Compositional Relational Semantics for Indeterminate Dataflow Networks

Eugene W. Stark*

Department of Computer Science

State University of New York at Stony Brook

Stony Brook, NY 11794 USA

Abstract

Given suitable categories $\mathbf{T}, \mathbf{C}$ and functor $F : \mathbf{T} \to \mathbf{C}$, if X, Y are objects of $\mathbf{T}$, then we define an (X,Y)-*relation in* $\mathbf{C}$ to be a triple $(R, \underline{r}, \bar{r})$, where R is an object of $\mathbf{C}$ and $\underline{r} : R \to FX$ and $\bar{r} : R \to FY$ are morphisms of $\mathbf{C}$. We define an algebra of relations in $\mathbf{C}$, including operations of "relabeling," "sequential composition," "parallel composition," and "feedback," which correspond intuitively to ways in which processes can be composed into networks. Each of these operations is defined in terms of composition and limits in $\mathbf{C}$, and we observe that any operations defined in this way are preserved under the mapping from relations in $\mathbf{C}$ to relations in $\mathbf{C}'$ induced by a continuous functor $G : \mathbf{C} \to \mathbf{C}'$.

To apply the theory, we define a category **Auto** of concurrent automata, and we give an operational semantics of dataflow-like networks of processes with indeterminate behaviors, in which a network is modeled as a relation in **Auto**. We then define a category **EvDom** of "event domains," a (non-full) subcategory of the category of Scott domains and continuous maps, and we obtain a coreflection between **Auto** and **EvDom**. It follows, by the limit-preserving properties of coreflectors, that the denotational semantics in which dataflow networks are represented by relations in **EvDom**, is "compositional" in the sense that the mapping from operational to denotational semantics preserves the operations on relations. Our results are in contrast to examples of Brock and Ackerman, which imply that no compositional semantics is possible in terms of set-theoretic relations.

1 Introduction

Dataflow networks (see, *e.g.* [4, 3, 5, 7, 9, 10]) consist of a collection of concurrently and asynchronously executing sequential processes that communicate by transmitting sequences

*Research supported in part by NSF Grant CCR-8702247.

or "streams" of "value tokens" over FIFO communication channels. Typically, a network is described as a directed graph, whose nodes are processes and whose arcs are communication channels. Each channel serves to connect an "output port" of one process to an "input port" of another process. "Determinate" (or functional) networks were first studied by Kahn [9], who gave an elegant fixed-point principle for determining the function computed by a network from the functions computed by the components.

"Indeterminate" (or non-functional) networks remain less well understood, despite extensive study. An interesting class of indeterminate processes are the "merge" processes, which shuffle together sequences of values from two input channels onto a single output channel. Brock and Ackerman have shown [3, 4] that no naive generalization of Kahn's theory to indeterminate networks, obtained by replacing input/output functions with input/output relations, can be "compositional" in the sense that the mapping from operational to denotational semantics preserves the operations by which networks are built from component processes. Their examples use only functional processes and a weak form of merge process; thus, their results apply to essentially any interesting class of indeterminate networks.

The title of this paper advertises "compositional relational semantics for indeterminate dataflow networks." This reason this is not a contradiction with Brock and Ackerman's results is that their results apply only to the usual *set-theoretic* notion of a relation, whereas here we use instead a category-theoretic generalization. Our notion of relation involves two suitably complete categories $\mathbf{T}$ and $\mathbf{C}$, and a suitable functor $F : \mathbf{T} \to \mathbf{C}$. If X and Y are objects of $\mathbf{T}$, then by an "(X, Y)-relation in $\mathbf{C}$" we mean a triple $(R, \underline{r}, \bar{r})$ where R is an object of $\mathbf{C}$, and $\underline{r} : R \to FX$ and $\bar{r} : R \to FY$ are morphisms of $\mathbf{C}$. We think of the objects of $\mathbf{T}$ as "types," and in our semantics of dataflow networks the types X and Y are simply the sets of input ports and output ports over which a process may communicate. An algebra of relations in $\mathbf{C}$ can be defined, including operations of "relabeling," "sequential composition," "parallel composition," and "feedback" that correspond intuitively to ways in which networks can be built from component processes. These operations have categorical definitions in terms of composition and limits in $\mathbf{C}$. It is a simple observation (Theorem 1) that any operations definable in this way are preserved by the mapping from relations in $\mathbf{C}$ to relations in $\mathbf{C}'$ induced by a continuous functor $G : \mathbf{C} \to \mathbf{C}'$.

To obtain a correct semantics of process networks, we must choose properly the category $\mathbf{C}$, and this is a bit tricky. In this paper, we give two examples for $\mathbf{C}$: (1) a category **Auto** of concurrent automata, and (2) a certain (non-full) subcategory **EvDom** of the category of Scott domains and continuous maps. The semantics based on **Auto** is "operational," because it involves automata and computation sequences, and it is not difficult to become convinced that it can serve as accurate model of the usual informal "token-pushing" operational semantics usually given for dataflow networks (see, *e.g.* [3]). The semantics based on **EvDom** is more "denotational," because it is defined using order-theoretic notions that do not necessarily have to do with computation. The objects of **EvDom** are the "event domains," which have been studied previously [6, 20], but the morphisms we use are apparently new, and are crucial to our results. We obtain a coreflection between **Auto** and **EvDom**, and from the limit-preserving properties of coreflectors it follows immediately that the denotational semantics based on **EvDom** is "compositional" in the sense that the map from operational to denotational semantics preserves the operations of our relational algebra. Moreover, the denotational semantics is not "too abstract," in the sense that networks having distinct input/output relations in the ordinary set-theoretic sense, also determine distinct relations in **EvDom**. The semantics is not fully abstract, though.

In this paper, FX or $F(X)$ denotes the application of functor (or function) F to argument X. If $F : X \to Y$ and $G : Y \to Z$, then GF or $G \circ F$ denotes the composition of F and G. We use parentheses freely to increase readability. For basic definitions and terminology of category theory, we refer the reader to [8, 11]. The text [15] and the unpublished [14] contain material on domain theory.

2 An Algebra of Relations

We begin by defining a category-theoretic generalization of the notion of a relation, and we show how to obtain various operations on relations that correspond intuitively to ways in which processes can be composed into networks.

2.1 Relations

Let $\mathbf{T}$ be a category equipped with a specified terminal object and binary products, and let $\mathbf{C}$ be a category with a specified terminal object, binary products, and equalizers of parallel pairs of arrows. Let $F : \mathbf{T} \to \mathbf{C}$ be a functor that preserves the specified terminal object and products. Intuitively, we think of $\mathbf{T}$ as a category whose objects are "types" and whose morphisms are "relabeling maps." Pairs of objects of $\mathbf{T}$ will be used to index classes of relations, which are built in the category $\mathbf{C}$.

Formally, if X and Y are objects of $\mathbf{T}$, then an (X, Y)-*relation in* $\mathbf{C}$ (with respect to the functor F) is a triple $(R, \underline{r}, \bar{r})$, where R is an object of $\mathbf{C}$ and $\underline{r} : R \to FX$, $\bar{r} : R \to FY$ are morphisms of $\mathbf{C}$:

$$\begin{tikzcd} R \arrow[r, "\underline{r}"] \arrow[d, "\bar{r}"'] & FX \\ FY & \end{tikzcd}$$

When there can be no confusion, we usually say "let R be an (X, Y)-relation in $\mathbf{C}$," and use corresponding lower-case letters $\underline{r}$, $\bar{r}$ to denote the morphisms.

If R and S are (X, Y)-relations in $\mathbf{C}$, then a *morphism* from R to S is a morphism $f : R \to S$ in $\mathbf{C}$ such that the obvious diagram commutes:

$$
\begin{tikzcd}
R \arrow[r, "\underline{r}"] \arrow[d, "\bar{r}"'] \arrow[dr, "f"] & FX \\
FY & S \arrow[l, "\bar{s}"] \arrow[u, "\underline{s}"']
\end{tikzcd}
$$

Let $\mathbf{Reln_C}(X, Y)$ denote the category of (X, Y)-relations in $\mathbf{C}$ and their morphisms.

For example, if $\mathbf{T} = \mathbf{C} = \mathbf{Set}$, the category of sets and functions, and $F : \mathbf{T} \to \mathbf{C}$ is the identity functor, then an ordinary set-theoretic relation $R \subseteq X \times Y$ may be represented as an (X, Y)-relation in $\mathbf{C}$ by taking $\underline{r} : R \to X$ and $\bar{r} : R \to Y$ to be the projections on the first and second components, respectively.

2.2 Operations on Relations

We may define various operations on relations.

2.2.1 Relabeling

Suppose R is an (X, Y)-relation in $\mathbf{C}$. If $\phi : Y \to Y'$ is a morphism in $\mathbf{T}$, then the *output relabeling* of R by ϕ is the (X, Y')-relation $R; \phi = (R, \underline{r}, (F\phi)\bar{r})$. Similarly, if $\psi : X \to X'$ is a morphism in T, then the *input relabeling* of R by ψ is the (X', Y)-relation $\psi; R = (R, (F\psi)\underline{r}, \bar{r})$. Output relabeling by ϕ extends to a functor

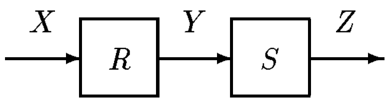
$$(-); \phi : \mathbf{Reln_C}(X, Y) \to \mathbf{Reln_C}(X, Y').$$

and input relabeling by ψ extends to a functor

$$\psi; (-) : \mathbf{Reln_C}(X, Y) \to \mathbf{Reln_C}(X', Y).$$

2.2.2 Sequential Composition

Relations may be composed in sequence as suggested by the picture:



Formally, if R is an (X, Y)-relation in $\mathbf{C}$ and S is a (Y, Z)-relation in $\mathbf{C}$, then their *sequential composition* is the (X, Z)-relation $R; S = (R; S, \underline{r}s', \bar{s}r')$, where r' and s' are defined to make the square in the following diagram a pullback:

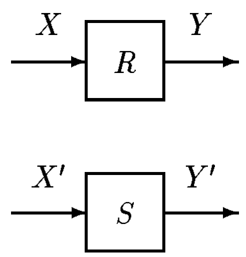
$$\begin{tikzcd}
R;S \arrow[r, "s'"] \arrow[d, "r'"'] & R \arrow[r, "\underline{r}"] \arrow[d, "\bar{r}"] & FX \\
S \arrow[r, "\underline{s}"'] \arrow[d, "\bar{s}"'] & FY & \\
FZ & &
\end{tikzcd}$$

It should be noted that in the case $\mathbf{T} = \mathbf{C} = \mathbf{Set}$, with $F : \mathbf{T} \to \mathbf{C}$ the identity functor, the above definition of sequential composition of relations does not agree exactly with the standard set-theoretic definition. Specifically, if $R \subseteq X \times Y$ and $S \subseteq Y \times Z$, then

$$R; S \simeq \{(x, y, z) : (x, y) \in R, (y, z) \in S\}.$$

2.2.3 Parallel Composition

Relations may also be composed in parallel, as suggested by the picture:



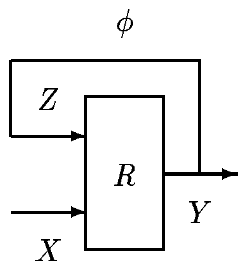
Formally, suppose R is an (X, Y)-relation in $\mathbf{C}$ and S is an (X', Y')-relation in $\mathbf{C}$. Then their *parallel composition* is the $(X \times X', Y \times Y')$-relation $R \| S = (R \times S, \underline{r} \times \underline{s}, \bar{r} \times \bar{s})$, where we have used the assumption that $F(X \times X') = FX \times FX'$ and $F(Y \times Y') = FY \times FY'$.

