains of higher-order functions are very "abstract"—they are in fact the prime examples of domains which are *not* concrete in the sense of [58]—and notions of temporality are left quite far behind. (There have attempts to capture more of these notions by varying the definition of the order on function spaces, but these have not been completely successful—and in some cases, provably cannot be).

Recursive Types Remarkably, the idea of the Fixpoint Theorem, and its use to give meaning to recursive definitions of elements of domains, can be lifted to the level of domains themselves, to give meaning to *recursive definitions of types*. This even extends to the free use of function spaces in recursive definitions of domains, leading to the construction of domains D whose continuous function spaces $[D \longrightarrow D]$ are isomorphic to D or to a subspace of D. This allows models of the type-free λ-calculus, and of various strongly impredicative type theories, to be given within Domain Theory.

Powerdomains There are also a number of *powerdomain* constructions $P(D)$, which build a domain of subsets of D. This allows various forms of non-deterministic and concurrent computation to be described. There is also a *probabilistic powerdomain* construction, which provides semantics for probabilistic computation.

Some suggestions for further reading on Domain Theory The Handbook article [18] is a comprehensive technical survey. The monograph [42] focusses on the connections to topology and lattice theory. The texts [77, 25] show how domain theory is used in the semantics of programming languages.

2.2 Dynamic Logic

Dynamic Logic originates at the confluence of two sources: modal logic and its Kripke semantics; and Hoare logic of programs.

Modal Logic Modal Logic adds to a standard background logic (say classical propositional calculus) the propositional operators $\Box$ and $\Diamond$, expressing ideas of "necessity" and "possibility". This was transformed from a philosophical curiosity to a vibrant and highly applicable branch of mathematical logic by the introduction of Kripke semantics. This is based on Kripke structures (W, R, V), where W is a set of worlds, $R \subseteq W \times W$ is an "accessibility relation", and $V : \mathbb{P} \to \mathcal{P}(W)$ is a valuation which for each propositional atom in $\mathbb{P}$ assigns the set of

worlds in which it is true. This valuation is then extended to one on formulas, with the key clauses:

$$w \models \Box\phi \;\equiv\; \forall w'.\, wRw' \;\Rightarrow\; w' \models \phi$$
$$w \models \Diamond\phi \;\equiv\; \exists w'.\, wRw' \;\wedge\; w' \models \phi.$$

The importance of the Kripke semantics is that it gives modal logic a clear mathematical purpose: it is a logical language for talking about such structures, which strikes a good balance between expressive power and tractability. Computer Science provides a wealth of situations where such structures arise naturally, and where there is a clear need for the verification of their logical properties. The dominant interpretation of Kripke structures in Computer Science replaces metaphysical talk of "possible worlds" by the more prosaic terminology of *states*. Here we think of states of a system, which are generally characterized by the information we have about them. In a Kripke structure, the *direct* information we have about a state is which atomic propositions are true in that state. However, while we seem again to be speaking about information states, as in our discussion of Domain Theory, there is an important difference. In Domain theory, (as in Kripke semantics for Intuitionistic Logic), information is in general *partial*, but also *persistent*. Information can only *increase* along a computation. We may never reach total information, but we will never lose what we had—just as we can never (in current Physics) change the past. (Indeed, the two are intimately related. In the implicit temporality of Domain Theory, the current information state summarizes all the information produced in the computation up till now; whatever happens in the future cannot change that). By contrast, Kripke structures for modal logics correspond to an imperative world. We may have perfect knowledge of the current state, but the dynamics of the system, as described by the accessibility relation, allow in general for arbitrary state change. A basic Computer Science model for this scenario is provided by taking the states to be memory states of a computer. At some instant of time we may have a complete snap-shot of the memory. But our repertoire of actions allow us to assign an arbitrary new value into any memory cell, so we can go from any given state to any other (possibly by a sequence of basic actions). In particular, the key feature of computer memory, the fact that we can destructively over-write the previous contents of a memory cell, (a feature which is not, apparently, available for our own memories), ensures that the past is not in general carried forward.

Hoare Logic Hoare Logic provides a compositional proof theory for reasoning about imperative programs. It is a two-sorted system. We have a syntax for *programs* P, and one for *formulas* ϕ, which are generally taken to be formulas of predicate calculus. Such formulas can be used to express properties of program states (*i.e.* memory state snap-shots as in our previous discussion, or more formally assignments of values to the variables appearing in the program), by a **variable pun** by which the individual variables used in formulas are identified with the program variables. The basic assertions of the system are taken to be *Hoare triples* $\phi\{P\}\psi$. Such a triple is said to be valid if, in any initial state satisfying the formula ϕ (the *precondition*), execution of the program P will, if it terminates, result in a final state satisfying the formula ψ (the *post-condition*).

The **variable pun** is put to use in the axiom for assignment statements:

$$\phi[e/x]\{x := e\}\phi$$

which says that ϕ is true after executing the assignment statement $x := e$ if ϕ with e substituted for x was true before.

The key rules of the system allow for compositional derivation of assertions about complex programs from assertions about their immediate sub-programs.

$$\frac{\phi\{P\}\psi \quad \psi\{Q\}\theta}{\phi\{P;Q\}\theta} \qquad \frac{\phi \wedge B\{P\}\psi \quad \phi \wedge \neg B\{Q\}\psi}{\phi\{\textbf{if } B \textbf{ then } P \textbf{ else } Q\}\psi} \qquad \frac{\phi \wedge B\{P\}\phi}{\phi\{\textbf{while } B \textbf{ do } P\}\phi \wedge \neg B}$$

Here $P;Q$ is the *sequential composition* which firstly performs P, then Q; **if** B **then** P **else** Q is the *conditional* which evaluates B in the current state; if it is **true** then P is performed, while if it is **false**, Q is performed. Finally, **while** B **do** P evaluates B; if it is **true**, then P is performed, after which the whole statement is repeated; while if it is **false**, the statement terminates immediately.

Dynamic Logic Dynamic Logic arises by combining salient features of these two systems. Note that we are reasoning about programs in terms of the *input-output relations on states* which they define. If the program is deterministic, this relation will actually be a partial function, but there is no need to insist on this. We can thus view each program P as defining a relation $R \subseteq S \times S$, where S is the set of states. Thus for each individual program, we obtain a Kripke structure (S, R, V), where V is the valuation which assigns truth conditions on states for some repertoire of state predicates. The key point of contact between the two systems is that validity of the Hoare triple $\phi\{P\}\psi$ corresponds exactly to the validity of the modal formula

$$\phi \rightarrow \Box\psi$$

in the Kripke structure (S, R, V), where by validity we mean that

$$s \models \phi \rightarrow \Box\psi$$

for every $s \in S$.

As a first extension, we can consider multiple programs, each defining an accessibility relation R. To keep track of which program we are talking about at any given point, we replace $\Box$ by $[R]$, so that the formula corresponding to the Hoare triple now reads as

$$\phi \rightarrow [R]\psi.$$

This is just *multi-modal logic*, with mutiple accessibility relations, each with its own modalities. Note that it is now completely meaningful to consider modal formulas which make assertions about programs which go well beyond Hoare triples, e.g.

$$[R]\langle S\rangle\phi \rightarrow \langle S\rangle[R]\phi.$$

However, at this point we lack the compositional analysis of programs offered by Hoare Logic.

The final step to (propositional) Dynamic Logic comes by considering a two-sorted system with a mutually recursive syntax. We have a set $\mathbb{P}$ of propositional atoms as before, and also a set Rel of basic relations. The syntax of formulas is given by

$$\phi \;::=\; p \in \mathbb{P} \mid \neg\phi \mid \phi \wedge \psi \mid [R]\phi$$

while the syntax of relations R is given by

$$R \;::=\; r \in \mathsf{Rel} \mid R;S \mid R \cup S \mid R^* \mid \phi?$$

We have not included the modal operator $\langle R \rangle$ as primitive syntax, since we can define

$$\langle R \rangle \phi \quad \equiv \quad \neg [R] \neg \phi.$$

In this syntax, any program is allowed to appear as a modal operator on formulas, while in addition to the usual regular operations of relational algebra (composition, union, and reflexive transitive closure), any formula is allowed to appeared as a program test (we may call this the **formula pun**). In general, this is too strong, and only a restricted class of tests should be allowed. Tests are interpreted as sub-identity relations—so $\phi?$ is the set of all (s, s) such that ϕ is true in s.

Note that the usual imperative program constructs can be recovered from these relational constructs. Sequential composition is provided directly, while

$$\textbf{if } b \textbf{ then } R \textbf{ else } S \;\equiv\; b; R \cup \neg b; S \qquad \textbf{while } b \textbf{ do } R \;\equiv\; (b; R)^*; \neg b.$$

The Hoare Logic axioms can now be derived from the following modal axioms:

$$\begin{array}{rcl} [R; S]\phi & \leftrightarrow & [R][S]\phi \\ [R \cup S]\phi & \leftrightarrow & [R]\phi \;\wedge\; [S]\phi \\ [\psi?]\phi & \leftrightarrow & \psi \rightarrow \phi \end{array}$$

and the rule

$$\frac{\phi \rightarrow [R]\phi}{\phi \rightarrow [R^*]\phi}.$$

2.2.1 Discussion

While Hoare Logic is specifically tailored to the needs of conventional imperative programming languages, Dynamic Logic is much more generic in style; and indeed, it has been applied in a range of contexts, including Natural Language and Quantum Logic. In the Chapter in this Handbook by Baltag and Moss, a version of Dynamic Logic is described in which the states are *information states of agents*, and the actions are *epistemic actions* by these agents, such as public announcements.

As a general formalism, though, Dynamic Logic offers only a limited analysis of information dynamics. Indeed, despite its name, it is not really very dynamic, as it is limited to speaking of the input-output behaviour of relations. This is confirmed by the simple translation it admits into first-order logic (augmented with fixpoints to account for the reflexive transitive closure operation on relations).

Suggestion for further reading The book [50] is a comprehensive technical reference, while [30] is a wide-ranging study. Applications to Natural Language appear in [49].

2.3 Process Algebra

2.3.1 Background

One of the major areas of activity in Theoretical Computer Science over the past three decades has been Concurrency Theory, and in particular Process Algebra. Whereas modelling sequential computation in terms of input-output functions or relations essentially uses off-the-shelf tools from Discrete Mathematics and Logic, albeit in novel combinations and with new technical twists, and even Domain Theory can be seen as an off-shoot of General Topology and

Lattice Theory, Concurrency Theory has really entered some new territory. In Concurrency Theory, the computational processes themselves become the objects of study; concurrent systems are executed for the behaviour they produce, rather than to compute some pre-specified function. What function does the Internet compute? In this setting, even such corner-stones of computation as Turing's analysis of computability do not provide all the answers. For all its conceptual depth, Turing's analysis of computability was still calibrated using familiar mathematical objects: which *functions* or *numbers* are computable? When we enter the vast range of possibilities for the behaviour of computational systems in general, the whole issue of what it means for a concurrent formalism to be *expressively complete* must be re-examined. There is in fact no generally accepted form of Church-Turing thesis for concurrency; and no widely accepted candidate for a universally expressive formalism. Instead, there are a huge range of concurrency formalisms, embodying a host of computational features.

Another question which ramifies alarmingly in this context is what is the right notion of *behavioural equivalence* of processes. Again, a large number of candidates have arisen. Experts use what seems most appropriate for their purpose; it is not even plausible that a single notion will gain general acceptance as "the right one".

In fact, a great deal of progress has been achieved, and the situation is much more positive than might appear from these remarks. There is a great diversity of particular formalisms and definitions in Concurrency Theory; but underpinning these are a much smaller number of underlying paradigms and technical tool-kits, which do provide effective intellectual instruments, both for fundamental research and applications.

Examples include:

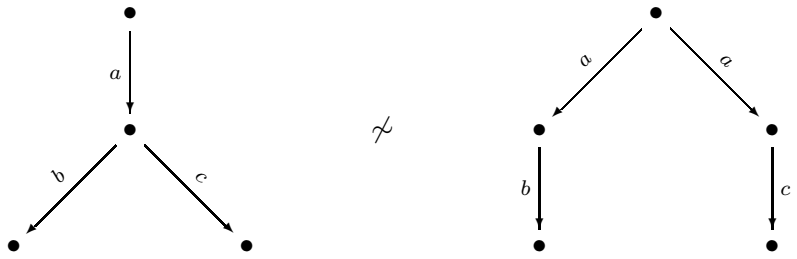
- labelled transition systems and bisimulation
- naming and scope restriction and extrusion
- the automata-theoretic paradigm for model-checking

These tool-kits are the real fruits of these theories. They may be compared to the traditional tool-kits of physics and engineering: Differential Equations, Laplace and Fourier Transforms, Numerical Linear Algebra, etc. They can be applied to a wide range of situations, going well beyond those originally envisaged, e.g. Security, Computational Biology, ...

2.3.2 Some Basics of Process Algebra

We now turn to a brief description of a few basic notions, in a subject on which there is a vast literature. We begin with the key semantic structure, namely *labelled transition systems*. A labelled transition system is a structure (S, Act, T), where S is a set of states, Act is a set of actions, and $T \subseteq S \times \mathsf{Act} \times S$ is the transition relation. We write $s \xrightarrow{a} t$ for $(s, a, t) \in T$. Note how close this is to the notion of Kripke structure we have already encountered. However, that notion is tuned to a *state-based* view of computation, in which we focus on assertions which are true in given states. The transition relation plays an indirect rôle, in controlling the behaviour of the modal operators. By contrast, the point of view in labelled transition systems is that states are not directly observable, and hence do not have properties directly attributable to them. Rather, it is the actions which are the basic observables, and we infer information about states indirectly from their potential for observable behaviour. Thus the point of view here is closer to automata theory. A key difference from classical automata theory, however, is that we look beyond the classical notion of behaviour in terms of the words or traces (sequences of

actions) accepted or generated by the system, and also encompass *branching behaviour*. The classical example which illustrates this is the following:



These systems have the same linear traces $\{ab, ac\}$. However, if we think of a scenario where we can perform experiments by pressing buttons labelled with the various actions, and observe if the experiments succeed, *i.e.* whether the system performs the corresponding action, then after observing an a in the first system, it is clear that whether we press the b button or the c button, we will succeed; whereas in the second system, one button will succeed and the other won't. A fundamental notion of process equivalence which enforces this distinction is *bismulation*. We define a *bisimulation* on a labelled transition system (S, Act, T) to be a relation $R \subseteq S \times S$ such that:

$$\begin{array}{l} sRt \ \wedge \ s \xrightarrow{a} s' \ \Rightarrow \ \exists t'.\, t \xrightarrow{a} t' \ \wedge \ s'Rt' \\ \wedge \\ sRt \ \wedge \ t \xrightarrow{a} t' \ \Rightarrow \ \exists s'.\, s \xrightarrow{a} s' \ \wedge \ s'Rt' \end{array}$$

We write $s \sim t$ if there is a bisimulation R such that sRt. We can see that indeed the root states of the two trees in the above example are not bisimilar.

