Dataflow Networks are Fibrations

Eugene W. Stark*

Department of Computer Science

State University of New York at Stony Brook

Stony Brook, NY 11794 USA

(stark@cs.sunysb.edu)

Abstract

Dataflow networks are a paradigm for concurrent computation in which a collection of concurrently and asynchronously executing processes communicate by sending messages over FIFO message channels. In a previous paper, we showed that dataflow networks could be represented as certain spans in a category of automata, or more abstractly, in a category of domains, and we identified some universal properties of various operations for building networks from components. Not all spans corresponded to dataflow processes, and we raised the question of what might be an appropriate categorical characterization of those spans that are "dataflow-like." In this paper, we answer this question by obtaining a characterization of the dataflow-like spans as *split right fibrations*, either in a 2-category of automata or a 2-category of domains. This characterization makes use of the theory of fibrations in a 2-category developed by Street. In that theory, the split right fibrations are the algebras of a certain doctrine (or 2-monad) R on a category of spans. For the 2-categories we consider, R has a simple interpretation as an "input buffering" construction.

1 Introduction

Dataflow networks [4, 5] are a paradigm for concurrent computation in which a collection of concurrently and asynchronously executing processes communicate by sending messages over FIFO message channels. *Determinate* dataflow networks compute continuous functions from input message histories to output message histories, and have a well-understood theory. Less developed is the theory of *indeterminate* or non-functional networks. These more general networks are especially interesting because they exhibit both concurrency and indeterminacy, and insight gained from their study will likely contribute to a better overall understanding of these two concepts.

This paper is part of a research program aimed at finding the correct algebraic setting for the study of indeterminate dataflow networks. We wish to view dataflow networks as the elements of an algebra whose operations represent ways to build networks from

*Research supported in part by NSF Grant CCR-8902215.

components, and we would like to understand fully the notions of behavioral equivalence that are appropriate in this context. For some time, we have been studying a particular automata-theoretic model for dataflow networks, in an attempt to identify whatever useful algebraic structure might be present. Based on the progress we have made so far [8, 9, 10, 13], a general structure appears to be emerging. However, all is not yet completely clear, and it continues to be difficult to identify and separate the important structure from the incidental artifacts of the model.

In a previous paper [9] we showed that a dataflow network with input "ports" X and output ports Y could be represented as a span from FX to FY (*i.e.* a diagram $FY \xleftarrow{g} A \xrightarrow{f} FX$) in a finitely complete category **Auto** of concurrent automata. Here F is a suitable functor that associates "objects of inputs" FX and FY in **Auto** with finite sets of ports X and Y. We showed that various constructions, corresponding intuitively to ways of composing smaller networks into larger ones, could be defined in terms of limits in **Auto**. In particular, the operation of "feeding back" outputs to inputs was defined in terms of equalizers. We also showed that dataflow networks could be modeled more abstractly as spans in a category of **EvDom** of "conflict event domains," and that this model is related to the more concrete automaton model by a coreflection. Consequently, operations defined in terms of limits are preserved in the passage from the more concrete model to the more abstract version.

At the end of the previous paper, we noted several interesting properties, valid in the domain-theoretic model, of spans corresponding to dataflow processes, and we raised the question of what might be the correct categorical characterization of the "dataflow-like" spans. A proper answer to this question would be prerequisite to the construction of a fully categorical theory of dataflow networks. In the present paper, we obtain a characterization of dataflow-like spans as *split right fibrations*, either in a 2-category of automata, or in a 2-category of domains. The fact that essentially the same characterization holds in both cases lends credence to the idea that it is in fact the correct categorical notion. Further support comes from an intuitive interpretation of the definition of fibration. Fibrations in a 2-category are defined to be the algebras of a certain "doctrine," or 2-monad. In the present situation, this doctrine corresponds to the construction "compose with an input buffer." Thus, the dataflow-like spans are those spans that are algebras of the input buffering doctrine.

The theory of fibrations was first developed in terms of concrete constructions on categories [3]. Then, Street [14, 15], building on work of Gray [2], showed that this theory has a bicategorical formulation, which can be applied not only to the 2-category **Cat**, but to any bicategory with sufficient completeness properties. Here, we examine how the theory applies to the category **Auto** of automata and the category **EvOrd** of "conflict event orderings," which is equivalent to the category **EvDom** of our previous paper. These categories have 2-categorical structure we have not exploited until now. As a technical matter, the 2-categories **Auto** and **EvOrd** do not quite have the necessary completeness properties (existence of comma objects and certain 2-pullbacks), so our results are complicated somewhat by the necessity of enlarging them to 2-categories **AutoWk**, of "automata and weak morphisms," and **EvOrdWk**, of "conflict event orderings and sup-preserving maps." We determine the structure of the split right fibrations in each of the

2-categories **AutoWk** and **EvOrdWk**. Our main results are: (1) the dataflow-like spans in **Auto** are exactly those that are split right fibrations in **AutoWk** with an **Auto**-morphism as cleavage, and (2) the dataflow-like spans in **EvOrd** are exactly those that are split right fibrations in **EvOrdWk** with an **EvDom**-morphism as cleavage.

Some further comments are in order concerning the general directions envisioned for this research. We would like very much to be able to define a notion of "dataflow model," which would permit both the study of the algebra of network-forming operations, and the comparison of different dataflow models by means of homomorphisms. In the author's opinion, the evidence at present suggests quite strongly that a dataflow model ought to be a certain kind of bicategory, whose objects are "types," whose arrows are "processes," and whose 2-cells are morphisms of processes. Composition of arrows would correspond to "sequential composition" of processes, in which the output of one process becomes the input of another. The bicategory would also be equipped with a functorial tensor product $\otimes$, which would provide a way to form the "parallel composition" $p \otimes p' : X \otimes X' \to Y \otimes Y'$ of processes $p : X \to Y$ and $p' : X' \to Y'$. It ought to be possible to define other operations on processes using bicategorical constructions, although in some cases (in particular the feedback operation) this remains problematic. At present, we have several examples of this type of bicategorical model for dataflow networks, but the correct general formulation seems elusive. The "bicategories of spans" of Carboni and Walters [1] represent an example of the structure we might hope to find.

We hope that the organization chosen for the rest of this paper will make the motivations from the intended application area clear both to category theorists and to computer scientists, and will hold the attention, at least for a little while, of computer scientists unfamiliar with 2-categories. To this end, we have kept Section 2 of the paper almost completely free of 2-categories. Instead, we define the ordinary categories **Auto** of automata and **AutoWk** of automata and weak morphisms, ignoring their 2-categorical structure, and we show how certain spans in these categories model processes that consume inputs and produce outputs. We show that these "monotone automata" are in fact the algebras of an "input buffering monad" on a category of spans in **AutoWk**. This result prepares the connection, made in Section 3, with fibrations in **AutoWk**. For this, the use of 2-categories is necessary, and the reader is referred to [6, 14, 15] for the basic terminology and notations. In Section 4, we apply the theory to the 2-category **EvOrdWk** of domains and obtain a similar characterization of the dataflow-like spans in **EvOrdWk**.

Finally, a comment on notation. In this paper, fx or $f(x)$ denotes the application of a mapping f to its argument x. Compositions are written in reverse diagrammatic order, so that gf denotes f followed by g. We extend this convention to all types of composition. For example, if $e_1, e_2, \ldots, e_n$ are elements of a set E, then the string "e_1 followed by e_2 followed by ... followed by e_n" will be denoted $e_n \ldots e_2 e_1$.

