A Generalized Kahn Principle for Abstract Asynchronous Networks

Samson Abramsky

Department of Computing

Imperial College of Science, Technology and Medicine

180 Queen's Gate

London SW7 2BZ

England

Published in the Proceedings of the Symposium on

Mathematical Foundations of Programming Language Semantics,

Springer *Lecture Notes in Computer Science* 442, pp. 1–21

November 3, 1989

Abstract

Our general motivation is to answer the question: "What is a model of concurrent computation?". As a preliminary exercise, we study dataflow networks. We develop a very general notion of model for asynchronous networks. The "Kahn Principle", which states that a network built from functional nodes is the least fixpoint of a system of equations associated with the network, has become a benchmark for the formal study of dataflow networks. We formulate a generalized version of the Kahn Principle, which applies to a large class of non-deterministic systems, in the setting of abstract asynchronous networks; and prove that the Kahn Principle holds under certain natural assumptions on the model. We also show that a class of models, which represent networks that compute over arbitrary event structures, generalizing dataflow networks which compute over streams, satisfy these assumptions.

1 Introduction

There are by now a proliferation of mathematical structures which have been proposed to model concurrent systems. These include synchronization trees [Win85], event structures [Win86], Petri nets [Rei85], failure sets [Hoa85], trace monoids [Maz89], pomsets [Pra82] and many others. One is then led to ask: what general structural conditions should a model of concurrency satisfy? There is an obvious analogy with the λ-calculus, where a consensus on the appropriate notions of model only emerged some time after a number of particular model constructions had been discovered (*cf.* [Bar84]). Indeed, we would like to pose the question:

"What is a model of concurrent computation?"

in the same spirit as the title of Meyer's excellent paper [Mey82].

One important disanalogy with the λ-calculus is that the field of concurrent computation so far lacks a canonical syntax; and at a deeper level, there is as yet no analogue of Church's thesis for concurrent computation. The various formalisms which have been proposed actually draw inspiration from a highly varied phenomenology: synchronous, asynchronous, real-time, dataflow, shared-memory, declarative, object-oriented, systolic, SIMD, neural nets, etc. etc. In these circumstances, some more modest and circumscribed attempts at synthesis seem justified. At the same time, merely finding general definitions which subsume a number of concrete models is not enough; good definitions should show their cutting edge by yielding some non-trivial results.

In the present study, we start from a particular class of concurrent systems, the *non-deterministic dataflow networks* [Par82]. A problem which has established itself as a benchmark for the formal study of such systems is the *Kahn Principle* [Kah74], which states that if a network is composed of functional nodes, its behaviour is captured by the least fixpoint of a system of equations associated with the network in a natural way.

We attempt to formulate a notion of model for such networks in the most general and abstract form which still allows us to prove the Kahn Principle. In this way, we hope both to shed light on the initial motivating question of the axiomatics of process semantics, and to expose the essence of the Kahn Principle. In the course of doing so, we shall attain a level of generality, both as regards the notion of asynchronous network we consider, and the statement of the Kahn Principle, far in excess of anything we have seen in the literature.

The structure of the remainder of the paper is as follows. In section 2, we review some background on domain theory and dataflow networks. Then in section 3 we introduce our general notion of model, state a generalized version of the Kahn Principle, and prove that certain conditions on models are sufficient to imply the Kahn Principle. As far as I know, these are the first results of this form, as opposed to proofs of the Kahn Principle for specific models. Some directions for further research are given in section 4.

2 Background

We begin with a review of some notions in Domain theory; see e.g. [GS89] for further information and motivation.

We write $\mathsf{Fin}(X)$ for the set of finite subsets of a set X; and $A \subseteq^{\mathsf{f}} X$ for the assertion that A is a finite subset of X. A *poset* is a structure $(P, \leqslant)$, where P is a set, and $\leqslant$ a reflexive, transitive, anti-symmetric relation on P. Let $(P, \leqslant)$ be a poset. We write $\downarrow x = \{y \in P \mid y \leqslant x\}$, $\uparrow x = \{y \in P \mid y \geqslant x\}$ for $x \in P$; and $\downarrow X = \bigcup_{x \in X} \downarrow x$, $\uparrow X = \bigcap_{x \in X} \uparrow x$ for $X \subseteq P$. A subset $S \subseteq P$ is *directed* if every finite subset of S has an upper bound in S. A poset is *directed-complete* if every directed subset S has a least upper bound, written $\bigsqcup S$. A cpo (complete partial order) is a directed-complete poset with a least element, written $\perp$. An element $b \in D$ of a cpo $(D, \sqsubseteq)$ is *compact* if whenever $S \subseteq D$ is directed, and $b \sqsubseteq \bigsqcup S$, then $b \sqsubseteq d$ for some $d \in S$. We write $K(D)$ for the set of compact elements of D, and $K(d) = \downarrow d \cap K(D)$ for $d \in D$. A cpo D is *algebraic* if for all $d \in D$, $K(d)$ is directed, and $d = \bigsqcup K(d)$; and ω-algebraic if in addition $K(D)$ is countable. An *ideal* over a poset P is a directed subset $I \subseteq P$ such that $x \leqslant y \in I \Rightarrow x \in I$. The *ideal completion* of a poset P is the set of ideals over P, ordered by inclusion. If P has a least element, this is an algebraic cpo; it is ω-algebraic if P is countable.

A map $f : D \to E$ of cpo's is *continuous* if for every directed subset $S \subseteq D$, $f(\bigsqcup S) = \bigsqcup f(S)$; and *strict* if $f(\perp_D) = \perp_E$. A subset $U \subseteq D$ of a cpo D is *Scott-open* if $U = \uparrow U$, and whenever $\bigsqcup S \in U$ for a directed subset $S \subseteq D$, then $S \cap U \neq \varnothing$. The Scott-open subsets form a topology on D; a function between cpo's is continuous as defined above iff it is continuous in the topological sense with respect to the Scott topology. The Scott-open subsets of an algebraic cpo D are those of the form $\bigcup_{i \in I} \uparrow b_i$, where $b_i \in K(D)$ for all $i \in I$.

We define some standard constructions on cpo's. Given a set X, the algebraic cpo of *streams* over X, $\mathsf{Str}(X)$, is the set of finite and infinite sequences over X, with the prefix ordering. If D, E are cpo's, $[D \to E]$ is the cpo of continuous functions from D to E, with the pointwise ordering; if $\{D_i\}_{i \in I}$ is a family of cpo's, $\prod_{i \in I} D_i$ is the cartesian product cpo, with the componentwise ordering. If $f : D \to D$ is a continuous map on a cpo D, it has a least fixed point, defined by

$$\mathsf{lfp}(f) = \bigsqcup_{k \in \omega} f^k(\perp).$$

We shall assume some small knowledge of category theory in the sequel; suitable references are [ML71, AM75]. We write **Cpo** for the category of cpo's and continuous maps, $\mathbf{Cpo}^s$ for the subcategory of strict continuous maps; and $\omega\mathbf{Alg}$, $\omega\mathbf{Alg}^s$ for the corresponding categories of ω-algebraic cpo's.

We define the *weak covering relation* on a poset $(P, \leqslant)$ by:

$$x \preceq y \quad \stackrel{\text{def}}{\iff} \quad x \leqslant y \;\&\; \forall z.\, (x \leqslant z \leqslant y \;\Rightarrow\; (x = z \text{ or } y = z))$$

and the *covering relation* by

$$x \prec y \quad \stackrel{\text{def}}{\iff} \quad x \preceq y \;\&\; x \neq y.$$

The computational intuition behind the covering relation as used in Domain theory is that it represents an atomic computation step, or the occurrence of an atomic event; this idea can be traced back to [KP78].

A *covering sequence* in an algebraic cpo D is a non-empty finite or infinite sequence of compact elements (b_n), such that $b_0 = \perp$, and $b_n \prec b_{n+1}$ for all terms b_n, b_{n+1} in the sequence.