We now turn to a suitable modal logic for labelled transition systems. The basic form for such a logic is Hennessy-Milner Logic. This has modal operators $[a]$, $\langle a \rangle$ for each action a. In general, this logic does not have (or require) any propositional atoms; just constants tt (true) and ff (false). The semantic clauses are as expected for a multi-modal logic, where we view the transition relation as an Act-indexed family of relations $\{T_a\}_{a \in \mathsf{Act}}$, where $T_a \subseteq S \times S$ is defined by

$$T_a = \{(s,t) \mid (s,a,t) \in T\}.$$

Thus we have the clauses

$$s \models [a]\phi \ \equiv \ \forall t.\, s \xrightarrow{a} t \ \Rightarrow \ t \models \phi$$

$$s \models \langle a \rangle \phi \ \equiv \ \exists t.\, s \xrightarrow{a} t \ \wedge \ t \models \phi.$$

The basic result here is that, under suitable hypotheses, two states in a labelled transition system are bisimilar if and only if they satisfy the same formulas in this modal logic. Thus in our example above, the first system satisfies the formula $\langle a \rangle (\langle b \rangle \mathsf{tt} \ \wedge \ \langle c \rangle \mathsf{tt})$, while the second does not.

We now turn, finally, to the *algebraic* aspect of process algebra. Just as we structured the programs in Dynamic Logic using relational algebra, so we seek an algebraic structure to generate a wide class of process behaviours. As we have already discussed, there is no one universally adopted set of process combinators, but we shall consider a standard set of operations, essentially a fragment of Milner's CCS. The syntax of process terms P is defined, assuming a set Act of actions, as follows:

$$P \ ::= \ a.P \ \ (a \in \mathsf{Act}) \mid P + Q \mid \ 0 \ \mid P \parallel Q.$$

Here $a.P$ is *action prefixing*; first do a, then behave as P. $P + Q$ is *non-deterministic choice* between P and A, while 0 is *inaction*; the process which can do nothing. Finally, $P \parallel Q$ is *parallel composition*, which we take here in a simple form, not involving any interaction between P and Q.

We formalize these intuitions as a labelled transition system in which the states are the process terms, while the transition relation is defined by structural induction on the syntax of terms—the Structural Operational Semantics paradigm.

The transition relation is specified as follows.

$$\frac{}{a.P \xrightarrow{a} P} \qquad \frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$$

$$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q} \qquad \frac{Q \xrightarrow{a} Q'}{P \parallel Q \xrightarrow{a} P \parallel Q'}$$

This labelled transition system gives rise to a notion of bimulation, which is an equivalence relation, and in fact a congruence for the process algebra. The corresponding equational theory for the algebra can be axiomatized as follows:

$$\begin{aligned} P + P &= P \\ P + 0 &= 0 \\ P + Q &= Q + P \\ P + (Q + R) &= (P + Q) + R \end{aligned}$$

together with the following equational scheme. If $P \equiv \sum_{i \in I} a_i.P_i$ and $Q \equiv \sum_{j \in J} b_j.Q_j$, then:

$$P \parallel Q = \sum_{i \in I} a_i.(P_i \parallel Q) + \sum_{j \in J} b_j.(P \parallel Q_j).$$

This is an infinite family of equations. In fact, the equational theory of bisimulation on process terms is not finitely axiomatizable; however, with the aid of an auxiliary operator (the "left merge"), a finite axiomatization can be achieved.

2.3.3 Discussion

Process Algebra can be used as a *vehicle* for discussions of information flow and information dynamics. It does not in itself offer a fully fledged theory of these notions.

Process Algebra is a qualitative theory of process behaviours. It is our first example of a *dynamic* theory, since it makes temporality and the flow of events explicit.

Suggestions for further reading Introductory textbooks include [54, 71, 27, 72]. The Handbook of Process Algebra [31] provides wide technical coverage of the field.

3 Does Information Increase in Computation?

Let us begin with an inane question:

Why do we compute?

The natural answer is: to gain information (which we did not previously have)! But how is this possible?

Problem 1: Isn't the output *implied* by the input?

Problem 2: Doesn't this contradict the second law of thermodynamics?

A logical form of Problem 1 This problem lies adjacent to another one at the roots of logic. If we extract logical consequences of axioms, then surely the answer was already there implicitly in the axioms; what has been added by the derivation? Since computation can itself, via the Curry-Howard isomorphism, be modelled as performing cut elimination on proofs, or *normalization* of terms, the same question can be asked of computation. A normal form which is presented as the result of a computation is logically *equal* to the term we started with:

$$M \longrightarrow^* N \quad \Longrightarrow \quad [\![M]\!] = [\![N]\!].$$

so what has been added by computing it?

The challenge here is to build a useful theory which provides convincing and helpful answers to these questions. We simply make some preliminary observations. Note that normal forms are in general *unmanagably big*. Useful output has two aspects:

- Making information explicit—*i.e.* extracting the normal form.
- Data reduction—getting rid of a lot of the information in the input.

(Note that it is *deletion of data* which creates thermodynamic cost in computation). Thus we can say that much (or all?) of the actual usefulness of computation lies in getting rid of the hay-stack, leaving only the needle.

Problem 2: Discussion While information is presumably conserved in the *total* system, there can be information flow between, and information increase in, *subsystems*. (A body can gain heat from its environment). Subsystems which can *observe* incoming information from their environment, and *act* to send information to their environment, have the capabilities of *agents*.

Moral: Agents and their interactions are intrinsic to the study of information flow and increase in computation. The classical theories of information *do not reflect this adequately*.

But before turning to study agents and interaction, we shall look at two major *extensional* theories of information.

3.1 Scott domain theory and Shannon information theory

Two important theories of information give contrasting views on the question of information increase[1]. Information theory *à la* Shannon is a quantitative theory which considers how *given* information can be transmitted losslessly on noisy channels. In this process, information may only be lost, never increased. Domain theory *à la* Scott is, as we have seen, a qualitative theory in which the key notion is the partial order $x \sqsubseteq y$, which can be interpreted as: "y has more information content than x". This theory is able to model a wide range of computational phenomena. To take a classical example, consider the interval bisection methods for finding

[1]Indeed, I was once challenged on this point by an eminent physicist (now knighted), who demanded to know how I could speak of information increasing in computation when Shannon Information theory tells us that it cannot! My failure to answer this point very convincingly at the time led me to continue to ponder the issue, and eventually gave rise to this section of the Chapter.

the root of a function. We start with an interval in which the root is known to lie. At each step, we halve the length of the interval being considered. This represents an increase in our information about the location of the root, in an entirely natural sense. In the limit, this nested sequence of intervals contains a single point, the root – we have perfect information about the solution.

More generally, in Domain theory recursion (and thereby control mechanisms such as iteration) is modelled as the least fixed point of a monotonic and (order-)continuous function:

$$\bot \sqsubseteq f\bot \sqsubseteq f^2\bot \sqsubseteq \cdots \bigsqcup_k f^k\bot$$

since $f(\bigsqcup_k f^k\bot) = \bigsqcup_k f^{k+1}\bot = \bigsqcup_k f^k\bot$. Thus a basic tenet of this theory is that information *does* increase during computation, and in particular this is how the meaning of recursive definitions is given.

It is intriguing to consider that the different viewpoints taken by Information theory and Domain theory may have been influenced by their technological roots. Information theory was summoned forth by the needs of the telecommunications industry, whose task is to transmit the customer's data with the highest possible fidelity. Domain theory arose as a mathematical theory of *computation*; the task of computation is to "add value" to the customer's data[2].

How can these views be reconciled? Information theory is a thermodynamic theory; Shannon information is negative entropy. From this viewpoint, the *total information* of a system can only decrease; however, information can flow from one subsystem into another, just as a body can be warmed by transferring heat from its environment.

The Domain theory view, we suggest, arises most naturally if we think of adding an observer to a system. It is the observer's information which increases during a computation. This reading has a precise mathematical analogue in the view of Domain theory as a "logic of observable properties" [1]. Information increase is always, necessarily it seems, *relative to a sub-system*. Moreover, this is a subsystem which can *observe* its environment, and which may, symmetrically, act itself to direct information to the environment. It is then a small step to viewing such sub-systems as *agents*.

It is worth adding that Shannon Information Theory also relies on such a view for its guiding intuitions. One of the standard ways of motivating Shannon information is in terms of "twenty questions": the number of bits of information in a message is how many yes/no questions we would need to have answered in order to know the contents of the message. Again, implicit here is some interaction between agents. And of course, the purpose of communication itself is to transfer information from one agent to another.

We need a quantitative theory to deal with essentially quantitative issues such as complexity, information content, rate of information flow etc. However, the *weakness* of a purely quantitative theory is that numbers always compare, so that some more subtle issues are obscured, such as, crucially, distinguishing different *directions* of information increase. Beyond this, by combining quantitative and qualitative aspects, e.g. in formulating conditions on "informatic processes", a unified theory can be more than the sum of its parts.

[2]Which of course begs our question of how this can be possible thermodynamically. The answer is, again, that it is the *customer's data* which is having value added to it; just as buying energy from the National Grid does not violate the Second Law.

3.2 Domains with measurements: connecting the quantitative and qualitative views

An important step towards unifying the qualitative and the quantitative points of view was taken in Keye Martin's Ph.D. thesis [65] and subsequent publications [66, 67, 69]. Martin introduced a simple notion of *measurement* on domains. In its most concrete form, a measurement assigns real numbers to domain elements, which can be said to measure the degree of undefinedness or uncertainty of the element. Thus the maximal elements, which can be regarded as having perfect information, will have measurement 0. The axioms for measurements, while quite simple and intuitive, tie the quantitative notion in with the qualitative domain structure in a very rich way. Just to mention some of the highlights:

- There is a rich theory of fixpoints which applies to increasing, *but not necessarily monotonic*, functions on domains. This is already a remarkable departure from 'classical' domain theory, in which monotonicity is always assumed. However, Martin shows that there are compelling natural examples, such as interval bisection, which require this broader framework. Not only are there existence and uniqueness theorems for fixpoints in this frameworks, but also novel induction principles.

- As the previous point suggests, there is a move away from the use of domain theory to model purely extensional aspects of computation, and towards using it to capture important features of *computational processes*. This leads to a notion of 'informatic derivative' which can be used to gain information about the *rate of convergence* of a computational process.

- A notable aspect of this development is the unified basis on which it puts the study of both discrete and continuous (e.g. real-number) computation.

It is also important that there are many natural examples of measurement covering most of the domains standardly arising as data-types for computation, including the domain of intervals (for real-number computation); finite lists and other standard finite data-structures; streams; partial functions on the natural numbers; and both non-deterministic and probabilistic powerdomains.

However, the example which has really revealed the possibilities of this framework has only appeared recently, and is a major development in its own right.

3.3 Combining Scott Information and Shannon Information

Recently, Bob Coecke and Keye Martin have produced a very interesting construction which can be seen as a first step towards a unification of these two theories of information [34]. The problem which they attacked can be formulated as follows. Consider the set of probability distributions on a finite set. For an n-element set, these are the "classical n-states" of Physics:

$$\Delta^n := \{x \in [0,1]^n : \sum_{i=1}^{n} x_i = 1\}.$$

This is the setting in which Shannon entropy, the fundamental quantitative notion in classical Information Theory, is defined. It assigns a number, the 'expected information', to each classical state. The question is: can we place a *partial order* on Δ^n such that:

1. This partial order forms a domain.

2. Shannon entropy is a measurement with respect to this domain.

3. The order extends to quantum states (density operators).

These are highly non-trivial requirements to satisfy. Note that the set of probability distributions on a 3-element set, seen as a subset of Euclidean space, form a (solid) triangle, and in general those on a n-element set form an n-simplex. The distribution corresponding to maximum uncertainty is the uniform distribution, with each point assigned probability $1/n$ — geometrically, the barycenter of the simplex; while the maximal elements, corresponding to perfect information, are the *pure states* assigning probability 1 to one element, and 0 to all others — geometrically, the vertices of the simplex. This geometrical aspect brings a rich mathematical structure to this example which seems different to anything previously encountered in Domain theory.

Note also the contrast with previous work on the probabilistic powerdomain [57]. Classical probability distributions are *maximal elements* in the probabilistic powerdomain; non-standard elements (valuations) are introduced which provide approximations to measures, but the order restricted to the measures themselves is discrete. By contrast, we are seeking a rich informatic structure on the standard objects of probability (distributions) and quantum mechanics (density operators) *themselves*, without introducing any non-standard elements. It is by no means *a priori* obvious that this can be done at all; once we see that it can, many new possibilities will unfold.

A classical state $x \in \Delta^n$ is *pure* when $x_i = 1$ for some $i \in \{1, \ldots, n\}$; we denote such a state by e_i. Pure states $\{e_i\}_i$ are the actual states a system can be in, while general mixed states x and y are epistemic entities.