2 Trace Automata

In this section, we review the automata-theoretic model for dataflow networks presented in [9], along with its associated intuition. The objects of this model are certain spans in a category **Auto** of "trace automata." We develop various properties of these spans, to motivate the idea that they ought to be examples of fibrations.

2.1 Concurrent Alphabets and Traces

A *concurrent alphabet* is a pair $(E, \|)$, where E is a set, and $\|$ is a symmetric, irreflexive relation on E, called the *concurrency relation.* If $e \| e'$, then we say that e and e' are *concurrent* or that they *commute.* A set $U \subseteq E$ is called *commuting* if $e \| e'$ for all $e, e' \in U$ with $e \neq e'$, and we use $\mathrm{Comm}(E)$ to denote the set of all *finite* commuting subsets of E. Intuitively, if E is the set of basic observable actions of interest for some system, then $\mathrm{Comm}(E)$ is the set of all possible instantaneous occurrences that might be observed during an execution of that system. If $U, V \in \mathrm{Comm}(E)$, then U and V are called *orthogonal*, and we write $U \perp V$, if $U \cup V \in \mathrm{Comm}(E)$ and $U \cap V = \emptyset$.

A *morphism* from a concurrent alphabet E to a concurrent alphabet E' is a function $h : \mathrm{Comm}(E) \to \mathrm{Comm}(E')$, such that

1. $h(\emptyset) = \emptyset$.
2. If $U \cup V \in \mathrm{Comm}(E)$, then $h(U) \cup h(V) \in \mathrm{Comm}(E')$, and $h(U \setminus V) = h(U) \setminus h(V)$.

The symbol $\setminus$ denotes set difference. It can be shown [9] that if $U \cup V \in \mathrm{Comm}(E)$, then $h(U \cup V) = h(U) \cup h(V)$, so that a morphism of concurrent alphabets is uniquely determined by its action on singleton sets. We often use this fact in defining particular morphisms. It can also be shown $U \perp V$ implies $f(U) \perp f(V)$ — as a special case we see that $e \| e'$ implies $f(\{e\}) \cap f(\{e'\}) = \emptyset$. Let **Alph** denote the category of concurrent alphabets and their morphisms. The identity morphism on E is just the identity function from $\mathrm{Comm}(E)$ to $\mathrm{Comm}(E)$. Composition of morphisms is function composition.

The *product* $E \otimes E'$ of concurrent alphabets E and E' is the concurrent alphabet whose set of elements is the disjoint union of the sets of elements of E and E', and whose concurrency relation extends those of E and E' by making each element of E concurrent with each element of E'. Equipping $E \otimes E'$ with the restriction maps

$$\begin{array}{rl} \text{-} \cap E: & E \otimes E' \to E \\ \text{-} \cap E': & E \otimes E' \to E' \end{array}$$

makes $E \otimes E'$ into a categorical product.

Suppose E is a concurrent alphabet. The *free partially commutative monoid* generated by E is obtained by factoring the monoid of finite sequences of elements of E by the least congruence that relates ee' and $e'e$ whenever $e \| e'$. We use E^* to denote this monoid, whose elements are called *traces* [7]. We shall find it convenient in the sequel to identify a set $U = \{e_1, \ldots, e_n\} \in \mathrm{Comm}(E)$ with the corresponding trace $e_n \ldots e_1 \in E^*$. In this way we give meaning to expressions such as $U_m U_{m-1} \ldots U_1$, where $U_i \in \mathrm{Comm}(E)$ for $1 \leq i \leq m$.

2.2 Trace Automata

"Trace automata" are transition systems whose transition labels are drawn from a concurrent alphabet. To capture the idea that "concurrent transitions commute," the definition of trace automata includes the requirement that if two transitions with concurrent labels are enabled in the same state, then they can be executed in either order with the same effect.

Formally a *trace automaton* (or more simply, an *automaton*) is a four-tuple

$$A = (E, Q, T, q_{\mathrm{I}}),$$

where

- E is a concurrent alphabet of *actions.*
- Q is a set of *states*, with $q_{\mathrm{I}} \in Q$ a distinguished *initial state.*
- $T : Q \times E \to Q$ is a partial function, called the *transition map*, such that whenever $e \| e'$, $T(q, e) = r$, and $T(q, e') = r'$, then there exists $s \in Q$ with $T(r, e') = s = T(r', e)$.

We often write $e : q \to r$ or $q \xrightarrow{e} r$ to assert that $T(q, e) = r$, and call the triples (q, e, r) such that $q \xrightarrow{e} r$ the *transitions* of A. Condition (3) in the above definition embodies the intuitive idea that the order of occurrence of concurrent actions is immaterial.

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

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

Each computation sequence γ determines a corresponding trace $\mathrm{tr}(\gamma) = e_n e_{n-1} \ldots e_1 \in E^*$. Two computation sequences are *equivalent* if they have the same trace, and the equivalence classes of computation sequences are called the *computations* of A.

A *morphism* from a trace automaton A to a trace automaton A' is a pair of maps $h = (h_{\mathrm{a}}, h_{\mathrm{s}})$, where $h_{\mathrm{a}} : E \to E'$ is a morphism of concurrent alphabets, and $h_{\mathrm{s}} : Q \to Q'$ is a function, such that:

- $h_{\mathrm{s}}(q_{\mathrm{I}}) = q'_{\mathrm{I}}$.
- For all q, $r \in Q$ and $e \in E$, if $e : q \to r$ in A and $h_{\mathrm{a}}(e) = \{e_1, e_2, \ldots, e_n\}$, then $T'(h_{\mathrm{s}}(q), e_i)$ is defined for all i with $1 \leq i \leq n$, and A' has a computation sequence

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

  with $q_0 = h_{\mathrm{s}}(q)$ and $q_n = h_{\mathrm{s}}(r)$.

The above notion of morphism was introduced in [9], where it was also shown that the resulting category **Auto** has reasonable properties. (In particular, **Auto** has finite limits, which are created by the forgetful functor from **Auto** to the product of **Alph** and the category $\mathbf{Set}_*$ of pointed sets.) The intuition behind the definition is that finite commuting sets of transitions are what a trace automaton can do in a single instantaneous step, and these are the things that ought to be preserved by their morphisms. The reason why finite commuting sets of transitions, rather than single transitions, are what correspond to instantaneous steps, can be seen by considering a finite product $A = A_1 \times \ldots \times A_n$ of automata. We wish to think of A as representing a system $\{A_1, \ldots, A_n\}$, executing concurrently and independently. In such a system, there is the possibility of the simultaneous occurrence of a transition from each of the A_i. Formally, this is reflected in the requirement that a collection of morphisms $h_i : B \to A_i$ $(1 \leq i \leq n)$ induce a unique morphism $h : B \to A$. In order for h to be a morphism, we must allow morphisms to map single transitions of B to finite commuting sets of transitions of A.

The category **Alph** is isomorphic to a full reflective subcategory of **Auto** via the functor that takes each concurrent alphabet E to the one-state automaton with alphabet E, having a transition for each of its actions. Since the embedding of **Alph** in **Auto** is right-adjoint to the forgetful functor from **Auto** to **Alph**, it preserves limits. In the sequel, it will be convenient for us to identify a concurrent alphabet E with the corresponding one-state automaton.

