doi:10.1093/comjnl/bxh052

Events, Causality and Symmetry

Glynn Winskel

University of Cambridge Computer Laboratory, England
Email: Glynn.Winskel@cl.cam.ac.uk

The article discusses causal models, such as Petri nets and event structures, how they have been rediscovered in a wide variety of recent applications, and why they are fundamental to computer science. A discussion of their present limitations leads to their extension with symmetry. The consequences, actual and potential, are discussed.

Received 31 October 2008; revised 28 April 2009

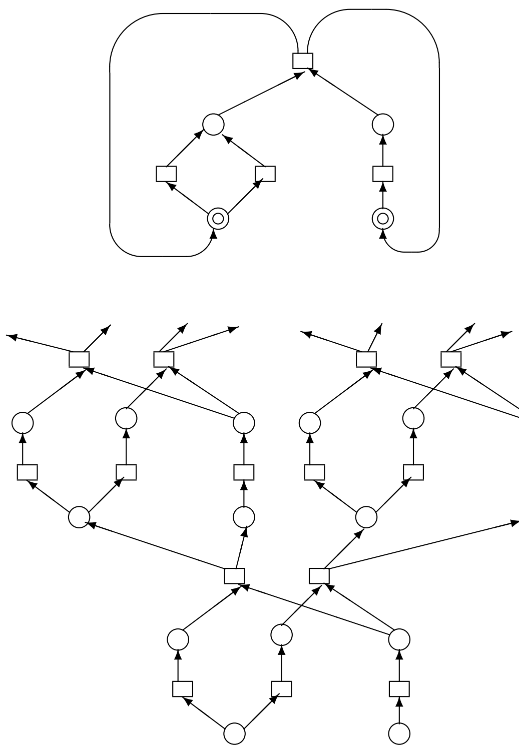
1. INTRODUCTION

We are witnessing a rebirth of interest in causal models. Causal models are alternatively described in a variety of ways, as: causal-dependence models; independence models; non-interleaving models; true-concurrency models; and partial-order models. They include Petri nets, event structures, Mazurkiewicz trace languages, transition systems with independence, multiset rewriting, and many more. Fortunately this diversity is to a large extent only apparent. The models share the central feature that they represent processes in terms of the events they can perform, and that they make explicit the causal dependency and conflicts between events. Although this might be done through different mechanisms, *e.g.* the effect events have on local states (as is the case for Petri nets) or more abstractly through relations which directly express the causal dependency and the conflict/consistency of events (in event structures), the different models can be formally related—see Section 2.2.

Causal models have arisen, and have sometimes been rediscovered as *the* natural model, in many diverse and often unexpected areas of application:

- *Security protocols:* for example, strand spaces are a form of event structure which support reasoning about secrecy and authentication through causal relations [19, 23, 92];
- *Systems biology:* biologists rediscovered Petri nets in the analysis of chemical pathways (with conditions standing for molecular species and events for reactions). Ideas from Petri nets and event structures are exploited in recent descriptive and analysis tools in tracking chemical pathways [28, 82];
- *Physics:* as causal sets in theories of quantum gravity [90];
- *Hardware:* in the design and analysis of asynchronous circuits [59];
- *Types and proof:* as representations of propositions as types, and of proofs [4, 25, 41];
- *Nondeterministic dataflow:* where numerous researchers have used or rediscovered causal models in providing a compositional semantics—see [76] and its references;
- *Network diagnostics:* in the monitoring and fault diagnosis of communication networks [10];
- *Logic of programs:* in concurrent separation logic where some artificialities in Brookes' pioneering soundness proof are obviated through a Petri-net model [49];
- *Partial order model checking:* following the seminal work of McMillan [64] the unfolding of nets is exploited in the automated analysis of systems [32];
- *Distributed computation:* event structures appear both classically [60] and recently in the Bayesian analysis of trust [71].

To illustrate the close relationship between Petri nets (based on the changes events incur on local states) and the 'partial-order models' of occurrence nets and event structures (possessing a global partial order of causal dependency on events), we consider how a Petri net can be unfolded first to a net of occurrences and from there to an event structure [72]. The unfolding construction is analogous to the well-known method of unfolding a transition system to a tree, and is central to several analysis tools in the applications above. In the figure, the net on top has loops. It is an example of a (1-safe) Petri net. Its conditions drawn as circles stand for local states. An event, a rectangle, when it occurs ends the holding of its preconditions (those conditions with arcs into the event) and begins the holding of its postconditions (those conditions with arcs from the event). Initially the two conditions at the bottom are imagined to hold, shown by their being marked. Initially any of the three events with marked preconditions can occur, ending the


A Petri net and its unfolding

holding of its respective precondition and beginning the holding of its postcondition. Though of them, the two events on the left are in *conflict*, in the sense that only one of them can occur—they compete to end the holding of their common precondition. Either of those two events can occur *concurrently* with the third event to the right, in the sense that the third event shares no pre- or postconditions with them and so can occur independently. Once one of the two conflicting events and the event to the right have occurred all the preconditions of the top event will hold and it can occur, restoring the marking of conditions to its original state.

The net below it is its *occurrence-net unfolding.* It consists of all the occurrences of conditions and events of the original net, and is infinite because of the original repetitive behaviour. The occurrences keep track of what enabled them. Notice how the shared postcondition of the conflicting events on left splits into two occurrences according to which of the two conflicting events gave rise to it. Similarly the top event splits into occurrences according to the nature of the occurrence of its left precondition.

The conditions in the occurrence net play two roles. They provide links of causal dependency between event occurrences. They also show when event occurrences are in conflict through sharing a common precondition. The simplest form of event structure arises by abstracting away the conditions in the occurrence net and capturing their two roles in relations of causal dependency and conflict on event occurrences.

Another way in which causal models can arise is through making explicit the independence of actions. A *Mazurkiewicz trace language* consists of a set of sequences of actions (corresponding to a sequence of actions one might observe) together with a relation on actions, saying when one action is independent of another. If an action a is independent of an action b, then a sequence of actions of the form $s\,a\,b\,t$, where s and t are sequences of actions, is equivalent to the sequence $s\,b\,a\,t$. The occurrence of a is seen not to depend on the occurrence of b. In this way one reveals a relation of causal dependency between occurrences of actions. In fact, a Mazurkiewicz trace language determines an event structure where the events are associated with occurrences of actions [99]. A more involved construction shows that when a transition system is equipped with a suitable independence relation on its actions (technically, so that it forms an *asynchronous transition system* [7, 88]), it determines a Petri net [99].

The relations between the different forms of causal models are well understood. Despite this and their often very successful, specialized applications, causal models lack a *comprehensive* theory which would support:

- Their systematic use in giving *structured operational semantics* to a broad range of programming and process languages; while many examples exist of Petri-net semantics of processes and languages there are presently no *generally-accepted* standard techniques for describing operational semantics using Petri nets on similar lines to Plotkin's 'Structural Operational Semantics' [81].
- An expressive '*domain theory*' with rich higher-order type constructions needed by mathematical semantics. It should for example cover the standard event-structure semantics of CCS [99], extend to higher-order CCS, and support the formalization and analysis of distributed algorithms. Such a domain theory would go beyond traditional domain theory, in which types are represented as partial orders of information, in that causal models (perhaps enriched, *e.g.* with probability) would feature as denotations. (The very concreteness of causal models appears to belie this possibility.)

This paper argues for a research programme towards such a comprehensive theory, its potential benefits, and why a comprehensive theory, including a 'domain theory,' of causal models is within reach. A remedy to the overly-concrete nature of causal models is presented: a formal treatment of symmetry in causal models reveals connections between causal models *with symmetry* and the rich and expressive world of higher-dimensional algebra, in which theories of types, homotopy, geometry and combinatorics also find a mathematical home [35]. In fact much of the work reported here started in the search for an operational reading of specific semantics using higher-dimensional algebra [18, 74, 76] and the realization that denotations (as presheaves and profunctors) could sometimes be represented by event structures [73, 85].

2. HISTORY

As we experience the ever-broader uses of computers so must we adapt our understanding of what a computational process is.

In the earliest days of computer science it became accepted that a computation was essentially an (effective) partial function

$$f : \mathbb{N} \to \mathbb{N}$$

between the natural numbers. This view underpins the Church-Turing thesis on the universality of computability.

As computer science matured it demanded increasingly sophisticated mathematical representations of processes. The pioneering work of Strachey and Scott in the denotational semantics of programs assumed a view of a process still as a function

$$f : D \to D' \ ,$$

but now acting in a continuous fashion between datatypes represented as special topological spaces, 'domains' D and D'; reflecting the fact that computers can act on complicated, conceptually-infinite objects, but only by virtue of their finite approximations. Denotational semantics and domain theory set the standard for semantics of computation. The theory provided a global mathematical setting for sequential computation, and thereby placed programming languages in connection with each other; connected with the mathematical worlds of algebra, topology and logic; and inspired programming languages, type disciplines and methods of reasoning.

In the 1960's, around the time that Strachey started the programme of denotational semantics [91], Petri advocated his radical view of a process, expressed in terms of its events and their effect on local states [79]—a model which addressed directly the potentially distributed nature of computation, but which, in common with many other current models, ignored the distinction between data and process implicit in regarding a process as a function. Here it is argued that today an adequately-broad notion of process requires a marriage of Petri's view of a process and the vision of Scott and Strachey.

