An Algebra of Dataflow Networks

Eugene W. STARK*

Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400 USA†

Abstract. This paper describes an algebraic framework for the study of *dataflow networks*, which form a paradigm for concurrent computation in which a collection of concurrently and asynchronously executing processes communicate by sending messages between *ports* connected via FIFO message channels. A syntactic *dataflow calculus* is defined, having two kinds of terms which represent *networks* and *computations*, respectively. By imposing suitable equivalences on networks and computations, we obtain the *free dataflow algebra*, in which the dataflow networks with m input ports and n output ports are regarded as the objects of a category S_m^n, and the computations of such networks are represented by the arrows. Functors defined on S_m^n label each computation by the *input buffer* consumed and the *output buffer* produced during that computation, so that each S_m^n is a span in **Cat**. It is shown that the free dataflow algebra construction underlies a monad in the category of collections $S = \{S_m^n : m, n \geq 0\}$ of spans in **Cat**. The algebras of this monad, called *dataflow algebras*, have a monoid structure representing *parallel composition*, and are also equipped with an action of a certain collection of continuous functions, thereby representing the formation of *feedback loops*. The two structures are related by a distributive law of feedback over parallel composition. We also observe the following connection with the theory of fibrations: if S is a dataflow algebra, then each S_m^n is a *split bifibration* in **Cat**.

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 between *ports* connected via FIFO message channels. *Determinate* dataflow networks compute continuous functions from input message histories to output message histories, and have a pleasant, well-understood theory [4]. Less developed is the theory of *indeterminate* or non-functional networks, which present a puzzle because naive attempts to model such networks by generalizing the continuous functions that work for the determinate case, fail to yield theories that are compatible with an intuitive operational semantics [1, 6]. From a more general concurrency theory perspective, indeterminate dataflow networks are interesting because they exhibit both concurrency and indeterminacy, and it seems likely that insight gained from their study will contribute to a better overall understanding of these two concepts.

---

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

†E-mail address: `stark@cs.sunysb.edu` (Internet)

For several years I have been studying dataflow networks with the goal of finding the "correct" algebraic setting for the study of indeterminate networks. I have taken the view that, in such a setting, dataflow networks should be the elements of an algebra whose operations represent ways to build networks from components. Although the particular selection of network operations is mostly a presentational matter, included among the operations should be a *parallel composition* operation for aggregating a collection of components into a network without any interconnections, and a *feedback* operation for introducing feedback loops from outputs to inputs. Further, a notion of computation should be integral to the theory, so that connections can be drawn between the abstract algebraic properties of the network-building operations and a more concrete operational semantics. Finally, the theory should provide some sort of principle, for reasoning about the behavior of networks with feedback loops, that somehow generalizes the least fixed-point principle originally described by Kahn [4] for determinate networks.

In previous work toward these goals [9], I observed that some algebraic properties of the network operations are clarified if dataflow networks are represented as spans in a finitely complete category **Auto** of concurrent automata. Intuitively, a span $X \xleftarrow{f} A \xrightarrow{g} Y$ from X to Y represents a dataflow network with "object of inputs" X, "object of outputs" Y, and "underlying automaton" A. The morphisms f and g serve to label the computations of A by the inputs consumed and outputs produced during those computations. When networks are represented as spans in **Auto**, parallel composition has a simple description as cartesian product of spans, and feedback of outputs to inputs can be described in terms of the equalizer of the two maps that label computations by their feedback output and the feedback input. The latter characterization depends heavily on the particular structure of automata in **Auto** and on having just the right notion of morphism for these automata. Specifically, the automata support the notion that certain transitions represent simultaneous occurrences of independent computational steps, and morphisms are required to preserve these transitions. This property of the morphisms ensures that the subautomaton formed by equalizing output and input is capable only of "causal" computations, in which the production of feedback output in a step of computation does not depend on the consumption of that output as feedback input in the same step.

In a a subsequent paper [10] I explored the networks-as-spans idea further by observing that "dataflow-like" spans in **Auto** have certain special properties pertaining to inputs and outputs, and then attempting to identify categorical properties that characterize the dataflow-like spans. I found that the dataflow-like spans in **Auto** could be described in terms of Street's notion of *0-fibration* [12, 13], which adapts to more general 2-categories the notion of "opfibration" in **Cat** [2]. Street's theory characterizes fibrations in a 2-category as being the algebras of a certain kind of 2-monad called a "KZ-doctrine." For dataflow networks, the endo-2-functor underlying this 2-monad corresponds to the construction "compose with an input buffer." Thus, in this setting, we can say that the dataflow-like spans are those spans that are algebras of the input buffering doctrine.