If we know x and by some means determine that outcome i is not possible, our knowledge improves to

$$p_i(x) = \frac{1}{1 - x_i}(x_1, \ldots, \hat{x_i}, \ldots, x_{n+1}) \in \Delta^n,$$

where $p_i(x)$ is obtained by first removing x_i from x and then renormalizing. The partial mappings which result,

$$p_i : \Delta^{n+1} \rightharpoonup \Delta^n$$

with $\mathrm{dom}(p_i) = \Delta^{n+1} \setminus \{e_i\}$, are called the *Bayesian projections* and lead one directly to the following relation on classical states.

Definition 3.1 For $x, y \in \Delta^{n+1}$,

$$x \sqsubseteq y \equiv (\forall i)(x, y \in \mathrm{dom}(p_i) \Rightarrow p_i(x) \sqsubseteq p_i(y)). \tag{2}$$

For $x, y \in \Delta^2$,

$$x \sqsubseteq y \equiv (y_1 \leq x_1 \leq 1/2) \text{ or } (1/2 \leq x_1 \leq y_1)\,. \tag{3}$$

The relation $\sqsubseteq$ on Δ^n is called the *Bayesian order.*

See [34] for motivation and results showing that the order on Δ^2 is uniquely determined under minimal assumptions.

The key result is:

Theorem 3.2 $(\Delta^n, \sqsubseteq)$ *is a domain with maximal elements*

$$\max(\Delta^n) = \{e_i : 1 \leq i \leq n\}$$

and least element $\perp := (1/n, \ldots, 1/n)$. *Moreover, Shannon entropy*

$$\mu x = -\sum_{i=1}^{n} x_i \log x_i$$

is a measurement of type $\Delta^n \to [0, \infty)^*$.

The Bayesian order can also be described in a more direct manner, the *symmetric characterization.* Let $S(n)$ denote the group of permutations on $\{1, \ldots, n\}$, and

$$\Lambda^n := \{x \in \Delta^n : (\forall i < n)\, x_i \geq x_{i+1}\}$$

the collection of *monotone* classical states.

Theorem 3.3 *For* $x, y \in \Delta^n$, *we have* $x \sqsubseteq y$ *iff there is a permutation* $\sigma \in S(n)$ *such that* $x \cdot \sigma, y \cdot \sigma \in \Lambda^n$ *and*

$$(x \cdot \sigma)_i (y \cdot \sigma)_{i+1} \leq (x \cdot \sigma)_{i+1} (y \cdot \sigma)_i$$

for all i *with* $1 \leq i < n$.

Thus, the Bayesian order is order isomorphic to $n!$ many copies of Λ^n identified along their common boundaries. This fact, together with the pictures of $\uparrow x$ at representative states x in Figure 1, will give the reader a good feel for the geometric nature of the Bayesian order.

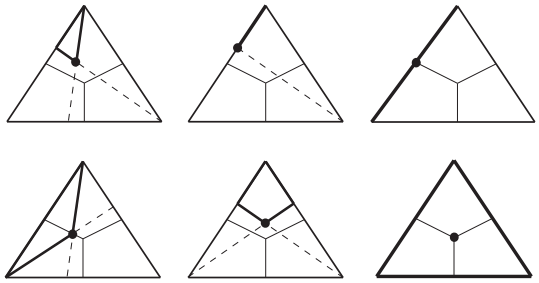


Figure 1: Pictures of $\uparrow x$ for $x \in \Delta^3$.

3.4 The Quantum Case

The real force of the construction for classical states becomes apparent in the further development in [34], to show that it can be lifted to analogous constructions for *quantum states.* Here, rather than probability distributions on finite sets, one is looking at *mixed states on finite-dimensional Hilbert spaces.* Let $\mathcal{H}^n$ denote an n-dimensional complex Hilbert space. A *quantum state* is a density operator $\rho : \mathcal{H}^n \to \mathcal{H}^n$, i.e., a self-adjoint, positive, linear operator with $\mathrm{tr}(\rho) = 1$. The quantum states on $\mathcal{H}^n$ are denoted Ω^n. A quantum state ρ on $\mathcal{H}^n$ is *pure* if

$$\mathrm{spec}(\rho) \subseteq \{0, 1\}.$$

The set of pure states is denoted Σ^n. They are in bijective correspondence with the one dimensional subspaces of $\mathcal{H}^n$. Classical states are distributions on the set of pure states $\max(\Delta^n)$. By Gleason's theorem [48], an analogous result holds for quantum states: Density operators encode distributions on Σ^n.

If our knowledge about the state of a system is represented by density operator ρ, then quantum mechanics predicts the probability that a measurement of observable e yields the value $\lambda \in \mathrm{spec}(e)$. It is

$$\mathrm{pr}(\rho \to e_\lambda) := \mathrm{tr}(p_e^\lambda \cdot \rho),$$

where p_e^λ is the projection corresponding to eigenvalue λ and e_λ is its associated eigenspace in the *spectral representation* of e.

Let e be an observable on $\mathcal{H}^n$ with $\mathrm{spec}(e) = \{1, \ldots, n\}$. For a quantum state ρ on Ω^n,

$$\mathrm{spec}(\rho|e) := (\mathrm{pr}(\rho \to e_1), \ldots, \mathrm{pr}(\rho \to e_n)) \in \Delta^n.$$

So what does it mean to say that we have more information about the system when we have $\sigma \in \Omega^n$ than when we have $\rho \in \Omega^n$? It means that there is an experiment e which (a) serves as a physical realization of the knowledge each state imparts to us, and (b) that we have a better chance of predicting the result of e from state σ than we do from state ρ. Formally, (a) means that $\mathrm{spec}(\rho) = \mathrm{Im}(\mathrm{spec}(\rho|e))$ and $\mathrm{spec}(\sigma) = \mathrm{Im}(\mathrm{spec}(\sigma|e))$, which is equivalent to requiring $[\rho, e] = 0$ and $[\sigma, e] = 0$, where $[a, b] = ab - ba$ is the commutator of operators.

Definition 3.4 Let $n \geq 2$. For quantum states $\rho, \sigma \in \Omega^n$, we have $\rho \sqsubseteq \sigma$ iff there is an observable $e : \mathcal{H}^n \to \mathcal{H}^n$ such that $[\rho, e] = [\sigma, e] = 0$ and $\mathrm{spec}(\rho|e) \sqsubseteq \mathrm{spec}(\sigma|e)$ in Δ^n.

Theorem 3.5 *$(\Omega^n, \sqsubseteq)$ is a domain with maximal elements*

$$\max(\Omega^n) = \Sigma^n$$

and least element $\perp = I/n$, where I is the identity matrix. Moreover, von Neumann entropy

$$\sigma\rho = -\mathrm{tr}(\rho \log \rho)$$

is a measurement of type $\Omega^n \to [0, \infty)^*$.

This order can be characterized in a similar fashion to the Bayesian order Δ^n, in terms of symmetries and projections. In its symmetric formulation, *unitary operators* on $\mathcal{H}^n$ take the place of permutations on $\{1, \ldots, n\}$, while the projective formulation of $(\Omega^n, \sqsubseteq)$ shows that each classical projection $p_i : \Delta^{n+1} \rightharpoonup \Delta^n$ is actually the restriction of a special quantum projection $\Omega^{n+1} \rightharpoonup \Omega^n$.

3.5 The Logics of Birkhoff and von Neumann

Quantum Logic in the sense of Birkhoff and von Neumann [32] consists of the propositions one can make about a physical system. Each proposition takes the form "The value of observable e is contained in $E \subseteq \mathrm{spec}(e)$." For classical systems, the logic is $\mathcal{P}\{1, \ldots, n\}$, while for quantum systems it is $\mathbb{L}^n$, the lattice of (closed) subspaces of $\mathcal{H}^n$. In each case, implication of propositions is captured by inclusion, and a fundamental distinction between classical and quantum — that there are pairs of quantum observables whose exact values cannot be simultaneously measured at a single moment in time — finds lattice theoretic expression: $\mathcal{P}\{1, \ldots, n\}$ is distributive; $\mathbb{L}^n$ is not.

The classical and quantum logics can be *derived* from the Bayesian and spectral orders using the *same* order theoretic construction.

Definition 3.6 An element x of a dcpo D is *irreducible* when

$$\bigwedge(\uparrow x \cap \max(D)) = x$$

The set of irreducible elements in D is written $\mathrm{Ir}(D)$.

The order dual of a poset $(D, \sqsubseteq_D)$ is written D^*; its order is $x \sqsubseteq y \Leftrightarrow y \sqsubseteq_D x$.
The following result is proved in [33].

Theorem 3.7 *For $n \geq 2$, the classical lattices arise as*

$$\mathrm{Ir}(\Delta^n)^* \simeq \mathcal{P}\{1, \ldots, n\} \setminus \{\emptyset\},$$

and the quantum lattices arise as

$$\mathrm{Ir}(\Omega^n)^* \simeq \mathbb{L}^n \setminus \{0\}.$$

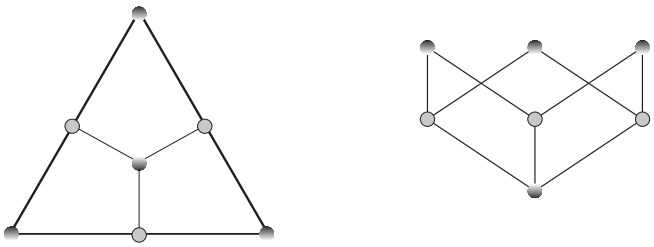


Figure 2: The irreducibles of Δ^3 with the corresponding Hasse diagram.

3.6 Discussion

The foregoing development has been quite technical, but the underlying programme which these ideas illustrate has a clear conceptual interest. The broad agenda of developing a unified quantitative/qualitiative theory of information, applicable to a wide range of situations in logic and computation, is highly attractive, and likely to lead to new perspectives on information in general.

Our discussion thus far has largely been couched in terms of *static* theories, although we have already hinted at the importance of agents and explicit dynamics. We now turn to *interactive models* of logic and computation.

4 Games, Logical Equilibria and Conservation of Information Flow

In this Section and the next, we shall discuss some dynamical theories of computation which are explicitly based on *interaction* between agents, and which expose a structure of information flow which is both *geometrical* and *logical* in character. These theories, which go under the names of *Game Semantics* and *Geometry of Interaction*, have played a considerable rôle in recent work on the semantics both of programming languages, and of logical proofs.

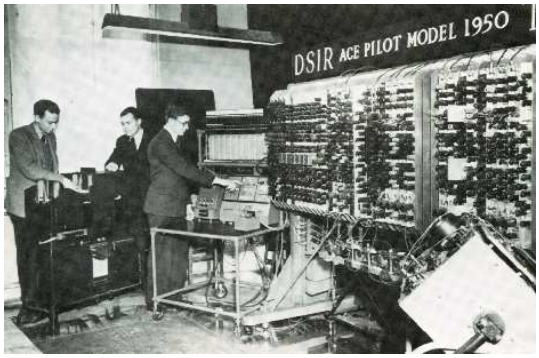
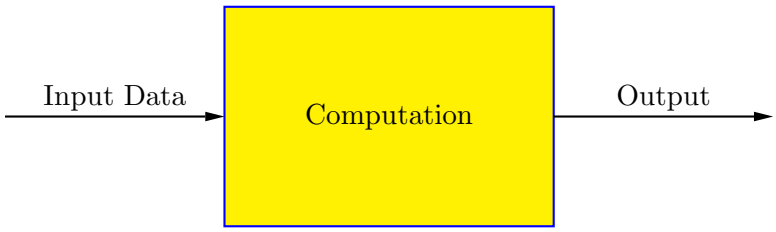


Figure 3: Computing "in the isolation ward".

4.1 Changing Views of Computation

To set the scene, we begin by recalling how perspectives on computation have changed since the first computers appeared. The early practice of computing can be pictured as in Figure 3. This is the era of stand-alone machines and programs: computers are served by an elite priesthood, and have only a narrow input-output interface with the rest of the world.

First-generation models of computation Given this limited vision of computing, there is a very natural abstraction of computation, in which programs are seen as computing *functions* or *relations* from inputs to outputs.[3]



These models live on the existing intellectual inheritance from discrete mathematics and logic. *Time* and *processes* lurk in the background, but are largely suppressed.

Computation in the Age of the Internet As we know, the technology has changed dramatically. Even a conventional Distributed Systems picture, as illustrated in Figure 4, which has been common-place for the last 20 years, tells a very different story. We have witnessed the progression

multitasking → distributed systems → Internet → "mobile" and "global" computing

Key features of this unfolding new computational universe include: *agents interacting* with each other, and *information flowing* around the system.

The insufficiency of the first-generation models of computation for this new computational environment is evident:

What does the Internet compute?

[3]This is the exactly the point of view on which, as we have seen, Dynamic Logic is based.