2.1. Classical domain theory

It is helpful to recall the basics of traditional domain theory, the mathematical foundations of denotational semantics. A *domain* is a partial order $(D, \sqsubseteq)$ with, at the very least, the completeness property that any infinite chain

$$d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$$

has a least upper bound $\bigsqcup_n d_n$. The order $\sqsubseteq$ can be mysterious to a beginner and is quite abstract. It stands for an order of increasing computational information, information which can be presented as a limit (the least upper bound) of a chain of approximations. Accordingly, a function between domains $f : D \to E$ should be *continuous* in that f should preserve limits, *i.e.* it should preserve the information order and least upper bounds of chains.

If a domain D has a least element $\bot$ and a function $f : D \to D$ is continuous, then f has a least fixed point given by $\bigsqcup_n f^n(\bot)$. This is a central tool in giving meaning to recursive programs. By pushing this technique for solving recursive definitions up to the level of domains (treating the category of domains with continuous functions as analogous to a domain itself), Scott achieved the breakthrough in the late sixties of producing a nontrivial solution to $D \cong [D \to D]$ (a recursively defined domain), so providing a model of the λ-calculus, and, by the same techniques, the semantics of recursive types [87].

2.1.1. Representations of domains

What is the information order? Can any sense be made of its 'units' of information? There are essentially two answers in the literature, the '*topological*,' the most well-known from Scott's work, and the '*temporal*,' arising from the work of Berry [11]:

- *Topological*: the basic units of information are *propositions* describing finite properties; more information corresponds to more propositions being true. Functions are ordered pointwise. Domains are represented by logical theories in the form of 'information systems' or 'logic of domains.'
- *Temporal*: the basic units of information are *events*; more information corresponds to more events having occurred over time. Functions are restricted to 'stable' functions and ordered by the intensional 'stable order,' in which common output has to be produced for the same minimal input. Berry's specialized domains 'dI-domains' are represented by event structures.

In truth, Berry developed 'stable domain theory' by a careful study of how to obtain a suitable category of domains with stable rather than all continuous functions. He arrived at the axioms for his 'dI-domains' because he wanted function spaces (so a cartesian-closed category). The realization that dI-domains were precisely those domains which could be represented by event structures, a fact explained now, came a little later [101, 105].

2.1.2. Event structures

An *event structure* comprises $(E, \leq, \mathrm{Con})$, consisting of a set E, of *events* (event occurrences), partially ordered by $\leq$, the *causal dependency relation* which satisfies $\{e' \mid e' \leq e\}$ is finite for all $e \in E$, and a family Con of finite subsets of E, the *consistency relation*, which satisfy

$$\begin{aligned}&\{e\} \in \mathrm{Con} \text{ for all } e \in E,\\ &Y \subseteq X \in \mathrm{Con} \implies Y \in \mathrm{Con}, \quad \text{and}\\ &X \in \mathrm{Con} \;\&\; e \leq e' \in X \implies X \cup \{e\} \in \mathrm{Con}.\end{aligned}$$

The relation $e' \leq e$ expresses that the event e can only occur after the event e'. The consistency relation picks out those events which can occur together.

Consider a record of the events that have occurred. If an event e has occurred then so must all events e' on which e depends have occurred previously, and any finite set of events that have occurred should be consistent. Accordingly, the *configurations*, $\mathcal{C}(E)$, of an event structure E consist of those subsets $x \subseteq E$ which are

Down-closed: $\forall e, e'.\ e' \leq e \in x \implies e' \in x$ and
Consistent: $\forall X \subseteq_{\mathrm{fin}} x.\ X \in \mathrm{Con}$.

Configurations stand for histories, given in terms of the events that have occurred; the events inherit an order from the event structure. In particular, for an event e the set $[e] =_{\mathrm{def}} \{e' \in E \mid e' \leq e\}$ is a configuration including the whole causal history of the event e. The inclusion relation $x \subseteq x'$ means that x is a sub-history of x'. Ordered by inclusion the configurations form a domain $(\mathcal{C}(E), \subseteq)$—in fact, when E is countable, a dI-domain as discovered by Berry, and all such are so obtained. In this way event structures can represent a rich variety of types, even for polymorphism [21, 105].

2.1.3. Anomalies in domain theory

Nondeterminism: For traditional ('topological') domain theory the problem of adjoining nondeterminism was solved by Plotkin through the introduction of powerdomains [80]. Powerdomains can be seen as concerned with information about the possible and eventual properties of a nondeterministic process [102]. But for stable domain theory the information order of all but the simplest powerdomain fail to be temporal. Section 2.3 will provide an alternative way to adjoin nondeterminism.

Concurrency/interaction: The intricacy of models for distributed computation such as Petri nets and event structures (modelling both processes and types), means that they don't fit comfortably within a partial order of information. Rather their intricacy suggests that they belong more rightfully to an extended notion of domain as a category—see Section 2.2. Only rarely do the wanted equivalences on processes arise from traditional domain theory.

Probability and *nondeterminism:* There are powerdomains both for nondeterminism and probability. Sometimes one needs both. Combining probability and nondeterminism is problematic because the two forms of powerdomain together do not satisfy a distributive law (their combination forces extra laws to be imposed [67]). However, if one works with the *indexed probabilistic powerdomain* where the probability distribution is carried by the *ways* in which values are computed, one recovers a distributive law [95, 96].

Nondeterministic dataflow: While, as Kahn was early to show [57], deterministic dataflow is a shining application of simple domain theory, nondeterministic dataflow is beyond its scope. The compositional semantics of nondeterministic dataflow needs a form of generalized relation which specifies the *ways* input-output pairs are realized—see Section 2.3.

Traditional denotational semantics and domain theory appear to have abstracted away from operational concerns too early and the problems point towards a more intensional 'domain theory' which expresses the *set of ways* of computing. Early suggestions along these lines were made by Lehmann with a definition of a categorical powerdomain [62], Benson in 'counting paths' of nondeterministic processes [9] and by Girard in his 'quantitative' domain theory [40]. Lehmann and Girard's work fit within the broader view of domains as presheaf categories [18]. Causal models have reappeared in providing an operational reading of the ways computations are realized, for example as the finite configurations of an event structure [73].

2.2. An abundance of models

Partly in reaction to the difficulties of traditional denotational semantics and domain theory, today we find a range of ways to model a process in computer science, for example as: a transition system; an equivalence class of transition systems w.r.t. some equivalence such as bisimulation; a coalgebra; a resumption in a powerdomain; an IO-automaton; a Petri net; a game; a (generalized) relation; ...

Fortunately many models can be formally related by adjunctions whose adjoints give translations of one model into another. The adjunctions make explicit how to translate from one kind of model (say Petri nets) to another (say occurrence nets, or event structures); this relies on regarding a kind of model (say Petri nets) as a category (a category of Petri nets) with suitable 'simulation' maps (we define the maps on event structures below). For example the unfolding of a Petri net illustrated in the Introduction is the right adjoint to the inclusion functor from the category of occurrence nets to the category of (1-safe) Petri nets. There is an intuitively obvious map $f : \mathcal{U}(N) \to N$ from the occurrence net $\mathcal{U}(N)$ back to the original net N; it takes occurrences of conditions and events back to those conditions and events in the original net of which they are occurrences. A way to express the adjunction is through the following *universal characterization* of the unfolding. Given any map $g : O \to N$ from an occurrence net O to the original net, there is a unique map $h : O \to \mathcal{U}(N)$ such that $f \circ h = g$:

$$\begin{tikzcd}
\mathcal{U}(N) \arrow[r, "f"] & N \\
O \arrow[u, dotted, "h"] \arrow[ur, "g"'] &
\end{tikzcd}$$

[For a definition of the maps of Petri nets, see [99, 103].]

As a right adjoint, unfolding automatically preserves limits, for example products and pullbacks, in the category, and this can be useful in relating parallel compositions in one model to those in another [33]. There is a further adjunction between event structures and occurrence nets; its right adjoint 'strips' away the conditions of an occurrence net to reveal its underlying event structure, while its left adjoint 'saturates' an event structure with conditions to make an occurrence net. Adjunctions compose so we obtain an adjunction between event structures and safe nets—its right adjoint acts as the operation from Petri nets to event structures sketched in the Introduction.

The use of categories exposes a uniformity across different models. Semantics of synchronising processes [52, 65], whether they be in transition systems, Petri nets, event structures or many other models, are given in precisely the same way in terms of the categorical constructions used. Presented as categories, models support a general, diagrammatic definition of an important equivalence, *strong bisimulation* and its extension beyond transition systems, via open maps—see Section 3.2.

The categories depend on a choice of simulation map which we illustrate for event structures. The maps arise in relating compound constructions to their components, for example, as projections from a parallel composition of event structures to one of its components. Earlier we saw event structures as representations of domains, so types. Now we are seeing them in the role of processes (in the sense of the synchronising processes of Hoare and Milner in CSP and CCS). Let E and E' be event structures. A *partial map* of event structures $f : E \rightharpoonup E'$ is a partial function on events $f : E \rightharpoonup E'$ such that for all $x \in \mathcal{C}(E)$

$$\begin{aligned}
&fx \in \mathcal{C}(E') \text{ and} \\
&\text{if } e_1, e_2 \in x \text{ and } f(e_1) = f(e_2)\,,\ \text{both being defined,} \\
&\qquad\qquad\qquad\qquad\text{then } e_1 = e_2.
\end{aligned}$$

The idea is that the occurrence of an event e in E induces the *coincident* occurrence of the event $f(e)$ in E', whenever it is defined. Because events are thought of as essentially without duration, two distinct events which occur in a history of the input cannot be coincident with a common event in the image—the reason for the 'local injectivity' condition.