The mapping that takes the pair (R, S) to $R \| S$ extends to a functor

$$\| : \mathbf{Reln_C}(X, Y) \times \mathbf{Reln_C}(X', Y') \to \mathbf{Reln_C}(X \times X', Y \times Y').$$

2.2.4 Feedback

We can also define an operation of "feedback," corresponding to the picture:



Formally, suppose R is an $(X \times Z, Y)$-relation in $\mathbf{C}$, and $\phi : Y \to Z$ is a morphism in $\mathbf{T}$. The *feedback* of R by ϕ is the (X, Y)-relation $(R_{\circlearrowleft\phi}, \pi_{FX}\underline{r}e, \bar{r}e)$, where e is the equalizer of $\pi_{FZ}\underline{r}$ and $(F\phi)\bar{r}$ in C:

$$
\begin{tikzcd}
 & & FX & \\
 & & F(X \times Z) \arrow[u, "\pi_{FX}"'] \arrow[dr, "\pi_{FZ}"] & \\
R_{\circlearrowleft\phi} \arrow[uurr] \arrow[r, "e"] \arrow[ddrr] & R \arrow[ur, "\underline{r}"] \arrow[dr, "\bar{r}"'] & & FZ \\
 & & FY \arrow[ur, "F\phi"'] \arrow[d, "\mathrm{id}_{FY}"] & \\
 & & FY &
\end{tikzcd}
$$

The map $(\text{ - })_{\circlearrowleft\phi}$ extends to a functor

$$(\text{ - })_{\circlearrowleft\phi} : \mathbf{Reln}_{\mathbf{C}}(X \times Z, Y) \to \mathbf{Reln}_{\mathbf{C}}(X, Y).$$

The operations we have defined are not independent: in particular, sequential composition is definable (up to isomorphism) in terms of parallel composition, output relabeling, and feedback. Formally, suppose R is an (X, Y)-relation in $\mathbf{C}$ and S is a (Y, Z)-relation in $\mathbf{C}$. Let $\pi_Y : Y \times Z \to Y$ and $\pi_Z : Y \times Z \to Z$ be the projections associated with the product $Y \times Z$ in $\mathbf{T}$. Then

$$R; S \simeq ((R \| S)_{\circlearrowleft\pi_Y}); \pi_Z.$$

2.3 Preservation of Operations by Functors

Suppose now we have two categories, $\mathbf{C}$ and $\mathbf{C}'$, equipped with specified terminal object, binary products, and equalizers, and two functors $F : \mathbf{T} \to \mathbf{C}$ and $F' : \mathbf{T} \to \mathbf{C}'$ that preserve the specified terminal object and products. If $G : \mathbf{C} \to \mathbf{C}'$ is a functor such that $GF = F'$, then each (X,Y)-relation $R = (R, \underline{r}, \bar{r})$ in $\mathbf{C}$ determines an (X,Y)-relation $GR = (GR, G\underline{r}, G\bar{r})$ in $\mathbf{C}'$. The map taking R to GR extends, for each (X,Y), to a functor

$$G_{X,Y} : \mathbf{Reln}_{\mathbf{C}}(X,Y) \to \mathbf{Reln}_{\mathbf{C}'}(X,Y).$$

Moreover, if G preserves the specified limits, then the ensemble of functors $\{G_{X,Y} : X, Y \in \mathbf{T}\}$ preserves the operations of relabeling, sequential composition, parallel composition, and feedback on relations in a way made precise by the following result:

Theorem 1 *Suppose* $\mathbf{C}, \mathbf{C}', F, F'$ *and* G *are as above. Then*

1. *If* $R \in \mathbf{Reln}_{\mathbf{C}}(X,Y)$, $\phi : Y \to Y'$, *and* $\psi : X \to X'$, *then*

$$G_{X,Y'}(R;\phi) = G_{X,Y}(R);\phi, \qquad G_{X',Y}(\psi;S) = \psi;G_{X,Y}(S).$$

2. *If* $R \in \mathbf{Reln}_{\mathbf{C}}(X,Y)$ *and* $S \in \mathbf{Reln}_{\mathbf{C}}(Y,Z)$, *then* $G_{X,Z}(R;S) = G_{X,Y}(R);G_{Y,Z}(S)$.

3. *If* $R \in \mathbf{Reln}_{\mathbf{C}}(X,Y)$ *and* $R' \in \mathbf{Reln}_{\mathbf{C}}(X',Y')$, *then*

$$G_{X\times X',Y\times Y'}(R\|R') = G_{X,Y}(R)\|G_{X',Y'}(R').$$

4. *If* $R \in \mathbf{Reln}_{\mathbf{C}}(X \times Z, Y)$ *and* $\phi : Y \to Z$, *then* $G_{X,Z}(R_{\circlearrowleft\phi}) = G_{X\times Z,Y}(R)_{\circlearrowleft\phi}$.

Proof – All of these operations have categorical definitions in terms of composition and limits, and these are preserved by the functor G. ∎

Obviously, the same reasoning applies to show that the functors $G_{X,Y}$ preserve any other operations on relations in $\mathbf{C}$ we might wish to consider, provided those operations can be defined in terms of composition and limits in $\mathbf{C}$.

3 Concurrent Automata

In this section, we define a category **Auto** of concurrent automata. This category serves as our operational semantics for dataflow networks, in that we shall model networks as relations in **Auto**. The kind of automata we consider incorporate concurrency in the form of a binary *concurrency relation* $\|$ on a set E of *events*. The pair $(E, \|)$ is called a "concurrent alphabet." Intuitively, events represent primitive occurrences during computation. If events e and e' are both possible occurrences when in state q, and if e and e' are related by the concurrency relation, then e and e' can be executed in either order, with equivalent effect. Similar automata have been studied by Bednarczyk [2], and by the author [18, 19].

3.1 Concurrent Alphabets

A *concurrent alphabet* is a set E, equipped with a symmetric, irreflexive binary relation $\|_E$, called the *concurrency relation*. Elements $e, e' \in E$ are said to *commute* if $e\|_E e'$, and a subset U of E is called *commuting* if every pair of its elements commutes. Let $\mathrm{Com}(E)$ denote the set of all finite commuting subsets of E. Suppose $U, V \in \mathrm{Com}(E)$. Then U and V are called *orthogonal*, and we write $U \perp_E V$, if $U \cup V \in \mathrm{Com}(E)$ and $U \cap V = \emptyset$.

A *morphism* from a concurrent alphabet E to a concurrent alphabet F is a function $\mu : \mathrm{Com}(E) \to \mathrm{Com}(F)$ such that

1. $\mu(\emptyset) = \emptyset$.

2. If $U \cup V \in \mathrm{Com}(E)$, then $\mu(U) \cup \mu(V) \in \mathrm{Com}(F)$, and $\mu(U \setminus V) = \mu(U) \setminus \mu(V)$.

Here the symbol $\setminus$ denotes set difference. Let **Alph** denote the category of concurrent alphabets and their morphisms.

Concurrent alphabets constitute the starting point for "trace theory" [1, 12]. The above definition of morphism of concurrent alphabets was motivated by the author's study of "concurrent transition systems" [16, 17, 18]. The category **Alph** has a number of pleasant properties, although the author is not aware of it having been studied previously.

Lemma 3.1 *Suppose $\mu : E \to F$ is a morphism of concurrent alphabets. If $U \cup V \in \mathrm{Com}(E)$, then (1) $\mu(U \cup V) = \mu(U) \cup \mu(V)$ and (2) $\mu(U \cap V) = \mu(U) \cap \mu(V)$.*

Proof – (1) Observe that for arbitrary sets A, B, C, we have $C = A \cup B$ iff $A \setminus C = \emptyset = B \setminus C$, $C \setminus A = B \setminus A$, and $C \setminus B = A \setminus B$. Clearly, $\emptyset = \mu(U \setminus (U \cup V)) = \mu(U) \setminus \mu(U \cup V)$, and similarly $\mu(V) \setminus \mu(U \cup V) = \emptyset$. Since $\mu(U \cup V) \setminus \mu(V) = \mu((U \cup V) \setminus V) = \mu(U \setminus V) = \mu(U) \setminus \mu(V)$, and similarly, $\mu(V \cup U) \setminus \mu(U) = \mu(V) \setminus \mu(U)$, it follows that $\mu(U \cup V) = \mu(U) \cup \mu(V)$.

(2) $\mu(U \cap V) = \mu(U \setminus (U \setminus V)) = \mu(U) \setminus (\mu(U) \setminus \mu(V)) = \mu(U) \cap \mu(V)$. ▮

Lemma 3.2 *Suppose $\mu : E \to F$ is a morphism of concurrent alphabets. Then $\mu(e) \perp_F \mu(e')$ whenever $e\|_E e'$. Conversely, any function $\mu : E \to \mathrm{Com}(F)$ having this property extends uniquely to a morphism $\mu : E \to F$.*

Proof – If $\mu : E \to F$ is a morphism, and $e\|_E e'$, then $\mu(\{e\}) = \mu(\{e\} \setminus \{e'\}) = \mu(\{e\}) \setminus \mu(\{e'\})$, and similarly, $\mu(\{e'\}) = \mu(\{e'\}) \setminus \mu(\{e\})$, so $\mu(\{e\}) \cap \mu(\{e'\}) = \emptyset$.

Conversely, if $\mu : E \to \mathrm{Com}(F)$ has the stated property, then it extends to a morphism by defining $\mu(U) = \bigcup\{e \in U : \mu(e)\}$. Moreover, by Lemma 3.1, any extension of μ to a morphism must satisfy this relation, so the extension of μ is uniquely determined. ▮

Theorem 2 **Alph** *has finite limits.*

Proof – We show that **Alph** has: (1) a terminal object, (2) binary products, and (3) equalizers of parallel pairs.

(1) The empty concurrent alphabet is easily seen to be a terminal object (in fact a zero object) in **Alph**.

(2) Suppose E and F are concurrent alphabets. Let $E \otimes F$ denote the concurrent alphabet with elements $E+F$ (disjoint union), and with $\|_{E\otimes F} = \|_E \cup \|_F \cup (E \times F) \cup (F \times E)$. Note that the sets in $\mathrm{Com}(E \otimes F)$ are precisely those of the form $U + V$ with $U \in \mathrm{Com}(E)$ and $V \in \mathrm{Com}(F)$. Define projections $\pi_E : E \otimes F \to E$ and $\pi_F : E \otimes F \to F$ by $\pi_E(U) = U \cap E$ and $\pi_F(U) = U \cap F$. It is easy to check that π_E and π_F are morphisms, and that $E \otimes F$, equipped with π_E and π_F, has the universal property required of a categorical product.