In spite of the pleasant intuitive connections, there are some technical problems with the treatment of dataflow networks as fibrations in **Auto**, which seem to indicate that this is not exactly the right way to proceed. These problems stem from the delicate issue of the choice of morphisms for **Auto**. In a nutshell, the morphisms that yield the characterization of feedback in terms of limits do not lead to a 2-category **Auto** with sufficient completeness properties (in particular, the existence of comma objects). To get around this problem, **Auto** was regarded as embedded in a 2-category **AutoWk**, which had a larger class of morphisms that behaved somewhat better, and the dataflow-like spans were described as those 0-fibrations in **AutoWk** having an **Auto**-morphism as the structure map. Unfortunately, it is difficult to see how to apply this somewhat messy characteration to 2-categories very much different than **Auto**, or to give categorical proofs of theorems such as: "the unwinding 2-functor, taking automata to the prefix-ordered sets of their computations, preserves dataflow-like spans."

In [11], I took a more syntactic approach, and defined a "dataflow calculus" to be a formal system in which there are two sets of expressions: one set denoting networks and another set denoting computations. Network expressions are constructed, via formal network-building operations, from certain basic standard networks used for "wiring" and basic nonstandard networks which we think of as variables. Computation expressions are proof terms which are constructed via inference rules from axioms associated with the basic networks. The inference rules are associated with the network-building operations, and in fact can be thought of as a structured operational semantics (SOS) [8] description of their computational behavior. The main result of that paper was a sound and complete axiomatization for a certain equivalence relation on networks, defined in terms of bisimulation [7].

The goal of the present paper is to forge a link between the syntactic approach of [11] and the more abstract networks-as-spans approaches of [9] and [10]. We retain the idea that dataflow networks are represented using spans, but discard the idea that they must be spans in **Auto**. Instead, it turns out that with some care in the treatment of feedback, **Cat** can be used as the category of automata. We begin by defining a syntactic "dataflow calculus," which is concerned with networks and their computations. Then, we describe a "free dataflow algebra" construction on certain diagrams of spans in **Cat**. This construction is concerned both with objects, which we regard as network states, and arrows, which we regard as computations. In this way we manage to treat simultaneously the network-building operations and their operational semantics. We show how the free dataflow algebra supports an interpretation of both the network expressions and computation expressions of dataflow calculus, so that the latter can be viewed as a concrete syntax for the former. Our main result is a theorem stating that the free dataflow algebra construction is the underlying functor of a monad, whose algebras we call "dataflow algebras." Using Street's characterization of fibrations as algebras, an easy consequence of the result is that every dataflow algebra consists of split bifibrations in **Cat**.

2 Dataflow Calculus

Our calculus for dataflow networks is a language with two kinds of terms: "network expressions" and "computation expressions." Network expressions are constructed by applying syntactic network-forming operations to a set of "basic standard networks," which consists of constants that are common to every dataflow calculus, and a set of "basic nonstandard networks," which is given as a parameter and whose elements we think of as variables. Computation expressions are proofs, constructed via inference rules associated with the network-forming operations, from "nonstandard computation axioms" associated with the basic nonstandard networks, and "standard computation axioms" associated with the basic standard networks. The conclusions of such proofs are labeled arrows of the form $P \overset{\gamma}{\underset{\beta}{\Longrightarrow}} Q$, which represent computational steps in which input data β is consumed, output data γ is produced, and network state P is transformed into network state Q. The inference system for computation expressions may be viewed as a structured operational semantics (SOS) definition [8] of the computational behavior of network expressions.

Network expressions in dataflow calculus are segregated into classes by numbers of input and output ports; so that, if Ord denotes the set of finite ordinals, then for each pair $(m, n) \in \mathrm{Ord} \times \mathrm{Ord}$ we have a set of "network expressions of type (m, n)." We use ordinals because we regard the input and output ports of a network as having a definite left-to-right orientation.

To give a formal definition of dataflow calculus, we begin by postulating a set V of *data values*, representing the quantities that can be sent in messages between components of a network. For each finite ordinal n, a *buffer of width* n is a function

$$\beta : \{0, 1, \ldots, n-1\} \to V^*,$$

where V^* is the free monoid generated by V. We use ϵ_n to denote the buffer of width n such that $\epsilon_n(i)$ is the empty string for $0 \leq i < n$. We will drop the subscript n when the width is clear from the context.