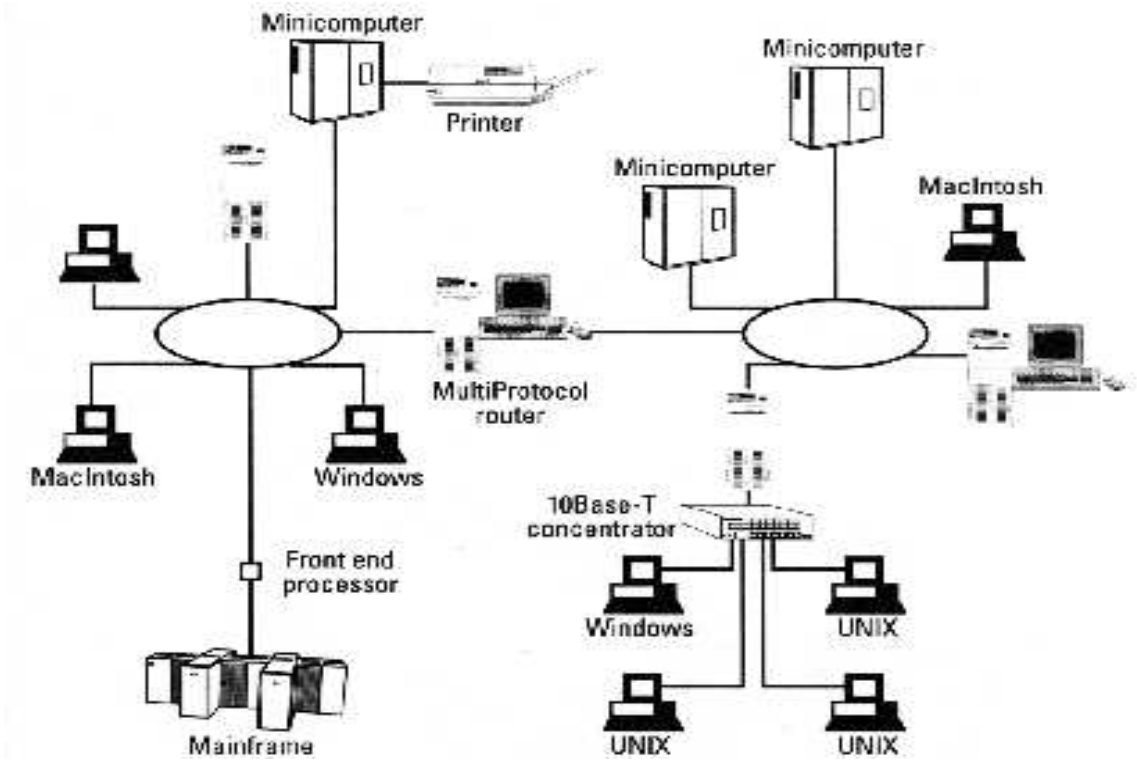


Figure 4: Distributed Computing

Surely not a mathematical function ...

The old concepts fail to match the modern world of computing and its concerns:

Robustness in the presence of failures.

Atomicity of transactions.

Security of information flows.

Quality of user interface.

Quantitative aspects.

Processes vs. Products We see a shift in emphasis and importance between *How* we compute *vs. What* we compute. Processes were in the background, but now come to the fore: the "how" *becomes* the new "what".

This leads ineluctably to the need for **Second-generation models** of computation, and in particular *Process Models* such as Petri nets, Process Algebra, etc. Whereas 1st-generation models lived off the intellectual inheritance from mathematics and logic, there is no adequate pre-existing theory of *processes* or *agents*, *interaction*, and *information flow*, as we see by considering the following questions (some of which have already been mentioned in Section 2):

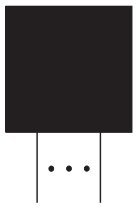
- *What* is computed?
- What *is* a process?

- What are the analogues to Turing-completeness, universality?

There are indeed a plethora of models, but no definitive conceptual analysis, comparable to Turing's analysis of computation in its "classical" sense. —Not least, perhaps, because it is indeed *a harder problem*!

4.2 Some New Perspectives

Instead of isolated systems, with rudimentary interactions with their environment, the standard unit of description or design becomes a *process* or *agent*, the essence of whose behaviour is *how it interacts* with its environment.

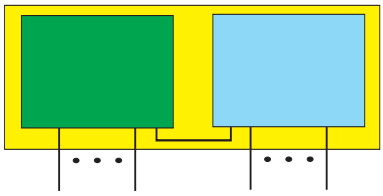


Who is the System? Who is the Environment? This depends on point of view. We may designate some agent or group of agents as the System currently under consideration, with everything else as the Environment; but it is always possible to contemplate a rôle interchange, in which the Environment becomes the System and vice versa. (This is, of course, one of the great devices, and imaginative functions, of creative literature). This *symmetry* between System and Environment carries a first clue that there is some structure here; it will lead us to a key *duality*, and a deep connection to logic.

4.3 Interaction

Complex behaviour arises as the global effect of a *system* of *interacting agents* (or processes).

The key building block is the agent. The key operation is *interaction* – plugging agents together so that they interact with each other



This conceptual model works at all "scales" :

- Macro-scale: processes in operating systems, software agents on the Internet, transactions.
- Micro-scale: how programs are implemented (subroutine call-return protocols, register transfer) all the way down into hardware.

It is applicable both to *design* (synthesis) and to *description* (analysis); to *artificial* and to *natural* information-processing systems.

There are of course large issues lurking here, e.g. in the realm of "Complex Systems": *emergent behaviour* and even *intelligence*. Is is helpful, or even feasible, to understand this complexity *compositionally*? We need new conceptual tools, new theories, to help us analyze and synthesize these systems, to help us to *understand* and to *build*.

4.4 Towards a "Logic of Interaction"

Specifying and reasoning about the behaviour of computer programs takes us into the realm of logic. For the first-generation models, logic could be taken "as it was"—static and timeless. For our second-generation models, getting an adequate account—a genuine "logic of interaction"—may require a fundamental reconceptualization of logic itself. This radical revision of our view of logic is happening anyway—prompted partly by the applications, and partly by ideas arising within logic.

4.4.1 The Static Conception of Logic

We provide an unfair caricature of the standard logical idea of tautology to make our point. The usual "static" notion of tautology is as "a statement which is vacuously true because it is compatible with all states of affairs".

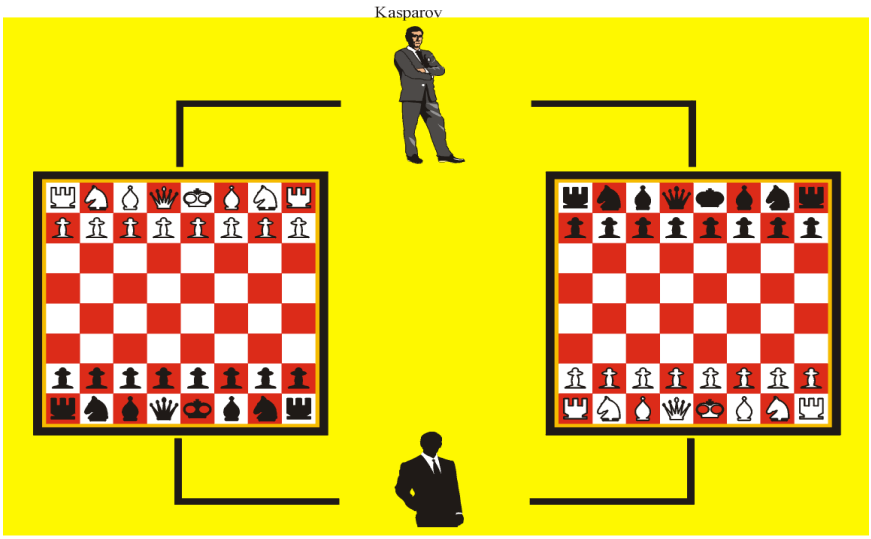
$$A \vee \neg A$$

"It is raining *or* it is not raining"—truth-functional semantics. This is illustrated (subversively) in Figure 5. But what could a *dynamic notion of tautology* look like?

Figure 5: Tertium non datur?

4.4.2 The Copy-Cat Strategy

We begin with a little fable:



The copy-cat strategy

How to beat an International Chess Grandmaster by the power of pure logic.

Since we are relying on logic, rather than on any talent at Chess, we proceed as follows. We arrange to play two games of Chess with the grandmaster, say Gary Kasparov, once as White and once as Black. Moreover, we so arrange matters that we start with the game in which we play as Black. Kasparov makes his opening move; we respond by playing the *same* move in the *other* game—this makes sense, since we are playing as White there. Now Kasparov responds (as Black) to our move in that game; and we copy that response back in the first game. We simply proceed in this fashion, copying the moves that our opponent makes in one board to the other board. The net effect is that *we play the same game twice—once as White, and once as Black*. (We have essentially made Kasparov play against himself). Thus, whoever wins that game, we can claim a win in one of our games against Kasparov! (Even if the game results in a stale-mate, we have done as well as Kasparov over the two games—surely still a good result!)[4]

Of course, this idea has nothing particularly to do with Chess. It can be applied to any two-person game of a very general form. We shall continue to use Chess-boards to illustrate our discussion, but this underlying generality should be kept in mind.

What are the salient features which can be extracted from this example?

A dynamic tautology There is a sense (which will shortly be made more precise) in which the copy-cat strategy can be seen as a *dynamic version* of the tautology $A \vee \neg A$. Note, indeed, that an essential condition for being able to play the copy-cat is that the rôles of the two players are inter-changed on one board as compared to the other. Note also the disjunctive quality of the argument that we must win in one or other of the two games. But the copy-cat strategy is a *dynamic process*: a two-way channel which maintains the correlation between the plays in the two games.

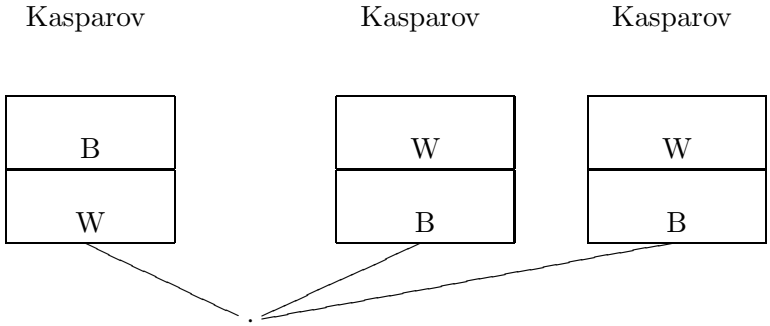
[4]Our fable is actually recorded as having happened at least once in the chronicles of Chess. Two players conspired to play this copy-cat strategy against Alekhine in the 1920's. Alekhine realized what was happening, and made a tempting offer of a sacrifice to one of his opponents. That opponent was not able to resist such a coup against the great Alekhine, and departed from the copy-cat strategy to swallow the bait. Then the symmetry was broken, and Alekhine was able to win easily in both games. Thus we are reminded of the familiar truth, that logic rarely prevails over psychology in "real life".

Conservation of information flow The copy-cat strategy does not *create* any information; it reacts to the environment in such a way that information is conserved. It ensures that exactly the same information flows out to the environment as flows in from it. Thus one gets a sense of logic appearing in the form of *conservation laws for information dynamics*.

The power of copying Another theme which appears here, and which we will see more of later, concerns the surprising power of simple processes of copying information from one place to another. Indeed, as we shall eventually see, such processes are *computationally universal*.

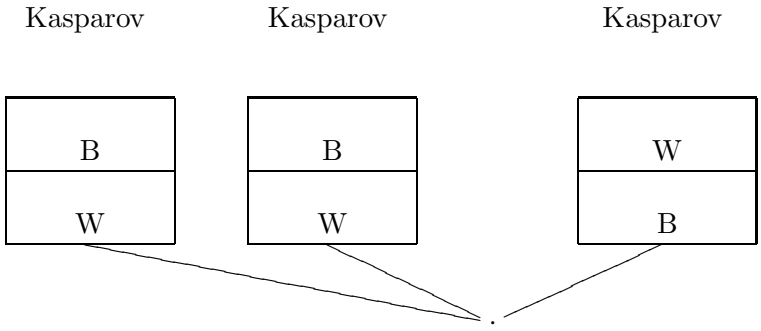
The geometry of information flow From a dynamical point of view, the copy-cat strategy realizes a channel between the two game boards, by performing the *actions* of copying moves. But there is also some implicit *geometry* here. Indeed, the very idea of two boards laid out side by side appeals to some basic underlying spatial structure. In these terms, the copy-cat channel can also be understood geometrically, as creating a graphical link between these two spatial locations.These two points of view are complementary, and link the logical perspective to powerful ideas arising in modern geometry and mathematical physics.

To provide further evidence that the copy-cat strategy embodies more substantial ideas than might at first be apparent, we consider varying the scenario. Consider now the case where we play against Kasparov on *three boards*; one as Black, two as White.



Does the Copy-Cat strategy still work here? In fact, we can easily see that it does *not*. Suppose Kasparov makes an opening move m_1 in the left-hand board where he plays as White; we copy it to the board where we play as White; he responds with m_2; and we copy m_2 back to the board where Kasparov opened. So far, all has proceeded as in our original scenario. But now Kasparov has the option of playing a *different* opening move, m_3 say, in the rightmost board. We have no idea how to respond to this move; nor can we copy it anywhere, since the board where we play as White is already "in use". This shows that these simple ideas already lead us naturally to the setting of a *resource-sensitive* logic, in which in particular the Contraction Rule, which can be expressed as $A \to A \wedge A$ (or equivalently as $\neg A \vee (A \wedge A)$) cannot be assumed to be valid.

What about the other obvious variation, where we play on two boards as White, and one as Black?



It seems that the copy-cat strategy *does* still work here, since we can simply ignore one of the boards where we play as White. However, a geometrical property of the original copy-cat strategy has been lost, namely a *connectedness* property, that information flows to every part of the system. This at least calls the corresponding logical principle of Weakening, which can be expressed as $A \wedge A \rightarrow A$, (or equivalently as $\neg A \vee \neg A \vee A$) into question.

We see from these remarks that we are close to the realm of Linear Logic and its variants; and, mathematically, to the world of monoidal (rather than cartesian) categories.

4.4.3 Game Semantics

These ideas find formal expression in *Game Semantics*. Games play the role of:

- Interface types for computation modules
- Propositions with dynamic content.

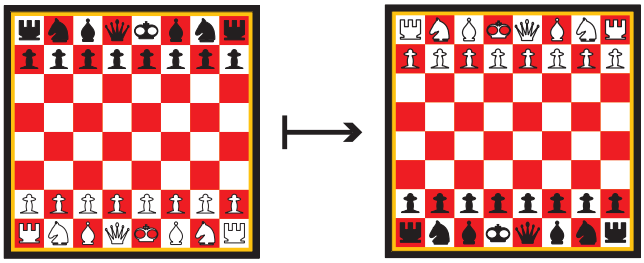
In particular, 2-person games capture the duality of:

- Player *vs.* Opponent
- System *vs.* Environment.