2.3 Monotone Automata

We wish to discuss automata that consume inputs and produce outputs. Therefore, if X and Y are concurrent alphabets, we define an *automaton from X to Y* to be a span $Y \xleftarrow{g} A \xrightarrow{f} X$ from X to Y in **Auto**. Intuitively, if $Y \xleftarrow{g} A \xrightarrow{f} X$ is an automaton from X to Y, then we think of X as an "object of inputs," whose transitions represent the possible inputs that A might consume. Similarly, we regard Y as an "object of outputs," whose transitions represent the possible outputs that A might produce. If $Y \xleftarrow{g} A \xrightarrow{f} X$ and $Y \xleftarrow{g'} A' \xrightarrow{f'} X$ are automata from X to Y, then an *arrow of spans* from A to A' is a morphism $h : A \to A'$ such that $g'h = g$ and $f'h = f$. Let $\mathbf{Auto}(X, Y)$ denote the category of automata from X to Y, with arrows of spans as morphisms.

We wish to single out automata with the following properties:

1. They are always prepared to consume arbitrary input.
2. The act of consuming input can never cause enabled output transitions to become disabled, although it may cause more output transitions to become enabled.

In previous papers, we have formalized such automata using variants of the following definition: an automaton A from X to Y is *monotone* iff the alphabet E of A is (up to isomorphism) of the form $Z \otimes X$, with $f_{\mathrm{a}} : E \to X$ the restriction to X, with $g_{\mathrm{a}} e = \emptyset$ for all $e \in X$, and in addition with the following condition satisfied:

(Receptivity) For all states q of A, and all actions $e \in X$, there exists a transition $q \xrightarrow{e} r$ of A.

The "port automata" defined in [9], are special cases of monotone automata, in which X and Y are concurrent alphabets whose elements represent the transmission of data values over finite sets of "ports," and E is required to be of the form $Y \otimes Z \otimes X$, with g the restriction to Y. We use the more general monotone automata here because they have a clean characterization in terms of fibrations.

The next result, easily proved from the definitions, gives the essential properties of monotone automata. Statement (1) is a generalization of receptivity to arbitrary traces, rather than single actions. Statement (2) states that under certain conditions, a transition of A in state q can be "pushed out" along an input computation sequence, to yield a transition of A from the final state of that computation sequence.

Proposition 2.1 *Suppose A is a monotone automaton from X to Y. Then*

1. *For all states q of A and all traces x in X^*, there exists a computation sequence of A, starting from state q and having trace x. Moreover, any two such computation sequences arrive in the same final state, which we denote by $q \cdot x$.*

2. *Suppose $q \xrightarrow{e} r$ is a transition of A, and x and x' are traces in X^*, such that $x'e = ex$. Then A also has a transition $q \cdot x \xrightarrow{e} q \cdot x'$.*

2.4 The Input Buffering Monad

Suppose $Y \xleftarrow{g} A \xrightarrow{f} X$ is an automaton from X to Y. The *input buffering construction* is a way to obtain a monotone automaton $Y \xleftarrow{g'} BA \xrightarrow{f'} X$ by "composing A with an input buffer." Formally, if $A = (E, Q, T, q_{\rm I})$, then $BA = (E', Q', T', q'_{\rm I})$, where:

- $E' = E \otimes X$.
- $Q' = Q \times X^*$. That is, the states of BA are pairs (q, x), where q is a state of A, and x is a trace in X^*. The initial state $q'_{\rm I}$ of BA is the pair $(q_{\rm I}, \epsilon)$, where ϵ is the empty trace.
- The transition map $T' : Q' \times E' \to Q'$ of BA is defined as follows:
  1. If $e \in X$, then $T'((q, x), e) = (q, ex)$.
  2. If $e \in E$, then $T'((q, x), e)$ is defined iff A has a transition $q \xrightarrow{e} r$ and there exists x' with $x = x'(fe)$, in which case $T'((q, x), e) = (r, x')$.

Intuitively, a state of BA is a pair (q, x) consisting of a state q of A and an "input buffer" x. There are two types of transitions of BA: *input* transitions (case (1) above), in which input $e \in X$ arrives and is appended to the tail of the input buffer, and *noninput* transitions (case (2) above), in which A performs a transition $q \xrightarrow{e} r$, absorbing any necessary input fe from the head of the input buffer.

Define $f' : BA \to X$ to take (q, x) to the unique state of X, and to take $e \in X$ to $\{e\}$ and $e \in E$ to $\emptyset$. Define $g' : BA \to Y$ to take (q, x) to the unique state of Y, to take $e \in X$ to $\emptyset$, and $e \in E$ to ge. It is straightforward to verify that f' and g' are morphisms of automata. We have the following result:

Proposition 2.2 *For any automaton A from X to Y, the automaton BA from X to Y is a monotone automaton. Moreover, the map taking A to BA extends in an obvious way to a functor $B : \mathbf{Auto}(X, Y) \to \mathbf{Auto}(X, Y)$.*

In fact, more can be shown. For each automaton A from X to Y, there is a morphism $\mu_A : BBA \to BA$. Intuitively, μ_A is the morphism that collapses two tandem input buffers into one, hiding actions corresponding to the transfer of input between the two buffers. Formally, the map μ_A takes a state $((q, x), x')$ of BBA to the state $(q, x'x)$ of BA. To define the behavior of μ_A on actions, observe that the alphabet of BBA is of the form $(E \otimes X) \otimes X$, and the alphabet of BA is of the form $E \otimes X$. The map μ_A takes $e \in E$ to $\{e\}$, it takes e in the "outer copy" of X to $\{e\}$, and it takes e in the "inner copy" of X to $\emptyset$.

Proposition 2.3 *The maps $\mu_A : BBA \to BA$ are morphisms of automata, which are the components of a natural transformation $\mu : BB \to B$ that satisfies the associative law $\mu \circ (B\mu) = \mu \circ (\mu B)$.*

If A is a monotone automaton from X to Y, then we may also obtain a morphism $h_A : BA \to A$. To see this, note that the alphabet E of A has the form $Z \otimes X$, and the alphabet of BA has the form $(Z \otimes X) \otimes X$. Let $h_A : BA \to A$ take each state (q,x) of BA to the state $q \cdot x$ of A (see Proposition 2.1), each action e in Z to $\{e\}$, each action e in the "outer copy" of X to $\{e\}$, and each action e in the "inner copy" of X to $\emptyset$.

Proposition 2.4 *If A is a monotone automaton from X to Y, then the map $h_A : BA \to A$ is an arrow of spans. Moreover, $h_A \circ \mu_A = h_A \circ Bh_A$.*

Proof – If h_A is a morphism of automata, then the fact that it is an arrow of spans is immediate from the definitions. To see that h_A is a morphism of automata, suppose $(q,x) \xrightarrow{e} (q',x')$ is a transition of BA. If e is an input action (*i.e.* it is in the outer copy of X), then $x' = ex$ and $q' = q$. In this case, A has a transition $q \cdot x \xrightarrow{e} q \cdot ex$, which is then the image of $(q,x) \xrightarrow{e} (q',x')$ under h_A. If e is a noninput action in the inner copy of X, then $x'e = x$ and A has a transition $q \xrightarrow{e} q'$. By Proposition 2.1, A also has a transition $q \cdot x'e \xrightarrow{e} q' \cdot x'$ which is then the image of $(q,x) \xrightarrow{e} (q',x')$ under h_A. Finally, if $e \in Z$, then $x' = x$ and A has a transition $q \xrightarrow{e} q'$. By Proposition 2.1, A also has a transition $q \cdot x \xrightarrow{e} q' \cdot x'$, which is the image of $(q,x) \xrightarrow{e} (q',x')$ under h_A. ▮

Proposition 2.3 states that B is almost the underlying functor of a monad, with the natural transformation μ as multiplication. Proposition 2.4 states that every monotone automaton from X to Y almost carries a structure of B-algebra. What is missing to make B an actual monad, and the monotone automata its actual algebras, is the monad unit, which would be a natural transformation $\eta : 1 \to B$. The components $\eta_A : A \to BA$ of such a natural transformation almost exist. More precisely, the map η_A would have to take a state q of A to the state (q, ϵ) of BA, and an action $e \in E$ to the disjoint sum $\{e\} + fe \in \mathrm{Comm}(E \otimes X)$. The only problem is that such a map η_A need not be a morphism.