A covering sequence can be taken as a representation of $d = \bigsqcup b_n$, which gives a step-by-step description of how it was computed.

Given an algebraic cpo D, we can form the algebraic cpo $\mathcal{C}(D)$ of covering sequences over D, with the prefix ordering. There is a continuous map $\mu : \mathcal{C}(D) \to D$, with $\mu((b_n)) = \bigsqcup b_n$.

Finally, we define the *relative covering relation* in D by:

$$[b, c] \sqsubseteq d \stackrel{\text{def}}{\iff} b, c \in K(d) \ \&\ b \prec c.$$

We can think of $b \prec c$ as an atomic step at some finite stage in the computation of d.

A *prime event structure* [Win86] is a structure $\mathcal{E} = (E, \leqslant, \mathsf{Con})$, where $(E, \leqslant)$ is a countable poset, and $\mathsf{Con} \subseteq \mathsf{Fin}(E)$ a family of finite subsets of E, satisfying:

- $\forall e \in E. (\downarrow e \text{ is finite})$.
- $\forall e \in E. (\{e\} \in \mathsf{Con})$.
- $A \subseteq B \in \mathsf{Con} \Rightarrow A \in \mathsf{Con}$.
- $A \in \mathsf{Con} \Rightarrow \downarrow A \in \mathsf{Con}$.

We refer to elements of E as *events*, to $\leqslant$ as the *causality* or *enabling* relation, and to Con as the *consistency* predicate. A *configuration* of $\mathcal{E}$ is a set $x \subseteq E$ such that

- $e \leqslant e' \in x \Rightarrow e \in x$
- $A \subseteq^{\mathsf{f}} x \Rightarrow A \in \mathsf{Con}$.

The set $|\mathcal{E}|$ of configurations of $\mathcal{E}$, ordered by inclusion, is an algebraic cpo; the compact elements are the finite configurations. Note that in $|\mathcal{E}|$, $x \prec y$ iff $y \setminus x = \{e\}$ for some $e \in E$; and that if $x \sqsubseteq y$ for compact elements x, y, there is a sequence $e_1, \ldots, e_n$ such that $x = z_0 \prec \cdots \prec z_n = y$, where $z_i = x \cup \{e_1, \ldots, e_i\}$. The algebraic cpo's which arise from prime event structures are characterized in [Win86]; we refer to them as *event domains*. They form quite an extensive class, containing models of type-free and polymorphic lambda calculi (using stable functions), as well as the usual datatypes of functional programming [CGW87].

We now turn to the dataflow model of concurrent computation. Consider a process network, represented by a directed (multi)graph $G = (N, A, s, t)$, where N is the set of nodes, A the set of arcs, and $s, t : A \to N$ are the source and target functions. Each node is labelled with a sequential process, while each arc corresponds to a buffered message channel, which behaves like an unbounded FIFO queue. In addition to the usual sequential constructs, each node n can read from its input channels (those α with $t(\alpha) = n$), and write to its output channels (those α with $s(\alpha) = n$). Although this computational model might be criticised as unrealistic because of the unbounded buffering, this very feature enables a high degree of parallelism, and the model is appealingly simple, and quite close to a number of actually proposed and implemented dataflow languages and architectures [WA85, KLP79, KM77, GGKW84]. Kahn's brilliant insight in his seminal paper [Kah74] was that the behaviour of such networks could be captured denotationally in a very simple and elegant fashion, using some elementary domain theory. The key idea is to model the behaviour of each message channel α, on which atomic values from the set D_α can be transmitted, as a stream from the domain $\mathsf{Str}(D_\alpha)$. Using standard denotational techniques, the behaviour of the process at node n, with input channels $\alpha_1, \ldots, \alpha_k$, and output channnels $\beta_1, \ldots, \beta_l$, can be modelled by a continuous function

$$f : \mathsf{Str}(D_{\alpha_1}) \times \cdots \times \mathsf{Str}(D_{\alpha_k}) \to \mathsf{Str}(D_{\beta_1}) \times \cdots \times \mathsf{Str}(D_{\beta_l}).$$

The behaviour of the whole system can be modelled by setting up a system of equations, one for each channel in the network, of the overall form

$$\mathbf{X} = G(\mathbf{X}),$$

where $G : \prod_\alpha \mathsf{Str}(D_\alpha) \to \prod_\alpha \mathsf{Str}(D_\alpha)$; and solving by taking the least fixed point $\mathsf{lfp}(G) \in \prod_\alpha \mathsf{Str}(D_\alpha)$.

It is worth noting that Kahn never *proved* the coincidence of this denotational semantics with an operational semantics based directly on the computational model sketched above; indeed, he never defined any formal operational semantics for dataflow networks. Nevertheless, no-one has ever seriously doubted the accuracy of his semantics. A number of subsequent attempts have been made to fill this gap in the theory [Fau82, LS88]; it has proved surprisingly difficult to give a clean and elegant account.

In another direction, many attempts have been made to overcome one crucial limitation built into Kahn's framework; namely, the assumption that all processes in the network are deterministic, and hence their behaviour can be described by functions. This limitation must be overcome in order for these networks to be sufficiently expressive to model general-purpose concurrent systems (see e.g. [Hen82, Abr84]). However, as soon as non-deterministic processes are allowed, the denotational description of dataflow networks becomes much more complicated. In fact, naive attempts to extend Kahn's model have been shown to be doomed to failure by certain "anomalies" which were found by Keller [Kel78] and Brock and Ackerman [BA81]. In particular, Brock and Ackerman exhibited a pair of deterministic processes N_1, N_2 with the same Kahn semantics, and a non-deterministic context $C[\cdot]$ such that $C[N_1] \neq C[N_2]$ with respect to the intended operational semantics. The main point of this is to show that in the presence of non-determinism, the behaviour of a system is no longer adequately modelled by a "history tuple" $d \in \prod_\alpha \mathsf{Str}(D_\alpha)$. Such a tuple records the order in which values are realized on each channel, but fails to record causality relations which may exist between items of data on *different* channels. A number of more detailed models have been proposed which reflect this kind of information. Two in particular have received some attention.

Definition 2.1 *Let S be a set of channel names, where for each $\alpha \in S$, there is a set D_α of atomic data which can be transmitted over α. The domain of* linear traces *over S, LTr_S, is the stream domain $\mathsf{Str}(E_S)$, where*

$$E_S = \{(\alpha, d) \mid \alpha \in S, d \in D_\alpha\}.$$

The idea is that a linear trace represents a sequential observer's view of a computation in the network, as a sequence of atomic events (α, d)—namely, the production of the atomic value d on the channel α. We can regard linear traces as more detailed—perhaps even *over-specified*—representations of history tuples; indeed, there is an obvious "result" or "output" map $\mu_S : \mathsf{LTr}_S \to \prod_{\alpha \in S} \mathsf{Str}(D_\alpha)$. It is a useful exercise to verify that this is strict and continuous.

Given $S \supseteq T$, we can define a (strict, continuous) *restriction map*, $\rho^S_T : \mathsf{LTr}_S \to \mathsf{LTr}_T$, where $\rho^S_T(s)$ is obtained by deleting all (α, d) from s such that $\alpha \notin T$.