Despite the abundance of categories of pre-existing models, they form a rather patchy landscape and are, for example, insufficient to represent *higher-order* processes (which might take a process itself as input and deliver another process as output). Presheaf categories fill out the landscape of models to provide a versatile range of models for processes [18]. The idea is to build models for processes directly out of computation paths, regarding a nondeterministic process as a presheaf on a category of paths; essentially, a presheaf is a glueing together of computation paths. Presheaf categories are as versatile as the notion of computation path. With suitable choices of paths, presheaf categories subsume existing models such as event structures, while supporting a range of type constructions, also for higher-order processes and name generation [17, 18, 73, 107]. Presheaf categories and the relations between them, expressed as profunctors, connect with the rich world of higher-dimensional algebra. In particular, the little-explored representation of processes as 'bundles' is crucial in the general treatment of weak bisimulation, and its extension to causal models [14, 34]. But the mathematical advantages come at a cost, that of finding an operational reading.

2.3. Relations rediscovered

From a historical perspective it is remarkable that so many of the models of processes listed in Section 2.2 are not associated with input and output types; while the structure of models has become much more sophisticated the explicit connections with input an output data have often been lost.

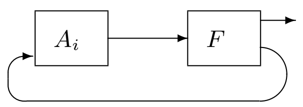
A central exception is that of generalized relations in the form of *profunctors* (or *distributors*, or *bimodules*). Profunctors are relations between categories in which the category of sets takes over the role of truth values; instead of simply saying whether or not an object of input is related to an object of output, a profunctor provides a set of ways in which that particular input-output instance is realized. Technically, a profunctor F from a category A to a category B, written $F : A \mathrel{\rightarrow\!\!\!\!\!+} B$, is a functor $F : A \times B^{\text{op}} \to \mathbf{Set}$, covariant in input and contravariant in output. When A is the trivial category $\mathbf{1}$ with just a single object and the single identity map, the profunctor amounts to just a contravariant functor from B to $\mathbf{Set}$, in other words a presheaf on B. (See [108] for more intuition.)

Profunctors have arisen independently in a range of areas: in logic and types, e.g. Girard's normal functors [40] and 'container types' [3]; combinatorics through Joyal's theory of species [55] and its extensions [35]; nondeterministic dataflow [76]; higher-order programming languages and processes [18]; categories of models for concurrent computation [18]; as a starting point for the theory of operads [24]; ...

The 'relations' arising from computation can often be represented, in a more computationally informative way, in terms of event structures, with event structures playing both the role of input and output *types*, as well as the *process* of computation between them. A compelling example comes from the early work of Brock and Ackerman who were the first to emphasize the difficulties in giving a compositional semantics to nondeterministic dataflow [13], though our example is based on simplifications in the later work of Rabinovich and Trakhtenbrot, and Russell.

Nondeterministic dataflow—Brock-Ackerman anomaly

$C[A_i] =$



There are two simple nondeterministic processes A_1 and A_2, which have the same input-output relation, and yet behave differently in the common feedback context $C[-]$, illustrated above. The context consists of a fork process F (a process that copies every input to two outputs), through which the output of the automata A_i is fed back to the input channel, as shown in the figure. Process A_1 has a choice between two behaviours: either it outputs a token and stops, *or* it outputs a token, waits for a token on input and then outputs another token. Process A_2 has a similar nondeterministic behaviour: either it outputs a token and stops, *or* it waits for an input token, then outputs two tokens. For both automata, the input-output relation relates empty input to the eventual output of one token, and non-empty input to one or two output tokens. But $C[A_1]$ can output two tokens, whereas $C[A_2]$ can only output a single token. Notice that A_1 has two ways to realize the output of a single token from empty input, while A_2 only has one. It is this extra way, not caught in a simple input-output relation, that gives A_1 the richer behaviour in the feedback context.

Over the years there have been many solutions to giving a compositional semantics to nondeterministic dataflow (see [76] for fuller references). But they all hinge on some form of generalized relation, to distinguish the different ways in which output is produced from input. A compositional semantics can be given using *stable spans* of event structures, an extension of Berry's stable functions to include nondeterminism [85, 86]. A process of nondeterministic dataflow, with input type given by an event structure A and output by an event structure B, is captured by a pair of maps (a span)

$$\begin{tikzcd} & E \arrow[dl, "dem"'] \arrow[dr, "out"] & \\ A & & B \end{tikzcd}$$

where E is also an event structure. The map $out : E \to B$ is a *rigid* map, *i.e.* a total map of event structures as in Section 2.2 which preserves the relation of causal dependency, or equivalently, a total map with the property that for a configuration x of E if y is a subconfiguration of $out\, x$ then there is a (necessarily unique) subconfiguration x' of x such that $out\, x' = y$:

$$\begin{tikzcd} x' \arrow[r, dotted, phantom, "\subseteq"] \arrow[d, dotted, mapsto] & x \arrow[d, mapsto] \\ y' \arrow[r, phantom, "\subseteq"] & out\, x \end{tikzcd}$$

The map $dem : E \to A$, associated to input, is of a different character. It is a *demand* map, *i.e.* a function from $\mathcal{C}(E)$ to $\mathcal{C}(A)$ which preserves finite configurations and unions; $dem\, x$ is the minimum input for x to occur and is the union of the demands of its events. The occurrence of an event e in E demands minimum input $dem\,[e]$ and is observed as the output event $out(e)$. *Deterministic* stable spans, where consistent demands in A lead to consistent behaviour in E, correspond to Berry's stable functions.

The stable span determines a *profunctor* $\widetilde{E}$ from the finite configurations p of A to the finite configurations q of B:

$$\widetilde{E}(p, q) = \{x \in \mathcal{C}(E) \mid dem\, x \subseteq p \;\&\; out\, x = q\}\,,$$

the set of ways the input-output pair (p, q) is realized.

Stable spans can be composed one after the other (essentially by a pullback construction, as rigid maps extend to special demand maps between configurations)—their composition coincides with the composition of their profunctors. They also have a nondeterministic sum, and compose in parallel, and most significantly allow a feedback operation [85].

In fact, stable spans were first discovered as a way to represent, and give operational meaning to, the profunctors that arose as denotations of terms in affine-HOPLA, an affine Higher Order Process LAnguage [73, 74]. The spans helped explain the tensor of affine-HOPLA as the parallel juxtaposition of event structures and a form of entanglement which appeared there as patterns of consistency and inconsistency on events. The use of stable spans in nondeterministic dataflow came later as a representation of the profunctors used in an earlier semantics [76, 85].

3. CAUSAL MODELS AND SYMMETRY

3.1. Anomalies in traditional causal models

The insufficiency of stable spans: Although one can easily encode CCS within affine-HOPLA this induces the usual interleaving semantics of CCS. One can prove that the spans denotable by terms of affine-HOPLA can never be those of the usual event-structure semantics of CCS [99]. More generally, because output maps of stable spans are rigid, they are too restrictive when considering parallel compositions via synchronizations. Such a parallel composition of event structures can augment extra causal dependency to that which is present in the event structures originally. So 'projections' from a parallel composition to a component are rarely rigid.

Varying maps: There are uses for several different forms of maps on event structures: rigid, total and partial maps, demand maps and a variant of demand maps due to Abbes [1]. Changing the category generally changes important categorical constructions. One would like to settle on some basic maps and then have a systematic way to vary the nature of maps within it.

Unfoldings of general Petri nets: In general nets conditions can hold with multiplicities. While their occurrence net unfoldings can be defined, there is no universal characterisation like that of Section 2.2. The symmetry intrinsic to nets with multiplicities spoils uniqueness.

Unfoldings of higher-dimensional automata: Higher-dimensional automata are essentially glueings together of cubes of concurrent actions [46]. Although they can be unfolded to event structures the identifications due to glueing spoil uniqueness in trying to get a universal characterization.

Weak bisimulation: Just as for labelled transition systems weak bisimulation between labelled event structures (in which we abstract from invisible actions, generally labelled τ) can be explained as strong bisimulation between the results of 'hiding' the invisible actions [34]. Whereas the 'hiding' operation on a transition system is again a transition system, the hiding operation on an event structures does not always yield an event structure [34].

Name generation: There are methods to represent the generation of new names in causal models, *e.g.* [23, 92, 108]. But the methods are overly concrete in the sense that they ignore the implicit symmetry on names. Presently there are difficulties in extending work on an event-structure semantics of the pi-Calculus [66] to the whole language because of the absence of a key algebraic operation—a form of new-name abstraction on event structures [22] .

The anomalies have a common solution: a formal treatment of symmetry in processes. Consider the first, that stable spans are insufficient. One wishes to free up the choice of maps for the input and output legs of the span. In answer to the second difficulty one would like to do this systematically.

A systematic way to modify maps is through Kleisli maps associated with monads [68]. Starting from an original category—a good choice would be the category of event structures with rigid maps—one could hope to obtain other kinds maps from an object E to an object B as original maps from E to $T(B)$, where T is an appropriate monad. This suggests that stable spans be generalized to general spans of event structures

$$\begin{tikzcd} & E \arrow[dl] \arrow[dr] & \\ S(A) & & T(B) \end{tikzcd}$$

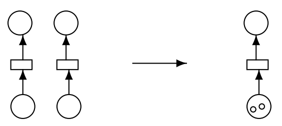
with input the event structure A, output B and process E, w.r.t. suitable monads S and T to moderate the regimes of input and output. More should hold for the spans to compose [109, 110].