One can see what goes wrong by considering the case $A = Z \otimes X$, with $f : A \to X$ the restriction to X and $g : A \to Y$ the zero map. In this case, A has just one state $*$, so up to isomorphism BA has as its states the traces in X^*, and the alphabet of BA is $(Z \otimes X) \otimes X$. The map η_A would have to take the unique state $*$ of A to the empty trace ϵ, and each action $a \in X$ to the set $\{a, a'\} \in \mathrm{Comm}((Z \otimes X) \otimes X)$, where a' is in the "inner copy" of X and a is in the "outer copy." Since A has a transition $a : * \to *$ whenever $a \in X$, for η_A to be a morphism, BA would have to have transitions from state ϵ for both actions a and a'. In fact, BA has transitions $\epsilon \xrightarrow{a} a \xrightarrow{a'} \epsilon$, but no transition for action a' in state ϵ. Intuitively, BA has the capability of accepting input a in one step and then processing it on the next step, but not of doing both in a single step. There are two ways around this problem: we can enlarge the class of objects of **Auto** to include automata capable of the behavior required of BA, or we can weaken the properties required of morphisms **Auto**, so as to include all the maps η_A. Ultimately, the first approach is probably the correct way to proceed, but the definition of a suitably general class of automata (such as the "concurrent transition systems" of [11, 12]) introduces an additional layer of abstraction that would tend to obscure the ideas we wish to convey here. We therefore follow the second approach in this paper.

Thus, a *weak morphism* from a trace automaton A to a trace automaton A' is a pair of maps $h = (h_{\mathrm{a}}, h_{\mathrm{s}})$, where $h_{\mathrm{a}} : E \to E'$ is a morphism of concurrent alphabets, and $h_{\mathrm{s}} : Q \to Q'$ is a function, such that:

- $h_{\mathrm{s}}(q_{\mathrm{I}}) = q'_{\mathrm{I}}$.
- For all $q, r \in Q$ and $e \in E$, if $e : q \to r$ in A, then for some enumeration $\{e_1, e_2, \ldots, e_n\}$ of $h_{\mathrm{a}}(e)$, the automaton A' has a computation sequence

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

with $q_0 = h_{\mathrm{s}}(q)$ and $q_n = h_{\mathrm{s}}(r)$.

The change is that we have deleted the requirement that $T'(h_{\mathrm{s}}(q), e_i)$ be defined for all i with $1 \leq i \leq n$.

Let **AutoWk** denote the category of trace automata and weak morphisms. This category is not as nice as the category **Auto**: although **AutoWk** does have finite products, which are constructed the same way as in **Auto**, it fails to have all equalizers. In addition, there is no way to extend the coreflection between **Auto** and **EvDom**, exhibited in [9], to a coreflection between **AutoWk** and some expansion of **EvDom**. For these reasons, we shall have to work with both categories **Auto** and **AutoWk**.

Proposition 2.5 *The functor B extends to an endofunctor of* $\mathbf{AutoWk}(X, Y)$. *The maps $\eta_A : A \to BA$ are weak morphisms of trace automata, which are the components of a natural transformation $\eta : 1 \to B$ that satisfies the unit laws $\mu \circ B\eta = 1 = \mu \circ \eta B$. Thus, the triple (B, η, μ) is a monad in* $\mathbf{AutoWk}(X, Y)$.

We now arrive at the main result of this section.

Theorem 1 *Suppose A is an automaton from X to Y. Then A is a monotone automaton iff there is a* **Auto***-morphism $h_A : BA \to A$ enriching A with a structure of B-algebra.*

Proof – Suppose $Y \xleftarrow{g} A \xrightarrow{f} X$ is a monotone automaton from X to Y. Let the map $h_A : BA \to A$ be as in Proposition 2.4. It is a straightforward use of the definitions to check that $h_A \circ \eta_A = 1$ and $h_A \circ \mu_A = h_A \circ (Bh_A)$, so that h_A is a B-algebra structure on A.

Conversely, suppose $h_A : BA \to A$ is an **Auto**-morphism enriching the span $Y \xleftarrow{g} A \xrightarrow{f} X$ with a structure of B-algebra. Now, the alphabet of BA is of the form $E \otimes X$, with f the restriction to X. Suppose $e \in X$. Because $fh_A = f' : BA \to A$, we must have $fh_A e = f'e = \{e\}$. There must therefore exist some particular $e' \in h_A e$ with $fe' = \{e\}$. So, for each $e \in X$, there exists $e' \in E$ with $fe' = \{e\}$. Moreover, if $e_1 \| e_2$, then $h_A e_1 \perp h_A e_2$, so we must have $e'_1 \| e'_2$. Conversely, if $e'_1 \| e'_2$, then $\{e_1\} = fe'_1 \perp fe'_2 = \{e_2\}$, so $e_1 \| e_2$. Thus, $E \simeq (E \setminus \{e' : e \in X\}) \otimes \{e' : e \in X\}$, with f the restriction on the second factor. Since the second factor is isomorphic to X, we have shown that the alphabet E of

A has the form required of a monotone automaton. To prove that A has the receptivity property, observe that given e' corresponding to $e \in X$, applying h_A to the transition $(q, \epsilon) \xrightarrow{e} (q, e)$ of BA gives a computation $h_A e : q \to r$ of A. Using the fact that $e' \in h_A e$ and the assumption that h_A is a morphism, not just a weak morphism, we see that e' is enabled for A in state q, yielding the required transition $q \xrightarrow{e'} q \cdot e$ of A. ∎

3 Fibrations between Automata

The definition of monotone automaton given in Section 2, although successful at capturing our intuition, and having a number of interesting consequences as well, is not categorical, and is therefore unsuitable as a basis for a category-theoretic study of the properties of dataflow-like networks, viewed as spans in a category of automata. The purpose of this section is to show that monotone automata can be characterized categorically as certain *split right fibrations* in a suitable 2-category of automata.

In the theory of fibrations in a 2-category developed by Street [14, 15], the (two-sided) split fibrations from X to Y in a 2-category $\mathbf{K}$ are defined to be the algebras of a certain "doctrine" (or "2-monad") M on the 2-category of spans from X to Y in $\mathbf{K}$. The structure map for an M-algebra is called a "cleavage" for the underlying span. The one-sided "left" and "right" fibrations (essentially corresponding to what were earlier called split fibrations and split op-fibrations) are also algebras of doctrines L and R, which compose according to certain distributive laws to form M.

The connection between the abstract theory of fibrations and the concrete definitions we have given so far is made when one realizes that in the automata-theoretic case, the functor R is essentially the input buffering construction B, defined concretely for trace automata in the previous section, which takes an automaton A from X to Y and composes it with an input buffer to yield a "free X-input-buffered automaton" BA from X to Y. It then follows from Theorem 1 that the monotone automata from X to Y are exactly those automata from X to Y that are split right fibrations in **AutoWk** having an **Auto**-morphism as cleavage. Dually, it is possible to identify the doctrine L as an "output buffering construction," however we do not develop that idea further in this paper.