In the linear trace model, a process is modelled by a pair (S, P), where S is the set of channels incident to the process, and $P \subseteq \mathsf{LTr}_S$ describes its (possibly non-deterministic) behaviour. The key definition is that of the operation of *network composition*, which glues together a family of processes along their coincident channels. Let $\{(S_j, P_j)\}_{j \in J}$ be a family of processes; we define $\|_{j \in J}\ (S_j, P_j) = (S, P)$, where

$$\begin{array}{rcl} S & = & \bigcup_{j \in J} S_j \\ P & = & \{s \in \mathsf{LTr}_S \mid \forall j \in J.\,(\rho^S_{S_j}(s) \in P_j)\}. \end{array}$$

Note that this definition of the behaviour of a net is quite different in form to the Kahn semantics; we have replaced continuous functions by sets of traces, and the iterative construction of a least fixed point by a product-like construction. It thus becomes a matter of some importance to see if this definition actually *coincides* with the Kahn semantics in the case when each node in the network is in fact computing some continuous function. (Of course, we must firstly define what that means in terms of sets of traces). We refer to this task as the proof of the *Kahn Principle* for the linear trace model.

The linear trace model has recently been proved to be *fully abstract* in a certain sense [Jon89]; however, some other models have also received considerable attention, and avoid the apparent over-specification of linear traces. In particular there are the *pomset* models [Pra82], which were inspired by Brock and Ackerman's *scenarios* [BA81]. The idea is to allow partial orders of events, rather than insisting on purely sequential observations.

Definition 2.2 *The domain of partially ordered traces* PTr_S *is the ideal completion of the finite partially-ordered traces with the prefix ordering, where:*

- *A finite partially-ordered trace is an isomorphism type of finite labelled partial orders* $(V, \leqslant, \ell)$*, where* $\ell : V \to E_S$*, and for each* $\alpha \in S$*, the subposet*

  $$\{v \in V \mid \exists d \in D_\alpha(\ell(v) = (\alpha, d))\}$$

  is linearly ordered.

- *The prefix ordering is defined on representatives by:*

$$\begin{array}{rcl} (V, \leqslant, \ell) \sqsubseteq (V', \leqslant', \ell') & \stackrel{\text{def}}{\iff} & V \subseteq V' \;\&\; \leqslant \,=\, \leqslant' \cap V^2 \;\&\; \ell = \ell' \restriction V \\ & & \&\; v \leqslant' v' \in V \;\Rightarrow\; v \in V. \end{array}$$

Note that, if we identify sequences with isomorphism types of labelled *linear* orders, we have the inclusion $\mathsf{LTr}_S \subseteq \mathsf{PTr}_S$. Once again, there is an evident definition of a restriction map $\rho^S_T : \mathsf{PTr}_S \to \mathsf{PTr}_T$ for $S \supseteq T$, and, by virtue of the stipulation that events at each channel are linearly ordered, a map $\mu_S : \mathsf{PTr}_S \to \prod_{\alpha \in S} \mathsf{Str}(D_\alpha)$.

We can then define the notion of network composition in the partially ordered trace model in *exactly the same way* as we did for the linear traces, modulo the different notions of "trace" and "restriction"; and formulate the Kahn Principle in exactly the same terms. The main previous work on proving the Kahn Principle for (essentially) the partially ordered trace model is described in [GP87].

Our aim is firstly to extract the essential properties of this situation to arrive at a general notion of model, and then to prove the Kahn principle in this general setting. Apart from yielding the particular results for the linear and partially-ordered trace models for dataflow networks as instances of our general result, there are a number of other insights that we hope this work provides:

- The abstract networks we consider compute over a much broader class of domains than just the stream domains of dataflow—our results apply at least to the event domains.
- The version of the Kahn Principle we formulate and prove in fact applies not only to the deterministic case, but to a broad class of *non-deterministic* networks—namely those in which each node computes one of a *set* of possible continuous functions. This includes for example the so-called "infinity-fair merge", though not the "angelic merge" [PS88]. As far as I know, this major extension to the Kahn Principle is new, even for the specific models described above.
- Although our notion of model is abstracted from the dataflow family, and cannot be claimed to be fully general, we hope it is a useful step along the way to answering the question raised in the opening paragraph, namely: "what is a model of concurrent computation?".

3 Results

3.1 Models

We assume a class Chan of *channel names*, ranged over by α, β, γ. We refer to sets of channels as *sorts*; the class of sorts, partially ordered by inclusion, is denoted by **Sort**. We use S, T, U to range over sorts.

Definition 3.1 *A model $\mathcal{M} = (\mathcal{T}, \mathcal{V}, \mu)$ comprises:*

- *functors* $\mathcal{T}, \mathcal{V} : \mathbf{Sort}^{\mathsf{op}} \to \mathbf{Cpo}^s$
- *a natural transformation* $\mu : \mathcal{T} \dot{\to} \mathcal{V}$

such that $\mathcal{V}$ preserves limits.

We refer to $\mathcal{T}_S$ as the *traces* of sort S, $\mathcal{V}_S$ as the *values* of sort S, and μ as the *output* or *evaluation* map.

More explicitly, $\mathcal{T}$ assigns to each sort S a cpo $\mathcal{T}_S$, and to each $S \supseteq T$ a strict, continuous *restriction map* $\rho^S_T : \mathcal{T}_S \to \mathcal{T}_T$, such that:

- $S \supseteq T \supseteq U \;\Rightarrow\; \rho^T_U \circ \rho^S_T = \rho^S_U$
- $\rho^S_S = \mathsf{id}_{\mathcal{T}_S}$.

Similarly, $\mathcal{V}$ assigns a cpo $\mathcal{V}_S$ to each sort S. The requirement that $\mathcal{V}$ preserves limits amounts to asking that $\mathcal{V}$ takes *unions* in **Sort** to *products* in $\mathbf{Cpo}^s$. Since each sort is the union of its singletons, this means that if $\mathcal{V}_\alpha$ is the value domain of sort $\{\alpha\}$,

$$\mathcal{V}_S = \prod_{\alpha \in S} \mathcal{V}_\alpha;$$

and that the restriction maps will be the projections onto sub-products: for $S \supseteq T$, $\pi^S_T : \mathcal{V}_S \to \mathcal{V}_T$. Thus $\mathcal{V}$ is completely determined by the $\mathcal{V}_\alpha$.

Finally, for each sort S there is a strict, continuous map $\mu_S : \mathcal{T}_S \to \mathcal{V}_S$, such that for all $S \supseteq T$,

$$\mu_T \circ \rho^S_T = \pi^S_T \circ \mu_S.$$

Notation. We write $\nu^S_T = \mu_T \circ \rho^S_T = \pi^S_T \circ \mu_S$.

Examples

(1). Firstly, from the discussion in the previous Section, it is easy to see that both linear and partially-ordered traces yield examples of models. More precisely, for each channel α fix a set D_α of atomic values; then define $\mathcal{V}_\alpha = \mathsf{Str}(D_\alpha)$, and $\mathcal{T}_S = \mathsf{PTr}_S$ (LTr_S), ρ^S_T, μ_S as in Section 2. The verification of the required functoriality and naturality conditions is straightforward.

(2). We now describe a general class of models. For each channel α, fix an event structure $\mathcal{E}_\alpha = (E_\alpha, \leqslant_\alpha, \mathsf{Con}_\alpha)$. Define $\mathcal{V}_\alpha = |\mathcal{E}_\alpha|$, the domain of configurations over $\mathcal{E}_\alpha$. For a sort S, we define $\mathcal{E}_S = \prod_{\alpha \in S} \mathcal{E}_\alpha$, where the product of event structures is defined as their *disjoint union* [Win86]: $\mathcal{E}_S = (E_S, \leqslant_S, \mathsf{Con}_S)$, where

$$\begin{array}{rcl} E_S & \stackrel{\text{def}}{=} & \{(\alpha, e) \mid \alpha \in S, e \in E_\alpha\} \\ (\alpha, e) \leqslant_S (\beta, e') & \stackrel{\text{def}}{\Longleftrightarrow} & \alpha = \beta \;\&\; e \leqslant_\alpha e' \\ A \in \mathsf{Con}_S & \stackrel{\text{def}}{\Longleftrightarrow} & \forall \alpha \in S.\, (\{e \mid (\alpha, e) \in A\} \in \mathsf{Con}_\alpha). \end{array}$$

We have [Win86]: $|\mathcal{E}_S| \cong \prod_{\alpha \in S} |\mathcal{E}_\alpha|$, and we shall take $\mathcal{V}_S = |\mathcal{E}_S|$. For $S \supseteq T$, the projections $\pi^S_T : |\mathcal{E}_S| \to |\mathcal{E}_T|$ are defined by $\pi^S_T(x) = x \cap E_T$.