Agents are strategies In this setting, we model our agents or processes as *strategies* for playing the game. These strategies *interact* by playing against each other. We obtain a notion of correctness which is *logical* in character in terms of the idea of *winning* strategy—one which is guaranteed to reach a successful outcome however the environment behaves. This in a sense replaces (or better, *refines*) the logical notion of "truth": winning strategies are our dynamic version of tautologies (more accurately, of *proofs*).

Building complex systems by combining games We shall now see how games can be combined to produce more complex behaviours while retaining control over the interface. This provides a basis for the *compositional* understanding of our systems of interacting agents—understanding the behaviour of a complex system in terms of the behaviour of its parts. This is crucial for both analysis and synthesis, *i.e.* for both description and design. These operations for building games can be seen as (dynamic forms of) "type constructors" or "logical connectives". (The underlying logic here will in fact be Linear Logic).

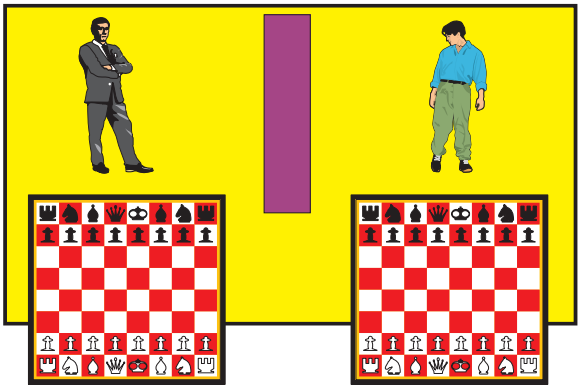
Duality—"Linear Negation" $A^\perp$ — interchange rôles of Player and Opponent (reflecting the symmetry of interaction).



Note that, with this interpretation, negation is involutive:

$$A^{\perp\perp} = A.$$

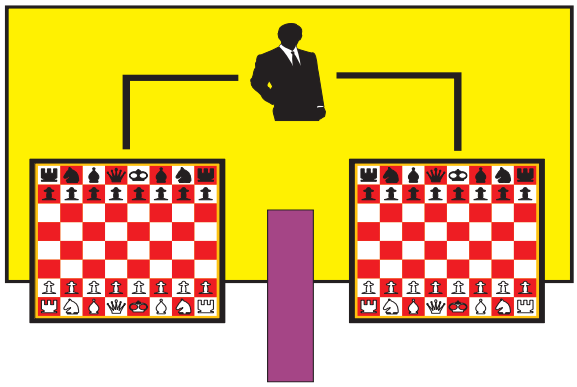
Tensor — "Linear conjunction"



$A \otimes B$

The idea here is that we combine the two game boards into one system, *without any information flow between the two sub-systems.* (This is the significance of the "wall" separating our two players, who we shall refer to as Gary (Kasparov) and Nigel (Short)). This connective has a conjunctive quality, since we must independently be able to play (and to win) in each conjunct. Note however, that there is no constraint on information flow for the environment, as it plays against this compound system.

Par — "Linear disjunction"



$A \parr B$

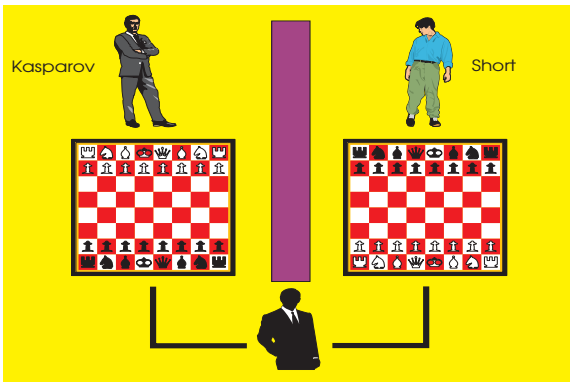
In this case, we have two boards, but one player (who we refer to as the Copy-Cat), indicating that we *do* allow information flow for this player between the two game boards. This for example allows information revealed in one game board by the Opponent to be used against him on the other game board—as exemplified by the copy-cat strategy. However, note that the wall appears on the environment's side now, indicating that the environment is constrained to play separately on the two boards, with no communication between them.

Thus we have a De Morgan duality between these two connectives, mediated by the Linear negation:

$$\begin{aligned} (A \otimes B)^{\perp} &= A^{\perp} \parr B^{\perp} \\ (A \parr B)^{\perp} &= A^{\perp} \otimes B^{\perp} \end{aligned}$$

The idea is that on one side of the mirror of duality (Player/System for the Tensor, Opponent/Environment for the Par), we have the constraint of no information flow, while on the other side, we do have information flow.

We can now reconstrue the Copy-Cat strategy in logical terms:



We see that it is indeed a winning strategy for $A^{\perp} \parr A$. Moreover, we can define $A \multimap B$ ("Linear implication") by

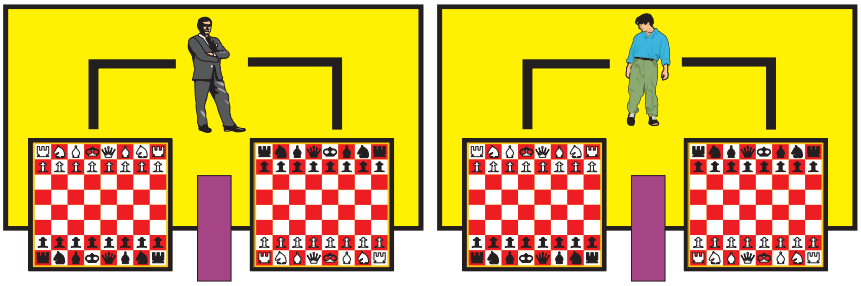
$$A \multimap B \equiv A^{\perp} \parr B$$

(*cf.* $A \supset B \equiv \neg A \vee B$.) The information flow possibilities of Par receive a more familiar logical interpretation in terms of the Linear implication; namely, that we can use information about the antecedent in proving the consequent (and conversely with respect to their negations, if we think of proof by contraposition).

Thus an entire "linearized" logical structure opens up before us, with a natural interpretation in terms of the dynamics of information flow.

4.4.4 Interaction

We now turn to a key step in the development: the modelling of *interaction* itself. Constructors create "potentials" for interaction; the operation of plugging modules together so that they can communicate with each other releases this potential into actual computation.



$A^{\perp} \parr B \;\equiv\; A \multimap B$ $\qquad$ $B^{\perp} \parr C \;\equiv\; B \multimap C$

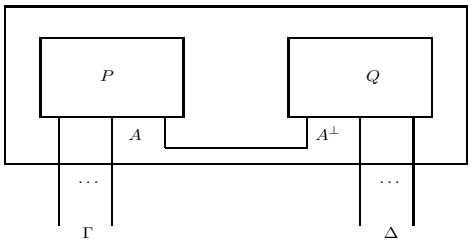
Here we see two separate sub-systems, each with a compound structure, expressed by the *logical types of their interfaces*. What these types tell us is that these systems are *composable*; in particular, the *output type* of the first system, namely B, matches the input type of the second system. Note that this "logical plug-compatibility" makes essential use of the duality, just as the copy-cat strategy did. What makes Gary (the player for the first system) a fit partner for interaction with Nigel (the player for the second system), is that they have *complementary views* of their locus of interaction, namely B. Gary will play in this type "positively", as Player (he sees it as B), while Nigel will play "negatively", as Opponent (he sees it as $B^{\perp}$). Thus each will become part of the environment of the other—part of the potential environment of each will be realized by the other, and hence part of the *potential* behaviour of each will become *actual* interaction.

This leads to a dynamical interpretation of the fundamental operation of *composition*, in mathematical terms:

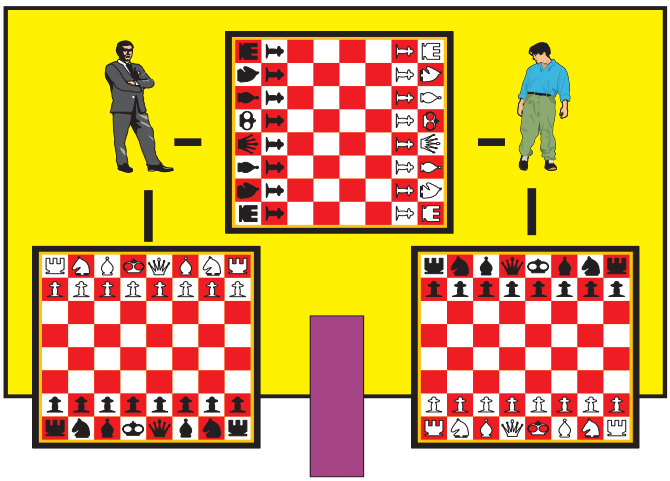
$$\frac{A \xrightarrow{\text{Gary}} B \xrightarrow{\text{Nigel}} C}{A \xrightarrow{\text{Gary;Nigel}} C}$$

or of the *Cut rule*, in logical terms:

$$\text{Cut:} \qquad \frac{\vdash \Gamma, A \quad \vdash A^{\perp}, \Delta}{\Gamma, \Delta}$$



Composition as Interaction



The Interaction Game

The picture here shows the new system formed by plugging together the two sub-systems. The "external interface" to the environment now shows just the left hand board A as input, and the right hand board C as output. The Cut formula B is hidden from the environment, and becomes the locus of interaction inside the black box of the system. Suppose that the Environment makes some move m in C. This is visible only to Nigel, who as a strategy for $B \multimap C$ has a response. Suppose this response m_1 is in B. This is a move by Nigel as Player in $B^{\perp}$, hence appears to Gary as a move by Opponent in B. Gary as a strategy for $A \multimap B$ has a response m_2 to this move. If this response is again in B, Nigel sees it as a response by the environment to his move, and will have a response again; and so on. We thus have a sequence of moves $m_1, \ldots, m_k$ in B, ping-ponging back and forth between Nigel and Gary. If, eventually, Nigel responds to Gary's last move by playing in C, or Gary responds to Nigel's last move by playing in A, then we have the response of the *composed strategy* Gary; Nigel to the original move m. Indeed, all that is visible to the Environment is that it played m, and eventually some response appeared, in A or C.

Moreover, if both Nigel and Gary are winning strategies, then so is the composed strategy; and the composed strategy will not get stuck forever in the internal ping-pong in B. To see this, suppose for a contradiction that it did in fact get stuck in B. Then we would have an infinite play in B following the winning strategy Gary for Player in B, and the *same* infinite play following the winning strategy Nigel for Player in $B^{\perp}$, hence for Opponent in B. This yields the desired contradiction.

4.5 Discussion

Game Semantics in the sense discussed in this section has had an extensive development over the past decade, with a wealth of applications to the semantics of programming languages, type theories and logics [15, 16, 20, 21, 19, 22, 23, 56]. More recently, there has been an algorithmic turn, and some striking applications to verification and program analysis [39, 4, 9, 73].

From the point of view of the general analysis of Information, we see the following promising lines of development:

- Game semantics provides a promising arena for exploring the combination of quantitative and qualitative theories of information, as discussed in Section 3, but now in a

dynamic setting. In particular, it provides a setting for quantifying information flow between agents. We would like to ask quantitative questions about *rate of information flow* through a strategy (representing a program, or a proof), how can a system gain *maximum* information from its environment while providing *minimal* information in return, robustness in the presence of noise, etc.

- As we saw in our discussion of the copy-cat strategy, there is an intuition of logical principles arising as *conservation laws for information flow*. (And indeed, in the case of Multiplicative Linear Logic, the proofs correspond exactly to "generalized copy-cat strategies"). Can we develop this intuition into a fully-fledged theory? Can we *characterize* logical principles as those expressing the conservation principles of this information flow dynamics?

- There is also the hope that the more structured setting of game semantics will constrain the exuberant variety of possibilities offered by process algebra, and allow a sharper exploration of the logical space of possibilities for information dynamics. This has already been borne out in part, by the success of game semantics in exploring the space of programming language semantics. It has been possible to give crisp characterizations of the "shapes" of computations carried out within certain *programming disciplines*: including purely functional programming [16, 56], stateful programming [20, 21], general references [12], programming with non-local jumps and exceptions [59, 61], non-determinism [51], probability [37], concurrency [40, 41], names [10], polymorphism [55, 17] and more. See [22] for an overview (now rather out of date).

  There has also been a parallel line of development of giving *full completeness* results for a range of logics and type theories, characterizing the "space of proofs" for a logic in terms of informatic or geometric constraints which pick out those processes which are proofs for that logic [15, 23]. This allows a new look at such issues as the boundaries between classical and constructive logic, or the fine structure of polymorphism and second-order quantification.

- This also gives some grounds for optimism that we can capture—in a "machine-independent", and moreover "geometrical", non-inductive way—what *computational processes* are, *without* referring back to Turing machines or any other explicit machine model.

- In the same spirit as for computability, can we characterize *polynomial-time computation* and other complexity classes in such terms?

5 Emergent Logic: The Geometry of Information Flow

Game Semantics carries many vivid intuitions arising from our experiences of game-playing as a human activity. We were able to take advantage of this in the previous section to explain some key ideas without resorting to any explicit formalization. We now turn to a related but somewhat different development of interactive models for logic and computation, known loosely as "Geometry of Interaction particle-style models".[5] We will use this setting to carry forward our discussion of dynamic models for information flow, with particular emphasis on the following themes:

[5] See [44, 45, 46, 64, 35, 36], and [14, 15, 2, 3, 13, 11].