3.1 AutoWk as a 2-Category

Suppose A is an automaton. Let A^* be the category whose objects are the states of A, and whose arrows are its computations (computation sequences modulo trace equivalence). Empty computations serve as identities and computations are composed by concatenation. We call A^* the *computation category* of A. It can be shown [12] that: (1) A^* has no nontrivial isomorphisms, (2) every arrow of A^* is both epi and mono, (3) every span $s \xleftarrow{g} q \xrightarrow{f} r$ that can be completed to a commuting square has a pushout. Moreover, if $f : A \to B$ is a morphism of **AutoWk**, then f determines a pushout-preserving functor $f^* : A^* \to B^*$, in such a way that the map taking A to A^* becomes a functor $(\text{ - })^* : \mathbf{AutoWk} \to \mathbf{Cat}$. Specifically, f^* takes each state q of A^* (which is nothing more than a state q of A) to the state fq of B^*. The action of f^* on arrows of A^* (that is, on computations of A) is determined by the fact that it is to be a functor from A^* to B^*

(hence it must preserve empty computations and concatenation of computations), and that it takes each single-transition computation $\gamma : q \xrightarrow{e} r$ of A to its image $f\gamma$ in B.

The category **AutoWk** can be made into a 2-category in such a way that $(\text{ - })^* : \mathbf{AutoWk} \to \mathbf{Cat}$ becomes a 2-functor. Specifically, if $f, g : A \to B$ in **AutoWk**, then a 2-cell from f to g is a natural transformation $\tau : f^* \Rightarrow g^*$. Identity 2-cells, along with vertical and horizontal composition, are inherited from **Cat**, and the interchange law holds because it does in **Cat**. The subcategory **Auto** becomes a 2-category in the same way.

It should be emphasized here that although **Auto** is finitely complete as a 1-category, it is not the case that all limits are 2-limits. In particular, it is not the case that every pullback is a 2-pullback. Similarly, even for the cases in which ordinary limits exist in **AutoWk**, these need not be 2-limits. We shall see, though, that enough 2-pullbacks do exist in **AutoWk** to satisfy our needs.

3.2 Comma Objects

A *comma object* [14] for an opspan $Y \xrightarrow{g} A \xleftarrow{f} X$ from X to Y in a 2-category $\mathbf{K}$ is a span $Y \xleftarrow{d_0} g/f \xrightarrow{d_1} X$ from X to Y, together with a 2-cell $\lambda : gd_0 \Rightarrow fd_1$, such that composition with λ yields a 2-natural isomorphism

$$\mathbf{K}(\text{ - }, g/f) \simeq \mathbf{K}(\text{ - }, g)/\mathbf{K}(\text{ - }, f).$$

Here for each object X the expression $\mathbf{K}(X, g)/\mathbf{K}(X, f)$ denotes the usual comma category of the functors $\mathbf{K}(X, g)$, $\mathbf{K}(X, f)$. An equivalent elementary description of this situtation is that, for each span $Y \xleftarrow{u_0} B \xrightarrow{u_1} X$ from X to Y in $\mathbf{K}$, we have the following properties:

1. For every 2-cell $\sigma : gu_0 \Rightarrow fu_1$, there exists a unique $h : B \to g/f$, such that $\sigma = \lambda h$.

2. Given 2-cells $\xi : d_0 h \Rightarrow d_0 h'$ and $\eta : d_1 h \Rightarrow d_1 h'$ such that $\lambda h' \cdot g\xi = f\eta \cdot \lambda h$, then there exists a unique 2-cell $\phi : h \Rightarrow h'$ such that $\xi = d_0\phi$, $\eta = d_1\phi$.

Lemma 3.1 *The 2-category* **AutoWk** *has a comma object for every opspan.*

Proof – Given an opspan $Y \xrightarrow{g} A \xleftarrow{f} X$ from X to Y in **AutoWk**, define an automaton g/f as follows:

- The alphabet of actions of g/f is the product $Y \otimes X$.

- The states of g/f are arrows $gr \xrightarrow{\gamma} fq$ of A^*, or more precisely, triples (r, γ, q) with r a state of Y, q a state of X, and $\gamma : gr \to fq$ an arrow of A^*. The initial state of g/f is the identity computation on the initial state q_{I} of A.

- Suppose $gr \xrightarrow{\gamma} fq$ and $gr' \xrightarrow{\gamma'} fq'$ are states of g/f. There are two cases in which there are transitions of g/f from γ to γ':

  1. In case $q' = q$, then the transitions from γ to γ' are the transitions $r \xrightarrow{e} r'$ of Y such that $\gamma'(ge) = \gamma$ in A^*:

$$\begin{tikzcd} gr \arrow[r, "\gamma"] \arrow[d, "ge"'] & fq \arrow[d, "1"] \\ gr' \arrow[r, "\gamma'"'] & fq' \end{tikzcd}$$

2. In case $r' = r$, then the transitions from γ to γ' are the transitions $q \xrightarrow{e} q'$ of X such that $\gamma' = (fe)\gamma$:

$$\begin{tikzcd} gr \arrow[r, "\gamma"] \arrow[d, "1"'] & fq \arrow[d, "fe"] \\ gr' \arrow[r, "\gamma'"'] & fq' \end{tikzcd}$$

These are the only types of transitions that g/f has. One may verify that g/f satisfies the commutativity condition required of an automaton.

Define maps $d_0 : g/f \to Y$ and $d_1 : g/f \to X$ as follows:

- d_0 takes a state $gr \xrightarrow{\gamma} fq$ of g/f to the state r of Y, and restricts the alphabet of g/f to the Y component.
- d_1 takes a state $gr \xrightarrow{\gamma} fq$ of g/f to the state q of X, and restricts the alphabet of g/f to the X component.

It is easily seen that d_0 and d_1 are morphisms of trace automata. The automaton g/f is also equipped with a 2-cell $\lambda : gd_0 \Rightarrow fd_1$, which associates with each state $gr \xrightarrow{\gamma} fq$ of g/f the computation γ of A.

We claim that g/f, equipped with the maps d_0 and d_1 and the 2-cell λ, is a comma object in **AutoWk** for the opspan $Y \xrightarrow{g} A \xleftarrow{f} X$. To see this, suppose $Y \xleftarrow{u_0} B \xrightarrow{u_1} X$ is a span in **AutoWk**.

1. Suppose we are given a 2-cell $\sigma : gu_0 \Rightarrow fu_1$. With each state q of D, this 2-cell (which is actually a natural transformation from $(gu_0)^*$ to $(fu_1)^*$) associates a computation $\gamma_q : gu_0q \to fu_1q$ of A, in other words with each state q of D is associated a state of g/f.

   Define the map $h : B \to g/f$ to take each state q of B to the corresponding state γ_q of g/f, and each action e of B to the disjoint sum $ge + fe$, which is a commuting set of actions of g/f. Clearly, if h is a weak morphism of automata, then it is the unique such morphism such that $\sigma = \lambda h$.

   To see that h is a weak morphism of automata, suppose we are given a transition $q \xrightarrow{e} r$ of B. This transition determines the following commuting diagram in A^*:

$$\begin{tikzcd}
fu_1q \arrow[r, "fu_1e"] & fu_1r \\
gu_0q \arrow[u, "\gamma_q"] \arrow[r, "gu_0e"'] & gu_0r \arrow[u, "\gamma_r"']
\end{tikzcd}$$

which factors as follows:

$$\begin{tikzcd}
fu_1q \arrow[r, "fu_1e"] & fu_1r \arrow[r, "1"] & fu_1r \\
gu_0q \arrow[u, "\gamma_q"] \arrow[r, "1"'] & gu_0q \arrow[u] \arrow[r, "gu_0e"'] & gu_0r \arrow[u, "\gamma_r"']
\end{tikzcd}$$

Since the left and right squares clearly determine computations of g/f, their composite does too. This composite computation is the image of the transition $q \xrightarrow{e} r$ under h. (A map h behaving in this way is in general not a morphism of automata, but rather only a weak morphism.)