In order to define the traces over $\mathcal{E}_S$, we follow the idea that

$$\text{traces} = \text{data} + \text{causality}.$$

Thus a trace is a configuration together with extra information about the order in which data was actually produced in a particular computation, reflecting some causal constraints.

Definition 3.2 *A trace over an event structure $\mathcal{E} = (E, \leqslant, \mathsf{Con})$ is a pair $t = (x_t, \leqslant_t)$, where $x_t \in |\mathcal{E}|$, and $\leqslant_t$ is a partial order on x_t such that:*

- $\forall e \in x_t. (\{e' \in x_t \mid e' \leqslant_t e\}$ *is finite*)
- $(\leqslant \cap x_t^2) \subseteq \leqslant_t$.

Traces are partially ordered as follows:

$$t \sqsubseteq t' \overset{\text{def}}{\iff} x_t \subseteq x_{t'} \;\&\; \leqslant_t = \leqslant_{t'} \cap x_t^2 \;\&\; (e \leqslant_{t'} e' \in x_t \;\Rightarrow\; e \in x_t).$$

Clearly, traces with this ordering form an algebraic cpo $\mathbb{P}\mathcal{E}$. A trace t is *linear* if $\leqslant_t$ is a linear order; the linear traces also form an algebraic cpo, $\mathbb{L}\mathcal{E}$, and $\mathbb{L}\mathcal{E} \subseteq \mathbb{P}\mathcal{E}$. The compact elements of $\mathbb{P}\mathcal{E}$ are those t for which x_t is a finite configuration of $|\mathcal{E}|$. Also, $t \prec u$ in $\mathbb{P}\mathcal{E}$ iff $x_u \setminus x_t = \{e\}$ for some e which is *maximal* in $\leqslant_u$. The following construction on trace domains will be useful. Given $t \in \mathbb{P}\mathcal{E}$, and $X \subseteq x_t$, we define $t \lceil X$ by:

$$\begin{aligned} x_{t\lceil X} &= \{e \in x_t \mid \exists e' \in X. e \leqslant_t e'\} \\ \leqslant_{t\lceil X} &= \leqslant_t \cap (x_{t\lceil X})^2. \end{aligned}$$

Clearly $t \lceil X$ is a well-defined trace, and $t \lceil X \sqsubseteq t$; moreover, $X \subseteq Y \;\Rightarrow\; t \lceil x \sqsubseteq t \lceil Y$. This construction can also be applied to $\mathbb{L}\mathcal{E}$.

We can now complete the definitions for our two families of models, $\mathcal{M}_{\mathbb{P}}$ (partially ordered traces over event structures) and $\mathcal{M}_{\mathbb{L}}$ (the sub-model of linearly ordered traces). The trace domains for $\mathcal{M}_{\mathbb{P}}$ are defined by $\mathcal{T}_S = \mathbb{P}\mathcal{E}_S$, and for $\mathcal{M}_{\mathbb{L}}$ by $\mathcal{T}_S = \mathbb{L}\mathcal{E}_S$. The evaluation maps are defined for both by

$$\mu_S(t) = x_t,$$

and the restriction maps by

$$\rho_T^S(t) = (x_t \cap E_T, \leqslant_t \cap E_T^2),$$

for $S \supseteq T$.

The verification that these definitions yield models is straightforward. Note that $\mathcal{M}_{\mathbb{P}}$ and $\mathcal{M}_{\mathbb{L}}$ are really *families* of models, parameterized by the choice of event structures $\mathcal{E}_\alpha$ for each α. Our results will apply to *all* models in these families.

We now show how the concrete dataflow models of (1) are special cases of $\mathcal{M}_{\mathbb{P}}$ and $\mathcal{M}_{\mathbb{L}}$. Fix a set D_α for each channel α, and define an event structure $\mathcal{E}_\alpha$ as follows:

- $E_\alpha = \{(s, sd) \mid s \in D_\alpha^\star, d \in D_\alpha\}$.
- $(s, sd) \leqslant_\alpha (s', s'd') \overset{\text{def}}{\iff} sd \sqsubseteq s'd'$.

- $A \in \mathsf{Con} \stackrel{\text{def}}{\iff} \forall (s, sd), (s', s'd') \in A.(sd \sqsubseteq s'd' \text{ or } s'd' \sqsubseteq sd)$.

It can easily be verified that $|\mathcal{E}_\alpha| \cong \mathsf{Str}(D_\alpha)$. Also, we have

Proposition 3.3 *For all sorts S,*

$$\begin{aligned} \mathsf{PTr}_S &\cong \mathbb{P}\mathcal{E}_S \\ \mathsf{LTr}_S &\cong \mathbb{L}\mathcal{E}_S. \end{aligned}$$

Proof. Given $t \in K(\mathbb{P}\mathcal{E}_S)$, we define a labelled poset $(x_t, \leqslant_t, \ell)$, where

$$\ell((\alpha, (s, sd))) = (\alpha, d).$$

This defines a map $\phi : K(\mathbb{P}\mathcal{E}_S) \to K(\mathsf{PTr}_S)$. (Note that the condition $(\leqslant_S \cap x_t) \subseteq \leqslant_t$ is needed to ensure that α-events are linearly ordered in $\phi(t)$ for each $\alpha \in S$). Now consider a trace in $K(\mathsf{PTr}_S)$ with representative labelled poset $(V, \leqslant, \ell)$. For each $v \in V$, let $\ell(v) = (\alpha, d)$. The set of α-labelled predecessors of v is linearly ordered, say

$$v_1 < \cdots < v_n < v,$$

and hence yields a finite sequence $s = d_1 \cdots d_n \in K(\mathsf{Str}(D_\alpha))$, where $d_i = \mathsf{snd} \circ \ell(v_i)$, $i = 1, \ldots, n$. We can thus define a new labelling function ℓ', which maps v to $(\alpha, (s, sd)) \in E_S$. Note that ℓ' is *injective*, and hence we can dispense with V, and take the induced order on $\ell'(V)$: $\ell'(v) \leqslant' \ell'(v') \stackrel{\text{def}}{\iff} v \leqslant v'$, yielding a trace $(\ell'(V), \leqslant')$ in $\mathbb{P}\mathcal{E}_S$. Thus we obtain a map $\psi : K(\mathsf{PTr}_S) \to K(\mathbb{P}\mathcal{E}_S)$. It is easily checked that ϕ and ψ are monotone and mutually inverse, yielding an order-isomorphism $K(\mathbb{P}\mathcal{E}_S) \cong K(\mathsf{PTr}_S)$, and hence by algebraicity, $\mathbb{P}\mathcal{E}_S \cong \mathsf{PTr}_S$. Finally, ϕ, ψ cut down to an isomorphism $K(\mathbb{L}\mathcal{E}_S) \cong K(\mathsf{LTr}_S)$, and so $\mathbb{L}\mathcal{E}_S \cong \mathsf{LTr}_S$. ∎

One further connection will be useful: the linear traces over an event structure are isomorphic to the covering sequences over its domain of configurations.