(3) Suppose $\sigma, \tau : E \to F$ are morphisms in **Alph**. Define a nonempty set $U \in \mathrm{Com}(E)$ to be *equalizing* if $\sigma(U) = \tau(U)$. Call U *minimal equalizing* if it is equalizing and it has no proper equalizing subsets. We observe the following fact: if U and V are distinct minimal equalizing subsets of E, and $U \cup V \in \mathrm{Com}(E)$, then U and V are disjoint. For, if $U \cap V = W \neq \emptyset$, then $\sigma(W) = \sigma(U) \cap \sigma(V) = \tau(U) \cap \tau(V) = \tau(W)$, so W is equalizing, a contradiction with the assumed minimality of U, V. It follows from this observation that every equalizing $U \in \mathrm{Com}(E)$ can be written uniquely as a finite union of minimal equalizing subsets.

Now, let D be the set of all minimal equalizing subsets of E, and define $U \|_D V$ iff $U \cup V \in \mathrm{Com}(E)$. Define $\mu : D \to E$ to be the morphism that satisfies $\mu(\{U_1, \ldots, U_n\}) = \bigcup_k U_k$ whenever $\{U_1, \ldots, U_n\} \in \mathrm{Com}(D)$. One may now check that μ is an equalizer of σ and τ. ∎

3.2 Automata

An *automaton* is a tuple $A = (E, Q, q^\circ, T)$, where

- E is a concurrent alphabet of *events*, not containing the special symbol ϵ, called the *identity event*.
- Q is a set of *states*.
- $q^\circ \in Q$ is a distinguished *start state*.
- $T \subseteq Q \times (E \cup \{\epsilon\}) \times Q$ is a set of *transitions*. We write $t : q \xrightarrow{e} r$, or just $q \xrightarrow{e} r$, to denote a transition $t = (q, e, r) \in T$ or to assert the existence of such a transition in T.

These data are required to satisfy the following conditions:

(Identity) $q \xrightarrow{\epsilon} r$ iff $q = r$.

(Disambiguation) If $q \xrightarrow{e} r$ and $q \xrightarrow{e} r'$, then $r = r'$.

(Commutativity) For all states q and events $e, e' \in E$, if $e \|_E e'$, $q \xrightarrow{e} r$, and $q \xrightarrow{e'} r'$, then for some state s there exist transitions $r \xrightarrow{e'} s$ and $r' \xrightarrow{e} s$.

If $t : q \xrightarrow{e} r$, then q is called the *domain* $\mathrm{dom}(t)$ of t and r is called the *codomain* $\mathrm{cod}(t)$ of t. Transitions t and u are called *coinitial* if $\mathrm{dom}(t) = \mathrm{dom}(u)$. We say that event $e \in E$ is *enabled* in state q if there exists a transition $q \xrightarrow{e} r$ in T.

Intuitively, if $e \in E$, then a transition $q \xrightarrow{e} r$ represents a potential computation step of A in which event e occurs and the state changes from q to r. *Identity* transitions $q \xrightarrow{\epsilon} q$ do not represent computation steps of A; they serve merely to "pad" computations. The (Identity) condition ensures that this is all they can do. The (Disambiguation) condition ensures that the new state r in a transition $q \xrightarrow{e} r$ is uniquely determined by q and e. The (Commutativity) condition says that if two commuting events are enabled in the same state, then they can occur in either order with the same effect.

A *finite computation sequence* for an automaton A is a finite sequence γ of transitions of the form:

$$q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \dots \xrightarrow{e_n} q_n.$$

The number n is called the *length* $|\gamma|$ of F. (By convention, if $n = 0$ then the computation sequence consists of the single state q_0, and no transitions.) Similarly, an *infinite computation sequence* for A is an infinite sequence of transitions:

$$q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \dots.$$

We extend notation and terminology for transitions to computation sequences, so that if γ is a computation sequence, then the *domain* $\mathrm{dom}(\gamma)$ of γ is the state q_0, and if γ is finite, then the *codomain* $\mathrm{cod}(\gamma)$ of γ is the state q_n. We write $\gamma : q \to r$ to assert that γ is a finite computation sequence with domain q and codomain r. A computation sequence γ is *initial* if $\mathrm{dom}(\gamma)$ is the distinguished start state q^{o}. If $\gamma : q \to r$ and $\delta : q' \to r'$ are finite computation sequences, then γ and δ are called *composable* if $q' = r$, in which case we define their *composition* to be the finite computation sequence $\gamma\delta : q \to r'$, obtained by concatenating γ and δ and identifying $\mathrm{cod}(\gamma)$ with $\mathrm{dom}(\delta)$.

If $A = (E, Q, q^{\mathrm{o}}, T)$ and $A' = (E', Q', (q^{\mathrm{o}})', T')$ are automata, then a *morphism* from A to A' is a pair $\rho = (\rho_{\mathrm{e}}, \rho_{\mathrm{s}})$, where $\rho_{\mathrm{e}} : E \to E'$ is a morphism of concurrent alphabets, and $\rho_{\mathrm{s}} : Q \to Q'$ is a function, such that

1. $\rho_{\mathrm{s}}(q^{\mathrm{o}}) = (q^{\mathrm{o}})'$.
2. Suppose $q \xrightarrow{e} r \in T$, with $e \neq \epsilon$. Then for every enumeration $\{e'_1, \dots, e'_n\}$ of $\rho_{\mathrm{e}}(\{e\})$, there exists a (necessarily unique) finite computation sequence
$$\rho_{\mathrm{s}}(q) = r'_0 \xrightarrow{e'_1} r'_1 \xrightarrow{e'_2} \dots \xrightarrow{e'_n} r'_n = \rho_{\mathrm{s}}(r)$$
of A'.

We usually drop the subscripts on ρ_{e} and ρ_{s}, writing ρ for both.

Let **Auto** denote the category of automata and their morphisms. There is an obvious forgetful functor $\mathrm{AuAl} : \mathbf{Auto} \to \mathbf{Alph}$, which takes each automaton $A = (E, Q, q^{\mathrm{o}}, T)$ to the concurrent alphabet E, and each morphism $(\rho_{\mathrm{e}}, \rho_{\mathrm{s}})$ of automata to the morphism ρ_{e} of concurrent alphabets.

Theorem 3 *The forgetful functor* AuAl : **Auto** $\to$ **Alph** *has a right adjoint* AlAu : **Alph** $\to$ **Auto**.

Proof – Given a concurrent alphabet E, let $\mathrm{AlAu}(E)$ be the one-state automaton $(E, \{*\}, *, T)$, where $T = \{*\} \times (E \cup \{\epsilon\}) \times \{*\}$. We claim that this construction defines the object map of a functor AlAu : **Alph** $\to$ **Auto**, which is right-adjoint to the forgetful functor AuAl : **Auto** $\to$ **Alph**. To prove this, it suffices to show that for each $E \in$ **Alph**, there exists an "evaluation map" $\varepsilon_E : \mathrm{AuAl}(\mathrm{AlAu}(E)) \to E$, universal from AuAl to E. We may simply take $\varepsilon_E = \mathrm{id}_E$. ∎

Theorem 4 **Auto** *has finite limits.*

Proof – We show that **Auto** has (1) a terminal object, (2) binary products, and (3) equalizers.

(1) It is easy to see that the one-state, one-transition automaton with the empty alphabet of events is a terminal object (in fact a zero object) in **Auto**.

(2) Suppose $A_1 = (E_1, Q_1, q_1^\circ, T_1)$ and $A_2 = (E_1, Q_2, q_2^\circ, T_2)$ are automata. Let $A_1 \times A_2$ be the automaton

$$A_1 \times A_2 = (E_1 \otimes E_2, Q_1 \times Q_2, (q_1^\circ, q_2^\circ), T),$$

where T is the set of all $((q_1, q_2), e, (r_1, r_2))$ such that one of the following conditions holds:

1. $e = \epsilon$, $q_1 = r_1$, and $q_2 = r_2$.
2. $e \in E_1$, $(q_1, e, r_1) \in T_1$, and $q_2 = r_2$.
3. $e \in E_2$, $(q_2, e, r_2) \in T_2$, and $q_1 = r_1$.

Define projections $\pi_i : A_1 \times A_2 \to A_i$ ($i \in \{1, 2\}$) by letting $(\pi_i)_\mathrm{e} : E_1 \otimes E_2 \to E_i$ be the projection in **Alph** and $(\pi_i)_\mathrm{s} : Q_1 \times Q_2 \to Q_i$ be the projection in **Set**. It is straightforward to verify that $A_1 \times A_2$, equipped with projections π_1 and π_2, is a product in **Auto**.

(3) Suppose $\sigma, \tau : A \to A'$ are morphisms in **Auto**, where $A = (E, Q, q^\circ, T)$ and $A' = (E', Q', (q^\circ)', T')$. Let $\mu_\mathrm{e} : D \to E$ be an equalizer of σ_e and τ_e in **Alph**, let $\mu_\mathrm{s} : R \to Q$ be an equalizer of σ_s and τ_s in **Set**. Recall that D is the set of all minimal equalizing sets $U \in \mathrm{Com}(E)$. Define $B = (D, R, q^\circ, T)$, where T consists of all triples (q, U, r) such that to each enumeration $\{e_1, \ldots, e_n\}$ of U there corresponds a computation sequence

$$q = q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} q_n = r$$

of A. Let $\mu = (\mu_\mathrm{e}, \mu_\mathrm{s})$, then it is straightforward to verify that μ is an equalizer of σ and τ in **Auto**. ∎

4 Operational Semantics of Dataflow Networks

In this section, we give a formal operational semantics for dataflow networks as relations in **Auto**.

4.1 Port Sets

Let **Port** be the category whose objects are finite or countably infinite sets (of *ports*), and whose morphisms are the opposites of functions; thus, a morphism $f : P \rightarrowtail P'$ is a function from P' to P. The empty set $\emptyset$ is clearly a terminal object in **Port**, and if $+$ denotes disjoint union, then the set $P + P'$ is obviously a product of P and P' in **Port**. Let V be a fixed universe of *data values*, which we assume contains the set of natural numbers as a subset. If P is an object of **Port**, then define a *P-event* to be an element of the concurrent alphabet $\text{Events}(P) = P \times V$, where $(p,v) \| (p',v')$ iff $p \neq p'$. Intuitively, we think of a P-event $e = (p,v)$ as representing the transmission of data value v over port p. We write $\text{port}(e)$ to denote the port component p and $\text{value}(e)$ to denote the value component v, of e.