Let B_n denote the set of buffers of width n. The set B_n is a monoid under the operation of concatenation lifted pointwise from V^*. Besides this *vertical composition*, which takes β and β' in B_n and produces $\beta; \beta' \in B_n$, the collection B of all sets B_n is also equipped with a *horizontal composition*, which takes $\beta \in B_m$ and $\gamma \in B_n$ and places them "side-by-side" to yield $\beta\gamma \in B_{m+n}$. The vertical and horizontal composition are connected by the familiar "middle-four interchange law:"

$$(\beta\gamma); (\beta'\gamma') = (\beta; \beta')(\gamma; \gamma').$$

The basic nonstandard networks and nonstandard computation axioms in dataflow calculus are given by a *dataflow scheme*, which we define to be a collection

$$S = \{S_m^n : (m, n) \in \mathrm{Ord}\},$$

of spans in **Cat**, where S_m^n is a span from B_m to B_n, and we regard the monoids B_m and B_n as one-object categories. The objects and arrows of S_m^n represent the basic nonstandard networks and the nonstandard computation axioms, respectively. For each (m, n), we think of the functors $d_0 : S \to B_m$ and $d_1 : S \to B_n$ forming the legs of the span as labeling each computation by the input buffer consumed and the output buffer produced, respectively, during that computation. We write $t : x \overset{\gamma}{\underset{\beta}{\Longrightarrow}} y$, or sometimes just $x \overset{\gamma}{\underset{\beta}{\Longrightarrow}} y$, to refer to a computation t in S_m^n with $d_0(t) = \beta$ and $d_1(t) = \gamma$. We generally omit writing β, γ, or both, when they are empty buffers. The notation $1 : x \Longrightarrow x$ denotes an identity computation.

Given a dataflow scheme S, let the sets $(NS)_m^n$ of *network expressions of type* (m, n) *over* S be defined inductively as follows:

- For all $n, n' \in \mathrm{Ord}$, $\delta, \eta \in B_n$, and $\delta' \in B_{n'}$, we have the following *basic standard networks*:
  - the *buffered identity network* $\mathbf{1}_n^\delta \in (NS)_n^n$.

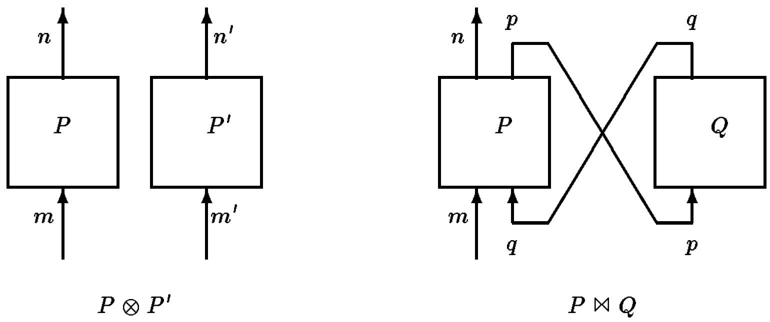


Figure 1: Parallel Composition and Feedback

- the *buffered exchange network* $\mathbf{s}^{\delta'\delta}_{n,n'} \in (NS)^{n'+n}_{n+n'}$,
- the *buffered duplicator network* $\mathbf{d}^{\delta\eta}_{n} \in (NS)^{n+n}_{n}$,
- the *terminator network* $\mathbf{t}_n \in (NS)^{0}_{n}$.

We often omit writing buffers in basic standard networks, under the convention that any omitted buffers are assumed to be empty.

- The *basic nonstandard network* x is in $(NS)^n_m$ whenever $x \in S^n_m$.
- The *input buffered network* $P \cdot \beta$ is in $(NS)^n_m$. whenever $P \in (NS)^n_m$ and $\beta \in B_m$.
- The *output buffered network* $\gamma \cdot P$ is in $(NS)^n_m$ whenever $P \in (NS)^n_m$ and $\gamma \in B_n$.
- The *parallel composition* $P \otimes P'$ of P and P' is in $(NS)^{n+n'}_{m+m'}$, whenever $P \in (NS)^n_m$ and $P' \in (NS)^{n'}_{m'}$.
- The *feedback* $P \bowtie Q$ of P and Q is in $(NS)^n_m$, whenever $P \in (NS)^{n+p}_{m+q}$ is a *standard network* (one not containing any occurrences of basic nonstandard networks) and $Q \in (NS)^q_p$ is an arbitrary network.

Figure 1 gives a schematic depiction of the parallel composition and feedback operations.

The *computation expressions over S* are all proofs that can be constructed from the axioms and inference rules listed below.

- The *nonstandard computation axiom*

$$x \xRightarrow[\beta]{\gamma} y,$$

for each arrow $x \xRightarrow[\beta]{\gamma} y$ of S^n_m.