2. Suppose we are given 2-cells $\xi : d_0h \Rightarrow d_0h'$ and $\eta : d_1h \Rightarrow d_1h'$, such that $\lambda h' \cdot g\xi = f\eta \cdot \lambda h$. For each state q of B, let ϕ_q denote the computation of g/f corresponding to the following commuting square in A^*:

$$\begin{tikzcd}
d_1hq \arrow[r, "f\eta q"] & d_1h'q \\
d_0hq \arrow[u, "\lambda hq"] \arrow[r, "g\xi q"'] & d_0h'q \arrow[u, "\lambda h'q"']
\end{tikzcd}$$

To see that this square does in fact determine a computation of g/f, use the same factoring trick as in (1) above. The map ϕ that assigns to each state q of B the corresponding computation ϕ_q of g/f, is now easily seen to be the unique 2-cell $\phi : h \Rightarrow h'$ such that $\xi = d_0\phi$, $\eta = d_1\phi$. ∎

The alphabet of actions of a comma object g/f of an opspan from X to Y is the product $Y \otimes X$. Let us call the actions in the Y component *output* actions, and those in the X component *input* actions. This terminology suggests an intuitive interpretation of g/f as kind of "nondeterministic X to Y transducer," which accepts as input a sequence of actions of X, records some information about these actions internally in the form of a computation of automaton A, and then outputs this information, perhaps after some delay, in the form of a sequence of actions of Y. Of special interest is the comma object ΦA for the opspan $A \xrightarrow{1} A \xleftarrow{1} A$ from A to A. We interpret ΦA as an "A-buffer."

The factoring trick used in the proof above represents an important property of comma objects in **AutoWk**, which can be formalized as follows:

Lemma 3.2 *Every 2-cell γ between morphisms from B to g/f has a unique* input/output factorization*; that is, γ factors uniquely as $\xi\eta$ with $d_0\eta$ and $d_1\xi$ both identity 2-cells.*

As noted above, **AutoWk** does not even have all ordinary pullbacks, let alone all 2-pullbacks. Fortunately, though, the 2-pullbacks that we need for the notion of fibration to make sense actually do exist:

Lemma 3.3 *Suppose $f : A \to C$, and $g : B \to C$ in* **AutoWk**. *Then all three indicated 2-pullbacks exist in* **AutoWk**.

$$\begin{tikzcd}
 & & & g^* \circ \Phi C \circ f \arrow[dl] \arrow[dr] & & & \\
 & & g^* \circ \Phi C \arrow[dl] \arrow[dr] & & \Phi C \circ f \arrow[dl] \arrow[dr] & & \\
 & B \arrow[dr, "g"'] & & \Phi C \arrow[dl, "d_0"] \arrow[dr, "d_1"'] & & A \arrow[dl, "f"] & \\
 & & C & & C & &
\end{tikzcd}$$

The proof of this result relies heavily on the input/output factorization property of computations of ΦC.

3.3 The Input Buffering Doctrine

We are now in a position to apply the theory of fibrations to **AutoWk**. Given concurrent alphabets X and Y, let $\mathbf{Span}(X, Y)$ denote the 2-category of spans in **AutoWk** from X to Y. Suppose $Z \xleftarrow{g} A \xrightarrow{f} X$ is a span from X to Z and $Y \xleftarrow{k} B \xrightarrow{h} Z$ is a span from Z to Y. If the following diagram is a 2-pullback,

$$\begin{tikzcd}
B \circ A \arrow[r, "h'"] \arrow[d, "g'"'] & A \arrow[d, "g"] \\
B \arrow[r, "h"'] & Z
\end{tikzcd}$$

then the span $Y \xleftarrow{kg'} B \circ A \xrightarrow{fh'} X$ from X to Y is called the *composite* of B and A. Composition on the right with the comma object ΦX, viewed as a span from X to X, yields an endo-2-functor

$$R : \mathbf{Span}(X, Y) \to \mathbf{Span}(X, Y).$$

Explicitly, if A is a span from X to Y, then RA is the span $A \circ \Phi X$ from X to Y. In view of our intuitive interpretation of ΦX as an "X-buffer," we may interpret R as an "input buffering construction," which takes a span from X to Y and places it in tandem with an input buffer, producing an "input-buffered" span from X to Y.

The 2-functor R comes equipped with 2-natural transformations $\eta : 1 \to R$ and $\mu : RR \to R$ making (R, η, μ) a monad in **2Cat** (also called a *2-monad* or a *doctrine*). The component ηA of η at a span $Y \xleftarrow{g} A \xrightarrow{f} X$ from X to Y is the unique arrow of spans $A \to RA$ whose composition with the projection $RA \to A$ is 1_A and whose composition with the projection $RA \to \Phi X$ is the map $i_f : A \to \Phi X$ corresponding to the identity 2-cell on $f : A \to X$. The component μA of μ at the span A is the unique arrow of spans $1_A \circ c : RRA \to RA$ induced by the map $1_A : A \to A$ and the map $c : \Phi X \circ \Phi X \to \Phi X$ corresponding to the composite 2-cell:

$$\begin{tikzcd}
& \Phi X \circ \Phi X \arrow[dl, "pr_1"'] \arrow[dr, "pr_0"] & \\
\Phi X \arrow[r, "d_0"] \arrow[dr, "d_1"'] & X \arrow[d, "1"] \arrow[dl, phantom, "\overset{\Leftarrow}{\lambda}"] \arrow[dr, phantom, "\overset{\Leftarrow}{\lambda}"] & \Phi X \arrow[l, "d_1"'] \arrow[dl, "d_0"] \\
& X &
\end{tikzcd}$$

The existence of R, η, and μ, and the fact that they form a doctrine, follows automatically from general properties of comma objects and 2-pullbacks (see [14]). The concrete form taken by these data in the 2-category **AutoWk** is given by the following result, which is easily proved by working through the definitions.

Lemma 3.4 *Suppose X and Y are one-state trace automata. Then the doctrine (R, η, μ), regarded as an ordinary monad on the category* $\mathbf{Span}(X, Y)$*, is nothing but the input buffering monad (B, η, μ) defined in Section 2.*

A span A from X to Y is called a *split right fibration* if it admits a structure of algebra for the doctrine R. This means that there exists an arrow of spans $h_A : RA \to A$ such that $h_A \circ (\eta A) = 1$ and $h_A \circ (Rh_A) = h_A \circ (\mu A)$. The structure map h_A is called a *right cleavage* for A. It is a consequence of the general theory that if a span A is a split right fibration, then the cleavage h_A is left-adjoint to the map $\eta A : A \to RA$, hence is determined uniquely up to an invertible 2-cell. In the case of **AutoWk**, there are no invertible 2-cells other than identities, so h_A is unique. Street also defines a more general notion of *right fibration*, which is a span A that bears a structure of *pseudo-R* algebra. For a pseudo R-algebra, the above diagrams are required to commute only up to invertible 2-cell, rather than exactly. Again, since **AutoWk** has no nontrivial invertible 2-cells, there is no difference between a right fibration and a split right fibration.

The following is the main result of this section. It is a direct consequence of Theorem 1 and Lemma 3.4.

Theorem 2 *Suppose X and Y are one-state automata. Then an automaton from X to Y is monotone iff it is a split right fibration in* **AutoWk** *having an* **Auto***-morphism as cleavage.*

4 Fibrations between Domains

Although the fact that the monotone automata coincide exactly with the split right fibrations in **AutoWk** provides some evidence that the latter is the correct categorical notion, stronger evidence comes from the fact that essentially the same coincidence occurs for domains of computations of monotone automata and split right fibrations in a suitably defined 2-category of domains. The purpose of this section is to develop these results.