Define the functor $\text{PoAl} : \mathbf{Port} \to \mathbf{Alph}$ to take each object P of **Port** to the concurrent alphabet $\text{Events}(P)$, and each morphism $f : P \rightarrowtail P'$ of **Port** to the morphism $\phi : \text{PoAl}(P) \to \text{PoAl}(P')$, defined by

$$\phi(U) = \{e \in \text{PoAl}(P') : (f(\text{port}(e)), \text{value}(e)) \in U\}$$

for all $U \in \text{Com}(\text{Events}(P))$. Intuitively, we think of a morphism $f : P \rightarrowtail P'$ in **Port** as a relabeling map that labels each port in P' by a port in P. The corresponding morphism $\phi : \text{PoAl}(P) \to \text{PoAl}(P')$ in **Alph** has the dual effects: (1) of deleting events for ports in P that are not the labels of ports in P', and (2) of duplicating events for ports in P that happen to be the labels of more than one port in P'.

Lemma 4.1 *The functor* PoAl *preserves finite products.*

Proof – Straightforward. ▮

Let $\text{PoAu} : \mathbf{Port} \to \mathbf{Auto}$ be the composite functor $\text{AlAu} \circ \text{PoAl}$. Then since PoAl preserves finite products and AlAu is a right adjoint, hence preserves limits, it follows that PoAu also preserves finite products.

4.2 Port Automata

Define a *port signature* to be a pair (X, Y) of objects of **Port**. The elements of X are called *input ports* and those of Y, *output ports*. If (X, Y) is a port signature, then an (X, Y)-*port automaton* is an automaton $A = (E, Q, q^\circ, T)$, such that $E = \text{Events}(X + Y + Z) = \text{Events}(X) \otimes \text{Events}(Y) \otimes \text{Events}(Z)$ and the following condition holds:

(Receptivity) For all $q \in Q$ and all $a \in \text{Events}(X)$, there exists a transition $q \xrightarrow{a} r$ in T.

Elements of $\text{Events}(X)$, $\text{Events}(Y)$, and $\text{Events}(Z)$ are called *input events*, *output events*, and *internal events*, respectively. Intuitively, the receptivity condition states that a port automaton is always prepared to receive arbitrary input.

A port automaton is *determinate* if the following additional condition holds:

(Determinacy) For all $q \in Q$ and all $b, b' \in \mathrm{Events}(Y + Z)$, if both b and b' are enabled in state q, then $b \| b'$.

Intuitively, determinate port automata make no internal choices between conflicting events.

As an example of how we can model a dataflow process as a port automaton, consider the case of a "merge" process, whose function is to shuffle together sequences of values arriving on two input ports into a single output sequence. Let i_1 and i_2 denote the two input ports and let o denote the single output port. We may represent the merge process as a port automaton

$$A_{\mathrm{mrg}} = (\mathrm{Events}(\{i_1, i_2\} + \{o\} + \{n\}), V^* \times V^* \times V^*, (\perp, \perp, \perp), T),$$

where $\perp$ denotes the empty string, and the set of transitions T contains a transition

$$((x_1, x_2, y), e, (x_1', x_2', y'))$$

iff one of the following conditions holds:

1. $e = \epsilon$, $x_1' = x_1$, $x_2' = x_2$, and $y' = y$.
2. $e = (i_1, v)$, $x_1' = x_1 v$, $x_2' = x_2$, and $y' = y$.
3. $e = (i_2, v)$, $x_1' = x_1$, $x_2' = x_2 v$, and $y' = y$.
4. $e = (n, 1)$, $v x_1' = x_1$, $x_2' = x_2$, and $y' = yv$.
5. $e = (n, 2)$, $x_1' = x_1$, $v x_2' = x_2$, and $y' = yv$.
6. $e = (o, v)$, $x_1' = x_1$, $x_2' = x_2$, and $v y' = y$.

It is straightforward to check that A_{mrg} satisfies the conditions for a port automaton, and that it is not determinate.

Intuitively, the state of A_{mrg} contains two "input buffers" and one "output buffer." Transitions of type (2) and (3) correspond to arriving input values being placed at the end of the appropriate input buffer. Transitions of type (4) and (5) are internal transitions that correspond to the indeterminate selection of input in one input buffer or the other to be moved to the output buffer. Transitions of type (6) correspond to the transmission of output from the output buffer. Similar constructions can be used to model many other kinds of dataflow processes.

4.3 Port Automata as Relations

An (X, Y)-port automaton $A = (E, Q, q^\circ, T)$ may be identified with the (X, Y)-relation $(A, \lambda_X, \lambda_Y)$ in **Auto** (with respect to the functor $\mathrm{AlAu} \circ \mathrm{PoAl} : \mathbf{Port} \to \mathbf{Auto}$), where

$$\lambda_X : A \to \mathrm{AlAu}(\mathrm{PoAl}(X)), \qquad \lambda_Y : A \to \mathrm{AlAu}(\mathrm{PoAl}(Y))$$

are the adjoint transforms of the projections

$$\pi_X : \mathrm{AuAl}(A) \to \mathrm{PoAl}(X), \qquad \pi_Y : \mathrm{AuAl}(A) \to \mathrm{PoAl}(Y).$$

(Recall that $\mathrm{AuAl}(A) = E = \mathrm{PoAl}(X) \otimes \mathrm{PoAl}(Y) \otimes \mathrm{PoAl}(Z)$.)

By making the above identification of port automata as relations in **Auto**, the definitions of Section 2.2 immediately become applicable, yielding operations of relabeling, sequential composition, parallel composition, and feedback on port automata. However, since not every (X,Y)-relation in **Auto** is an (X,Y)-port automaton, we do not know *a priori* that the class of port automata is closed under these operations.

Theorem 5 *The classes of port automata and of determinate port automata are closed under the following operations:*

1. *Input relabeling by bijections, and output relabeling by injections.*

2. *Sequential composition.*

3. *Parallel composition.*

4. *Feedback by injections.*

Proof – The proof simply requires substituting the characterizations of Theorem 4 into the definitions of the operations on relations, and checking that the conditions for port automata are satisfied in each case. We omit the details. ∎

By working through the details of the previous proof, one may convince oneself that the port automaton model is a reasonable operational semantics for dataflow networks. For example, suppose $A = (E, Q, q^\circ, T)$ is an $(X \times Z, Y)$-port automaton, and let ϕ be an injection from Z to Y, viewed as a morphism $\phi : Y \rightarrowtail Z$ in **Port**. Intuitively, we think of ϕ as specifying, for each input port in Z, a corresponding port in Y to which it is to be "connected" by a "feedback loop." Applying the feedback operation $\circlearrowleft_\phi$ to A results (up to isomorphism) in an (X,Y)-port automaton $A' = (E \setminus \mathrm{Events}(Z), Q, q^\circ, T')$. Each input transition $q \xrightarrow{e} r$ of A with $\mathrm{port}(e) \in X$ is also a transition of A', and if $q \xrightarrow{e} r$ is an output transition of A with $\mathrm{port}(e) \in Y \setminus \phi(Z)$, then A' has that same output transition. However, if $q \xrightarrow{e} r$ is an output transition of A with $e = (\phi(z), v)$ for some $z \in Z$, then A' has instead a transition $q \xrightarrow{e} r'$, where r' is the unique state such that $r \xrightarrow{(z,v)} r'$ is a transition of A. Intuitively, A' behaves like A, except that in A' outputs of values on ports in $\phi(Z)$ occur simultaneously with inputs of the same values on the corresponding ports in Z.

4.4 Set-Theoretic Input/Output Relations

We now complete our operational semantics of dataflow networks by showing how the usual set-theoretic input/output relation between domains of "port histories" can be extracted from port automata.

Formally, if P is a set of ports, then a *P-history* is a mapping from P to the CPO V^∞ of all finite and infinite sequences of values in V, equipped with the prefix ordering $\sqsubseteq$. If H is a P-history, and $P' \subseteq P$, then we write $H|P'$ to denote the P'-history H' such that $H'(p) = H(p)$ for all $p \in P'$.

Suppose $A = (\mathrm{Events}(P), Q, q^\circ, T)$ is an (X,Y)-port automaton, where $P = X + Y + Z$. Then each finite or infinite computation sequence

$$\gamma = q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \dots$$

of A determines a P-history H_γ as follows: for each $p \in P$, the sequence $H_\gamma(p)$ is the sequence of values $\mathrm{value}(e'_1), \mathrm{value}(e'_2), \dots$, where $e'_1, e'_2, \dots$, is the subsequence of $e_1, e_2, \dots$ consisting of precisely those $e_k \neq \epsilon$ with $\mathrm{port}(e_k) = p$.

A computation sequence γ of A is called *completed* if there exists no computation sequence δ, such that $\mathrm{dom}(\gamma) = \mathrm{dom}(\delta)$, $H_\gamma|X = H_\delta|X$, and $H_\gamma \sqsubset H_\delta$. That is, completed computation sequences are those whose histories are maximal among all computation sequences having the same domain and input port history. This is actually a kind of "fairness" property (see [13]), which intuitively is true when "every enabled output or internal event, not in conflict with some other enabled output or internal event, eventually occurs." The *set-theoretic input/output relation* $\mathrm{Rel}(A)$ of A is the set of all pairs $(H_\gamma|X, H_\gamma|Y)$ such that γ is a completed initial computation sequence of A.

For example, the input/output relation of the automaton A_{mrg} defined in the previous section is the set of all $(H^{\mathrm{in}}, H^{\mathrm{out}})$, with H^{in} an $\{i_1, i_2\}$-history and H^{out} an $\{o\}$-history, such that $H^{\mathrm{out}}(o)$ is a shuffle of a prefix x_1 of $H^{\mathrm{in}}(i_1)$ and a prefix x_2 of $H^{\mathrm{in}}(i_2)$, subject to the following conditions:

1. If $H^{\mathrm{in}}(i_1)$ is finite, then $x_2 = H^{\mathrm{in}}(i_2)$.
2. If $H^{\mathrm{in}}(i_2)$ is finite, then $x_1 = H^{\mathrm{in}}(i_1)$.
3. If both $H^{\mathrm{in}}(i_1)$ and $H^{\mathrm{in}}(i_2)$ are infinite, then either $x_1 = H^{\mathrm{in}}(i_1)$ or else $x_2 = H^{\mathrm{in}}(i_2)$.

Thus, the input/output relation of A_{mrg} is the *angelic merge* relation [13]. It is also possible to construct a port automaton having a slightly different merging relation, called *infinity-fair merge*, as its input/output relation. However, it is shown in [13] that no port automaton can have as its input/output relation the *fair merge* relation, which is the set of all $(H^{\mathrm{in}}, H^{\mathrm{out}})$ such that $H^{\mathrm{out}}(o)$ is a shuffle of *all* of $H^{\mathrm{in}}(i_1)$ and *all* of $H^{\mathrm{out}}(i_2)$.

We may regard the set-theoretic input/output relation $\mathrm{Rel}(A)$ of an (X,Y)-port automaton A as an (X,Y)-relation in the category **Set** of sets and functions. To do this, let F be the functor $\mathrm{Hist} : \mathbf{Port} \to \mathbf{Set}$ that takes a set P of ports to the set $\mathrm{Hist}(P)$ of all port histories over P, and that takes a morphism $f : P \to P'$ to the corresponding duplication/restriction map $\mathrm{Hist}(f) : \mathrm{Hist}(P) \to \mathrm{Hist}(P')$, given by $\mathrm{Hist}(f)(H)(p) = H(f(p))$ for all $p \in P'$. This functor is easily seen to preserve finite products. The set-theoretic input/output relation $\mathrm{Rel}(A)$ of an (X,Y)-port automaton A may now be regarded as the (X,Y)-relation $(\mathrm{Rel}(A), \pi_X, \pi_Y)$ in **Set**, where $\pi_X : \mathrm{Rel}(A) \to \mathrm{Hist}(X)$ and $\pi_Y : \mathrm{Rel}(A) \to \mathrm{Hist}(Y)$ are the obvious projections.