- The *standard computation axioms*

$$\mathbf{1}_n^{\gamma;\delta} \xRightarrow[\beta]{\gamma} \mathbf{1}_n^{\delta;\beta} \qquad \mathbf{d}_n^{(\gamma;\delta)(\gamma';\delta')} \xRightarrow[\beta]{\gamma\gamma'} \mathbf{d}_n^{(\delta;\beta)(\delta';\beta)}$$
$$\mathbf{t}_n \xRightarrow[\beta]{} \mathbf{t}_n \qquad \mathbf{s}_{n,n'}^{(\gamma';\delta')(\gamma;\delta)} \xRightarrow[\beta\beta']{\gamma'\gamma} \mathbf{s}_{n,n'}^{(\delta';\beta')(\delta;\beta)}.$$

- The *input and output buffering rules*

$$\frac{P \xRightarrow[\eta]{\gamma} Q}{P \cdot (\eta;\delta) \xRightarrow[\beta]{\gamma} Q \cdot (\delta;\beta)} \qquad \frac{P \xRightarrow[\beta]{\eta} Q}{(\gamma;\delta) \cdot P \xRightarrow[\beta]{\gamma} (\delta;\eta) \cdot Q}$$

- The *parallel composition rule*

$$\frac{P \xRightarrow[\beta]{\gamma} Q \quad P' \xRightarrow[\beta']{\gamma'} Q'}{P \otimes P' \xRightarrow[\beta\beta']{\gamma\gamma'} Q \otimes Q'}$$

- The *feedback rule*

$$\frac{P \xRightarrow{\gamma\delta} R \quad Q \xRightarrow[\delta]{\eta} Q' \quad R \xRightarrow[\beta\eta]{} P'}{P \bowtie Q \xRightarrow[\beta]{\gamma} P' \bowtie Q'}$$

- The *cut rule*:

$$\frac{P \xRightarrow[\beta]{\gamma} R \quad R \xRightarrow[\delta]{\eta} Q}{P \xRightarrow[\beta;\delta]{\gamma;\eta} Q}$$

3 Networks

The definition of dataflow calculus given in the previous section includes no notion of equivalence for network expressions. However, we have in mind that many pairs of network expressions, such as $P \otimes (Q \otimes R)$ and $(P \otimes Q) \otimes R$, ought to be regarded as equivalent. One way to obtain a suitable equivalence would be the operational approach taken in [11]: treat the network expressions as the states of a transition system whose transitions are the inferrable arrows $P \xRightarrow[\beta]{\gamma} Q$, define a version of bisimulation equivalence [7] for this transition system, and declare that two network expressions are equivalent if and only if they are bisimilar under all possible choices for the nonstandard transition axioms. However, it is also possible to obtain an equivalence more directly by defining an interpretation of network expressions as "networks," and to declare that two networks are equivalent iff they denote the same network. This is the approach we shall follow here. The equivalence we obtain is not as coarse as that defined in [11], since it does not relate distinct networks that differ by a permutation of their components, nor does it relate a network to the "reduced" network obtained by deleting all "unobservable" components not having data paths to an external output. It is possible, though, to extend the ideas of this paper to treat the coarser equivalence.

We begin by defining the interpretation of *standard network expressions*, which are those that contain no occurrences of basic nonstandard networks. These expressions will be interpreted as certain continuous functions on "histories." Formally, a *history of width n* is a function

$$\beta : \{0, 1, \dots, n-1\} \to V^\infty,$$

where V^∞ is the set of all finite and infinite sequences of elements of V, partially ordered by the relation: $u \leq u'$ iff u is finite and $u' = uw$ for some $w \in V^\infty$. Let H_n denote the set of all histories of width n; then H_n is a Scott domain (an ω-algebraic, bounded complete CPO) under the pointwise ordering. Clearly, we may regard B_n as a subset of H_n. As was the case for buffers, we can form the "horizontal composition" of a history $\beta \in H_m$ and a history $\eta \in H_n$, to obtain a history $\beta\eta \in H_{m+n}$. Although vertical composition of arbitrary histories does not make sense, in general, it does always make sense to form the vertical composition of a history $\beta \in H_n$ and a *buffer* $\delta \in B_n$, to obtain a history $\delta;\beta \in H_n$.