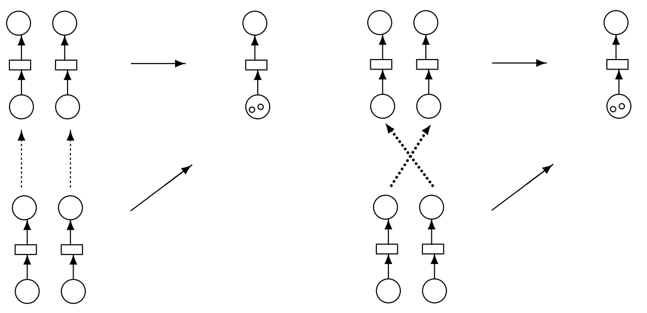
In the span shown above, one would hope in particular for a monad S such that demand maps from E to A are realized as Kleisli maps from E to $S(A)$. It becomes important that event structures are able to support a reasonable repertoire of monads. It is here we run into difficulties. Consider for example the useful operation of replication $!E$ forming the parallel composition of countably many copies of an event structure E. For this to be a monad we require a unit $\eta_E : E \to !E$ and multiplication $\mu_E : !!E \to !E$. It would seem reasonable that η_E takes E to the zeroth copy. Assuming an injection $[\_,\_] : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ encoding pairs of natural numbers as natural numbers, again it would seem reasonable that μ_E takes $(i,(j,e))$ to $([i,j],e)$. But then the laws for monads fail. It is easy to check that none of the monad diagrams commute:

$$\begin{tikzcd} !E \arrow[r, "\eta_{!E}"] \arrow[dr, "\mathrm{id}_{!E}"'] & !!E \arrow[d, "\mu_E"] \\ & !E \end{tikzcd} \qquad \begin{tikzcd} !E \arrow[r, "!\eta_E"] \arrow[dr, "\mathrm{id}_{!E}"'] & !!E \arrow[d, "\mu_E"] \\ & !E \end{tikzcd} \qquad \begin{tikzcd} !!!E \arrow[r, "\mu_{!E}"] \arrow[d, "!\mu_E"'] & !!E \arrow[d, "\mu_E"] \\ !!E \arrow[r, "\mu_E"'] & !E \end{tikzcd}$$

For example if the first two diagrams commuted, $i = [0,i] = [i,0]$, violating the injectivity of pairing. But in $!E$ one copy of E is similar to another. Up to this symmetry, allowing one copy to swap with another, the diagrams do commute. In answer to the first and second anomalies at least, a formal treatment of symmetry is needed.

It is illustrative to consider how symmetry is also important in the third anomaly, that of obtaining a universal characterisation of unfoldings of general Petri nets. In general a condition of a Petri net may not just hold or not hold, but hold with a certain multiplicity. For example, below, the net on the right has an initial marking in which a condition holds with multiplicity 2. It is generally agreed that its unfolding should be the occurrence net on the left, the two components of which correspond to the two ways in which the conditions and the events of the original general net can occur. There is a folding map taking the occurrences back to the conditions and events of the original net of which they are occurrences:





Earlier, in Section 2.2, we saw a universal characterization of the unfolding of (1-safe) Petri nets without multiply-holding conditions. An analogous result here would require that any map from an occurrence net to the original net factored *uniquely* through the folding map. But this does not hold of the folding map itself, where both the identity and the map 'swapping' the components in the unfolding provide a factorization—see the figure above. However the two maps, identity and 'swap,' are equal up to the symmetry implicit in the unfolding. The two components of the unfolding inherit their symmetry from the essentially symmetric marking of the initial condition in the original net.

The two examples, replication and unfolding, are alike and perhaps deceptively simple; symmetry in the behaviour of a process can be much less structurally apparent than in these examples. Still, they suggest that the extra structure of symmetry should express when a computation run, or path, can be swapped with another in the behaviour of a process. The method of adjoining symmetry should be tuned to process behaviour, and applicable to a wide range of models.

3.2. Symmetry

The treatment of symmetry on models proposed here makes use of a general method of open maps in defining bisimulation in a variety of models. The method shows one advantage of presenting models as categories—see [56] for more details.

Let $\mathcal{C}$ be a category with a distinguished subcategory $\mathcal{P}$ of path objects with path-extension maps. For example, the category $\mathcal{C}$ might be that of event structures and $\mathcal{P}$ the subcategory of finite 'elementary' event structures in which all subsets of events are consistent—so partial-order runs. Alternatively, the category $\mathcal{C}$ could be the category of transition systems and $\mathcal{P}$ the subcategory of special transition systems that have the form of single finite branches. (We might also allow infinite runs, important in treating 'fairness.') A map $f : X \to Y$ in $\mathcal{C}$ is a bisimulation map, traditionally called *open*, if, for any map $s : P \to Q$ in $\mathcal{P}$ and maps $p : P \to X$ and $q : Q \to Y$, if the diagram

$$\begin{tikzcd} P \arrow[r, "p"] \arrow[d, "s"'] & X \arrow[d, "f"] \\ Q \arrow[r, "q"'] & Y \end{tikzcd}$$

commutes then there is a map $h : Q \to X$ such that the diagram

$$\begin{tikzcd} P \arrow[r, "p"] \arrow[d, "s"'] & X \arrow[d, "f"] \\ Q \arrow[r, "q"'] \arrow[ur, dashed, "h"] & Y \end{tikzcd}$$

commutes, *i.e.* $p = hs$ and $q = fh$. This path-lifting property says that any extension, *viz.* qs of a path fp in Y can be matched, via f, by an extension hs of the path p in X. It can be shown by straightforward diagrammatic arguments that open maps include all isomorphisms, are closed under composition (and therefore form a subcategory). It is an instructive exercise to show that open maps are preserved under pullbacks [56].

We define a bisimulation between two objects X and Y in $\mathcal{C}$ to be a generalized relation in the form of a span of open maps in $\mathcal{C}$:

$$\begin{tikzcd} & R \arrow[dl] \arrow[dr] & \\ X & & Y \end{tikzcd}$$

A bisimulation specifies when paths in X are similar to paths in Y; that the two maps are open guarantees that similar paths have similar subpaths and similar extensions. When $\mathcal{C}$ has pullbacks, and categories such as those of event structures and transition systems do, bisimulation induces an equivalence relation on objects of $\mathcal{C}$ (using the fact that a pullback of an open map is open). In the case of transition systems the equivalence obtained is the usual strong bisimulation of Milner and Park [65], while for event structures it is also independently known, and called hereditary history-preserving bisimulation (unfortunately!) [8] .

We now show how to extend a category of models to a category of models with symmetry. We won't describe this in fullest generality, but for the case in which $\mathcal{C}$ is a category with pullbacks with distinguished subcategory $\mathcal{P}$ of path objects.

We define a new category, $\mathcal{SC}$, of objects with symmetry and symmetry-preserving maps, as follows.

The *objects* of $\mathcal{SC}$ are triples $(C; l, r : S \to C)$ comprising an object C of $\mathcal{C}$ together with *open* maps $l : S \to C$ and $r : S \to C$, from a common object S in $\mathcal{C}$,

$$\begin{tikzcd} & S \arrow[dl, "l"'] \arrow[dr, "r"] & \\ C & & C\,, \end{tikzcd}$$

so a bisimulation, which also forms a pseudo equivalence (see Appendix). The span expresses the relation of symmetry, when paths in C are similar according to the symmetry; its being a bisimulation ensures that similar paths will have similar pasts and futures. That it forms a pseudo equivalence ensures that similarity is reflexive, symmetric and transitive.

We call objects of $\mathcal{SC}$ *objects with symmetry.* When l, r form an equivalence relation (see Appendix) we shall call $(C; l, r : S \to C)$ a *symmetry equivalence.* In writing them we shall adopt the convention that for instance $(A; S_A)$ describes the object with symmetry $(A; l_A, r_A : S_A \to A)$.

In $\mathcal{SC}$, a *map* $f : (A; S_A) \to (B; S_B)$ is a map $f : A \to B$ in $\mathcal{C}$ which preserves symmetry in the sense that

$$\begin{tikzcd} A \arrow[d, "f"'] & S_A \arrow[l, "l_A"'] \arrow[r, "r_A"] \arrow[d, dashed, "h"'] & A \arrow[d, "f"] \\ B & S_B \arrow[l, "l_B"'] \arrow[r, "r_B"] & B \end{tikzcd}$$

commutes for some $h : S_A \to S_B$ in $\mathcal{C}$. This ensures that under f similar paths according to the symmetry in A go to similar paths according to the symmetry in B. Maps in $\mathcal{SC}$ compose as maps in $\mathcal{C}$ and share the same identity maps.

For maps $f, g : (A; S_A) \to (B; S_B)$, define $f \sim g$ iff there is a map $h : A \to S_B$ in $\mathcal{C}$ such that

$$\begin{tikzcd} & A \arrow[dl, "f"'] \arrow[d, dashed, "h"'] \arrow[dr, "g"] & \\ B & S_B \arrow[l, "l_B"'] \arrow[r, "r_B"] & B \end{tikzcd}$$

commutes. Then, under f and g a path in A is sent to similar paths according to the symmetry in B. Maps f and g in $\mathcal{SC}$ for which $f \sim g$ are thought of as *the same up to symmetry.*

The relation $\sim$ is an equivalence relation on maps $\mathcal{SC}(A, B)$ between objects with symmetry A and B. Composition respects $\sim$, and the category $\mathcal{SC}$ is enriched in the category of equivalence relations.