It is important to observe that, although we may regard the set-theoretic input/output relation of an (X, Y)-port automaton as an (X, Y)-relation in **Set**, and thereby obtain formal operations of relabeling, composition, product, and feedback on such relations, these operations are incompatible with the corresponding operations on port automata. Specifically, the feedback operation is not preserved by the mapping from port automata to input/output relations. Brock and Ackerman have given specific examples of this failure of commutativity, and it has come to be called the "Brock-Ackerman anomaly." The situation may be described succinctly as the failure of the set-theoretic relational semantics to be "compositional."

5 Denotational Semantics of Dataflow Networks

The goal of this section of the paper is to replace the category **Set**, which does not yield a compositional denotational semantics for dataflow networks, with a more highly structured category **EvDom**, for which a compositional semantics is obtained. We do this in two steps: first we define a category **TrDom** of "trace domains," which are domains that are embedded as certain normal subdomains of the domains of "traces" generated by a concurrent alphabet, then we throw away the embedding domains and obtain a category **EvDom**, whose objects are "event domains" and whose morphisms are certain continuous maps. We construct coreflections between **Auto** and **TrDom** and between **TrDom** and **EvDom**, with the right adjoints (coreflectors) going from **Auto** to **TrDom** and from **TrDom** to **EvDom**. It follows by Theorem 1 and the fact that right adjoints preserve limits, that the induced ensemble of functors, from (X, Y)-relations in **Auto** to (X, Y)-relations in **EvDom**, preserves the operations of relabeling, sequential and parallel composition, and feedback.

5.1 Domains

A (Scott) *domain* is an ω-algebraic, consistently complete CPO. A domain D is *finitary* if for all finite (=isolated=compact) elements $d \in D$ the set $\{d' \in D : d' \sqsubseteq d\}$ is finite. If D and E are domains, then a monotone map $f : D \to E$ is *continuous* if it preserves directed lubs, *strict* if $f(\perp_D) = \perp_E$, and *additive* if whenever d, d' are consistent elements of d, then $f(d), f(d')$ are consistent elements of E, and $f(e \sqcup e') = f(e) \sqcup f(e')$. Let **Dom** denote the category of domains and continuous maps.

A *subdomain* of D is a subset U of D, which is a domain under the restriction of the ordering on D, and is such that the inclusion of U in D is strict and continuous. A subdomain U of D is *normal* if for all $d \in D$, the set $\{e \in U : e \sqsubseteq d\}$ is directed.

Lemma 5.1 *A subdomain U of D is normal iff the inclusion of U in D is additive and reflects consistent pairs.*

Proof – Suppose U is a normal subdomain of D. If $e, e' \in U$ are consistent in D, then the set $V = \{e'' \in U : e'' \sqsubseteq e \sqcup_D e'\}$ is directed and contains e, e', so e and e' are consistent in U. Let $v = \bigsqcup_D V$, then $v \in U$ because $V \subseteq U$ and U is a subdomain of D. Clearly, $v \sqsubseteq e \sqcup_D e'$.

Also, $e \sqcup_D e' \sqsubseteq v$, because v is an upper bound of e and e'. Hence $v = e \sqcup_D e' = e \sqcup_U e'$ is an upper bound for e and e' in U.

Conversely, suppose that whenever e and e' are consistent in D, then e, e' are consistent in U, and $e \sqcup_U e' = e \sqcup_D e'$. We claim that for each $d \in D$, the set $U_d = \{e \in U : e \sqsubseteq d\}$ is directed. If $e, e' \in U_d$, then $e \sqsubseteq d$ and $e' \sqsubseteq d$, so e, e' are consistent in D, hence e, e' are consistent in U, and $e \sqcup_U e' = e \sqcup_D e' \sqsubseteq d$. Thus $e \sqcup_U e' \in U_d$. ∎

An *interval* of a domain D is a pair $I = (\mathrm{dom}(I), \mathrm{cod}(I)) \in D \times D$, with $\mathrm{dom}(I) \sqsubseteq \mathrm{cod}(I)$. Intervals I and J are *coinitial* if $\mathrm{dom}(I) = \mathrm{dom}(J)$. The interval I is an *identity* if $\mathrm{dom}(I) = \mathrm{cod}(I)$, and a nonidentity interval I is *prime* if there exists no $d \in D$ with $\mathrm{dom}(I) \sqsubset d \sqsubset \mathrm{cod}(I)$. It is $\sqcup$-*prime* if there exists a finite, nonempty set U of prime and identity intervals, such that $\mathrm{dom}(J) = \mathrm{dom}(I)$ for all $J \in U$, and $\mathrm{cod}(I) = \bigsqcup\{\mathrm{cod}(J) : J \in U\}$. Coinitial intervals I and J are *consistent* if $\mathrm{cod}(I)$ and $\mathrm{cod}(J)$ are consistent elements of D. If I and J are consistent, then the *residual of I after J* is the interval $I \backslash J = (\mathrm{cod}(J), \mathrm{cod}(I) \sqcup \mathrm{cod}(J))$.

In the sequel, we shall use the term "interval" exclusively to mean "interval with finite endpoints."

5.2 Trace Domains

Concurrent alphabets generate domains. Formally, suppose E is a concurrent alphabet. Let E^* denote the free monoid generated by E, then there is a least congruence $\sim$ on E^* such that $e \|_E e'$ implies $ee' \sim e'e$ for all $e, e' \in E$. The quotient $E^* /\!\sim$ is the *free partially commutative monoid* generated by E, and its elements are called *traces*. Define the relation $\sqsubseteq$ on $E^* /\!\sim$ by $[x] \sqsubseteq [y]$ iff there exists $z \in E^*$ with $[xz] = [y]$. This relation is a partial order with respect to which every consistent pair of elements has a least upper bound (this is not completely trivial to prove). We call $\sqsubseteq$ the *prefix* relation. By forming the ideal completion of the poset $(E^* /\!\sim, \sqsubseteq)$, we obtain a domain $\widehat{E}$. The map taking E to $\widehat{E}$ extends to a functor $(\widehat{-}) : \mathbf{Alph} \to \mathbf{Dom}$.

We may think of the elements of $\widehat{E}$ as equivalence classes of finite and infinite strings. The finite elements of E are the equivalence classes of finite strings. All the finite strings that are representatives of a given finite element of $\widehat{E}$ are permutations of each other, hence have the same length and the same number of occurrences of each element of E. The prime intervals of $\widehat{E}$ are those of the form $([x], [xe])$, where $x \in E^*$ and $e \in E$, and the $\sqcup$-prime intervals are those of the form $([x], [xe_1 \dots e_n])$, where $x \in E^*$ and $\{e_1, \dots, e_n\} \in \mathrm{Com}(E)$. Distinct coinitial prime intervals $I = ([x], [xe])$ and $I' = ([x], [xe'])$ are consistent iff $e \|_E e'$, in which case $I \setminus I' = ([xe'], [xe'e])$.

Lemma 5.2 *If E is a concurrent alphabet, then the domain $\widehat{E}$ is finitary.*

Proof – The finite elements of E are permutation equivalence classes of finite strings, and for finite $[x], [y]$ we have $[y] \sqsubseteq [x]$ iff there exists z with $[yz] = [x]$. Thus, the cardinality of $\{[y] : [y] \sqsubseteq [x]\}$ is bounded by the number of prefixes of permutations of x, which is finite. ∎

A *trace domain* is a pair (E, D), where E is a concurrent alphabet and D is a normal subdomain of $\widehat{E}$, such that every prime interval of D is also a prime interval of $\widehat{E}$. If (E, D)

and (E', D') are trace domains, then a *morphism* from (E, D) to (E', D') is a morphism $\mu : E \to E'$ of concurrent alphabets such that the following two conditions hold:

1. $\widehat{\mu}(D) \subseteq D'$.
2. Whenever (d, d') is a prime interval of D, then $(\widehat{\mu}(d), \widehat{\mu}(d'))$ is a $\sqcup$-prime interval of D'.

Let **TrDom** denote the category of trace domains and their morphisms.

We say that an element d of a domain D is *secured* if there exists a finite chain

$$\perp = d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_n = d,$$

such that each interval (d_k, d_{k+1}) is prime. We call such a chain a *securing chain* for d. We say that a domain D is secured if each of its finite elements is secured. It is easy to see that finitary domains are secured. We say that a subdomain U of D is *secured in D* if each finite element of U is secured as an element of D.

Lemma 5.3 *Suppose (E, D) is a trace domain. Then D is finitary and secured in $\widehat{E}$.*

Proof – Since D is a normal subdomain of $\widehat{E}$, every finite element of D is also a finite element of $\widehat{E}$. Since $\widehat{E}$ is finitary, so is D. Now, since D is finitary we know that D is secured (in itself). Since every prime interval in D is also a prime interval of E, it follows that every securing chain for $d \in D$ is also a securing chain for $d \in \widehat{E}$. Thus D is secured in E. ∎

Automata "unwind" to trace domains, and the unwinding map gives a coreflection between **Auto** and **TrDom**. Formally, suppose $A = (E, Q, q^{\circ}, T)$ is an automaton. Each finite or infinite computation sequence

$$\gamma = q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \ldots$$

of A, determines an element $\mathrm{tr}(\gamma)$ of the domain $\widehat{E}$, according to the definition

$$\mathrm{tr}(\gamma) = \bigsqcup_k e_1 e_2 \ldots e_k,$$

where concatenation denotes multiplication in the monoid of finite traces $E^* /\!\sim$, and we identify the identity event ϵ with the monoid identity, so that ϵ does not appear in the trace $e_1 \ldots e_k$. We call $\mathrm{tr}(\gamma)$ the *trace of* the computation sequence γ. Let $\mathrm{Traces}(A)$ denote the set of all traces of initial computation sequences of A.

Theorem 6 *Suppose $A = (E, Q, q^{\circ}, T)$ is an automaton. Then $(E, \mathrm{Traces}(A))$ is a trace domain. Moreover, the map taking A to $(E, \mathrm{Traces}(A))$ is the object map of a functor* AuTr : **Auto** $\to$ **TrDom**, *which is right-adjoint to a full embedding* TrAu : **TrDom** $\to$ **Auto**.

Proof – See the Appendix. ∎

Corollary 5.1 **TrDom** *has finite limits.*

Proof – This follows immediately from the previous result, but as an aid to intuition we give the explicit constructions.

(*Terminal Object*) The trace domain $(\emptyset, \{\perp\})$ is a terminal object (in fact a zero object) in **TrDom**.