A *Kahn function of type* (m,n) is a Scott-continuous function $f : H_m \to H_n$. We use the name "Kahn function" because these functions are what Kahn used in his original study [4] of determinate dataflow networks. Let K_m^n denote the set of all Kahn functions of type (m,n). Some simple Kahn functions are: the identity function $1_n \in K_n^n$, the "diagonal" or "duplicator" function $d_n \in K_n^{n+n}$ defined by $d_n(\eta) = \eta\eta$, the "terminator" function $t_n \in K_n^0$ defined by $t_n(\eta) = \epsilon$, and the "exchange" or "symmetry" function $s_{n,n'} \in K_{n+n'}^{n'+n}$ defined by $s_{n,n'}(\eta\eta') = \eta'\eta$. We also have, for each $\delta \in B_n$, the "buffering function" $\hat{\delta} \in K_n^n$ defined by $\hat{\delta}(\eta) = \delta;\eta$. If $f \in K_m^n$ and $f' \in K_{m'}^{n'}$ are Kahn functions, then we use the notation $f \otimes f'$ to denote the Kahn function in $K_{m+m'}^{n+n'}$ defined by

$$(f \otimes f')(\beta\beta') = (f(\beta))(f'(\beta')).$$

We now define the interpretation of each standard network expression P of type (m,n) as a Kahn function $[P]_0 : H_m \to H_n$.

$$\begin{aligned}
[\mathbf{1}_n^\delta]_0 &= \hat{\delta} \\
[\mathbf{s}_{n,n'}^{\delta'\delta}]_0 &= (\hat{\delta}' \otimes \hat{\delta}) \circ s_{n,n'} \\
[\mathbf{d}_n^{\delta\delta'}]_0 &= (\hat{\delta} \otimes \hat{\delta}') \circ d_n \\
[\mathbf{t}_n]_0 &= t_n \\
[P \cdot \delta]_0 &= [P]_0 \circ \hat{\delta} \\
[\delta \cdot P]_0 &= \hat{\delta} \circ [P]_0 \\
[P \otimes P']_0 &= [P]_0 \otimes [P']_0 \\
[P \bowtie Q]_0 &= [P]_0 \bowtie [Q]_0,
\end{aligned}$$

where, for the last definition, given a Kahn function $f \in K_{m+q}^{n+p}$, and a Kahn function $g \in K_p^q$, define Ψ to be the continuous functional

$$\Psi : K_m^{n+p} \to K_m^{n+p} : h \mapsto f \circ (1_m \otimes (t_n \otimes g)) \circ (1_m \otimes h) \circ d_m,$$

let $\mu\Psi$ denote the least fixed point of Ψ, and define $f \bowtie g = (1_n \otimes t_p) \circ (\mu\Psi)$.

We need two additional operations on Kahn functions. For $f \in K_{m+s}^{n+r}$ and $g \in K_{r+q}^{s+p}$, define $f\,{}_s\Diamond_r\, g \in K_{m+q}^{n+p}$ by

$$f\,{}_s\Diamond_r\, g = ((1_n \otimes s_{r+s,p}) \circ (f \otimes g) \circ (1_m \otimes s_{q,s+r}) \circ (1_{m+q} \otimes s_{r,s})) \bowtie 1_{r+s}.$$

For $f \in K^{n+p}_{m+q}$ and $g \in K^{n'+p'}_{m'+q'}$, define $f^p_q \Box^{p'}_{q'} g \in K^{n+n'+p+p'}_{m+m'+q+q'}$ by

$$f^p_q \Box^{p'}_{q'} g = (1_n \otimes s_{p,n'} \otimes 1_{p'}) \circ (f \otimes g) \circ (1_m \otimes s_{m',q} \otimes 1_{q'}).$$

We omit the subscripts and superscripts of $\Diamond$ and $\Box$ when they are clear from the context. The significance of these somewhat bizarre-looking definitions is made clearer by the following result:

Lemma 1

1. *$f \bowtie (g \bowtie h) = (f\,{}_s\Diamond_r\, g) \bowtie h$, whenever $f \in K^{n+r}_{m+s}$, $g \in K^{s+p}_{r+q}$, and $h \in K^q_p$.*
2. *$(f \bowtie g) \otimes (f' \bowtie g') = (f^p_q \Box^{p'}_{q'} f') \bowtie (g \otimes g')$, whenever $f \in K^{n+p}_{m+q}$, $f' \in K^{n'+p'}_{m'+q'}$, $g \in K^q_p$, and $g' \in K^{q'}_{p'}$.*

The operations $\Diamond$ and $\Box$ also have some familiar-looking properties, which we shall need later:

Lemma 2

1. *$s_{m,n}\ {}^{n}_{m}\Box^{n'}_{m'}\ s_{m',n'} = s_{m+m',n+n'}$.*
2. *For all $f \in K^{n+q}_{m+p}$, $g \in K^{n'+q'}_{m'+p'}$, and $h \in K^{n''+q''}_{m''+p''}$,*
$$(f\ {}^{q}_{p}\Box^{q'}_{p'}\ g)\ {}^{q+q'}_{p+p'}\Box^{q''}_{p''}\ h = f\ {}^{q}_{p}\Box^{q'+q''}_{p'+p''}\ (g\ {}^{q'}_{p'}\Box^{q''}_{p''}\ h).$$
3. *For all $f \in K^n_m$,*
$$f\ {}_m\Diamond_n\ s_{m,n} = f = s_{m,n}\ {}_n\Diamond_m\ f.$$
4. *For all $f \in K^{n+p}_{m+q}$, $g \in K^{q+r}_{p+s}$, and $h \in K^{s+n'}_{r+m'}$,*
$$(f\ {}_q\Diamond_p\ g)\ {}_s\Diamond_r\ h = f\ {}_q\Diamond_p\ (g\ {}_s\Diamond_r\ h).$$
5. *For all $f \in K^{n+p}_{m+q}$, $g \in K^{q+s}_{p+r}$, $f' \in K^{n'+p'}_{m'+q'}$, and $g' \in K^{q'+s'}_{p'+r'}$,*
$$(f_q\Diamond_p g)\ {}^{s}_{r}\Box^{s'}_{r'}\ (f'\ {}_{q'}\Diamond_{p'} g') = (f\ {}^{p}_{q}\Box^{p'}_{q'}\ f')\ {}_{q+q'}\Diamond_{p+p'}\ (g\ {}^{s}_{r}\Box^{s'}_{r'}\ g').$$

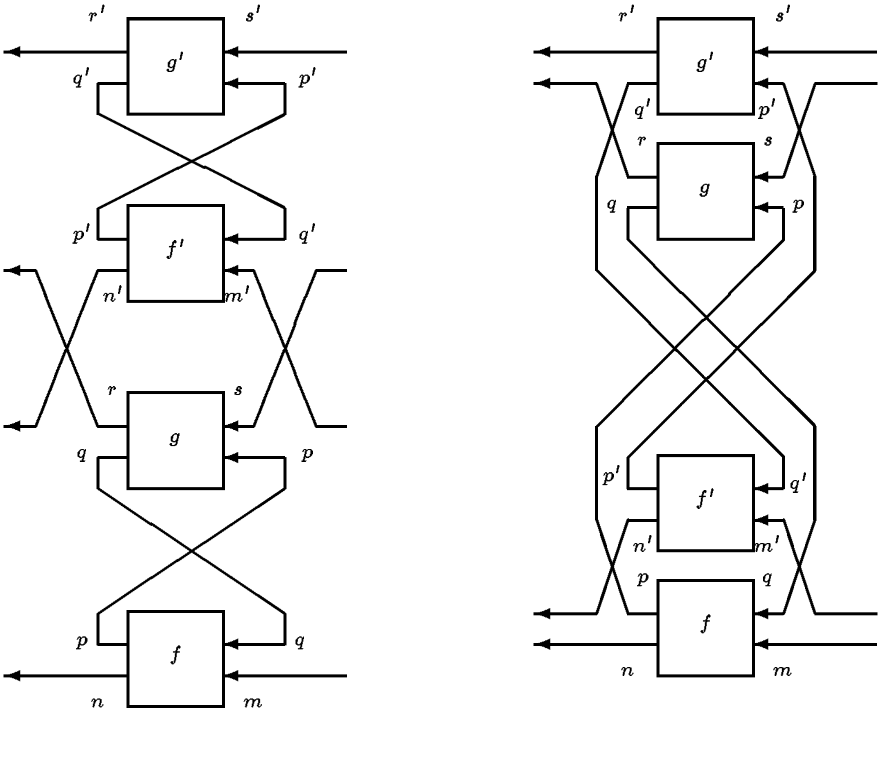
Figure 2 suggests why (5) is true. Similar figures may be drawn for the other equations.

Call a Kahn function *simple* if it is $[P]_0$ for some standard network expression $[P]$. If S is a dataflow scheme, and $m, n \in \mathrm{Ord}$, then define a *network of type* (m, n) *over* S to be a formal expression of the form

$$\langle f \mid x_1, x_2, \ldots, x_k \rangle,$$

where $f \in K^{n+p}_{m+q}$ is a simple Kahn function, $k \geq 0$, and $x_i \in S^{q_i}_{p_i}$ for $1 \leq i \leq k$ are such that $\sum_{i=1}^k p_i = p$ and $\sum_{i=1}^k q_i = q$. Intuitively, we have in mind a collection of k component networks, x_1, x_2, $\ldots$, x_k, connected in a mutual feedback loop through f (see Figure 3). We write $\langle f \mid \cdot \rangle$ when $k = 0$.