- Firstly, the model or family of models we shall discuss is technically simpler to formalize mathematically than Game Semantics, although also less cloaked in familiar intuitions. Thus we can introduce some more precision into our discussion without unduly taxing the reader.
- Secondly, the simple yet expressive nature of these models is itself of conceptual interest. They show how logic and computation can be understood in terms of simple processes of copying information from one "place" to another, generalizing what we have already seen of the copy-cat strategy. In fact, we shall see that *mere copying is computationally universal*. Moreover, models of logics and type theories arise from these models; because of the simplicity of the models, we may reasonably speak of *emergent logic*—where, as discussed in the previous section, we may think of the logical character of certain principles as arising from the fact that they express conservation laws of information flow.
- We will also be able to make visible how geometrical structure unfolds in these models, in a striking and unexpected fashion. This part of the development can be carried much further than we can describe here; there is a thread of ideas linking logical processes of cut-elimination to diagram algebras, knot theory and topological quantum field theory.
- We shall also begin to see the beginnings of links between *Logic* and *Physics*. The processes we shall describe will be *reversible* in a very strong sense. This link can in fact be carried much further, and the same kind of structures we are discussing here can be used to axiomatize Quantum Mechanics, and to give an incisive analysis of quantum entanglement and information flow.

5.1 Background: Combinatory Logic

It will be convenient to work in the setting of Combinatory Logic, which provides one of the simplest of all the formulations of computability—and moreover one which is purely algebraic. Combinatory Logic is also the basis for realizability constructions, which provide powerful methods for building extensional models of strong impredicative type theories and higher-order logics.

We recall that combinatory logic is the algebraic theory **CL** given by the signature with one binary operation (application) written as an infix $\_\cdot\_$, and two constants $\mathbf{S}$ and $\mathbf{K}$, subject to the equations

$$\begin{array}{lcl} \mathbf{K}\cdot x\cdot y & = & x \\ \mathbf{S}\cdot x\cdot y\cdot z & = & x\cdot z\cdot(y\cdot z) \end{array}$$

(application associates to the left, so $x\cdot y\cdot z = (x\cdot y)\cdot z$). Note that we can define $\mathbf{I} \equiv \mathbf{S}\cdot\mathbf{K}\cdot\mathbf{K}$, and verify that $\mathbf{I}\cdot x = x$.

The key fact about the combinators is that they are *functionally complete*, i.e. they can simulate the effect of λ-abstraction. Specifically, we can define bracket abstraction on combinatory terms built using a set of variables X:

$$\begin{array}{lcl} \lambda^* x.\, M & = & \mathbf{K}\cdot M \quad (x \notin \mathsf{FV}(M)) \\ \lambda^* x.\, x & = & \mathbf{I} \\ \lambda^* x.\, M\cdot N & = & \mathbf{S}\cdot(\lambda^* x.\, M)\cdot(\lambda^* x.\, N) \end{array}$$

Moreover (Theorem 2.15 in [53]):

$$\mathbf{CL} \vdash (\lambda^* x.\, M)\cdot N = M[N/x].$$

The **B** combinator can be defined by bracket abstraction from its defining equation:

$$\mathbf{B} \cdot x \cdot y \cdot z = x \cdot (y \cdot z).$$

The combinatory *Church numerals* are then defined by

$$\bar{n} \equiv (\mathbf{S} \cdot \mathbf{B})^n \cdot (\mathbf{K} \cdot \mathbf{I})$$

where we define

$$a^n \cdot b = a \cdot (a \cdots (a \cdot b) \cdots).$$

A partial function $\phi : \mathbb{N} \rightharpoonup \mathbb{N}$ is *numeralwise represented* by a combinatory term $M \in T_{\mathbf{CL}}$ if for all $n \in \mathbb{N}$, if $\phi(n)$ is defined and equal to m, then

$$\mathbf{CL} \vdash M \cdot \bar{n} = \bar{m}$$

and if $\phi(n)$ is undefined, then $M \cdot \bar{n}$ has no normal form.
The basic result on computational universality of **CL** is then the following (Theorem 4.18 in [53]):

Theorem 5.1 *The partial functions numeralwise representable in* **CL** *are exactly the partial recursive functions.*

Principal Types of Combinators Curry observed that the principal (*i.e.* the most general) types of the combinators corresponded to axiom schemes for a Hilbert-style proof system for Intuitionistic implicational logic—with the application operation corresponding to Modus Ponens. This is the "Curry" part of the Curry-Howard isomorphism. Thus combinators are to Hilbert-style systems as λ-calculus is to Natural Deduction.

These principal types can be computed by the Hindley-Milner algorithm [52] from the defining equations for the combinators:

$$\begin{array}{lcl} \mathbf{K} & : & \alpha \to (\beta \to \alpha) \\ \mathbf{S} & : & (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma). \end{array}$$

The Curry Combinators Curry's original set of combinators was not the Schönfinkel combinators **S** and **K**, but rather the combinators **B**, **C**, **K**, and **W**:

$$\begin{array}{lcl} \mathbf{B} \cdot x \cdot y \cdot z & = & x \cdot (y \cdot z) \\ \mathbf{C} \cdot x \cdot y \cdot z & = & x \cdot z \cdot y \\ \mathbf{W} \cdot x \cdot y & = & x \cdot y \cdot y \end{array}$$

These combinators are equivalent to the Schönfinkel combinators, in the sense that the two sets are inter-definable [28, 53]. In particular, **S** can be defined from **B**, **C**, **I** and **W**. They have the following principal types:

I	:	$\alpha \to \alpha$	Axiom
B	:	$(\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$	Cut
C	:	$(\alpha \to \beta \to \gamma) \to \beta \to \alpha \to \gamma$	Exchange
K	:	$\alpha \to \beta \to \alpha$	Weakening
W	:	$(\alpha \to \alpha \to \beta) \to \alpha \to \beta$	Contraction

Thus we see that in logical terms, **B** expresses the transitivity of implication, or the Cut rule; **C** is the Exchange rule; **W** is Contraction; and **K** is Weakening. Curry's analysis of *substitution* is close to Gentzen's analysis of *proofs*.

5.2 Linear Combinatory Logic

We shall now present another system of combinatory logic: *Linear Combinatory Logic* [3, 11, 13]. This can be seen as a finer-grained system into which standard combinatory logic, as presented in the previous section, can be interpreted. By exposing some finer structure, Linear Combinatory Logic offers a more accessible and insightful path towards our goal of mapping universal functional computation into a simple model of computation as copying.

Linear Combinatory Logic can be seen as the combinatory analogue of Linear Logic [43]; the interpretation of standard Combinatory Logic into Linear Combinatory Logic corresponds to the interpretation of Intuitionistic Logic into Linear Logic. Note, however, that the combinatory systems we are considering are type-free and "logic-free" (*i.e.* purely equational).

Definition 5.2 *A* Linear Combinatory Algebra $(A, \cdot, !)$ *consists of the following data:*

- *An applicative structure* $(A, \cdot)$
- *A unary operator* $! : A \to A$
- *Distinguished elements* $\mathbf{B}$, $\mathbf{C}$, $\mathbf{I}$, $\mathbf{K}$, $\mathbf{D}$, δ, $\mathbf{F}$, $\mathbf{W}$ *of* A

satisfying the following identities (we associate $\cdot$ *to the left and write* $x \cdot !y$ *for* $x \cdot (!(y))$*, etc.) for all variables* x, y, z *ranging over* A*):*

1.	$\mathbf{B} \cdot x \cdot y \cdot z$	$=$	$x \cdot (y \cdot z)$	Composition/Cut
2.	$\mathbf{C} \cdot x \cdot y \cdot z$	$=$	$(x \cdot z) \cdot y$	Exchange
3.	$\mathbf{I} \cdot x$	$=$	x	Identity
4.	$\mathbf{K} \cdot x \cdot !y$	$=$	x	Weakening
5.	$\mathbf{D} \cdot !x$	$=$	x	Dereliction
6.	$\delta \cdot !x$	$=$	$!!x$	Comultiplication
7.	$\mathbf{F} \cdot !x \cdot !y$	$=$	$!(x \cdot y)$	Monoidal Functoriality
8.	$\mathbf{W} \cdot x \cdot !y$	$=$	$x \cdot !y \cdot !y$	Contraction

The notion of LCA corresponds to a Hilbert style axiomatization of the $\{!, \multimap\}$ fragment of linear logic [3, 26, 76]. The *principal types* of the combinators correspond to the axiom schemes which they name. They can be computed by a Hindley-Milner style algorithm [52] from the above equations:

1.	$\mathbf{B}$	:	$(\beta \multimap \gamma) \multimap (\alpha \multimap \beta) \multimap \alpha \multimap \gamma$
2.	$\mathbf{C}$	:	$(\alpha \multimap \beta \multimap \gamma) \multimap (\beta \multimap \alpha \multimap \gamma)$
3.	$\mathbf{I}$	:	$\alpha \multimap \alpha$
4.	$\mathbf{K}$	:	$\alpha \multimap !\beta \multimap \alpha$
5.	$\mathbf{D}$	:	$!\alpha \multimap \alpha$
6.	δ	:	$!\alpha \multimap !!\alpha$
7.	$\mathbf{F}$	:	$!(\alpha \multimap \beta) \multimap !\alpha \multimap !\beta$
8.	$\mathbf{W}$	:	$(!\alpha \multimap !\alpha \multimap \beta) \multimap !\alpha \multimap \beta$

Here $\multimap$ is a *linear function type* (linearity means that the argument is used exactly once), and $!\alpha$ allows arbitrary copying of an object of type α.

A *Standard Combinatory Algebra* consists of a pair $(A, \cdot_s)$ where A is a nonempty set and $\cdot_s$ is a binary operation on A, together with distinguished elements $\mathbf{B}_s, \mathbf{C}_s, \mathbf{I}_s, \mathbf{K}_s$, and $\mathbf{W}_s$ of

A, satisfying the following identities for all x, y, z ranging over A:

$$\begin{array}{lrcl} 1. & \mathbf{B}_s \cdot_s x \cdot_s y \cdot_s z & = & x \cdot_s (y \cdot_s z) \\ 2. & \mathbf{C}_s \cdot_s x \cdot_s y \cdot_s z & = & (x \cdot_s z) \cdot_s y \\ 3. & \mathbf{I}_s \cdot_s x & = & x \\ 4. & \mathbf{K}_s \cdot_s x \cdot_s y & = & x \\ 5. & \mathbf{W}_s \cdot_s x \cdot_s y & = & x \cdot_s y \cdot_s y \end{array}$$

This is just a combinatory algebra with interpretations of the Curry combinators. Note that this is equivalent to the more familiar definition of **SK**-combinatory algebra as discussed in the previous sub-section.

Let $(A, \cdot, !)$ be a linear combinatory algebra. We define a binary operation $\cdot_s$ on A as follows: for $a, b \in A$, $a \cdot_s b \equiv a \cdot !b$. We define $\mathbf{D}' \equiv \mathbf{C} \cdot (\mathbf{B} \cdot \mathbf{B} \cdot \mathbf{I}) \cdot (\mathbf{B} \cdot \mathbf{D} \cdot \mathbf{I})$. Note that

$$\mathbf{D}' \cdot x \cdot !y = x \cdot y.$$

Now consider the following elements of A.

$$\begin{array}{lrcl} 1. & \mathbf{B}_s & \equiv & \mathbf{C} \cdot (\mathbf{B} \cdot (\mathbf{B} \cdot \mathbf{B} \cdot \mathbf{B}) \cdot (\mathbf{D}' \cdot \mathbf{I})) \cdot (\mathbf{C} \cdot ((\mathbf{B} \cdot \mathbf{B}) \cdot \mathbf{F}) \cdot \delta) \\ 2. & \mathbf{C}_s & \equiv & \mathbf{D}' \cdot \mathbf{C} \\ 3. & \mathbf{I}_s & \equiv & \mathbf{D}' \cdot \mathbf{I} \\ 4. & \mathbf{K}_s & \equiv & \mathbf{D}' \cdot \mathbf{K} \\ 5. & \mathbf{W}_s & \equiv & \mathbf{D}' \cdot \mathbf{W} \end{array}$$

Theorem 5.3 *Let $(A, \cdot, !)$ be a linear combinatory algebra. Then $(A, \cdot_s)$ with $\cdot_s$ and the elements $\mathbf{B}_s, \mathbf{C}_s, \mathbf{I}_s, \mathbf{K}_s, \mathbf{W}_s$ as defined above is a standard combinatory algebra.*

Finally, we mention a special case which will arise in our model. An *Affine Combinatory Algebra* is a Linear Combinatory Algebra such that the **K** combinator satisfies the stronger equation

$$\mathbf{K} \cdot x \cdot y = x.$$

Note that in this case we can *define* the identity combinator: $\mathbf{I} \equiv \mathbf{C} \cdot \mathbf{K} \cdot \mathbf{K}$.

5.3 A Linear Combinatory Algebra of Partial Involutions

Our aim is to describe an interactive model for logic and computation, which can be understood in two complementary ways:

- A model built from simple dynamical processes of copying information from one place to another.
- A model built from simple geometrical constructions, in which computation is interpreted as geometric simplification—tracing paths through tangles, and yanking them straight.

We begin with the dynamical interpretation. Here we think of an informatic *token* or *particle* traversing a path through logical (discrete) "space" and "time". For this purpose, we assume a set Pos of *positions* or *places* in "logical space". For the purposes of obtaining a type-free universal model of computation, it is important that Pos is (countably) infinite. (So we could just take it to be the set $\mathbb{N}$ of natural numbers). The only significant property of the instantaneous state of the particle is its current position $p \in \mathsf{Pos}$.

The *processes* we shall consider will be very simple, "history-free" or "time-independent" reversible dynamics, which we represent as *partial injective functions*

$$f : \mathsf{Pos} \rightharpoonup \mathsf{Pos}.$$

Such a process maps a particle in position p at any time t to the position $f(p)$ at time $t+1$; or may be undefined. In fact, we will have no need to make time explicit, since discrete time will be modelled adequately by *function composition*[6]. Thus the path traced by the particle starting from position p_0 under the dynamics f will be

$$p_0, p_1, p_2, \ldots, p_n, \ldots$$

where $p_{i+1} = f(p_i)$. This dynamics is clearly reversible. Since f is a partial injective map, its inverse f^{-1} (*i.e.* the relational converse of f) is also a partial injective function on Pos, and $p_i = f^{-1}(p_{i+1})$, so we can trace the reverse path using the inverse dynamics.