(*Products*) Suppose (E_1, D_1), (E_2, D_2), are trace domains. Let $E = E_1 \otimes E_2$ be the product of E_1 and E_2 in **Alph**, and let $\pi_i : E_1 \otimes E_2 \to E_i$ $(i \in \{1,2\})$ be the associated projections. Let D be the set of all $x \in \widehat{E}$ such that $\widehat{\pi}_1(x) \in D_1$ and $\widehat{\pi}_2(x) \in D_2$. Then (E, D), equipped with the morphisms π_1 and π_2, is a product of (E_1, D_1) and (E_2, D_2) in **TrDom**.

(*Equalizers*) Suppose $\sigma, \tau : (E_1, D_1) \to (E_2, D_2)$ is a parallel pair of morphisms in **TrDom**. Let $\mu : E \to E_1$ be an equalizer of σ and τ in **Alph**. Define an element $d \in \widehat{E}$ to be *reachable* if there exists a chain

$$\perp = d_0 \sqsubseteq d_1 \sqsubseteq \dots$$

of finite elements of $\widehat{E}$, such that $d = \bigsqcup_k d_k$, $\widehat{\mu}(d_k) \in D_1$ for all $k \geq 0$, and for all $k \geq 0$ the interval $(\widehat{\mu}(d_k), \widehat{\mu}(d_{k+1}))$ is $\sqcup$-prime. Let D be the set of all reachable elements of E. Then $\mu : (E, D) \to (E_1, D_1)$ is an equalizer of σ and τ in **TrDom**. ∎

5.3 Event Domains

We now wish to view a trace domain (E, D) as a domain in its own right, rather than as a normal subdomain of the domain $\widehat{E}$ of traces. In fact, this can be done. By discarding the concurrent alphabet component of trace domains we obtain a class of domains called "event domains," which have been studied previously. It is known that event domains are exactly those domains that are isomorphic to the "domains of configurations" of a certain kind of "event structure" [6, 20]. A byproduct of our investigation is the following representation theorem, which the author has not seen explicitly stated before.

- Event domains are precisely those domains D for which there exists a concurrent alphabet E and an embedding $f : D \to \widehat{E}$ of D as a normal subdomain of $\widehat{E}$, such that if (d, d') is a prime interval of D, then $(f(d), f(d'))$ is a prime interval of $\widehat{E}$.

The formal definition of event domains requires a few preliminaries. Suppose D is a domain with the following property:

1. $I \setminus J$ is a prime interval whenever I and J are distinct, consistent prime intervals.

Then it is not difficult to see that the same property holds if "prime" is replaced by "$\sqcup$-prime."

If $I = (d, d')$ is a $\sqcup$-prime interval of D, then define

$$\mathrm{pr}(I) = \{(d, d'') : d \sqsubset d'' \sqsubseteq d' \text{ and } (d, d'') \text{ is prime}\}.$$

Call coinitial $\sqcup$-prime intervals I and J *orthogonal* if they are consistent and $\mathrm{pr}(I) \cap \mathrm{pr}(J) = \emptyset$. Note that coinitial prime intervals are orthogonal iff they are distinct and consistent. Let $\equiv$ be the least equivalence relation on $\sqcup$-prime intervals of D such that $I \equiv I \setminus J$ whenever I and J are orthogonal $\sqcup$-prime intervals. A straightforward induction using property (1) shows that if I is prime and $I \equiv I'$, then I' is also prime.

An *event domain* is a finitary domain D that satisfies property (1) above, and in addition satisfies:

2. $I \equiv J$ implies $I = J$, whenever I, J are coinitial prime intervals.

3. If I, I', J, J' are prime intervals such that $I \equiv I'$, $J \equiv J'$, I and J are coinitial, and I' and J' are coinitial, then I and I' are consistent iff J and J' are consistent.

A *morphism* from an event domain D to an event domain D' is a strict, additive, continuous function $f : D \to D'$ with the following two additional properties:

1. Whenever I is a prime interval of D, then $f(I)$ is a $\sqcup$-prime interval of D'.

2. Whenever I, J are distinct consistent prime intervals of D, then $f(I)$ and $f(J)$ are orthogonal.

Let **EvDom** denote the category of event domains and their morphisms, then **EvDom** is a (non-full) subcategory of **Dom**.

Lemma 5.4 *If (E, D) is a trace domain, then D is an event domain. Moreover, the map taking (E, D) to D extends to a (forgetful) functor* $\mathrm{TrEv} : \mathbf{TrDom} \to \mathbf{EvDom}$.

Proof – We first show that if E is a concurrent alphabet, then $\widehat{E}$ is an event domain. We have already seen (Lemma 5.2) that $\widehat{E}$ is finitary. It remains to verify axioms (1)-(3) for event domains.

(1) An interval I of $\widehat{E}$ is prime iff it is of the form $([x], [xe])$, with $x \in E^*$ and $e \in E$. Distinct, coinitial prime intervals $I = ([x], [xe])$ and $I' = ([x], [xe'])$ are consistent iff $e \|_E e'$, in which case $I \setminus I'$ is the interval $([xe], [xee'])$, which is prime.

(2) A straightforward induction shows that if $I \equiv I'$, where $I = ([x], [xe])$ and $I' = ([x'], [x'e'])$ are prime intervals, then $e = e'$. Hence if $I \equiv I'$ and I, I' are coinitial, then $I = I'$.

(3) If distinct prime intervals $I = ([x], [xe])$ and $J = ([x], [xe'])$ are consistent, then $e \|_E e'$. If $I' = ([x'], [x'e]) \equiv I$ and $J' = ([x'], [x'e']) \equiv J$, then clearly I' and J' must also be consistent.

Next, we show that if D is is a normal subdomain of $\widehat{E}$, such that an interval of D is prime in D iff it is also prime in $\widehat{E}$, then D is also an event domain. Suppose D is such a domain. Clearly, D has property (1) of an event domain. By an induction we see that if I and J are prime intervals of D, then $I \equiv J$ in D iff $I \equiv J$ in $\widehat{E}$. Properties (2)-(3) of an event domain follow easily from this fact.

Finally, to see that TrEv is a functor, note that given a morphism $\mu : (E, D) \to (E', D')$ in **TrDom**, the map $\widehat{\mu} : \widehat{E} \to \widehat{E}'$ restricts to a map $\mathrm{TrEv}(\mu) : D \to D'$. The prime intervals of D are those of the form $I = ([x], [xe])$ where $x \in E^*$ and $e \in E$. Then $\widehat{\mu}(I) = (\widehat{\mu}([x]), \widehat{\mu}([x])\widehat{\mu}(e))$, which is a $\sqcup$-prime interval of D'. Also, if I and J are distinct, consistent prime intervals of D, then $I = ([x], [xe])$ and $J = ([x], [xe'])$, where $e \|_E e'$. Since $\mu(e) \cap \mu(e') = \emptyset$, it follows that $\widehat{\mu}(I)$ and $\widehat{\mu}(J)$ are orthogonal. Hence $\mathrm{TrEv}(\mu)$ is a morphism of event domains. ∎

Theorem 7 *The forgetful functor* $\mathrm{TrEv} : \mathbf{TrDom} \to \mathbf{EvDom}$ *has a left adjoint* EvTr*, such that* $\mathrm{TrEv} \circ \mathrm{EvTr} \simeq 1$.

Proof – See the Appendix. ∎

Corollary 5.2 **EvDom** *has finite limits.*

5.4 Summary

We have now reached our goal. Dataflow networks have an operational semantics as port automata, which we identify with the corresponding relations in **Auto**. The functor $G = \mathrm{TrEv} \circ \mathrm{AuTr} : \mathbf{Auto} \to \mathbf{EvDom}$ preserves limits, hence induces an ensemble of functors

$$G_{X,Y} : \mathbf{Reln}_{\mathbf{Auto}}(X, Y) \to \mathbf{Reln}_{\mathbf{EvDom}}(X, Y)$$

which preserve not only the operations of relabeling, sequential composition, parallel composition, and feedback, but any operations that have categorical definitions in terms of composition and limits in **Auto**. Thus, the denotational semantics, in which a dataflow network denotes an (X, Y)-relation in **EvDom**, agrees with the operational semantics given by port automata. Moreover, the semantics is not "too abstract," in the sense that networks with distinct set-theoretic input/output relations receive distinct denotations. This can be verified by observing that the set-theoretic input/output relation of a port automaton can still be extracted from the corresponding relation in **EvDom**.

6 Port Automata and Causal Relations

Not every (X, Y)-relation in **EvDom** is the relation corresponding to an (X, Y)-port automaton. There are a number of reasons why this is so, but the easiest to see is that although the receptivity condition in the definition of port automata introduces an asymmetry between input and output, there is no such asymmetry in the definition of relations in **EvDom**. We would like to have a theorem that characterizes exactly, in terms of categorical properties of **EvDom**, those (X, Y)-relations that are the relations corresponding to (X, Y)-port automata and those that are the relations corresponding to determinate (X, Y)-automata.

Note that the category **EvDom** has a substantial amount of structure with which to express such properties; for example, its hom-sets are partially ordered by virtue of its being a subcategory of the category **Dom**. At the moment, we have no exact characterization theorem, but we do have some partial results. In particular, we can show that the relation in **EvDom** corresponding to an (X, Y)-port automaton is *causal*, in a sense to be defined formally below, and that the class of causal relations in **EvDom** is closed under those relational operations that make sense for port automata. The notion of causality captures some of the input/output asymmetry exhibited by port automata.

Formally, let categories $\mathbf{T}$ and $\mathbf{C}$, and functor $F : \mathbf{T} \to \mathbf{C}$ be as in Section 2. Suppose further that the category $\mathbf{C}$ is *pointed*; that is, it has a unique zero morphism $0_{A,B} : A \to B$ for every pair of objects A, B. Then an (X, Y)-relation R in $\mathbf{C}$ is *causal* if there exists a morphism $m : FX \to R$ such that $\underline{r}m = \mathrm{id}_{FX}$ and $\bar{r}m = 0$. Then $\underline{r}$ is a retraction and m is a section.