We now define a mapping that interprets each network expression P of type (m, n) as a network $[P]$ of type (m, n).



$(f \diamond g) \square (f' \diamond g')$ $(f \square f') \diamond (g \square g')$

Figure 2: Assertion (5) of Lemma 2

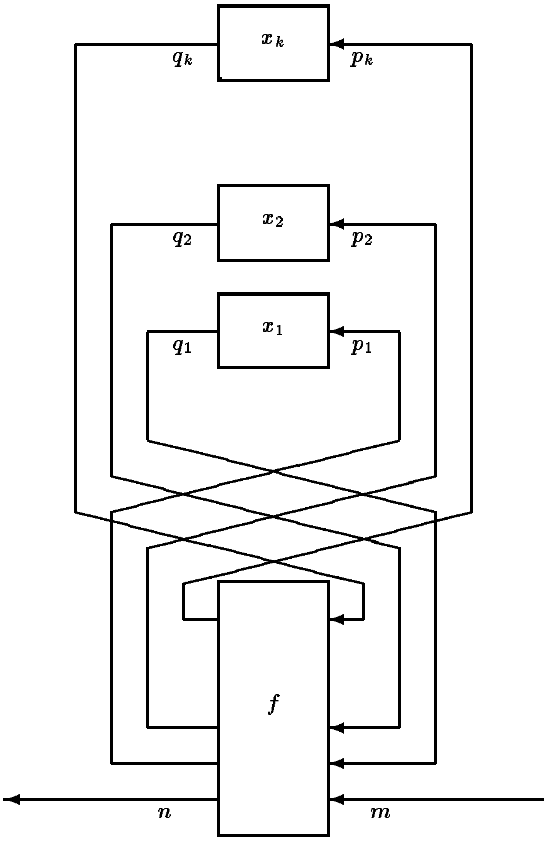


Figure 3: The Network $\langle f \mid x_1, x_2, \ldots, x_k \rangle$

- If P is a standard network expression, then $[P] = \langle [P]_0 \mid \cdot \rangle$.
- If $x \in S^n_m$, then $[x] = \langle s_{m,n} \mid x \rangle$.
- If $[P] = \langle f \mid x_1, x_2, \ldots, x_k \rangle$ with $f \in K^{n+p}_{m+q}$, then

$$[P \cdot \beta] = \langle f \circ (\hat{\beta} \otimes \hat{\epsilon_q}) \mid x_1, x_2, \ldots, x_k \rangle.$$

- If $[P] = \langle f \mid x_1, x_2, \ldots, x_k \rangle$ with $f \in K^{n+p}_{m+q}$, then

$$[\gamma \cdot P] = \langle (\hat{\gamma} \otimes \hat{\epsilon_p}) \circ f \mid x_1, x_2, \ldots, x_k \rangle.$$

- If $[P] = \langle f \mid x_1, x_2, \ldots, x_k \rangle$ and $[Q] = \langle g \mid y_1, y_2, \ldots, y_l \rangle$, then

$$[P \otimes Q] = \langle f \Box g \mid x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_l \rangle.$$

- If $[P] = \langle f \mid \cdot \rangle$ and $[Q] = \langle g \mid x_1, x_2, \ldots, x_k \rangle$, then

$$[P \bowtie Q] = \langle f \Diamond g \mid x_1, x_2, \ldots, x_k \rangle.$$

Network expressions P and Q in $(NS)^n_m$ are defined to be *equivalent* if the networks $[P]$ and $[Q]$ are identical.

With a suitable definition of "network computation," we can extend the interpretation of network expressions to computation expressions as well. Suppose $M = \langle f \mid x_1, \ldots, x_k \rangle$ and $N = \langle g \mid y_1, \ldots, y_k \rangle$ are networks. A *transition from M to N* is an expression of the form

$$\frac{\begin{array}{c} x_1 \overset{\eta_1}{\underset{\delta_1}{\Longrightarrow}} y_1 \\ x_2 \overset{\eta_2}{\underset{\delta_2}{\Longrightarrow}} y_2 \\ \cdots \\ x_k \overset{\eta_k}{\underset{\delta_k}{\Longrightarrow}} y_k \end{array}}{M \overset{\gamma}{\underset{\beta}{\Longrightarrow}} N}$$

such that for some simple Kahn function h we have

$$f = (\hat{\gamma} \otimes \hat{\delta}) \circ h \qquad \text{and} \qquad g = h \circ (\hat{\beta} \otimes \hat{\eta}),$$