Proposition 3.4 *For any event structure $\mathcal{E}$, $\mathbb{L}\mathcal{E} \cong \mathcal{C}(|\mathcal{E}|)$.*

Proof. From our description of covering relations in event domains, it follows that any covering sequence in $|\mathcal{E}|$ has the form

$$x_0 \prec \cdots x_n \prec \cdots$$

where $x_0 = \varnothing$, $x_{n+1} \setminus x_n = \{e_n\}$ for some $e \in E$. We can then define the linear trace t with $x_t = \bigcup x_n$, $e_n \leqslant_t e_m \Leftrightarrow n \leqslant m$. Conversely, any linear trace must, by countability of E and the well-foundedness property of traces, amount to a (finite or infinite) sequence (e_n), from which we can define a covering sequence (x_n), where $x_n = \{e_j \mid j \leqslant n\}$. The fact that each $x_n \in |\mathcal{E}|$ follows from the conditions on traces. These passages between $\mathbb{L}\mathcal{E}$ and $\mathcal{C}(\mathcal{E})$ are easily checked to be monotone and mutually inverse, establishing the required isomorphism. ∎

For the remainder of this section, we fix a model $\mathcal{M} = (\mathcal{T}, \mathcal{V}, \mu)$.

Definition 3.5 *A process in $\mathcal{M}$ is a pair (S, P), where $P \subseteq \mathcal{T}_S$. Let $\{(S_j, P_j)\}_{j\in J}$ be a family of processes. The network composition of this family is defined by:*

$$\|_{j\in J}\,(S_j, P_j) = (S, P),$$

where

$$\begin{array}{rcl} S & = & \bigcup_{j\in J} S_j \\ P & = & \{t \in \mathcal{T}_S \mid \forall j \in J.\,(\rho^S_{S_j}(t) \in P_j)\}. \end{array}$$

This definition was predictable from our discussion of concrete dataflow models in the previous section. The next definition is a key one, which answers the question of how to characterize when a process, *qua* set of traces, is computing a function. In fact, we deal with the more general situation when a process is computing any one (non-deterministically chosen) from a *set* of functions.

Definition 3.6 *Let (S, P) be a process, with $S = I \cup O$, and let $F \subseteq [\mathcal{V}_I \to \mathcal{V}_O]$ be a set of continuous functions. We say that (S, P) computes F if for all $t \in \mathcal{T}_S$:*

$$\begin{array}{l} t \in P \iff \exists f \in F : \\ (1)\;\; \nu^S_O(t) = f(\nu^S_I(t)) \\ (2)\;\; [u, v] \sqsubseteq t \;\Rightarrow\; \nu^S_O(v) \sqsubseteq f(\nu^S_I(u)). \end{array}$$

Condition (1) in this definition is the obvious stipulation that the overall effect of the trace is to compute an input-output pair in the graph of one of the functions $f \in F$. Condition (2) is more subtle; it insists that the *way* this input-output pair is computed must be "causally consistent", in the sense that for any step $u \prec v$ towards computing t, the output values realized after the step—at v—are no more than what was justified as f applied to the input values available before the step—at u.[1]

As regards the generality conferred by the use of *sets* of functions, consider the following example from dataflow [Par82]: the deterministic merge function

$$\mathsf{dmerge} : \mathsf{Str}(X) \times \mathsf{Str}(X) \times \mathsf{Str}(\{0,1\}) \to \mathsf{Str}(X)$$

which uses an oracle to guide its choices. This satisfies the equations:

$$\begin{array}{rcl} \mathsf{dmerge}(a : x, y, 0 : o) & = & a : \mathsf{dmerge}(x, y, o) \\ \mathsf{dmerge}(x, b : y, 1 : o) & = & b : \mathsf{dmerge}(x, y, o). \end{array}$$

Now for any set of oracles O we can define:

$$F = \{\lambda x, y.\mathsf{dmerge}(x, y, o) \mid o \in O\}.$$

[1]These conditions were directly inspired by Misra's "limit" and "smoothness" conditions in his notion of *descriptions* [Mis89]; his definition was made in the specific setting of the linear trace domain LTr_S, and in a rather different context.

If we take O to be the set of *fair* oracles, *i.e.* infinite binary sequences containing infinitely many zeroes and infinitely many ones, then F corresponds to the "infinity-fair merge" [PS88]; however, note that the "angelic merge" cannot be obtained in this way.

Now let $\{(S_j, P_j)\}_{j \in J}$ be a family of processes, with $(S, P) = \|_{j \in J} (S_j, P_j)$. We say that $\{(S_j, P_j)\}_{j \in J}$ is a *non-deterministic functional network* if the following conditions hold:

1. For all $j \in J$, $S_j = I_j \cup O_j$ and (S_j, P_j) computes $F_j \subseteq [\mathcal{V}_{I_j} \to \mathcal{V}_{O_j}]$.
2. For all $\alpha \in S$, there is exactly one $j \in J$ with $\alpha \in O_j$.

If F_j is a singleton for all $j \in J$, we say that the network is *deterministic*.

Condition (2) is worth some comment. The constraint that each channel has *at most* one producer precludes non-determinism by "short circuit". The requirement that there be *exactly* one producer is a technical convenience; it means that we can avoid considering input channels—*i.e.* those with no producer in the system—separately. Of course, we can still handle input channels, in a "pointwise" fashion; for each given input value, we add a process which behaves like the constant function producing that value on the channel. Indeed, in our approach this is immediately generalized to allow a *set* of values to be produced.

Now we generalize the Kahn semantics for dataflow in the obvious way. For each $f \in \prod_{j \in J} F_j$, we define $G_f : \mathcal{V}_S \to \mathcal{V}_S$ by:

$$G_f = \langle \pi_\alpha^{O_j} \circ f_j \circ \pi_{I_j}^S \rangle_{\alpha \in S, \alpha \in O_j}.$$

By virtue of condition (2) on the network, there is exactly one component of the tuple defining G_f for each $\alpha \in S$.

We say that the network satisfies the *Generalized Kahn Principle* if the following condition holds:

$$\text{(GKP)} \quad \mu_S(P) = \{\mathsf{lfp}(G_f) \mid f \in \prod_{j \in J} F_j\}.$$

We say that $\mathcal{M}$ satisfies the Generalized Kahn Principle if (GKP) holds for every non-deterministic functional network in $\mathcal{M}$. We say that $\mathcal{M}$ satisfies the (ordinary) Kahn Principle if (GKP) holds for every deterministic functional network. Note that in this case, $\prod_{j \in J} F_j$ is a singleton, and hence so is the right-hand side of (GKP).

Our main objective will be to give sufficient conditions on $\mathcal{M}$ to ensure that (GKP) holds. (GKP) states an equality between two sets; it is convenient to consider the two inclusions separately. Firstly, we have

$$\text{(GKP}_s\text{)} \quad \mu_S(P) \subseteq \{\mathsf{lfp}(G_f) \mid f \in \prod_{j \in J} F_j\}.$$

This is a *safety* property, since it asserts that every behaviour of the network computes one of the values specified by the (generalized) Kahn semantics. The converse:

$$\text{(GKP}_l\text{)} \quad \{\mathsf{lfp}(G_f) \mid f \in \prod_{j \in J} F_j\} \subseteq \mu_S(P)$$

is a *liveness* property, since it asserts that every specified value is realized by some computation.

3.2 Safety

Definition 3.7 *An ω-algebraic cpo is* incremental *if whenever $b \sqsubseteq c$ in $K(D)$, there is a finite covering sequence*

$$b = b_0 \prec \cdots \prec b_n = c.$$

A strict, continuous function $f : D \to E$ on incremental domains is an incremental morphism *if:*

- *f weakly preserves relative covers:*

$$[b, c] \sqsubseteq d \;\Rightarrow\; [f(b), f(c)] \sqsubseteq f(d) \text{ or } f(b) = f(c) \in K(d).$$

- *f lifts relative covers:*

$$[b', c'] \sqsubseteq d' = f(d) \;\Rightarrow\; \exists b, c.([b, c] \sqsubseteq d \;\&\; f(b) = b', f(c) = c').$$

Incremental domains and morphisms form a category **IncDom**. We say that a functor $F : \mathbb{C} \to \mathbf{Cpo}^s$ is incremental if it factors through the inclusion $\mathbf{IncDom} \hookrightarrow \mathbf{Cpo}^s$, and that a model $\mathcal{M} = (\mathcal{T}, \mathcal{V}, \mu)$ is incremental if $\mathcal{T}$ is.