Theorem 8 *The class of causal relations in* $\mathbf{C}$ *is closed under the following operations:*

1. *Output relabeling by arbitrary morphisms and input relabeling by retractions.*
2. *Sequential composition on the output with arbitrary relations (hence also sequential composition on the input with causal relations).*
3. *Parallel composition.*
4. *Feedback.*

Proof – Omitted. ∎

Lemma 6.1 *The category* **EvDom** *is pointed.*

Proof – The one-point domain is a zero object. ∎

Theorem 9 *If R is the (X, Y)-relation in* **EvDom** *corresponding to an (X, Y)-port automaton, then R is causal.*

Proof – The receptivity property of an (X, Y)-port automaton A implies that for each input history $H \in \mathrm{Hist}(X)$, there is a corresponding initial computation sequence γ such that γ consists entirely of input or identity transitions, and $H_\gamma | X = H$. The map $m_0 : \mathrm{Hist}(X) \to R$ that takes each such H to the trace $\mathrm{tr}(\gamma) \in R$ is the m required to show R causal. ∎

It appears that we can go quite a bit further than this. By taking into account the order relation $\leq$ on the hom-sets of **EvDom**, one can see that if R is an (X, Y)-relation in **EvDom** corresponding to an (X, Y)-port automaton, then the section $m_0 : \mathrm{Hist}(X) \to R$ produced by the construction in the proof of the previous theorem is characterized by the properties $\underline{r}m_0 = \mathrm{id}_{\mathrm{Hist}(X)}$ and $m_0\underline{r} \leq \mathrm{id}_R$. That is, $(m_0, \underline{r})$ is an embedding-projection pair. One may also observe that for relations R corresponding to determinate automata, the class of all sections m, such that $\underline{r}m = \mathrm{id}_{\mathrm{Hist}(X)}$, is directed by $\leq$ and has m_0 as a least element. By taking the least upper bound of this collection (working now in the full subcategory of **Dom** whose objects are the event domains), we obtain a continuous map $\mu : \mathrm{Hist}(X) \to R$. By composing with $\bar{r}$, we obtain a continuous map $\bar{r}\mu : \mathrm{Hist}(X) \to \mathrm{Hist}(Y)$. Thus we see that determinate automata determine continuous maps from input to output. We expect that this observation can form the basis for a nice connection between the category-theoretic semantics of feedback we gave here in terms of equalizers, and the order-theoretic version given by Kahn in terms of least fixed points. We are currently attempting to work out the details of this connection.

7 Conclusion

By showing that the algebra of causal relations in **EvDom** constitutes a correct semantics for dataflow networks, we obtain a substantial amount of algebraic machinery for reasoning about such networks. Further clarification of the structure of those relations in **EvDom** that correspond to port automata should yield even more machinery. The study of relations in **EvDom** may also yield useful characterizations of "observational equivalence" of networks, and information about the structure of the fully abstract semantics with respect to this equivalence.

A Appendix: Proofs of Theorems 6 and 7

Theorem 6 *Suppose $A = (E, Q, q^{\circ}, T)$ is an automaton. Then $(E, \mathrm{Traces}(A))$ is a trace domain. Moreover, the map taking A to $(E, \mathrm{Traces}(A))$ is the object map of a functor* $\mathrm{AuTr} : \mathbf{Auto} \to \mathbf{TrDom}$, *which is right-adjoint to a full embedding* $\mathrm{TrAu} : \mathbf{TrDom} \to \mathbf{Auto}$.

Proof – We first show that $\mathrm{Traces}(A)$ is a normal subdomain of $\widehat{E}$. To do this requires a detailed analysis of the structure of the set of initial computation sequences of A, and we perform this analysis using the notion of the "residual" $\gamma \setminus \delta$ of one finite computation sequence γ "after" another finite computation sequence δ. Intuitively, $\gamma \setminus \delta$ is obtained from γ by cancelling out transitions that, "up to permutation equivalence," also appear in δ.

Formally, the residual operation is a partial binary operation on coinitial pairs of computation sequences. We first define $\setminus$ for single transitions, and then extend to arbitrary finite computation sequences by induction on their length. For single transitions, suppose $t : q \xrightarrow{a} r$ and $u : q \xrightarrow{b} s$. If $a = \epsilon$, then $t \setminus u = (s \xrightarrow{\epsilon} s)$ and $u \setminus t = u$. If $a = b \neq \epsilon$, then we define $t \setminus u = (s \xrightarrow{\epsilon} s) = (r \xrightarrow{\epsilon} r) = u \setminus t$. If $\epsilon \neq a \neq b \neq \epsilon$, then $t \setminus u$ is defined iff $a \|_E b$, in which case the commutativity property of A implies there must exist (necessarily unique) transitions $s \xrightarrow{a} p$ and $r \xrightarrow{b} p$, which we take as $t \setminus u$ and $u \setminus t$, respectively.

Next, we extend to arbitrary finite computation sequences. Suppose $\gamma : q \to r$ and $\delta : q \to s$ are coinitial finite computation sequences. Then

1. If $|\gamma| = 0$ then $\gamma \setminus \delta = \mathrm{id}_s$, where id_s denotes the length-0 computation sequence from state s.

2. If $|\gamma| > 0$ and $|\delta| = 0$, then $\gamma \setminus \delta = \gamma$.

3. If $\gamma = t\gamma'$, δ is the single transition u, $t \setminus u$ is defined, and $\gamma' \setminus (u \setminus t)$ is defined, then
$$\gamma \setminus \delta = (t \setminus u)(\gamma' \setminus (u \setminus t)).$$

4. If $|\gamma| > 0$, $\delta = u\delta'$ with $|\delta'| > 0$, $\gamma \setminus u$ is defined, and $(\gamma \setminus u) \setminus \delta'$ is defined, then
$$\gamma \setminus \delta = (\gamma \setminus u) \setminus \delta'.$$

For a more detailed explanation of this operation and its properties, the reader is referred to [13, 18].

We now use $\setminus$ to define a relation $\precsim$ on initial computation sequences as follows: For finite sequences γ and δ, define $\gamma \precsim \delta$ to hold precisely when $\gamma \setminus \delta$ is a sequence of identity transitions. Extend this definition to infinite sequences by defining $\gamma' \precsim \delta'$ iff for every finite prefix γ of γ', there exists a finite prefix δ of δ', such that $\gamma \precsim \delta$.

We observe the following facts about $\precsim$:

1. The relation $\precsim$ is a preorder, and the set of $\precsim$-equivalence classes, equipped with the induced partial ordering, is a domain whose finite elements are precisely the equivalence classes of finite initial computation sequences.

2. The map taking each $\precsim$-equivalence class to its trace is strict, additive, and continuous.

3. Initial computation sequences γ and δ have a $\precsim$-upper bound (given by $\gamma(\delta \setminus \gamma)$ or $\delta(\gamma \setminus \delta)$, which are $\precsim$-equivalent) iff the traces $\mathrm{tr}(\gamma)$ and $\mathrm{tr}(\delta)$ are consistent.

4. $\gamma \precsim \delta$ iff $\mathrm{tr}(\gamma) \sqsubseteq \mathrm{tr}(\delta)$.

Facts (1) and (2) are shown in [13] using the properties of the residual operation developed there. Facts (3) and (4) can be verified by straightforward inductive arguments from the definition of the residual operation. It follows from these facts and Lemma 5.1 that the map taking each $\precsim$-equivalence class to its trace is an isomorphism of the domain of equivalence classes of initial computation sequences of A to a normal subdomain of $\widehat{E}$. Since this subdomain is $\mathrm{Traces}(A)$, we conclude that $\mathrm{Traces}(A)$ is a normal subdomain of $\widehat{E}$.

To complete the proof that $(E, \mathrm{Traces}(A))$ is a trace domain, we observe that the prime intervals $([\gamma], [\delta])$ in the domain of equivalence classes of initial computation sequences of A are those for which $\delta \setminus \gamma$ contains precisely one nonidentity transition. Since this implies that $\mathrm{tr}(\delta)$ is longer than $\mathrm{tr}(\gamma)$ by one symbol, it follows that prime intervals in $\mathrm{Traces}(A)$ are also prime intervals in $\widehat{E}$.

It remains to be shown the map AuTr, taking A to $(E, \mathrm{Traces}(A))$, is the object map of a functor that is right-adjoint to a full embedding $\mathrm{TrAu} : \mathbf{TrDom} \to \mathbf{Auto}$. Given a trace domain (E, D), define the automaton $\mathrm{TrAu}(E, D) = (E, Q, q^\circ, T)$ as follows:

- Let Q be the set of finite elements of D, with $q^o = \bot$. Then Q is a set of equivalence classes of finite strings.

- Let T contain all transitions $[x] \xrightarrow{\epsilon} [x]$ with $[x] \in Q$ and all transitions $[x] \xrightarrow{e} [xe]$ with both $[x]$ and $[xe]$ in Q.

It is straightforward to verify that $\mathrm{TrAu}(E, D)$ is an automaton. Moreover, each morphism $\phi : (E, D) \to (E', D')$ in **TrDom** determines a morphism $\mathrm{TrAu}(\phi) : \mathrm{TrAu}(E, D) \to \mathrm{TrAu}(E', D')$, where $\mathrm{TrAu}(\phi)_{\mathrm{e}} = \phi$, and $\mathrm{TrAu}(\phi)_{\mathrm{s}}$ is the restriction of $\widehat{\phi}$ to a map from finite elements of D to finite elements of D'. To see that $\mathrm{TrAu}(\phi)$ is in fact a morphism of automata, note that the nonidentity transitions t of $\mathrm{TrAu}(E, D)$ correspond bijectively to the prime intervals $([x], [xe])$ in D. For each such interval, $(\widehat{\phi}([x]), \widehat{\phi}([xe]))$ is a $\sqcup$-prime interval of D'. hence is of the form

$$([x'], [x'e'_1] \sqcup \ldots \sqcup [x'e'_n]),$$

where $\{e'_1, \ldots, e'_n\}$ is an arbitrary enumeration of $\phi(\{e\})$, and $[x'e'_1], \ldots, [x'e'_n]$ are all in D'. It follows from this that to each enumeration $\{e'_1, \ldots, e'_n\}$ of $\phi(\{e\})$ there corresponds a computation sequence

$$\mathrm{TrAu}(\phi)_{\mathrm{s}}([x]) = [x'] \xrightarrow{e'_1} [x'e'_1] \xrightarrow{e'_2} ([x'e'_1] \sqcup [x'e'_2]) \xrightarrow{e'_3} \ldots \xrightarrow{e'_n} ([x'e'_1] \sqcup \ldots \sqcup [x'e'_n]) = \mathrm{TrAu}(\phi)_{\mathrm{s}}([xe])$$

of $\mathrm{TrAu}(E', D')$.

Clearly, the functor TrAu is faithful and injective on objects. It is also full, because if $(\psi_{\mathrm{e}}, \psi_{\mathrm{s}}) : \mathrm{TrAu}(E, D) \to \mathrm{TrAu}(E', D')$, then (1) an induction on the length of securing chains shows that $\psi_{\mathrm{s}}([x]) = \widehat{\psi}_{\mathrm{e}}([x])$ for each finite trace $[x] \in D$, and (2) it follows from the defining properties of automata that if $([x], [xe])$ is a prime interval of $\mathrm{Traces}(\mathrm{TrAu}(E, D))$, then $(\widehat{\psi}_{\mathrm{e}}([x]), \widehat{\psi}_{\mathrm{e}}([xe]))$ is a $\sqcup$-prime interval of $\mathrm{Traces}(\mathrm{TrAu}(E', D'))$. Hence if $\phi : (E, D) \to (E', D')$ is the morphism in **TrDom** with ϕ and ψ_{e} the same morphism of concurrent alphabets, then $\mathrm{TrAu}(\phi) = \psi$.