In fact, it will be possible to restrict ourselves to an even simpler class of dynamics: namely the *fixed-point free partial involutions*, *i.e.* those partial injective functions $f : \mathsf{Pos} \rightharpoonup \mathsf{Pos}$ satisfying

$$f = f^{-1}, \qquad f \cap 1_{\mathsf{Pos}} = \varnothing.$$

Thus such a map satisfies:

$$f(x) = y \iff x = f(y), \qquad f(x) \neq x.$$

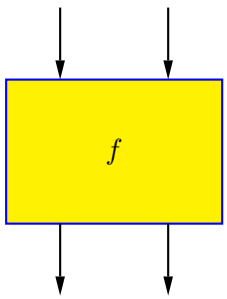
A partial involution on a set X is equivalently described as a partial partition of X into 2-element subsets:

$$X \supseteq \bigcup E, \qquad \text{where } E = \{\{x, y\} \mid f(x) = y\}.$$

This defines an undirected graph $G_f = (X, E)$. Clearly each vertex in this graph has at most one incident edge. Conversely, every graph $G = (X, E)$ with this property determines a unique partial involution f on X, with $G_f = G$. It is somewhat remarkable that such simple maps can form a universal computational model.

5.3.1 Function Application as Interaction

Our next and key step is to model *functional application* by *interaction* of these simple dynamical processes. This will in fact be a bare-bones version of the game-theoretic model of composition as interaction which we gave in the previous section. We shall view a "functional process" which can be applied to other processes as a two-input two-output function



[6]It is also the case that the paths or orbits we are considering have no fixed origin, and should really be considered as *cyclic*.

The interpretation of these two pairs of input-output lines is that the first will be used to connect the functional process to its argument, and the second to connect it to its external environment or context—which will interact with the function to consume its output. Formally, this is a function

$$f : \mathsf{Pos} + \mathsf{Pos} \longrightarrow \mathsf{Pos} + \mathsf{Pos}.$$

Note that we have the used the disjoint union (two copies of Pos) rather than the cartesian product $\mathsf{Pos} \times \mathsf{Pos}$ (infinitely many copies of Pos). This is because a particle coming in as input must *either* be on the first input line, *or* (in the exclusive sense) on the second input line; and similarly for the outputs.

However, since we want to make a type-free universal model of computation, we must reduce *all* our processes to one-input one-output functions. This is where our assumption that Pos is infinite becomes important. It allows us to define a *splitting function* s:

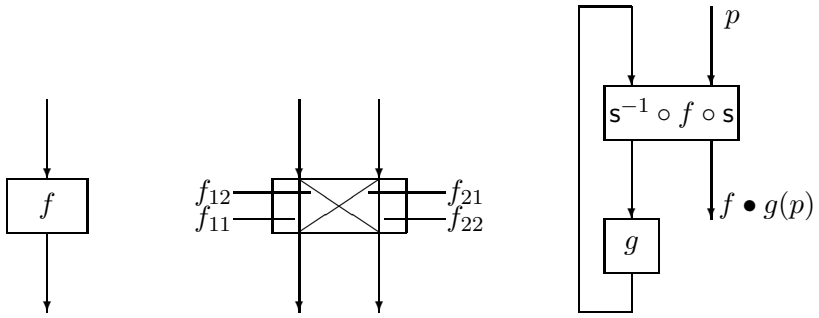
$$\mathsf{s} : \mathsf{Pos} + \mathsf{Pos} \xrightarrow{\cong} \mathsf{Pos}.$$

We can think of this as splitting logical space into two disjoint "address spaces". This allows us to transform any one-input/one-output function into a two-input/two-output function, or conversely, by conjugation. Thus if $f : \mathsf{Pos} \to \mathsf{Pos}$ is any process, we can view it as a two-input, two-output functional process, namely

$$\mathsf{s}^{-1} \circ f \circ \mathsf{s} : \mathsf{Pos} + \mathsf{Pos} \longrightarrow \mathsf{Pos} + \mathsf{Pos}.$$

5.3.2 Geometrical Representation of Application

Suppose we wish to apply f, *qua* functional process, to g, where both f and g are partial involutions on Pos. The application operation $f \bullet g$ is indicated pictorially as follows.



As already explained, we conjugate f by s to turn it into something with the right shape to be a functional process. Then we connect it to its argument, g, by a feedback loop using the first input and output lines of $f' = \mathsf{s}^{-1} \circ f \circ \mathsf{s}$. The residual behaviour by which the process resulting from the application communicates with its environment uses the second input and output lines. The full geometric significance of how this notion of application works will become apparent when we discuss the interpretation of the combinators in this setting. But we can give the dynamical interpretation of application immediately. Suppose the process $f \bullet g$ receives a token p on its input. The function f' may immediately dispatch this to its second output line as p'—in which case, that will be the response of $f \bullet g$. This would correspond to the behaviour of a constant function, which knows its output without consulting its input. Otherwise, f' may dispatch p to its *first* output line, as p_1. This is then fed as input to g. Thus this corresponds to the function represented by f' interrogating its argument. If

$g(p_1) = p_2$, then this is fed back around the loop as input to f' (now on its first input line). We may continue in this fashion, ping-ponging between f' (on its first input/output lines) and g around the feedback loop. Eventually, f' may have seen enough, and decide to despatch the token on its second output line, as p'. We then say that $f \bullet g(p) = p'$. In other words, to the external environment, the whole interaction between f' and g has been hidden inside the black box of the application $f \bullet g$; it only sees the final response p' to the initial entry of the token at p.

All of this should seem very familiar. It follows exactly the same general lines as the game-semantical interpretation of composition which we presented in the previous section. We note the following points of difference:

- The notion of composition we discussed in the previous section was fully symmetrical between the two agents involved, reflecting the classical nature of the underlying logic. Here, we are discussing *functional computation*, and our description of application reflects the asymmetry between function and argument.
- Since we are dealing with a type-free universal model of computation, we must allow some partiality in our model. The token may get trapped in the feedback loop for ever, for example, so the involutions giving the dynamics must be partial in general. This is unavoidable, for well-known metamathematical reasons.
- We are also considering a very restricted, simple notion of dynamics here. Certainly in the game semantics context, we would not want in general to limit ourselves to such a restricted class of strategies.

5.3.3 Algebraic Description of Application

We now give a formal definition of the application operation. Firstly, consider the map $f' = \mathsf{s}^{-1} \circ f \circ \mathsf{s} : \mathsf{Pos} + \mathsf{Pos} \longrightarrow \mathsf{Pos} + \mathsf{Pos}$. Each input lies in *either* the first component of the disjoint union, *or* (exclusive or) the second, and similarly for the corresponding output. This leads to a decomposition of f' into four *disjoint partial maps* f_{ij}, $i, j \in \{1, 2\}$, where f_{ij} maps the i'th input summand to the j'th output summand. Note that f' can be recovered as the union of these four maps. Since f' is a partial involution, these maps will also be partial involutions. The decomposition is indicated pictorially in the preceding diagram. Now we can define

$$f \bullet g = f_{22} \cup f_{21}; g; (f_{11}; g)^*; f_{12}\,,$$

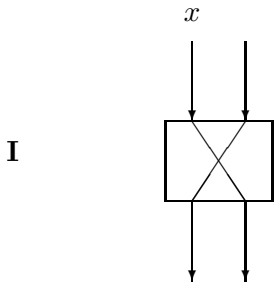
where we use relational algebra (union $R \cup S$, relational composition $R; S$ and reflexive transitive closure R^*) to write down formally exactly the information flow we described in our informal explanation of application above. It is a nice exercise to show that partial involutions are closed under application; that is, that $f \bullet g$ is again a partial involution.

5.4 Combinators as Partial Involutions

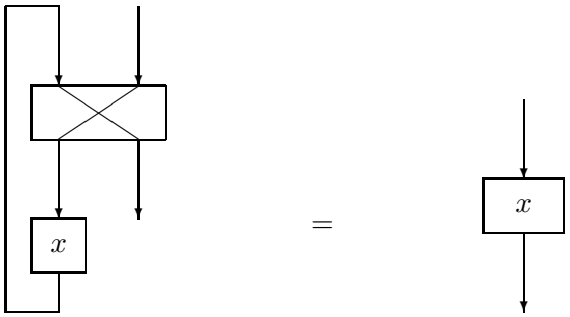
At this point, we have defined our applicative structure $(A, \bullet)$, where A is the set of partial involutions. We must now show that we can define combinators as partial involutions such that this structure will indeed form a Linear Combinatory Algebra. From now on, we shall mainly resort to drawing pictures, rather than writing algebraic expressions.

5.4.1 The Identity Combinator

Our first example is the simplest, and yet already shows the essence of the matter. The identity combinator **I** is represented by the *twist map*, which copies any token on its first input line to the second output line, and vice versa. This is depicted as follows.



What is surprising, and striking, is the geometric picture of *why* this works: that is, why the equation $\mathbf{I} \bullet x = x$ holds:



$\mathbf{I} \bullet x = x$

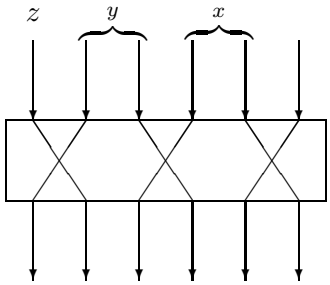
We see that geometrically, this is a matter of *yanking the string straight*; while dynamically, we picture the token flowing once around the feedback loop, and exiting exactly according to x.

Once again, we can recognize this combinator as a new description of an old friend from the previous section. *This is exactly the copy-cat strategy!* Reduced to its essence, it simply copies "tokens" or "moves" from one place to another, and *vice versa*; the logical requirement is that one of these places should be positive (or output); while the other should be negative (or input).

5.4.2 The Composition Combinator

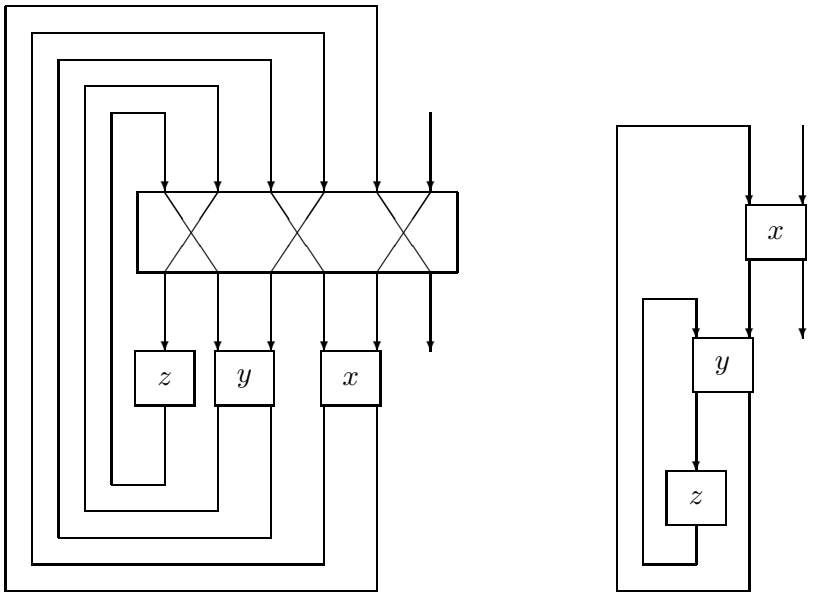
We now consider the composition combinator **B**. We interpret it as a *combination of copy-cats*. That is, it plays copy-cat between three pairs of input and output lines. (Thus, in particular, it is a partial involution).

B



Note that we can regard this combinator as having six inputs and six outputs, as shown in the diagram, simply by iterating the trick of conjugating it by the splitting map s. Our reason for giving it this many inputs and outputs is based on the *functionality* of **B**, *i.e.* its principal type. It expects to get three arguments, the first two of which will themselves be applied to arguments, and hence should each have two inputs and two outputs.

Once again, the real insight as to how this combinator works will come from the geometry, or equivalently the particle dynamics. We let the picture speak for itself.

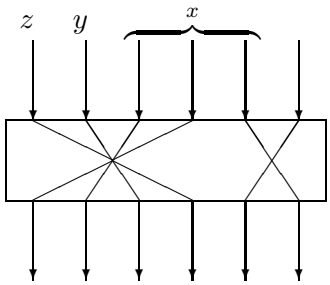


$$\mathbf{B} \cdot x \cdot y \cdot z = x \cdot (y \cdot z)$$

5.4.3 Other Affine Combinators

The remaining Linear Combinators can be described in similar style. We simply show the definition for **C**.

C

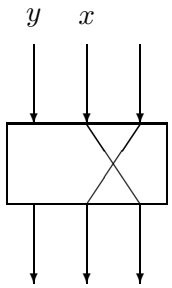


$$\mathbf{C} \cdot x \cdot y \cdot z = x \cdot z \cdot y$$

We note that geometrically, this is our first example of a *non-planar* combinator. This gives a hint of the geometrical possibilities lurking just below the surface. We shall not pursue this fascinating theme here for lack of space.

In fact, the algebra is naturally *affine*. We can define a **K** combinator:

K



$$\mathbf{K} \cdot x \cdot y = x$$

However, note that another geometric property is violated here; the first input and output lines are *disconnected* from the information flow. (Recall our discussion of the second variation of the Chess copy-cat scenario).

5.4.4 Duplication

We shall conclude our discussion of the algebra by sketching how explicit duplication of arguments is handled. This is needed for full expressive power.