Note that all event domains, and all ideal completions of countable posets satisfying both the ascending and descending chain conditions, are incremental. The reason for our terminology is that incremental domains are precisely the specialization to posets of the incremental categories introduced in [GJ88].

Proposition 3.8 *$\mathcal{M}_{\mathbb{P}}$ and $\mathcal{M}_{\mathbb{L}}$ are incremental.*

Proof. We have already observed that the domains $\mathbb{P}\mathcal{E}_S$, $\mathbb{L}\mathcal{E}_S$ are incremental. The fact the restriction maps weakly preserve relative covers follows easily from the definitions. We must verify the lifting property. We give the argument for $\mathcal{M}_{\mathbb{P}}$ only. Suppose then that $S \supseteq T$, $[u', v'] \sqsubseteq t'$ in $\mathbb{P}\mathcal{E}_T$, and $\rho^S_T(t) = t'$. We define $v = t \restriction x_{v'}$. Since $x_{v'} \subseteq x_{t'} \subseteq x_t$, this is well-defined, and yields $v \sqsubseteq t$. Let $w = \rho^S_T(v)$. Since $x_{v'} \subseteq x_v$, $x_{v'} \subseteq x_w$. For the converse, suppose $e \in x_w$. This implies that $e \in E_T$, and that for some $e' \in x_{v'}$, $e \leqslant_t e'$. But this implies $e \leqslant_{t'} e'$, since $\rho^S_T(t) = t'$, and hence $e \in x_{v'}$, since $v' \sqsubseteq t'$ and $e' \in x_{v'}$. Thus $x_w = x_{v'}$. The same reasoning shows that $\leqslant_w = \leqslant_{v'}$, and so $w = v'$.

To define u, recall that $u' \prec v'$ iff $x_{v'} \setminus x_{u'} = \{e\}$ for some $e \in E_T$ which is maximal in $\leqslant_{v'}$. But then e must also be maximal with respect to $\leqslant_v$, since otherwise we would have $e <_v e' \in x_{v'}$, which would imply $e <_{v'} e'$, contradicting $<_{v'}$-maximality of e. Thus if we define $x_u = x_v \setminus \{e\}$, $\leqslant_u = \leqslant_v \cap x_u^2$, we see that $v \restriction x_u = (x_u, \leqslant_u)$. Clearly $u \prec v$; and if $w = \rho^S_T(u)$,

$$x_w = x_u \cap E_T = (x_v \setminus \{e\}) \cap E_T = (x_v \cap E_T) \setminus \{e\} = x_{v'} \setminus \{e\} = x_{u'}.$$

Similarly $\leqslant_w = \leqslant_{u'}$, yielding $\rho^S_T(u) = u'$, and the proof is complete. ∎

Our main objective in the remainder of this subsection is to prove:

Theorem 3.9 *If $\mathcal{M}$ is incremental, it satisfies* (GKP$_s$).

Our strategy is to use incrementality of the restriction maps to move between local conditions expressing the functional behaviour of the nodes and global conditions expressing the functional behaviour of the whole network.

Lemma 3.10 *Let (S,P) be a non-deterministic functional process computing F, where $S = I \cup O$. For all $t \in P$ computing $f \in F$, and $u \sqsubseteq t$:*

$$\nu^S_O(u) \sqsubseteq f(\nu^S_I(u)).$$

PROOF. Suppose firstly that u is compact. Either $u = t$, in which case the conclusion follows directly from the first condition for $t \in P$, or by incrementality of $\mathcal{T}_S$, for some compact v, $[u,v] \sqsubseteq t$. Applying the second condition for $t \in P$,

$$\nu^S_O(u) \sqsubseteq \nu^S_O(v) \sqsubseteq f(\nu^S_I(u)).$$

The general result follows from this special case, since

$$\nu^S_O(u) = \bigsqcup_{v \in K(u)} \nu^S_O(v) \sqsubseteq \bigsqcup_{v \in K(u)} f(\nu^S_I(v)) = f(\nu^S_I(u)). \quad \blacksquare$$

Lemma 3.11 *Let $\{(S_j,P_j)\}_{j\in J}$ be a non-deterministic functional network computing F_j at each $j \in J$, where $S_j = I_j \cup O_j$. Let $(S,P) = \|_{j\in J} (S_j,P_j)$. Then for all $t \in \mathcal{T}_S$:*

$$\begin{array}{rcll} t \in P & \Leftrightarrow & \forall j \in J.\, \exists f_j \in F_j. & \\ & \bullet & \nu^S_{O_j}(t) = f_j(\nu^S_{I_j}(t)) & (1) \\ & \bullet & [u,v] \sqsubseteq t \;\Rightarrow\; \nu^S_{O_j}(v) \sqsubseteq f_j(\nu^S_{I_j}(u) & (2) \end{array}$$

PROOF. We shall write $t_j = \rho^S_{S_j}(t)$ for $t \in \mathcal{T}_S$. By definition of network composition,

$$\begin{array}{rcll} t \in P & \Leftrightarrow & \forall j \in J.\, t_j \in P_j & \\ & \Leftrightarrow & \forall j \in J.\, \exists f_j \in F_j. & \\ & \bullet & \nu^{S_j}_{O_j}(t) = f_j(\nu^{S_j}_{I_j}(t_j)) & (1') \\ & \bullet & [u_j,v_j] \sqsubseteq t_j \;\Rightarrow\; \nu^{S_j}_{O_j}(v) \sqsubseteq f_j(\nu^{S_j}_{I_j}(u_j) & (2') \end{array}$$

Now it suffices to show that for all $t \in \mathcal{T}_S$, $j \in J$, $f_j \in F_j$: (1) $\Longleftrightarrow$ (1′) and (2) $\Longleftrightarrow$ (2′). The equivalence of (1) and (1′) follows from the functoriality of ρ. To show that (2′) implies (2), we use the fact that ρ weakly preserves covers. Suppose $[u,v] \sqsubseteq t$. If $u_j = v_j$, we can apply Lemma 3.10 to get (2); if $u_j \prec v_j$, we can apply (2′). Finally, we show that (2) implies (2′). Suppose $[u',v'] \sqsubseteq t_j$. Since ρ lifts covers, for some $u,v \in \mathcal{T}_S$,

$$\rho^S_{S_j}(u) = u', \rho^S_{S_j}(v) = v', \;\&\; [u,v] \sqsubseteq t.$$

We can now apply (2) to get (2′), as required. $\blacksquare$

As an immediate Corollary of Lemma 3.11, we obtain:

Proposition 3.12 *With notation as in Lemma 3.11:*

$$
\begin{array}{rcll}
t \in P & \iff & \exists f \in \prod_{j \in J} F_j. & \\
& \bullet & \mu_S(t) = G_f(\mu_S(t)) & (1) \\
& \bullet & [u,v] \sqsubseteq t \;\Rightarrow\; \mu_S(v) \sqsubseteq G_f(\mu_S(u)) & (2)
\end{array}
$$

PROOF OF THEOREM 3.9. With notation as in Lemma 3.11, suppose $t \in P$. Applying Proposition 3.12 (1), for some $f \in \prod_{j \in J} F_j$, $\mu_S(t) = G_f(\mu_S(t))$, whence $\mathsf{lfp}(G_f) \sqsubseteq \mu_S(t)$. To show that $\mu_S(t) \sqsubseteq \mathsf{lfp}(G_f)$, let (t_k) be a covering sequence for t, which must exist by incrementality of $\mathcal{T}_S$; we show by induction on k that:

$$\forall k \in \omega.(\mu_S(t_k) \sqsubseteq G_f^k(\bot)).$$

The base case follows from the strictness of μ_S. For the inductive step,

$$
\begin{array}{rcll}
\mu_S(t_{k+1}) & \sqsubseteq & G_f(\mu_S(t_k)) & \text{Proposition 3.12 (2)} \\
& \sqsubseteq & G_f(G_f^k(\bot)) & \text{by induction hypothesis.} \quad \blacksquare
\end{array}
$$

3.3 Liveness

Consider an algebraic domain D, and a chain of compact elements $C = (b_k)$ in D, with $\bigsqcup b_k = d$. We can consider C as a (partial) specification of a particular way of computing d, which induces a causality relation on compact approximations of d, as follows. Define $\|\cdot\|_C : K(d) \to \omega$ by

$$\|b\|_C = \min\{k \mid b \sqsubseteq b_k\}.$$

Now we can define:

$$b <_C c \;\overset{\text{def}}{\iff}\; \|b\|_C < \|c\|_C,$$

for $b, c \in K(d)$.