4.1 Conflict Event Orderings

We begin by recalling some definitions from [9]. Suppose $D = (D, \sqsubseteq)$ is a partially ordered set. An *interval* of D is a pair $(d, d') \in D \times D$ with $d \sqsubseteq d'$. A *prime* (or *covering*) interval is an interval (d, d') with $d \sqsubset d'$ and such that for no $d'' \in D$ do we have $d \sqsubset d'' \sqsubset d'$. We say that an interval $I = (d, d')$ is $\sqcup$-*prime* if there exists a finite set of prime intervals $\{(d, d_1), \ldots, (d, d_n)\}$ such that $d' = \bigsqcup\{d_1, \ldots, d_n\}$. Call two intervals $I = (d_0, d_1)$ and $J = (d'_0, d'_1)$ *coinitial* if $d_0 = d'_0$. Coinitial intervals $I = (d_0, d_1)$ and $J = (d_0, d'_1)$ are called *consistent* if the set $\{d_1, d'_1\}$ has an upper bound. Call coinitial $\sqcup$-prime intervals I and J *orthogonal* if they are consistent, and there is no prime interval (d_0, d''_1) with $d''_1 \sqsubseteq d_1$ and $d''_1 \sqsubseteq d'_1$. We say that D is *finitely consistently complete* if every finite subset U of D having an upper bound, has a supremum $\bigsqcup U$. If D is finitely consistently complete, and if intervals $I = (d_0, d_1)$ and $J = (d_0, d'_1)$ are consistent, then let $I \setminus J$ denote the interval $(d'_1, d_1 \sqcup d'_1)$.

Suppose D is a nonempty, finitely consistently complete poset with the following additional property:

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

We may then define $\equiv$ to be the least equivalence relation on prime intervals of D, such that $I \equiv I \setminus J$ whenever I and J are distinct and consistent.

A *conflict event ordering* is a nonempty, finitely consistently complete poset D having property (1) above and in addition having the properties:

2. $I \equiv J$ implies $I = J$, whenever I and 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 *weak morphism* from a conflict event ordering D to a conflict event ordering D' is a function $f : D \to D'$ that preserves all finite suprema existing in D. A *morphism* from D to D' is a weak morphism $f : D \to D'$ with the additional properties:

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

2. If $\sqcup$-prime intervals I, J are orthogonal in D, then $f(I)$ and $f(J)$ are orthogonal in D'.

Let **EvOrd** denote the category of conflict event orderings and morphisms, and let **EvOrdWk** denote the category of conflict event orderings and weak morphisms. Note that the map taking a poset to its ideal completion determines an equivalence of categories between **EvOrd** and the category **EvDom** of conflict event domains defined in [9]. The adjoint maps a conflict event domain to its finite basis. Since we shall have no need for morphisms that map finite elements to infinite elements, we prefer here to dispense with infinite elements entirely, and work with the category **EvOrd** instead of **EvDom**.

We showed in [9] that the category **EvOrd** is finitely complete. We also showed the following:

Proposition 4.1 *If A is an automaton, then the set HA of (finite) computations of A from its initial state, partially ordered by prefix, is a conflict event ordering. Moreover, the map taking A to HA extends to a functor $H :$* **Auto** $\to$ **EvOrd**, *which is right-adjoint to a full and faithful embedding, with unit an isomorphism.*

We add that although the functor H extends to a functor from **AutoWk** to **EvOrdWk**, the adjunction does not.

4.2 EvOrdWk as a 2-Category

The categories **EvOrd** and **EvOrdWk** have partially ordered homs, with strict, monotone composition. Hence they are actually 2-categories. Although **EvOrd** is finitely complete as a 1-category, it is not finitely 2-categorically complete, for essentially the same reasons as for **Auto**.

Proposition 4.2 *The 2-category* **EvOrdWk** *has a comma object for every opspan.*

Proof – A comma object g/f for an opspan $Y \xrightarrow{g} D \xleftarrow{f} X$ in **EvOrdWk** is $g/f = \{(b, a) : gb \sqsubseteq fa\}$, ordered componentwise, and equipped with the evident projections and 2-cell. It is necessary to verify that g/f is a conflict event ordering—this can be done by a direct check of the axioms, using the fact that f and g preserve finite suprema. ∎

The same result concerning 2-pullbacks holds for **EvOrdWk** as for **AutoWk**:

Proposition 4.3 *Lemma 3.3 holds for* **EvOrdWk**.

Next, we describe the doctrine R on spans in **EvOrdWk** and obtain a characterization of its algebras, the split right fibrations in **EvOrdWk**. The object map of R takes a span $Y \xleftarrow{g} D \xrightarrow{f} X$ to the span $Y \xleftarrow{g'} RD \xrightarrow{f'} X$, where

$$RD \simeq \{(d,x) \in D \times X : fd \sqsubseteq x\},$$

the map f' takes (d,x) to $x \in X$, and g' takes (d,x) to $gd \in Y$. The unit $\eta : 1 \to R$ has components $\eta_D : D \to RD$ that take d to (d, fd). The multiplication $\mu : RR \to R$ has components $\mu_D : RRD \to RD$ that take $((d,x),x')$ to (d,x').

Theorem 3 *A span $Y \xleftarrow{g} D \xrightarrow{f} X$ in* **EvOrdWk** *is a split right fibration iff the following condition holds:*

- *For all $d \in D$ and all $x \geq fd$, there exists an element $d \sqcup x$ of D, which is the least $d' \geq d$ with $fd' \geq x$. Moreover, $f(d \sqcup x) = x$ and $g(d \sqcup x) = gd$.*

Proof – Suppose the condition. Let $h : RD \to D$ be the map taking $(d,x) \in RD$ to $d \sqcup x \in D$. It is easy to check that h preserves finite suprema, hence is a weak morphism. Since $f(d \sqcup x) = x$ and $g(d \sqcup x) = gd$, it follows that h is an arrow of spans. Also, if $fd \sqsubseteq x \sqsubseteq x'$ then $d \sqcup x' = (d \sqcup x) \sqcup x'$, so $h \circ \mu D = h \circ Rh$. Finally, $h(\eta_D(d)) = h(d, fd) = d$, so h is a right cleavage for D.

Conversely, suppose $h : RD \to D$ is a right cleavage for D. Then h is left-adjoint to $\eta D : D \to RD$, with counit the identity. It follows by properties of adjunctions that for all $(d,x) \in RD$, the element $h(d,x)$ of D is the least $d' \geq d$ with $fd' \geq x$. Since h is an arrow of spans, we have also $f(h(d,x)) = x$ and $g(h(d,x)) = gd$. ∎

4.3 Connection with Automata

In this section, we show that the "unwinding functor" H, which takes each automaton A to the poset HA of its computations from the initial state, preserves and reflects split right fibrations. This gives a connection, as in [9], between an "operational" semantics of dataflow networks, defined in terms of automata, and a more "denotational," order-theoretic semantics. The proofs of the results in this section involve a detailed examination of the structure of the posets of computations of monotone automata. It ought to be possible to prove at least some of these results categorically, though at present it is not clear to the author what satisfactory categorical versions of the proofs would look like.