We define another auxiliary function

$$\mathsf{p} : \mathbb{N} \times \mathsf{Pos} \xrightarrow{\cong} \mathsf{Pos}$$

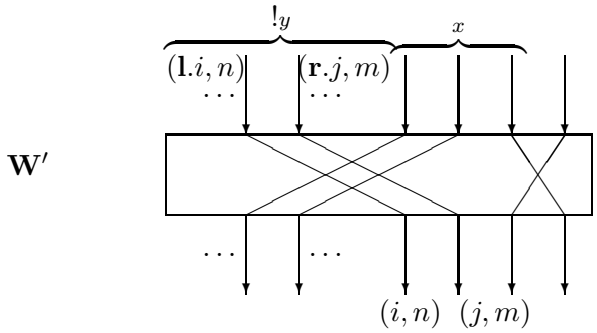
which splits logical space into countably many disjoint copies. Again, this requires the assumption that Pos is infinite. Using this, we can define an operation $!f$ which is intended to produce *infinitely many copies of* f. These are obtained by simply tagging each copy with a natural number, i.e. we define:

$$!f = \mathsf{p} \circ (1_{\mathbb{N}} \times f) \circ \mathsf{p}^{-1}.$$

We can then define **W** satisfying

$$\mathbf{W} \cdot x \cdot !y = x \cdot !y \cdot !y.$$

The W combinator



This combinator can be understood as effecting a "translation between dialects":

- x sees two arguments, each in many copies.
- $!y$ provides one argument, in as many copies as needed.

The combinator in effect decomposes into infinitely many copy-cat strategies, using a suitable splitting function to split the "address space" of the countably many copies of $!y$ into two infinite, disjoint parts, and copying between each of these and the corresponding argument position of x.

5.5 Putting the Pieces Together

We can round out the descriptions of the combinators as partial involutions to obtain a Linear Combinatory Algebra. By Theorem 5.3, this yields a standard Combinatory Algebra, and hence by Theorem 5.1 a universal model of computation. Moreover, realizability constructions over this Combinatory Algebra provide models for higher-order logics and type theories. Thus we have fulfilled our programme for this Section, of exhibiting the power of copying, leading to emergent models of logic and computation.

5.6 Discussion

Our gentle description of the partial involutions model in this section has merely indicated some first steps in this topic. We list some further directions:

- There is a general axiomatic formulation of this construction in terms of *traced monoidal categories*, with instances for deterministic, non-deterministic, probabilistic and quantum interaction [2, 11].
- The connections with reversible computation have been mentioned; this topic is carried further in [5].
- These models have some striking applications to the analysis of proofs, and of definability in various type theories, via *Full Completeness theorems* for models arising by realizability constructions over the basic Geometry of Interaction models [13].
- Current work is showing that the suggestive connections with geometry can be carried much further. In particular, there are connections with diagram algebras such as the Temperley-Lieb algebra, and hence with the Jones polynomial and ensuing developments.

- Finally, as already mentioned, there are strong connections with Quantum Information and Computation, which deserve a proper account of their own. Some references are [6, 7, 8].

References

[1] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic* 51, 1–77, 1991.

[2] S. Abramsky. Retracing some paths in process algebra. In *Proceedings of CONCUR 96*, Lectures Notes in Computer Science Volume 1119, pp. 1–17, Springer-Verlag, 1996.

[3] S. Abramsky, *Interaction, Combinators and Complexity*. Lecture Notes, Siena, Italy, 1997.

[4] S. Abramsky. Algorithmic game semantics: a tutorial introduction. In: *Proof and System-Reliability*, Kluwer, 2002.

[5] S. Abramsky. A structural approach to reversible computation. *Theoretical Computer Science* vol. 347(3), 441–464, 2005.

[6] S. Abramsky and B. Coecke. Physical traces: classical vs. quantum information processing. In Electronic Notes in Theoretical Computer Science, vol. 69, 2003, 1–26.

[7] S. Abramsky and B. Coecke, A Categorical Semantics of Quantum Protocols, in Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science: LICS 2004, IEEE Computer Society, 415–425, 2004.

[8] S. Abramsky and B. Coecke, Abstract Physical Traces, in Theory and Applications of Categories, vol 14, 111–124, 2005.

[9] S. Abramsky, D. R. Ghica, A. S. Murawski, and C.-H. L. Ong. Applying game semantics to compositional software modelling and verification. In *Proc. TACAS'04*, pages 421–435, 2004. LNCS 2988.

[10] S. Abramsky, D. R. Ghica, A. S. Murawski, I. D. B. Stark and C.-H. L. Ong. Nominal games and full abstraction for the nu-calculus. In *Proc. LICS'04*, pages 150–159, 2004. IEEE Computer Society Press.

[11] S. Abramsky and E. Haghverdi and P. J. Scott. *Geometry of Interaction and Linear Combinatory Algebras. Mathematical Structures in Computer Science* 12:625–665, 2002.

[12] S. Abramsky and K. Honda and G. McCusker. A fully abstract game semantics for general references. In Proceedings of the Thirteenth International Symposium on Logic in Computer Science, (Computer Society Press of the IEEE) 1998, 334–344.

[13] S. Abramsky and M. Lenisa. Linear realizability and full completeness for typed lambda-calculi, in *Annals of Pure and Applied Logic*, vol 134, 122–168, 2005.

[14] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. *Information and Computation* **111**, 53–119, 1994.

[15] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic* **59**, 543–574, 1994.

[16] S. Abramsky, R. Jagadeesan and P. Malacaria. Full abstraction for PCF. *Information and Computation* **163**, 409–470, 2000.

[17] S. Abramsky and R. Jagadeesan. A game semantics for generic polymorphism. In *Annals of Pure and Applied Logic*, 133: 3-37, 2005.

[18] S. Abramsky and A. Jung. Domain theory. In: *Handbook of Logic in Computer Science* Volume III, S. Abramsky, D. Gabbay and T. S. E. Maibaum, eds., 1–168. Oxford University Press, 1994.

[19] S. Abramsky and G. McCusker. Call-by-value games. In Proceedings of the Eleventh International Workshop on Computer Science Logic, M. Nielsen and W. Thomas, eds., Springer Lecture Notes in Computer Science Vol. 1414, (Springer-Verlag), 1998, 1–17.

[20] S. Abramsky and G. McCusker. Linearity, sharing and state. In: P. O'Hearn and R. D. Tennent, eds. *Algol-like languages*, pp. 317–348. Birkhauser, 1997.

[21] S. Abramsky and G. McCusker. Full abstraction for idealized Algol with passive expressions. *Theoretical Computer Science*, vol. 227 (1999), 3–42.

[22] S. Abramsky and G. McCusker. Game semantics. In: *Computational Logic: Proceedings of the 1997 Marktoberdorf Summer School*, pp.1–56. Springer-Verlag, 1999.

[23] S. Abramsky and P.-A. Mellies. Concurrent games and full completeness. In Proceedings of the Fourteenth International Symposium on Logic in Computer Science, (Computer Society Press of the IEEE) 1999, 431–442.

[24] P. M. Alberti and A. Uhlmann. *Stochasticity and Partial Order: Doubly Stochastic Maps and Unitary Mixing*. Math. Monographien 18. VEB Deutscher Verlag der Wissenschaften, 1982.

[25] R. Amadio and P.-L. Curien. *Domains and Lambda Calculi*. Cambridge University Press 1998.

[26] A. Avron, The semantics and proof theory of linear logic. *Theoretical Computer Science* 57:161–184, 1988.

[27] J.C.M. Baeten and W.P. Weijland, *Process algebra*, Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press 1990.

[28] H. P. Barendregt *The Lambda Calculus*, Studies in Logic, Vol. 103, North-Holland, 1984.

[29] J. Barwise and J. Seligman. *Information Flow: The Logic of Distributed Systems.* Cambridge University Press, 1997.

[30] J. van Benthem. *Exploring Logical Dynamics.* CSLI Publications, 1998.

[31] J. Bergstra and A. Ponse, eds. *Handbook of Process Algebra*. Elsevier 2000.

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

[33] B. Coecke. Entropic geometry from logic. In: *MFPS XIX*. 2003. `arXiv:quant-ph/0212065`

[34] B. Coecke and K. Martin. A partial order on classical and quantum states. Research Report PRG-RR-02-07 OUCL. `web.comlab.ox.ac.uk/oucl/publications/tr/rr-02-07.html`

[35] V. Danos and L. Regnier. Local and asynchronous beta-reduction. In *Proceedings of the Eighth International Symposium on Logic in Computer Science*, IEEE Press, 296–306, 1993.

[36] V. Danos and L. Regnier, Reversible, Irreversible and Optimal λ-machines, in *Electronic Notes in Theoretical Computer Science*, 1996.

[37] Vincent Danos, Russell Harmer: Probabilistic game semantics. ACM Trans. Comput. Log. 3(3): 359-382 (2002).

[38] J. Van Ejck and A. Vissers, eds. *Logic and Information Flow.* MIT Press, 1994.

[39] D. R. Ghica and G. McCusker. Reasoning about idealized algol using regular languages. In *Proc. ICALP'00*, pp. 103–116. 2000. LNCS 1853.

[40] D. R. Ghica and A. S. Murawski. Angelic semantics of fine-grained concurrency. In *Proc. FOSSACS'04*, pp. 211–225. 2004. LNCS 2987.

[41] D. R. Ghica and A. S. Murawski. Compositional model extraction for higher-order concurrent programs. In *Proc. TACAS'06*, LNCS, 2006.

[42] G. Gierz, K. H. Hofmann, K. Keimel. J. Lawson, M. Mislove and D. S. Scott. *Continuous Lattices and Domains.* Cambridge University Press, 2003.

[43] J.-Y. Girard, Linear Logic. *Theoretical Computer Science* 50(1):1-102, 1987.

[44] J.-Y. Girard, Geometry of Interaction I: Interpretation of System F, in: *Logic Colloquium '88*, ed. R. Ferro, et al. North-Holland, pp. 221-260, 1989.

[45] J.-Y. Girard, Geometry of Interaction II: Deadlock- free Algorithms. In *Proceedings of COLOG-88* (P. Martin-Lof, G. Mints, eds.) Springer LNCS Vol. 417, pp. 76-93, 1990.

[46] J.-Y. Girard, Geometry of Interaction III: accomodating the additives. In [47], 329–389.

[47] J.-Y. Girard, Y. Lafont, L. Regnier, eds. *Advances in Linear Logic*, London Math. Soc. Series 222, Camb. Univ. Press, 1995.

[48] A. M. Gleason. Measures on the closed subspaces of a Hilbert space. *Journal of Mathematics and Mechanics* **6**, 885–893, 1957.

[49] J. R. Groenendijk and M. R. Stokhof. Dynamic Predicate Logic. *Linguistics and Philosophy*, 1991.

[50] D. Harel, D. Kozen and J. Tiuryn. *Dynamic Logic.* MIT Press, 2000.

[51] Russell Harmer and Guy McCusker. A fully abstract game semantics for finite nondeterminism. In Proceedings, Fourteenth Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 1999.

[52] R. Hindley (1997) *Basic Simple Type Theory*, Cambridge Tracts in Theoretical Computer Science, no. 42, Cambridge Univ. Press.

[53] J. R. Hindley and J. P Seldin. *Introduction to Combinators and the λ-calculus*. Cambridge University Press, 1986.

[54] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall 1985.

[55] Dominic Hughes. *Hypergame Semantics: Full Completeness for System F*. D.Phil. Mathematical Sciences, Oxford University, 2000.

[56] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: i. models, observables and the full abstraction problem, ii. dialogue games and innocent strategies, iii. a fully abstract and universal game model. *Information and Computation* **163**, 285–408, 2000.

[57] C. Jones and G. Plotkin. A probabilistic powerdomain of valuations. In: *LiCS'89*.

[58] G. Kahn and G. Plotkin. Concrete Domains. *Theoretical Computer Science*, 121:187–277, 1993. Appeared as TR IRIA-Laboria 336 in 1978.

[59] J. Laird. A fully abstract games semantics of local exceptions. Extended abstract, in the *Proceedings of the 16th Annual Symposium on Logic in Computer Science*, LICS '01, 2001.

[60] J. Laird. *A semantic analysis of control*. Phd thesis, University of Edinburgh, 1998.

[61] J. Laird. Full abstraction for functional languages with control. Extended abstract, in the *Proceedings of the 12th Annual Symposium on Logic in Computer Science*, LICS '97, 1997.

[62] G. Lowe. Quantifying Information Flow. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, 2002.

[63] G. Lowe. Semantic Models for Information Flow. In *Theoretical Computer Science*, Volume 315, pages 209-256, 2004.

[64] P. Malacaria and L. Regnier. Some results on the interpretation of λ-calculus in Operator Algebras. In *Proceedings of the Sixth International Symposium on Logic in Computer Science*, IEEE Press, 63–72, 1991.

[65] K. Martin. *A Foundation for Computation*. Ph.D. Thesis, Department of Mathematics, Tulane University, 2000.

[66] K. Martin. A principle of induction. Lecture Notes in Computer Science Volume 2142, Springer Verlag, 2001.

[67] K. Martin. Unique fixed points in domain theory. Proceedings of MFPS XVII. *Electronic Notes in Theoretical Computer Science*, Volume 45, Elsevier, 2001.

[68] K. Martin. Entropy as a fixed point. ICALP 2004. Springer Lecture Notes in Computer Science Volume 3142, 2004.

[69] K. Martin, M. Mislove and J. Worrell. Measuring the probabilistic powerdomain. Lecture Notes in Computer Science Volume 2380, Springer Verlag 2002.

[70] J. McLean. Security models and information flow. In: *1990 IEEE Symposium on Security and Privacy*, 180–187, 1990.

[71] R. Milner. *Comunication and Concurrency*. Prentice Hall 1989.

[72] R. Milner. *Communicating and Mobile Systems: The Pi Calculus*. Cambridge University Press 1999.

[73] A. S. Murawski, C.-H. L. Ong, and I. Walukiewicz. Idealized Algol with ground recursion and DPDA equivalence. In *Proc. ICALP'05*, pp. 917–929. 2005. LNCS 3580

[74] D. S. Scott. *Outline of a Mathematical Theory of Computation*. Technical Monograph PRG-2 OUCL, 1970.

[75] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal* 27, 379–423 and 623–656, 1948.

[76] A. S. Troelstra, *Lectures on Linear Logic*. Center for the Study of Language and Information Lecture Notes No. 29, 1992.

[77] G. Winskel, *The Formal Semantics of Programming Languages*, MIT Press, 1993.