Now let t be a trace in $\mathcal{T}_S$, with $\mu_S(t) = d \in \mathcal{V}_S$. We can define a relation $<_t$ on $K(d)$ which reflects the causal constraints on how d can be realized introduced by t:

$$
\begin{array}{rcl}
b <_t c & \overset{\text{def}}{\iff} & \text{for every covering sequence } (t_k) \text{ for } t: \\
& & \min\{k \mid b \sqsubseteq \mu_S(t_k)\} < \min\{k \mid c \sqsubseteq \mu_S(t_k)\}.
\end{array}
$$

Definition 3.13 *Let $\mathcal{M} = (\mathcal{T}, \mathcal{V}, \mu)$ be an incremental model in which each value domain $\mathcal{V}_S$ is ω-algebraic. $\mathcal{M}$ is* causally expressive *if for every sort S, $d \in \mathcal{V}_S$, and chain of compact elements $C = (b_k)$ with $\bigsqcup b_k = d$, there exists $t \in \mathcal{T}_S$ such that:*

- $\mu_S(t) = d$
- $<_t \supseteq <_C$.

Proposition 3.14 *$\mathcal{M}_{\mathbb{P}}$ and $\mathcal{M}_{\mathbb{L}}$ are causally expressive.*

PROOF. Since $\mathcal{M}_{\mathbb{L}}$ is a sub-model of $\mathcal{M}_{\mathbb{P}}$, it suffices to prove causal expressiveness for $\mathcal{M}_{\mathbb{L}}$. Suppose then that a compact chain $C = (b_n)$ in $\mathcal{E}_S$ is given, with $\bigsqcup b_n = d$. Since $\mathcal{E}_S$ is incremental, C can be refined into a covering sequence C'; clearly $<_{C'} \supseteq <_C$. Now let t be the trace in $\mathbb{L}\mathcal{E}_S$ corresponding to C' under the isomorphism of Proposition 3.4. We note the general fact that for *any* algebraic cpo D, and covering sequence (c_n) in D, there is a *unique* covering sequence for (c_n) in $\mathcal{C}(D)$; a consequence of this is that $\mathcal{C}(\mathcal{C}(D)) \cong \mathcal{C}(D)$. If follows that $<_t = <_{C'} \supseteq <_C$, as required. ∎

We shall need a technical lemma about fixpoints in ω-algebraic cpo's. This was conjectured by the author, and proved under the hypothesis that the domain is SFP. The ingenious proof of the general result is due to Achim Jung (personal communication); it is reproduced here with his kind permission.

Lemma 3.15 (Jung) *Let D be an ω-algebraic cpo, and $f : D \to D$ a continuous function. There exists a chain (b_n) of compact elements in D such that:*

1. $b_0 = \bot$
2. $\forall n.\, b_{n+1} \sqsubseteq f(b_n)$
3. $\bigsqcup b_n = \mathsf{lfp}(f)$.

PROOF. For each $f^n(\bot)$ we choose a chain of compact elements (c^n_m) with least upper bound $f^n(\bot)$. By taking a diagonal sequence we find a chain (c_n) with the property $c^{n'}_{m'} \sqsubseteq c_n \sqsubseteq f^n(\bot)$ for all $n', m' \leqslant n$. The least upper bound of this chain is equal to $\mathsf{lfp}(f)$. Let $C_n = \uparrow c_n$.

We shall define the required sequence (b_n) inductively, to satisfy the following properties:

1. $b_n \sqsubseteq f(b_{n-1}), \quad n \geqslant 1$
2. $b_n \sqsubseteq f^n(\bot), \quad n \geqslant 0$
3. $b_n \in O_n = \bigcap_{m\in\omega, 0\leqslant 2m\leqslant n} g^{-n+2m}(C_{n-m}), \quad n \geqslant 0$.

For $n = 2k$, the last property implies in particular that $b_n \in C_k$, and together with (2) this ensures that the limit of the b_n is the least fixed point of f.

Let $b_0 = \bot$. Then (2) is obviously satisfied, and (3) evaluates to

$$O_0 = f^0(C_0) = C_0 = \uparrow c_0 = \uparrow\bot = D,$$

and is satisfied too.

Given $b_0, \ldots, b_n$ we find b_{n+1} as follows. First note that $b_n \sqsubseteq f(b_{n-1}) \sqsubseteq f(b_n)$ by (1) (for $n = 0$ this is trivially satisfied); and that $f(b_n) \sqsubseteq f^{n+1}(\bot)$ by (2). We shall select b_{n+1} below

$f(b_n)$ and above b_n, so (1) and (2) will be satisfied. As for (3), we calculate:

$$\begin{aligned}
& b_n \in O_n \;\Rightarrow\; f(b_n) \in f(O_n) \\
& \subseteq \textstyle\bigcap_{0 \leqslant 2m \leqslant n} f^{-n+2m+1}(C_{n-m}) \\
& = \textstyle\bigcap_{2 \leqslant 2m+2 \leqslant n+2} f^{-n-1+(2m+2)}(C_{n+1-(m+1)}) \\
& = \textstyle\bigcap_{2 \leqslant 2m' \leqslant n+2} f^{-n-1+2m'}(C_{n+1-m'}) \\
& \subseteq \textstyle\bigcap_{2 \leqslant 2m' \leqslant n+1} f^{-n-1+2m'}(C_{n+1-m'}).
\end{aligned}$$

Note that $f^{n+1}(\bot)$ is contained in C_{n+1}, so we have

$$\bot \in f^{-n-1}(f^{n+1}(\bot)) \subseteq f^{-n-1}(C_{n+1}),$$

which tells us that $f^{-n-1}(C_{n+1}) = D$. So

$$f(b_n) \in \bigcap_{0 \leqslant 2m' \leqslant n+1} f^{-n-1+2m'}(C_{n+1-m'}) = O_{n+1}.$$

Since O_{n+1} is Scott-open, it contains a compact element below $f(b_n)$; let b_{n+1} be such an element above b_n. ∎

Theorem 3.16 *If $\mathcal{M}$ is causally expressive, it satisfies* (GKP$_l$).