For the techniques to prove the following result, the reader is referred to [12]:

Lemma 4.4 *Suppose $Y \xleftarrow{g} A \xrightarrow{f} X$ is a monotone automaton. For each computation $\gamma : q_I \to r$ of A and trace $x \in X^*$ with $Hf\gamma \sqsubseteq x$, there exists a least computation $\gamma \sqcup x$ such that $\gamma \sqsubseteq \gamma \sqcup x$ and $x \sqsubseteq Hf(\gamma \sqcup x)$. Moreover, $Hf(\gamma \sqcup x) = x$, $Hg(\gamma \sqcup x) = Hg\gamma$, and the map taking (γ, x) to $\gamma \sqcup x$ is an* **EvOrd***-morphism from $R(HA)$ to HA.*

The following result makes use of technical properties of conflict event orderings. A complete proof would be rather lengthy, so we just sketch the main ideas.

Lemma 4.5 *Let X and Y be concurrent alphabets. Suppose $HY \xleftarrow{g} D \xrightarrow{f} HX$ is a span in* **EvOrd**, *and suppose $h_D : RD \to D$ is an* **EvOrd**-*morphism that is also an R-algebra structure on D. Then there exists a monotone automaton $Y \xleftarrow{g'} A \xrightarrow{f'} X$, and an order-isomorphism $\phi : HA \to D$, such that $g\phi = Hg'$ and $f\phi = Hf'$.*

Proof – (Sketch) We first observe the following facts about the poset D:

1. Every prime interval (d, d') in D satisfies exactly one of the following two conditions:

   (a) $d' = h_D(d, fd')$, with (fd, fd') a prime interval of HX.

   (b) $fd = fd'$.

   Call intervals of type (a) *input* intervals, and those of type (b) *noninput* intervals.

2. If I is an input interval $(d, h_D(d, x))$ and J is a noninput interval (d, d'), then $I \perp J$. Moreover, $I \setminus J = (d', h_D(d', x))$ and $J \setminus I = (h_D(d, x), h_D(d', x))$.

3. If $I = (d, d')$ is a noninput prime interval, and $fd' \sqsubseteq x$, then $(h_D(d, x), h_D(d', x))$ is also a noninput prime interval.

   The automaton A is then constructed as follows:

- The alphabet of A is $E \otimes X$, where E is the set of all $\equiv$-equivalence classes of noninput intervals of D and $[I] \| [J]$ in E iff there exist $I' = (d, d') \in [I]$ and $J' = (d, d'') \in [J]$ such that $I' \perp J'$.

- The states of A are the elements of RD, with $(\perp, \epsilon)$ as the initial state.

- The transitions of A are of two types:

  1. If $e \in X$, then for all states (d, x) of A there is a transition $(d, x) \xrightarrow{e} (h_D(d, ex), ex)$ of A.

  2. If $[I] \in E$, then A has a transition $(d, x) \xrightarrow{[I]} (d', x)$ whenever $(d, d') \in [I]$.

Verification that A satisfies the commutativity condition, hence is an automaton, requires a case analysis on the various ways in which actions of A can be concurrent. These arguments make use of the properties of D stated above, plus the hypothesis that h_D is an R-algebra structure on D.

Let the map $f' : A \to X$ take each state of A to the unique state of X, and on actions, let f' be the restriction to X. Let the map $g' : A \to Y$ take each state of A to the unique state of Y. On actions, let g' be the map that takes $e \in X$ to $\emptyset$ and takes each $[I] \in E$, where $I = (d, d')$, to the trace $gd' \setminus gd \in Y^*$, which must be in $\mathrm{Comm}(Y)$ because the **EvOrd**-morphism g preserves $\sqcup$-prime intervals. It then follows from the definitions that A has the receptivity property, hence the span $Y \xleftarrow{g'} A \xrightarrow{f'} X$ is a monotone automaton.

To complete the proof, one may check that the map $\phi : HA \to D$ that takes each computation $\gamma : (\perp, \epsilon) \to (d, x)$ of A to $d \in D$, is an isomorphism of spans in **EvOrdWk**, from the span HA to the span D. The verification of this fact involves the observation that prime intervals (d, d') in D correspond exactly to transitions of A from (d, fd) to (d', fd'), and thus the computation sequences of A from state $(\perp, \epsilon)$ correspond to covering chains from $\perp$ in D. Moreover, prime intervals (d, d') and (d, d'') in D are orthogonal iff the corresponding transitions of A are for commuting actions. These facts allow us to prove that the map ϕ is in fact an arrow of spans in **EvOrdWk**, with an inverse that is also an arrow of spans in **EvOrdWk**. ▮

We can now answer the question raised at the end of our previous paper [9], concerning a characterization of the dataflow-like spans in **EvOrd**.

Theorem 4 *Suppose X and Y are concurrent alphabets. A span D from HX to HY in* **EvOrd** *is HA for some monotone automaton A iff D is a split right fibration in* **EvOrdWk**, *having an* **EvOrd**-*morphism as cleavage.*

5 Conclusion

We have shown that spans arising as behaviors of dataflow networks can be characterized in terms of split right fibrations, both in a 2-category of automata and a 2-category of domains. This characterization should make it possible to give categorical proofs that this class of spans is closed under network-forming operations, such as parallel and sequential composition, and feedback. We hope also that it will facilitate the continuity proofs required in the development of a semantics for recursively defined networks. There remains, however, the problem of understanding the correct way to formulate the universal properties satisfied by the feedback operation.

It is a bit annoying that our characterizations had to be stated in terms of the 2-categories **AutoWk** and **EvOrdWk** and their sub-2-categories **Auto** and **EvOrd**. For intuitive reasons, though, it seems necessary that the results be stated in this way. In this paper, we tried to make the simplest extensions to the 2-categories **Auto** and **EvOrd** that would show the connection with fibrations. Perhaps a cleaner (though less concrete) formulation of the results might be achieved by making **AutoWk** and **EvOrdWk** much larger, and then giving categorical characterizations of **Auto** and **EvOrd** as sub-2-categories. For example, we expect that the 2-category **Cts** of "concurrent transition systems" [11, 12] would be a suitable replacement for **AutoWk**.

References

[1] A. Carboni and R. F. C. Walters. Cartesian bicategories I. *Journal of Pure and Applied Algebra*, 49:11–32, 1987.

[2] J. W. Gray. Fibred and cofibred categories. In *Proc. Conference on Categorical Algebra at La Jolla*, pages 21–83, Springer-Verlag, 1966.

[3] A. Grothendieck. Catégories fibrées et descente. In *Séminaire de Géométrie Algébrique de l'Institute des Hautes Études Scientifiques, Paris 1960/61 (SGA 1)*, pages 145–194, Springer-Verlag, 1971.

[4] 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.

[5] 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.

[6] G. M. Kelly and R. H. Street. Review of the elements of 2-categories. In *Lecture Notes in Mathematics 420*, pages 75–103, Springer-Verlag, 1974.

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

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

[9] E. W. Stark. Compositional relational semantics for indeterminate dataflow networks. In *Category Theory and Computer Science*, pages 52–74, Springer-Verlag. Volume 389 of *Lecture Notes in Computer Science*, Manchester, U. K., 1989.

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

[11] E. W. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64:221–269, 1989.

[12] 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, 1990. (to appear).

[13] E. W. Stark. A simple generalization of Kahn's principle to indeterminate dataflow networks. In M. Z. Kwiatkowska, M. W. Shields, and R. M. Thomas, editors, *Semantics for Concurrency, Leicester 1990*, pages 157–176, Springer-Verlag, 1990.

[14] R. H. Street. Fibrations and Yoneda's lemma in a 2-category. In *Lecture Notes in Mathematics 420*, pages 104–133, Springer-Verlag, 1974.

[15] R. H. Street. Fibrations in bicategories. *Cahier de Topologie et Géometrie Différentielle*, XXI-2:111–159, 1980.