The relation $\sim$ plays a central role. It allows the relaxation of concepts normally defined using equality on maps to analogous concepts up to symmetry. For example, traditionally two objects are described as isomorphic if there is a pair of mutual isomorphisms between them. In the presence of symmetry it is more appropriate to define another equivalence on objects: Let A and B be objects with symmetry. An *equivalence* from A to B is a pair of maps $f : A \to B$ and $g : B \to A$ in $\mathcal{SC}$ such that $f \circ g \sim \mathrm{id}_B$ and $g \circ f \sim \mathrm{id}_A$; then we say A and B are equivalent and write $A \simeq B$.

An object with symmetry B represents a presheaf $\mathcal{SC}(-, B)/\!\sim$ over $\mathcal{P}$, got by quotienting w.r.t. $\sim$, and this can have a marked effect on those presheaves which can be represented.

Under minor conditions, functors and adjunctions between categories with pullbacks and subcategories of paths lift to their extensions with symmetry.

Examples
(1) If the subcategory $\mathcal{P}$ of $\mathcal{C}$ is a groupoid (*i.e.* all its maps are isomorphisms), then all maps of $\mathcal{C}$ are open and the construction of $\mathcal{SC}$ from $\mathcal{C}$ coincides with its *exact completion* [16].
(2) For the categories of event structures (with partial, total or rigid maps), a symmetry equivalence on an event structure E corresponds to an *isomorphism family* $\mathcal{S}$ of bijections $\theta : x \cong_{\mathcal{S}} y$ between finite configurations such that
(i) $\mathrm{id}_x : x \cong_{\mathcal{S}} x$ for all finite $x \in \mathcal{C}(E)$; if $\theta : x \cong_{\mathcal{S}} y$ then $\theta^{-1} : y \cong_{\mathcal{S}} x$; if $\theta : x \cong_{\mathcal{S}} y$ and $\varphi : y \cong_{\mathcal{S}} z$ then $\varphi \circ \theta : x \cong_{\mathcal{S}} z$.
(ii) if $\theta : x \cong_{\mathcal{S}} y$ and $x' \subseteq x$ with $x' \in \mathcal{C}(E)$, then the restriction $\theta' : x' \cong_{\mathcal{S}} y'$ and $y' \subseteq y$ for some (unique) $y' \in \mathcal{C}(E)$.
(iii) if $\theta : x \cong_{\mathcal{S}} y$ and $x \subseteq x'$ for finite $x' \in \mathcal{C}(E)$, then an extension $\theta' : x' \cong_{\mathcal{S}} y'$ and $y \subseteq y'$ for some $y' \in \mathcal{C}(E)$.
Note that an event structure with symmetry now represents a *category* of finite configurations where maps are got by composing isomorphisms from its isomorphism family with inclusions; by (ii) all maps can be normalized to an isomorphism from the isomorphism family followed by an inclusion. There are characterizations in terms of isomorphism families of preservation of symmetry by maps of event structures and the relation $\sim$ on symmetry-preserving maps [109, 110].
(3) For transition systems (with the standard maps preserving initial states and transitions [99]) a symmetry equivalence on a transition system corresponds to an equivalence relation on the states of the transition system which is also a strong bisimulation [65].

3.3. Consequences of symmetry

A major consequence is that many wished-for monads on event structures, such as replication, do indeed become monads up to symmetry (technically forms of pseudo monad) once event structures are extended with symmetry. Starting with the category of event structures with rigid maps, its extension with symmetry has monads with which to realize demand maps (including the variant in [1]), total non-rigid maps, partial maps and replication [109, 111]. The monads adjust the notion of event, including possibly their atomicity, so that events can now have duration.

For example, on a (countable) event structure with symmetry A, the monad for demand maps creates events out of input histories, describing the way that input is explored in A. Such a *history* is a demand map $h : I \to A$ from an elementary event structure I with events lying in the natural numbers. We can order two histories $h : I \to A$ and $h' : I' \to A$ by $h \sqsubseteq h'$ iff there is a rigid inclusion map $j : I \hookrightarrow I'$ such that $h = h' \circ j$. So ordered, histories form a dI-domain. The domain is represented by an event structure $S(A)$, in which the events are those histories $h : I \to A$ for which I has a top element; the events inherit a causal order as a restriction of the ambient order $\sqsubseteq$ and a consistency relation from compatibility w.r.t. $\sqsubseteq$. A symmetry on $S(A)$ allows us to regard two histories $h : I \to A$ and $h : I' \to A$ as similar, when I and I' are isomorphic so that their corresponding images are related by the symmetry of A. Up to symmetry, demand maps between event structures with symmetry, from E to A correspond to rigid maps, preserving symmetry, from E to $S(A)$—for more details see [109, 111]. Using a monad based on a similar idea of history it is also possible to realize the effect of compound 'persistent' events [86, 108], events with duration, out of atomic events [111].

The ubiquity of monads up to symmetry opens the way to general spans providing semantics to potentially rich process languages and event types, supporting case analysis on events. This work is incomplete, but see [110] for an example of how such a higher-order language can induce the usual event-structure semantics for CCS, as well as an event-structure semantics for a higher-order variant of CCS. In particular, the stable spans used in the semantics of nondeterministic dataflow can be realized as particular general spans, with only one monad to modify the input map, a case studied by Burroni [12]. Such spans comprise a pair of rigid maps between event structures with symmetry

$$\begin{tikzcd} & E \arrow[dl] \arrow[dr] & \\ S(A) & & B \end{tikzcd}$$

where, as we have seen, the monad S makes events of $S(A)$ out of input histories, in such a way that rigid maps from E to $S(A)$ correspond to demand maps.

By adjoining symmetry to Petri nets we can obtain a universal characterisation of unfoldings of *general* Petri nets, like that of Section 2.2, but where instead of uniqueness we achieve uniqueness up to symmetry [50]. (An explicit notion of symmetry for Petri nets was also used in [84].)

By extending event structures with symmetry we are able to represent a broader category of presheaves [106, 110, 112]. Operations that previously only worked on presheaves now work on event structures with symmetry. We can now obtain a universal characterisation of an unfolding of higher-dimensional automata as an event structure with symmetry [110]. Now, in principle, the hiding operation on event structures with symmetry yields an event structure with symmetry, bringing a central operation of weak bisimulation back into causal models. Work on a domain theory [94] within the theory of nominal sets [78], was designed with the idea of extending it to presheaf models. Viewing event structures as presheaves (now in nominal sets) would yield constructs such as new-name abstraction on event structures, the missing key to an algebraic treatment of name generation in causal models.

In summary, the introduction of symmetry is bringing a new expressive power to causal models: previously unthought-of semantics to higher-order types and languages; mechanisms for event abstraction, allowing the switch from atomic to compound events; extensions to the usual Petri-net unfolding and its preservation results; new constructions, equivalences and new-name abstraction; and possibilities of new enrichments with further structure through the presentation of generalized relations as spans of causal models.

4. RESEARCH AREAS

So far it has been mainly shown how causal models and the introduction of symmetry arise from mathematical concerns, one thing leading to another. But of course the importance of causal models to computer science derives from their wide range of application, and the numerous research areas in which they have potential significance. In the past it has been not so much the mathematical richness of causal models, but their almost physical nature, making them close to what is modelled or to be implemented, that has most influenced their use. But it is their under-appreciated and under-explored mathematical richness that will drive their new uses in the research areas below.

Domain theory for causal models: The explanation and development of causal models here, in particular the introduction of symmetry, have been motivated by considering how causal models might support a form of 'domain theory' and denotational semantics, though of a revolutionary form in which causal models play the role of both types and maps (in the form of spans). A lot remains to be done. Especially pressing is the need for high-level syntax for general spans and associated types (generalizing Moggi's monadic metalanguage [68], which has been very successful in a more limited scenario); presently, in exploratory work, definitions proceed by an intriguing 'event induction' on the structure of events associated with the types—see [110].

Distributed algorithms: One motivation in boosting the power of causal models has been to make distributed algorithms, and the use of causal models there, amenable to semantic description and reasoning techniques. The area provides a valuable testing ground for the new domain theory and ideas on symmetry. The potential richness of event types and higher-order languages should, when developed more fully, be helpful in the design and analysis of distributed algorithms, especially where causal models or symmetry are involved. Lynch's book [63] provides a thorough introduction to distributed algorithms and many examples, some using symmetry, to try out. The techniques of distributed algorithms have become relevant in chip design, recognized in the use of Petri nets there [59], and become increasingly so as chip design is forced to count the cost of communication [70]. Significant advances have been made in the more limited regime of security protocols [23, 43, 92]—there are gaps in the treatment and exploitation of symmetry, and in relations with cryptography, which calls for probability. Two other specialized areas which rely on causal models, and probabilistic event structures [97], are the distributed diagnosis of communication networks [10], and the very recent Bayesian analysis of event-based trust [71].

Probability *and* nondeterminism: The use of spans suggests largely-unexplored methods to enrich computation with probability, in addition to nondeterminism, essentially by taking the vertex in a span of event structures to be a probabilistic event structure [2, 97]. The probabilistic event structure expresses both the ways, and with which probability, output is obtained. The output event structure (to the right of the span) would now stand for a type of probabilistic processes. In special cases the idea relates to, again largely-unexplored, *categorical* versions of the indexed-probability powerdomains [95, 96]. Each construction gives a category of random variables on a category, and mimics the corresponding construction on domains. To combine probability with nondeterminism, we expect distributive laws with the presheaf-category construction, as a form of categorical powerdomain.