Proof. We adopt the same notation as in Lemma 3.11. Suppose $f \in \prod_{j \in J} F_j$. We must show that for some $t \in P$, $\mu_S(t) = \mathsf{lfp}(G_f)$. We apply Lemma 3.15 to obtain a chain of compact elements $C = (b_k)$ with $\bigsqcup b_k = \mathsf{lfp}(G_f)$, $b_0 = \bot$, and $b_{k+1} \sqsubseteq G_f(b_k)$ for all k. Since $\mathcal{M}$ is causally expressive, for some $t \in \mathcal{T}_S$, $\mu_S(t) = \bigsqcup b_k = \mathsf{lfp}(G_f)$, and $<_t \supseteq <_C$. It remains to show that $t \in P$. By Proposition 3.12, it suffices to show that for all $[u, v] \sqsubseteq t$, $\mu_S(v) \sqsubseteq G_f(\mu_S(u))$, which in turn is equivalent to:

$$\forall b \in K(\mathcal{V}_S).\, (b \sqsubseteq \mu_S(v) \;\Rightarrow\; b \sqsubseteq G_f(\mu_S(u))).$$

Suppose then that $b \sqsubseteq \mu_S(v) \sqsubseteq \mu_S(t) = \mathsf{lfp}(G_f)$. Since b is compact, $b \sqsubseteq b_k$ for some k. If $b = \bot$ we are done; otherwise, for some k, $b \sqsubseteq b_{k+1}$, $b \not\sqsubseteq b_k$. This implies $b_k <_C b$, and hence $b_k <_t b$. By incrementality of $\mathcal{T}_S$, we can find a covering sequence (t_k) for t with $u = t_n$, $v = t_{n+1}$ for some n. But since $b \sqsubseteq \mu_S(v)$ and $b_k <_t b$, this implies $b_k \sqsubseteq \mu_S(u)$, and hence

$$b \sqsubseteq b_{k+1} \sqsubseteq G_f(b_k) \sqsubseteq G_f(\mu_S(u)),$$

as required. ∎

As an immediate Corollary of Propositions 3.8 and 3.14 and Theorems 3.9 and 3.16, we obtain:

Theorem 3.17 *$\mathcal{M}_{\mathbb{P}}$ and $\mathcal{M}_{\mathbb{L}}$ satisfy (GKP).*

4 Concluding Remarks

The results in this paper are of a preliminary nature. Even within the asynchronous network model, there are a number of interesting topics for further investigation. These include the characterisation of models in terms of properties of *extensionality* and *expressive completeness*; and connections with *full abstraction*. Also, it would be of interest to specify a uniform operational semantics for our general class of models $\mathcal{M}_{\mathbb{P}}$, and to prove some correspondence results. A good basis for this should be given by [Cur86]. It would also be interesting to formulate a notion of *continuous* (*e.g.* probabilistic) computation in a network, replacing algebraic domains by continuous ones. Much of the theory developed here should generalize; note in particular that Lemma 3.15 is valid for ω-continuous cpo's, replacing "compact" by "relatively compact". Beyond asynchronous networks, we would like to give a general notion of model in categorical terms, which would subsume a wide range of concurrency formalisms, including process algebras and Petri nets, as well as dataflow. The ideas of [Win88] should be relevant here.

Acknowledgements. I would like to thank Jay Misra for providing the initial stimulus to this work by sending me his paper [Mis89]; much of the present paper can be seen as an attempt to understand some of his ideas in a general setting. I would also like to thank Achim Jung for helpful discussions while the ideas developed, and for proving Lemma 3.15; and Mike Mislove for inviting me to the 1989 MFPS conference, where I presented this material at an informal "pre-meeting"; and for inviting me to submit this paper to the Proceedings. My thanks also to my hosts at the University of Pennsylvania for providing such a stimulating and friendly environment during my visit in the first half of 1989; the Nuffield Foundation, for their support in the form of a Science Research Fellowship for 1988–89; and the U.K. SERC and U.S.A. NSF for their financial support.

References

[Abr84] S. Abramsky. Reasoning about concurrent systems: a functional approach. In F. Chambers, D. Duce, and G. Jones, editors, *Distributed Computing*, volume 20 of *APIC Studies in Data Processing*, pages 307–319. Academic Press, 1984.

[AM75] M. A. Arbib and E. Manes. *Arrows, Structures and Functors: the Categorical Imperative*. Academic Press, 1975.

[BA81] J. D. Brock and W. B. Ackerman. Scenarios: a model of nondeterminate computation. In *Formalization of Programming Concepts*, pages 252–259. Springer-Verlag, 1981. Lecture Notes in Computer Science Vol. 107.

[Bar84] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[CGW87] T. Coquand, C. Gunter, and G. Winskel. dI-domains as a model of polymorphism. In *Third Workshop on the Mathematical Foundations of Programming Language Semantics*, pages 344–363. Springer-Verlag, 1987.

[Cur86] Pierre-Louis Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Pitman, 1986.

[Fau82] A. Faustini. *The equivalence of a denotational and an operational semantics for pure dataflow.* PhD thesis, University of Warwick, 1982.

[GGKW84] J. Glauert, J. Gurd, C. Kirkham, and I. Watson. The dataflow approach. In *Distributed Computing*, volume 20 of *APIC Studies in Data Processing*, pages 1–53. Academic Press, 1984.

[GJ88] C. Gunter and A. Jung. Coherence and consistency in domains. In *Third Annual Symposium on Logic in Computer Science*, pages 309–317. Computer Society Press of the IEEE, 1988.

[GP87] H. Gaifman and V. R. Pratt. Partial order models of concurrency and the computation of functions. In *Symposium on Logic in Computer Science*, pages 72–85. Computer Society Press of the IEEE, 1987.

[GS89] C. Gunter and D. S. Scott. Semantic domains. Technical Report MS-CIS-89-16, University of Pennsylvania, Department of Computer and Information Science, 1989.

[Hen82] P. Henderson. Purely functional operating systems. In J. Darlington, P. Henderson, and D. Turner, editors, *Functional Programming.* Cambridge University Press, 1982.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1985.

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

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information Processing 74*, pages 471–475, Amsterdam, 1974. North Holland.

[Kel78] R. M. Keller. Denotational models for parallel programs with indeterminate operators. In E. J. Neuhold, editor, *Formal Description of Programming Concepts*, pages 337–366. North Holland, 1978.

[KLP79] R. M. Keller, G. Lindstrom, and S. Patil. A loosely-coupled applicative multiprocessing system. In *AFIPS Conference Proceedings 46*, pages 613–622, 1979.

[KM77] G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. In B. Gilchrist, editor, *Information Processing 77*, pages 993–998, Amsterdam, 1977. North Holland.

[KP78] G. Kahn and G. Plotkin. Domaines concrets. Technical Report 336, IRIA-Laboria, 1978.

[LS88] N. A. Lynch and E. W. Stark. A proof of the Kahn principle for input/output automata. *Information and Computation*, 1988. To appear.

[Maz89] A. Mazurkiewicz. Basic notions of trace theory. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 285–363. Springer-Verlag, 1989. Lecture Notes in Computer Science Vol. 354.

[Mey82] Albert Meyer. What is a model of the lambda calculus? *Information and Control*, 52:87–122, 1982.

[Mis89] J. Misra. Equational reasoning about nondeterministic processes. Department of Computer Science, The University of Texas at Austin, 1989.

[ML71] S. Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, Berlin, 1971.

[Par82] D. Park. The "fairness" problem and non-deterministic computing networks. In *Foundations of Computer Science IV Part 2*, volume 159 of *Mathematical Centre Tracts*, pages 133–161. Centrum voor Wiskunde en Informatica, 1982.

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

[PS88] P. Panangaden and E. W. Stark. Computation, residuals and the power of indeterminacy. In *Automata, Languages and Programming*, pages 439–454. Springer-Verlag, 1988. Lecture Notes in Computer Science Vol. 317.

[Rei85] W. Reisig. *Petri Nets*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[WA85] W. W. Wadge and E. A. Ashcroft. *Lucid, the Dataflow Programming Language*, volume 22 of *APIC Studies in Data Processing*. Academic Press, 1985.

[Win85] G. Winskel. Synchronisation trees. *Theoretical Computer Science*, May 1985.

[Win86] G. Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to other Models of Concurrency*. Springer-Verlag, 1986. Lecture Notes in Computer Science Vol. 255.

[Win88] G. Winskel. A category of labelled Petri nets and compositional proof system. In *Third Annual Symposium on Logic in Computer Science*, pages 142–154. Computer Society Press of the IEEE, 1988.