Given an automaton A, define the "evaluation map" $\varepsilon_A : \mathrm{TrAu}(\mathrm{AuTr}(A)) \to A$ to be the identity on events and to take each state $[x]$ of $\mathrm{TrAu}(\mathrm{AuTr}(A))$ to the state $\mathrm{cod}(\gamma)$ of A, where γ is an initial computation sequence of A having trace $[x]$. Since all such computation sequences γ determine the same state $\mathrm{cod}(\gamma)$ by commutativity, we know that ε_A is well-defined. To see that ε_A is in fact a morphism of automata, suppose that $[x] \xrightarrow{e} [xe]$ is a transition of $\mathrm{TrAu}(\mathrm{AuTr}(A))$. Then there exist initial computation sequences γ and δ of A, such that γ has trace $[x]$ and δ has trace $[xe]$. Then $\delta \setminus \gamma : \mathrm{cod}(\gamma) \to \mathrm{cod}(\delta)$ is a computation sequence of A that contains precisely one nonidentity transition $t : \mathrm{cod}(\gamma) \to \mathrm{cod}(\delta)$. Hence, to every enumeration of $\{e\}$ of $\varepsilon_A(\{e\})$ there corresponds a computation sequence $\mathrm{cod}(\gamma) \xrightarrow{e} \mathrm{cod}(\delta)$ of A, as required to show that ε_A is a morphism.

We claim that ε_A is universal from TrAu to A. Suppose $\psi : \mathrm{TrAu}(E', D') \to A$ is a morphism in **Auto**. We claim that there is a unique morphism $\phi : (E', D') \to \mathrm{AuTr}(A)$ in **TrDom** such that $\varepsilon_A \circ \mathrm{TrAu}(\phi) = \psi$. Since ε_A is the identity on events, ϕ and ψ must be the same morphism of concurrent alphabets, thus there is only one possible definition for ϕ. To show that this definition actually yields a morphism $\phi : (E', D') \to \mathrm{AuTr}(A)$ of trace domains, we must show: (1) that $\widehat{\phi}(D') \subseteq \mathrm{Traces}(A)$, and (2) if $([x'], [x'e'])$ is a prime interval in D', then $(\widehat{\phi}([x']), \widehat{\phi}([x'e']))$ is a $\sqcup$-prime interval in $\mathrm{Traces}(A)$.

To show (1), observe that $\widehat{\phi}(D') = \widehat{\psi}(D')$. An induction on the length of securing chains in D' shows that each finite $[x'] \in D'$ determines an initial computation sequence γ of A with $\mathrm{tr}(\gamma) = \widehat{\psi}([x'])$ and $\mathrm{cod}(\gamma) = \psi_{\mathrm{s}}([x'])$. Thus, $\widehat{\psi}([x']) \in \mathrm{Traces}(A)$ for all finite $[x'] \in D'$. By continuity, $\widehat{\psi}(D') \subseteq \mathrm{Traces}(A)$.

To show (2), it suffices to show that if $[x] = \widehat{\phi}([x'])$ then $[xe] \in \mathrm{Traces}(A)$ for each $e \in \phi(\{e'\})$. Given $[x']$, obtain an initial computation sequence γ' of $\mathrm{TrAu}(E', D')$ with $\mathrm{cod}(\gamma') = [x']$, and a transition $t' : [x'] \xrightarrow{e'} [x'e']$ of $\mathrm{TrAu}(E', D')$. Using the fact that ψ is a morphism of automata, we may construct an initial computation sequence γ of A, with $\mathrm{tr}(\gamma) = [x]$ and $\mathrm{cod}(\gamma) = \psi_{\mathrm{s}}([x'])$. Now, e' is enabled in state $[x']$ of A', so each element e of $\phi(\{e'\})$ is enabled in state $\mathrm{cod}(\gamma) = \psi_{\mathrm{s}}([x'])$ of A. It follows that $[xe] \in \mathrm{Traces}(A)$ for each $e \in \phi(\{e'\})$. ∎

Theorem 7 *The forgetful functor* TrEv : **TrDom** $\to$ **EvDom** *has a left adjoint* EvTr*, such that* $\mathrm{TrEv} \circ \mathrm{EvTr} \simeq 1$.

Proof – We first define a mapping from event domains to trace domains. Given an event domain D, define the *events* of D to be the $\equiv$-classes of prime intervals of D. Let E_D be the set of all events of D. Say that events $e, e' \in E_D$ *commute*, and write $e \| e'$, when $e \neq e'$ and there exist representatives $I \in e$ and $I' \in e'$, such that I and I' are consistent. (By the properties of event domains, this implies that whenever $I \in e$ and $I' \in e'$ are coinitial, then they are consistent.) Then E_D, equipped with the relation $\|$, is a concurrent alphabet. Let A be the automaton $(E_D, Q, \perp, T)$, where Q is the set of finite elements of D and T contains all transitions $q \xrightarrow{\epsilon} q$ and all transitions $q \xrightarrow{[I]} r$ such that $I = (q, r)$ is a prime interval of D. Let $U_D = \mathrm{Traces}(A) \subseteq \widehat{E}_D$. It follows from Theorem 6 that (E_D, U_D) is a trace domain.

Next, we claim that D and U_D are isomorphic. To see this, note that the finite initial computation sequences γ of A are in bijective correspondence with the securing chains for finite elements d of D. Moreover, by definition of the commutativity relation $\|$ on E_D, any two securing chains for the same element d correspond to computation sequences having the same trace $[x] \in \widehat{E}_D$. Thus the mapping that takes d to the trace $[x]$ determined by a securing chain for d is a bijection between the finite elements of D and the finite elements of $U_D = \mathrm{Traces}(A)$. It can be shown that this mapping is monotone, hence by continuous extension we obtain an isomorphism $\mu_D : D \to U_D$.

Finally, we show that the forgetful functor TrEv has a left adjoint whose object map takes D to (E_D, U_D). Given D, let the "inclusion of generators" be the isomorphism $\mu_D : D \to U_D$ constructed above. We claim that μ_D is universal from D to TrEv. Suppose $(E', D') \in$ **TrDom** and $\phi : D \to D'$ in **EvDom** are given. Note that each prime interval I of D determines a $\sqcup$-prime interval $\phi(I)$ of D', which in turn determines a set $V_I \in \mathrm{Com}(E')$. If I and J are distinct consistent intervals, then $\phi(I)$ and $\phi(J)$ are orthogonal by the properties of the **EvDom**-morphism ϕ, hence $V_I \cap V_J = \emptyset$. Then an induction using the definition of

$\equiv$ shows that if $I \equiv I'$, then $\phi(I) \equiv \phi(I')$, hence $V_I = V_{I'}$. Thus, the map taking each equivalence class $[I]$ to the corresponding V_I determines an **Alph**-morphism $\rho : E_D \to E'$.

Now, an induction on the length of securing chains in U_D shows that $\widehat{\rho}([x]) = \phi(\mu_D^{-1}([x]))$ for all finite $[x] \in U_D$, so $\widehat{\rho}(\mu_D(d)) = \phi(d) \in D'$ for all finite $d \in D$, hence for all $d \in D$ by continuity. Also, if $I = ([x],[xe])$ is a prime interval of U_D, then $e = [(d,d')]$, where $d = \mu_D^{-1}([x])$ and $d' = \mu_D^{-1}([xe])$. Since then $\widehat{\rho}(I) = (\phi(d), \phi(d'))$, which is a $\sqcup$-prime interval of D', it follows that $\widehat{\rho}$ maps prime intervals of U_D to $\sqcup$-prime intervals of D'. Thus, we have shown that $\rho : (E_D, U_D) \to (E', D')$ is a **TrDom**-morphism, and $\mathrm{TrEv}(\rho) \circ \mu_D = \phi$.

Finally, we note that any **TrDom**-morphism $\rho : (E_D, U_D) \to (E', D')$ satisfying $\mathrm{TrEv}(\rho) \circ \mu_D = \phi$ must satisfy $\widehat{\rho}([xe]) = \phi(\mu_D^{-1}([xe]))$ for all finite prime intervals $I = ([x],[xe]) \in U_D$. Let $d = \mu_D^{-1}([x])$ and $d' = \mu_D^{-1}([xe])$, then since $\widehat{\rho}([xe]) = \widehat{\rho}([x])\widehat{\rho}(e) = \phi(d)\widehat{\rho}(e)$ and $\phi(d') = \phi(d)[e'_1 \dots e'_n]$ for some $\{e'_1, \dots, e'_n\} \in \mathrm{Com}(E')$, it must be the case that $\widehat{\rho}(e) = [e'_1 \dots e'_n]$ and $\rho(\{e\}) = \{e'_1, \dots, e'_n\}$. Since every $e \in E_D$ is $[(d,d')]$ for some prime interval $(d,d') \in D$, for each $e \in E_D$ we can find a corresponding prime interval $([x],[xe])$ in U_D by taking $[x] = \mu_D(d)$ and $[xe] = \mu_D(d')$. It follows that the condition $\mathrm{TrEv}(\rho) \circ \mu_D = \phi$ uniquely determines ρ. ▮

References

[1] I. J. Aalbersberg and G. Rozenberg. Theory of traces. *Theoretical Computer Science*, 60(1):1–82, 1988.

[2] M. Bednarczyk. *Categories of Asynchronous Systems.* PhD thesis, University of Sussex, October 1987.

[3] J. D. Brock. *A Formal Model of Non-Determinate Dataflow Computation.* PhD thesis, Massachusetts Institute of Technology, 1983. Available as MIT/LCS/TR-309.

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

[5] M. Broy. Nondeterministic data-flow programs: how to avoid the merge anomaly. *Science of Computer Programming*, 10:65–85, 1988.

[6] P.-L. Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming. Research Notes in Theoretical Computer Science*, Pitman, London, 1986.

[7] A. A. Faustini. An operational semantics for pure dataflow. In *Automata, Languages, and Programming, 9th Colloquium*, pages 212–224, Springer-Verlag. Volume 140 of *Lecture Notes in Computer Science*, 1982.

[8] H. Herrlich and G. E. Strecker. *Category Theory. Sigma Series in Pure Mathematics*, Heldermann Verlag, 1979.

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

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

[11] S. Mac Lane. *Categories for the Working Mathematician.* Volume 5 of *Graduate Texts in Mathematics*, Springer Verlag, 1971.

[12] A. Mazurkiewicz. Trace theory. In *Advanced Course on Petri Nets*, GMD, Bad Honnef, September 1986.

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

[14] G. D. Plotkin. Domains: lecture notes. 1979. (unpublished manuscript).

[15] D. A. Schmidt. *Denotational Semantics: A Methodology for Language Development.* Allyn and Bacon, 1986.

[16] E. W. Stark. Concurrent transition system semantics of process networks. In *Fourteenth ACM Symposium on Principles of Programming Languages*, pages 199–210, January 1987.

[17] E. W. Stark. Concurrent transition systems. *Theoretical Computer Science*, 1989. (to appear).

[18] E. W. Stark. Connections between a concrete and abstract model of concurrent systems. In *Fifth Conference on the Mathematical Foundations of Programming Semantics*, Springer-Verlag. *Lecture Notes in Computer Science*, New Orleans, LA, 1989 (to appear).

[19] E. W. Stark. *On the Relations Computed by a Class of Concurrent Automata.* Technical Report 88-09, SUNY at Stony Brook Computer Science Dept., 1988.

[20] G. Winskel. *Events in Computation.* PhD thesis, University of Edinburgh, 1980.