Systems biology: Causal models, with stochastic information, are already used in bio-computation [51] and there is reason to expect, by analogy with computer systems, that higher-order bio-processes (beginning to be accessible to the theory here) will become an important abstraction tool. Issues of compound versus atomic events are important in systems biology. In the presence of symmetry there are ways to realize forms of compound event, for example the 'persistent' events of [108], out of atomic events. Interestingly biologists are not interested in quite the same unfolding of Petri nets to event structures as most computer scientists. Biologists are more interested in the 'macroscopic' events of a biological process rather than the 'microscopic' events which are picked out by the traditional computer-science unfolding. For example, in the traditional unfolding of a Petri net the participation of the *same* molecule at several stages in a reaction would lead to a chain of causal dependencies. However to a biologist that one particular molecule rather than another has participated in a reaction is generally neither here nor there, and such incidental dependencies would be dropped [28]. The issue is closely related to a developing understanding of unfoldings for the 'collective-token' understanding of Petri nets [45]; such unfoldings involve the collapse of events according to equivalences induced by symmetry.

Unfoldings and tools: Unfoldings of Petri nets have provided strikingly successful methods for the analysis of distributed systems [32], and with some variation in systems biology [28]. The characterization of the unfolding as a right adjoint has been exploited in network diagnosis [33]. The techniques here, based on symmetry, extend unfoldings and their characterization to general nets, and begin to push Petri nets into the new territory of higher-order processes (although we have concentrated on event structures, similar ideas are working for nets). The compact, often finite, representations afforded by Petri nets accede to the known decision procedures for Petri nets and regular languages, and so potentially tools for the analysis of higher-order processes (in the manner of algorithmic game theory [39]). Symmetry is already exploited in model checking [30], and its potential role in net-unfolding techniques should be investigated.

Weak memory models: Weak memory models [54] are challenging as they do not respect a property fundamental to most causal models, that events can be globally serialized. This is because a memory transaction is more truly viewed as a compound event, comprising several different events, and which constituent event is in view depends on the processor [83]. Formalizing the relation between weak memory models and their implementation involves event abstractions.

Operational semantics: There are practical, design and dissemination issues in piloting structural operational semantics with Petri nets. In several separate Petri-net semantics, that of a language for security protocols [23], a language for biochemical systems [82], and a language with parallel commands and semaphores [49], the same technique has been used, and seems to be much more widely applicable. In defining a semantics via a Petri net (or its abbreviation as a form of coloured Petri net), first the net's basic conditions for data, names, resources and control are defined followed by definition of its events in terms of their pre- and postconditions. The definitions proceed in a syntax-directed way, much as in Plotkin's 'Structural Operational Semantics' (SOS) [81], but where in SOS it is generally transitions which are given inductively by rules, here it is events. One feature currently missing from such semantics is that of symmetry, for example between the different but similar conditions standing for names of resources. The latter suggests that the development might take place, more appropriately, within nominal or Fraenkel-Mostowski set theory [37, 78], rather than in traditional set theory (see 'Names and processes' below).

One intention in moving to a more intensional domain theory and denotational semantics is that the denotational semantics can more fully prescribe an operational reading. A guiding principle in obtaining the operational semantics for HOPLA was that elements in the presheaf denotation of a process (standing for the ways a computation could be realized) should correspond to derivations according to an operational semantics. This led to a 'strong-correspondence' theorem relating the denotational and operational semantics [73, 75]. But in general there is still some way to go in translating mathematical semantics to a rule-based operational semantics. For example, the fundamental higher-order process metalanguage introduced in [108], inspired by profunctors and exploiting their input-output duality, challenges rule-based operational semantics.

Causal models such as event structures have already played a key role in giving a more operational understanding of elements of presheaves (and so derivations in an operational semantics), as congurations of an event structure [73, 86, 108].

Quantum systems and event structures: There are intriguing parallels between quantum processes and processes described as event structures, where a degenerate form of entanglement appears in patterns of consistency and conflict amongst events [74], labels on copies of identical processes behave like amplitudes [110], and 'probabilistic tests' [97] resemble the consistent (or decoherent) families of the 'consistent-histories' approach to quantum theory.

An event structure arises quite naturally in describing the histories of quantum systems (with states in Hilbert space H). The following construction is inspired by the 'consistent-histories' approach to quantum theory [44]. A quantum history of the quantum system might be expressed as a finite sequence of quantum properties, or theoretical observations, taken to be projectors on H. The quantum histories possess the structure of a Mazurkiewicz trace language by interpreting actions as projectors, and independence as commutability of distinct projectors. By a standard construction we obtain an event structure from the Mazurkiewicz trace language [99]. It is an event structure where the events (occurrences of observations) are labelled by projectors in such a way that concurrent events are labelled by distinct commuting projectors. Each finite configuration is associated with an operator, that got by composing any sequence of projectors from which the configuration arises. In the manner of consistent histories, we can investigate those sets of configurations over which the operator weights (got via the trace inner product) determine a probability distribution. The underlying event structure begins to suggest variations on the form of decoherence conditions.

This is exploratory. But it does suggest studying constructions on Hilbert spaces, such as Fock space, in the light of event structures with symmetry, and, more generally, 'quantum event structures,' with events labelled by projectors on a Hilbert space; the hope is that quantum event structures and their constructions would furnish denotational semantics for quantum-process languages [5, 89, 93].

Reasoning techniques: Connections to the broader world of mathematics, strengthened by the addition of symmetry to causal models, pave the way to new methods of reasoning about processes. Through the addition of symmetry, causal models represent certain categories (in the case of event structures, categories of elements of certain presheaves [110, 112]). Through symmetry, equivalences such as weak bisimulation, defined via bundles [34], can be imported back into causal models, and potentially analyzed there. The richer discipline of types could play a role in establishing properties via type checking. The mathematical connections move us closer to topology and geometry and their use in reasoning about processes [31, 47, 48, 61].

For a long time it has been a puzzle how to exploit causal structure in useful specification logics for processes. There have been several suggestions and considerable ingenuity in getting logics which achieved some measure of expressivity according to one criterion or another [20, 56, 98]; most of the logics possess some form of backwards modality which can push aside independent actions. One such logic arose from the general categorical method for obtaining bisimulation from open maps [56]. But such logics have yet to become useful practical tools in the specification and analysis of processes.

At the same time there are many instances of informal reasoning through symmetry and chains of dependencies arising naturally in reasoning about distributed algorithms. In protocol design and analysis considerable skill is used in specifying the identity of events often through the generation of random or fresh names, which play an essential role in establishing causal dependencies. This is manifest in the methods of strand spaces [23, 92]. We are beginning to see convincing logics, designed specifically for security protocols, to support reasoning along causal dependency in the manner of strand spaces [26]. Symmetry is currently being exploited in tools for strand spaces [29].

Causal models, their equivalences and logics are making a surprising appearance in reversible computation [27, 77].

The influence between computer science and mathematics goes both ways; sometimes the computer-science need for frequent efficient and safe calculation can lead to methods which streamline or remove the hand-waving from the usual mathematical arguments [15, 58].

Mathematics and logic: The research cannot be divorced from questions in mathematics and logic.

Games and proofs: Event structures represent types, underlie game semantics [6, 53] and begin to appear as denotations of proofs [4, 25]; the need for symmetry also appears here and in the earlier but related geometry of interaction of Girard [42]. It is not clear if the bipartite nature of games ('opponent' vs. 'player') and the often intricate structure associated with it can be accommodated within spans of causal models. Perhaps it can: there are, for example, ways to express the composition of sequential algorithms as relational composition and this is taking us close to the composition of spans.

Names and processes: The need for fresh-name assumptions and new-name abstraction within causal models calls for projects in nominal and Fraenkel-Mostowski sets [37, 78]: the development of category theory, up to presheaves and profunctors, within nominal sets; the development of causal models in nominal sets. The first project is needed to treat new-name generation in the full range of presheaf models (by analogy with work on domain theory and semantics in nominal sets [94]), the second to import new-name abstraction and freshness assumptions systematically into causal models. Already, we can see an interesting byproduct of the second project, that symmetry relations (like those of Section 3.2) appear automatically, induced by the symmetry on names. The second project is anticipated in two converging lines of work on automata for calculi with names; 'history dependent automata' [69] are seen to equal automata in nominal sets once it is realized that 'named sets with symmetry' are a compact representations of nominal sets [36, 38].

Higher-dimensional algebra: The connection with higher-dimensional algebra is woven into the development of the new domain theory as several threads (initially via presheaves, profunctors and pseudo monads, and now also through symmetry and the enriched categories it leads to) and has already been fruitful [18, 35] and is likely to lead to the refinement and development of causal models, as we seek to give computational interpretation to mathematical constructions. We cannot expect to separate causal models from the more purely mathematical structures of higher-dimensional algebra. In particular, we need to understand better the relationship between the two kinds of generalized relations, spans of event structures and (certain kinds of) profunctors.

ACKNOWLEDGEMENTS

I have benefited from discussions with Marcelo Fiore, Jonathan Hayman, Martin Hyland, Jean Krivine, Ugo Montanari, Prakash Panangaden, Gordon Plotkin, Lucy Saunders-Evans, Pawel Sobocinski, Peter Sewell and Sam Staton, and the advice of the anonymous referees.

REFERENCES

[1] Abbes, S. (2006) A cartesian closed category of event structures with quotients. Discrete Mathematics and Theoretical Computer Science (DMTCS) 8(1), 249-272.

[2] Abbes, S. and Benveniste, A. (2006) Probabilistic models for true-concurrency: branching cells and distributed probabilities for event structures. Inf. and Comp. 204(2), 231-274.

[3] Abbott, M., Altenkirch, T. and Ghani, N. (2003) Categories of containers. Proceedings of FSSCS'03, LNCS 2620. Springer-Verlag, Berlin.

[4] Abramsky, S. (2007) Event Domains, Stable Functions and Proof Nets. In Computation, Meaning and Logic: Articles dedicated to Gordon Plotkin, ed. Cardelli, L., Fiore, M. and Winskel, G., ENTCS 172, 33–67.

[5] Abramsky, S. and Coecke, B. (2004) A categorical semantics of quantum protocols. Proceedings of LICS'04.

[6] Abramsky, S., Jagadeesan, R. and Malacaria, P. (2000) Full Abstraction for PCF. Inf. and Comp. 163, 409–470.

[7] Bednarczyk, M.A. (1988) Categories of asynchronous systems, PhD thesis in Computer Science, report no.1/88, University of Sussex, UK.

[8] Bednarczyk, M.A. (1991) Hereditary History Preserving Bisimulations or What is the Power of the Future Perfect in Program Logics. ICS PAS Report. Available from `http://www.ipipan.gda.pl/~marek/papers.html`.

[9] Benson, D.B. (1984) Counting Paths: Nondeterminism as Linear Algebra. IEEE Trans. Software Eng. 10(6), 785-794.

[10] Aghasaryan, A., Fabre, E., Benveniste, A., Boubour, R. and Jard, C. (1998) Fault detection and diagnosis in distributed systems: an approach by partially stochastic Petri Nets. Discrete event dynamic systems, theory and applications, vol 8 No 2.

[11] Berry, G. (1979) Modèles completement adéquats et stables des λ-calculs typés. Thèse de Doctorat d'Etat, Université de Paris VII.

[12] Burroni, A. (1971) T-catégories. Cahiers de topologie et géométrie différentielle, XII 3.

[13] Brock, J. and Ackerman, W. (1981) Scenarios: A model of non-determinate computation. Proceedings of the International Colloquium on Formalization of Programming Concepts, LNCS 107. Springer-Verlag, Berlin.

[14] Bunge, M. and Fiore, M.P. (2000) Unique factorisation lifting functors and categories of linearly-controlled processes. MSCS 10(2). CUP.

[15] Caccamo, M. and Winskel, G. (2005) Limit preservation from naturality. CTCS'04, ENTCS 122, 3-22.

[16] Carboni, A. and Vitale, E.M. (1998) Regular and exact completions Pure and Applied Algebra 125, 79–116.

[17] Cattani, G.L., Stark, I. and Winskel, G. (1997) Presheaf models for pi-Calculus. Proceedings of CTCS'97, LNCS 1290. Springer-Verlag, Berlin.

[18] Cattani, G.L. and Winskel, G. (2005) Profunctors, open maps and bisimulation. MSCS 15(3), 553–614. CUP.

[19] Cervesato, I., Durgin, N.A, Lincoln, P.D., Mitchell, J.C. and Scedrov, A. (2000) Relating strands and multiset rewriting for security protocol analysis. Proceedings 13th IEEE Computer Security Foundations Workshop.

[20] Nielsen, M. and Clausen, C. (1994) Bisimulations, Games, and Logic. In Karhumki, Maurer and Rozenberg, eds, Results and Trends in Theoretical Computer Science: Colloquium in Honor of Arto Salomaa, LNCS 812. Springer-Verlag, Berlin.

[21] Coquand,T., Gunter, C.A. and Winskel, G. (1987) DI-Domains as a Model of Polymorphism. Proceedings of MFPS'97, LNCS 298. Springer-Verlag, Berlin.

[22] Crafa, S., Varacca, D. and Yoshida, N. (2007) Event Structure Semantics for the Internal pi-calculus. Proceedings of CONCUR'07, LNCS 4703. Springer-Verlag, Berlin.

[23] Crazzolara, F. and Winskel, G. (2001) Events in security protocols. Proceedings of ACM Conference on Computer and Communications Security.

[24] Curien, P-L. (2006) Operads, clones, and distributive laws. Notes for an invited talk at Operads 2006, Strasbourg. Available from `http://www.pps.jussieu.fr/users/curien/`.

[25] Curien, P-L. and Faggian, C. (2005) L-Nets, Strategies and Proof-Nets. Proceedings of CSL'05, LNCS 3634. Springer-Verlag, Berlin.

[26] Datta, A., Derrick, A., Mitchell, J.C. and Roy, A. (2007) Protocol composition logic (PCL). In Computation, Meaning and Logic: Articles dedicated to Gordon Plotkin, ed. Cardelli, L., Fiore, M. and Winskel, G., ENTCS 172.

[27] Danos, V. and Krivine, J. (2004) Reversible Communicating Systems. Proceedings of CONCUR04, LNCS 3170. Springer-Verlag, Berlin.

[28] Danos, V., Feret, J., Fontana, W. and Krivine, J. (2007) Scalable Simulation of Cellular Signaling Networks. Proceedings of APLAS'07, LNCS 4807. Springer-Verlag, Berlin.

[29] Doghmi, S.F., Guttman, J.D. and Thayer, F.J. (2007) Searching for shapes in cryptographic protocols. TACAS'07.

[30] Emerson, E.A. and Sistla, A.P. (1994) Symmetry and model checking. Formal Methods in System Design.

[31] Eppendahl, A. (2007) Knot Theory and Data Distribution. Talk at Universal Structures in Mathematics and Computing (USMC), Australian National University Canbera, February 2007.

[32] Esparza, J. and Heljanko, K. (2008) Unfoldings: A Partial-Order Approach to Model Checking. EATCS Monographs in Theoretical Computer Science.

[33] Fabre, E. (2007) Bayesian Networks of Dynamic Systems. Habilitation thesis, IRISA Rennes.

[34] Fiore, M.P, Cattani, G.L. and Winskel, G. (1999) Weak Bisimulation and Open Maps. Proceedings of LICS'99.

[35] Fiore, M., Gambino, N., Hyland, J.M.E. and Winskel, G. (2007) The cartesian closed bicategory of generalised species of structures. Journal of the London Math. Soc., 77 2, 203–220. OUP.

[36] Fiore, M. P. and Staton, S. (2006) Comparing operational models of name- passing process calculi. Inf. and Comp. 204 (4), 524–560.

[37] Gabbay, M.J. (2001) A Theory of Inductive Definitions with Alpha-Equivalence. PhD thesis, Cambridge University.

[38] Gadducci, F., Miculan, M. and Montanari, U. (2006) About permutation algebras, (pre)sheaves and named sets. Higher-Order and Symbolic Computation, Vol 19 (2-3), 283-304.

[39] Ghica , D.R. and McCusker, G. (2000) Reasoning about Idealized Algol using regular languages. Proceedings of ICALP 2000, LNCS 1853. Springer-Verlag, Berlin.

[40] Girard, J-Y. (1988) Normal functors, power series and lambda calculus. Ann. Pure Appl. Logic 37.

[41] Girard, J-Y. (1987) Linear Logic. Theoretical Computer Science, 50, 1–102.

[42] Girard, J-Y. (1989) Towards a geometry of interaction. Categories, Computer Science and Logic, Contempory Mathematics, AMS 92.

[43] Gordon, A. and Jeffrey, A. (2002) Typing One-to-One and One-to-Many Correspondences in Security Protocols. Proceedings of ISSS 2002, LNCS 2609. Springer-Verlag, Berlin.

[44] Griffiths, R.B. (2003) Consistent Quantum Theory. CUP.

[45] Glabbeek, R.J. van (2005) The Individual and Collective Token Interpretations of Petri Nets. Proceedings of CONCUR 2005, LNCS 3653. Springer-Verlag, Berlin.

[46] Glabbeek, R.J. van (2005) On the expressiveness of higher dimensional automata. Proceedings of EXPRESS 2004, ENTCS 128(2).

[47] Goubault, E. (2003) Some geometric perspectives in concurrency theory. In Homology, Homotopy and Applications, 5, 95–136.

[48] Gunawardena, J. (2001) Homotopy and concurrency. Bulletin EATCS, 54, 184–193, 1994. Selected for inclusion in Paun, G., Rozenberg, G., and Salomaa, A., (eds), "Current trends in Theoretical Computer Science: Entering the 21st Century", World Scientific.

[49] Hayman, J. and Winskel, G. (2006) Independence and Concurrent Separation. Proceedings of LICS'06.

[50] Hayman, J. and Winskel, G. (2008) The unfolding of general Petri nets. Electronic proceedings of FSTTCS 2008. `http://drops.dagstuhl.de/opus/volltexte/2008/1755`

[51] Heiner, M., Gilbert, D. and Donaldson, R. (2008) Petri nets for Systems and Synthetic Biology. LNCS 5016. Springer-Verlag, Berlin.

[52] Hoare, C.A.R. (1985) Communicating Sequential Processes. Prentice-Hall International.

[53] Hyland, J.M.E. and Luke Ong, C-H. (2000) On Full Abstraction for PCF: I, II, and III. Inf. Comput. 163(2), 285–408.

[54] Intel Corp., Pentium Processor User's Manual, Vol.1.

[55] Joyal, A. (1981) Une théorie combinatoire des séries formelles. Advances in Mathematics 42.

[56] Joyal, A., Nielsen, M. and Winskel, G. (1996) Bisimulation from open maps. LICS '93 special issue of Inf. and Comp., 127(2), 164–185.

[57] Kahn, G. (1974) The semantics of a simple language for parallel programming. Information Processing, vol. 74.

[58] Kozen, D. (2004) Toward the Automation of Category Theory. Technical Report TR2004-1964, Computer Science Department, Cornell University.

[59] Khomenko, V., Koutny, M. and Yakovlev, A. (2006) Logic Synthesis for Asynchronous Circuits Based on STG Unfoldings and Incremental SAT, Fundamenta Informaticae 70 (1-2).

[60] Lamport, L. (1978) Time, clocks and the ordering of events in a distributed system. CACM vol.21.

[61] Lafont, Y. (2007) Algebra and geometry of rewriting. Applied Categorical Structures 15, 415–437. Springer-Verlag, Berlin.

[62] Lehmann D.J. (1976) Categories for fixedpoint semantics. Warwick University Theory of Computation Report No 15.

[63] Lynch, N. (1996) Distributed Algorithms. Morgan Kaufmann Publishers, San Mateo, CA.

[64] McMillan, K.L. (1995) A technique of state space search based on unfolding. Formal Methods in System Design 6(1).

[65] Milner, R. (1989) Communication and Concurrency. Prentice-Hall International.

[66] Milner, R. (1999) Communicating and Mobile Systems: the Pi-Calculus. CUP.

[67] Mislove, M.W. (2006) On Combining Probability and Nondeterminism. ENTCS 162, 261-265.

[68] Moggi, E. (1989) Computational lambda-calculus and monads. Proceedings of LICS'89.

[69] Montanari, U. and Pistore, M. (2005) History-Dependent Automata: An Introduction. Proceedings of Formal Methods for Mobile Computing, LNCS 3465. Springer-Verlag, Berlin.

[70] Moore, S., and Greenfield, D. (2008) The Next Resource War: Computation vs. Communication. Proceedings of 10th International Workshop on System-Level Interconnect Prediction.

[71] Nielsen, M., Krukow, K. and Sassone, V. (2007) A Bayesian Model for Event-based Trust. In Computation, Meaning and Logic: Articles dedicated to Gordon Plotkin, ed. Cardelli, L., Fiore, M. and Winskel, G., ENTCS 172, 499–521.

[72] Nielsen, M., Plotkin, G.D. and Winskel, G. (1981) Petri nets, event structures and domains. TCS, 13(1), 85–108.

[73] Nygaard, M. (2003) Domain theory for concurrency. PhD Thesis, University of Aarhus.

[74] Nygaard, M. and Winskel, G. (2004) Domain theory for concurrency. *TCS* 316, 153–190.

[75] Nygaard, M. (2004) Strong correspondence for HOPLA. Note. Available from http://www.daimi.au.dk/~nygaard/pub/strongcorrespondence.pdf

[76] Hildebrandt, T., Panangaden, P. and Winskel, G. (2004) A relational model of non-deterministic dataflow. MSCS 14(5), 613–649. CUP.

[77] Phillips, I. and Ulidowski, I. (2007) Reversibility and models for concurrency. Proceedings of SOS 2007, ENTCS 192(1).

[78] Pitts, A.M. (2003) Nominal Logic, A First Order Theory of Names and Binding. TACS2001 Special issue, Inf. and Comp. 186.

[79] Petri, C. A. (1962) Kommunikation mit Automaten. Ph. D. Thesis, University of Bonn.

[80] Plotkin, G.D. (1976) A Powerdomain Construction. SIAM J. Comput. 5(3), 452-487.

[81] Plotkin, G. D. (1981) A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, Aarhus University.

[82] Plotkin, G.D. (2009) A calculus of biochemical systems. In preparation.

[83] Sarkar, S., Sewell, P., Nardelli, F.Z., Owens, S., Ridge, T., Braibant, T., Myreen, M.O. and Alglave, J. (2009) The Semantics of x86-CC Multiprocessor Machine Code. Proceedings of POPL'09.

[84] Sassone, V. (1998) An axiomatization of the category of Petri nets computations. MSCS 8, 117–151. CUP.

[85] Saunders-Evans, L. and Winskel, G. (2006) Event structure spans for non-deterministic dataflow. Proceedings of Express'06, ENTCS 175(3).

[86] Saunders-Evans(=Brace-Evans), L. (2007) Events with persistence. PhD thesis, University of Cambridge Computer Laboratory.

[87] Scott, D.S. (1975) Data types as lattices. Lecture Notes in Mathematics 499.

[88] Shields, M.W. (1985) Concurrent machines, Computer Journal 28, 449–465.

[89] Selinger, P. and Valiron, B. (2008) On a fully abstract model for a quantum linear functional language. Proceedings of QPL 2006, Oxford, ENTCS 210, 123-137.

[90] Sorkin, R. (2006) Geometry from order: causal sets. From http://www.einstein-online.info/en/spotlights/causal_sets/.

[91] Stoy, J. (1981) Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory. MIT Press.

[92] Thayer, J., Herzog, J. and Guttman, J. (1998) Strand spaces: Why is a security protocol correct? In IEEE Symposium on Security and Privacy.

[93] Tonder, A. van (2004) A lambda calculus for quantum computation. SIAM Journal on Computing, 33(5), 1109–1135.

[94] Turner, D. and Winskel, G. (2009) Nominal domain theory for concurrency. Submitted.

[95] Varacca, D. (2003) Two Denotational Models for Probabilistic Computation. PhD University of Aarhus.

[96] Varacca, D. and Winskel, G. (2006) Distributing Probabililty over Nondeterminism. MSCS 16(1), 87–113. CUP.

[97] Varacca, D., Voelzer, H. and Winskel, G. (2006) Probabilistic event structures and domains. TTCS 358(2-3), 173-199.

[98] Walukiewicz, I. (2002) Local logics for traces. Automata, Languages and Combinatorics 7, 259-290.

[99] Winskel, G. and Nielsen, M. (1995) Models for Concurrency. Handbook of Logic and the Foundations of Computer Science, vol. 4, 1–148. OUP.

[100] Winskel, G. (1980) *Events in Computation.* PhD thesis, Univ. of Edinburgh. Available from http://www.cl.cam.ac.uk/users/gw104.

[101] Winskel, G. (1982) Event structure semantics of CCS and related languages. Proceedings of ICALP 82, LNCS 140. Springer-Verlag, Berlin. Extended version available from http://www.cl.cam.ac.uk/users/gw104.

[102] Winskel, G. (1983) A Note on Powerdomains and Modalitiy. Proceedings of FCT, LNCS 158. Springer-Verlag, Berlin.

[103] Winskel, G. (1984) A new definition of morphism on Petri Nets. Proceedings of STACS'84, LNCS 166. Springer-Verlag, Berlin.

[104] Winskel, G. (1987) Petri nets, algebras, morphisms and compositionality. Inf. and Comp. 72, 197238.

[105] Winskel, G. (1987) Event structures. Invited lectures for the Advanced Course on Petri nets, September 1986, LNCS 255. Springer-Verlag, Berlin.

[106] Winskel, G. (1999) Event structures as presheaves—two representation theorems. Proceedings of CONCUR 1999. LNCS 1664. Springer-Verlag, Berlin.

[107] Winskel, G. (2004) Linearity and nonlinearity in distributed computation. In the book 'Linear Logic in Computer Science,' CUP.

[108] Winskel, G. (2005) Relations in concurrency. Invited talk. Proceedings of LICS'05.

[109] Winskel, G. (2007) Event structures with symmetry. In Computation, Meaning and Logic: Articles dedicated to Gordon Plotkin, ed. Cardelli, L., Fiore, M. and Winskel, G., ENTCS 172. See `http://www.cl.cam.ac.uk/users/gw104` for corrections.

[110] Winskel, G. (2007) Symmetry and Concurrency. Proceedings of CALCO'07, LNCS 4624. Springer-Verlag, Berlin.

[111] Winskel, G. (2009) The symmetry of stability. In preparation.

[112] Winskel, G. (2009) Event structures with symmetry as presheaves. In preparation.

A.1. APPENDIX: PSEUDO EQUIVALENCE

Assume a category with pullbacks. A pair of maps $l, r : S \to E$ forms a *pseudo equivalence* provided it is:
Reflexive: there is a map ρ such that

$$\begin{tikzcd}
 & E \arrow[dl, "\mathrm{id}_E"'] \arrow[d, dotted, "\rho"] \arrow[dr, "\mathrm{id}_E"] & \\
E & S \arrow[l, "l"] \arrow[r, "r"'] & E
\end{tikzcd}$$

commutes;
Symmetric: there is a map σ such that

$$\begin{tikzcd}
 & S \arrow[dl, "r"'] \arrow[d, dotted, "\sigma"] \arrow[dr, "l"] & \\
E & S \arrow[l, "l"] \arrow[r, "r"'] & E
\end{tikzcd}$$

commutes;
Transitive: there is a map τ such that

$$\begin{tikzcd}
 & & P \arrow[dl] \arrow[dr] \arrow[d, dotted, "\tau"] \arrow[dd, phantom, "\lrcorner", very near start] & & \\
 & S \arrow[dl, "l"'] \arrow[dr, "r"] & S \arrow[dll, "l"] \arrow[drr, "r"] & S \arrow[dl, "l"'] \arrow[dr, "r"] & \\
E & & E & & E
\end{tikzcd}$$

commutes.

When l, r are *jointly monic* (*i.e.* for all maps $x, y : D \to S$, if $lx = ly$ and $rx = ry$, then $x = y$) they form an *equivalence relation*.