where $\delta = \delta_1 \delta_2 \ldots \delta_k$ and $\eta = \eta_1 \eta_2 \ldots \eta_k$. A *computation sequence from M to N* is a sequence of transitions of the form:

$$\frac{\begin{array}{c} x_{01} \overset{\eta_{11}}{\underset{\delta_{11}}{\Longrightarrow}} x_{11} \\ x_{02} \overset{\eta_{12}}{\underset{\delta_{12}}{\Longrightarrow}} x_{12} \\ \cdots \\ x_{0k} \overset{\eta_{1k}}{\underset{\delta_{1k}}{\Longrightarrow}} x_{1k} \end{array}}{M_0 \overset{\gamma_1}{\underset{\beta_1}{\Longrightarrow}} M_1} \qquad \frac{\begin{array}{c} x_{11} \overset{\eta_{21}}{\underset{\delta_{21}}{\Longrightarrow}} x_{21} \\ x_{12} \overset{\eta_{22}}{\underset{\delta_{22}}{\Longrightarrow}} x_{22} \\ \cdots \\ x_{1k} \overset{\eta_{2k}}{\underset{\delta_{2k}}{\Longrightarrow}} x_{2k} \end{array}}{M_1 \overset{\gamma_2}{\underset{\beta_2}{\Longrightarrow}} M_2} \qquad \cdots \qquad \frac{\begin{array}{c} x_{l-1,1} \overset{\eta_{l1}}{\underset{\delta_{l1}}{\Longrightarrow}} x_{l1} \\ x_{l-1,2} \overset{\eta_{l2}}{\underset{\delta_{l2}}{\Longrightarrow}} x_{l2} \\ \cdots \\ x_{l-1,k} \overset{\eta_{lk}}{\underset{\delta_{lk}}{\Longrightarrow}} x_{lk} \end{array}}{M_{l-1} \overset{\gamma_l}{\underset{\beta_l}{\Longrightarrow}} M_l}$$

with $M = M_0$ and $M_l = N$. With each computation sequence from M to N we associate buffers $\beta = \beta_1; \beta_2; \ldots; \beta_l \in B_m$ and $\gamma = \gamma_1; \gamma_2; \ldots; \gamma_l \in B_n$, by composing "below the line," and a k-tuple of computations

$$\left\langle \begin{array}{c} x_1 \overset{\eta_1}{\underset{\delta_1}{\Longrightarrow}} y_1, \\ x_2 \overset{\eta_2}{\underset{\delta_2}{\Longrightarrow}} y_2, \\ \ldots, \\ x_k \overset{\eta_k}{\underset{\delta_k}{\Longrightarrow}} y_k \end{array} \right\rangle$$

by forming the compositions (in the categories $S^{n_i}_{m_i}$) of the sequences of computations appearing "above the line." Define two such computation sequences to be *equivalent* if the corresponding associated buffers and k-tuples of computations are equal. Define the *network computations from* M *to* N to be the equivalence classes of computation sequences from M to N.

Theorem 1 *For each* (m, n)*, the set* $(\mathcal{NS})^n_m$ *of networks of type* (m, n) *is the set of objects of a category having the network computations as arrows. The mappings, taking each network computation to the associated buffers, determine functors* $d_0 : (\mathcal{NS})^n_m \rightarrow B_m$ *and* $d_1 : (\mathcal{NS})^n_m \rightarrow B_n$*, so that* $\mathcal{NS}$ *is a dataflow scheme.*

We now give the interpretation of computation expressions as network computations. The standard computation axioms are interpreted as follows:

$$\begin{aligned}
[\mathbf{1}^{\gamma;\delta}_n \overset{\gamma}{\underset{\beta}{\Longrightarrow}} \mathbf{1}^{\delta;\beta}_n] &= \langle \widehat{\gamma;\delta} \mid \cdot \rangle \overset{\gamma}{\underset{\beta}{\Longrightarrow}} \langle \widehat{\delta;\beta} \mid \cdot \rangle \\
[\mathbf{s}^{(\gamma';\delta')(\gamma;\delta)}_{n,n'} \overset{\gamma'\gamma}{\underset{\beta\beta'}{\Longrightarrow}} \mathbf{s}^{(\delta';\beta')(\delta;\beta)}_{n,n'}] & \\
&= \langle (\widehat{\gamma';\delta'} \otimes \widehat{\gamma;\delta}) \circ s_{n,n'} \mid \cdot \rangle \overset{\gamma'\gamma}{\underset{\beta\beta'}{\Longrightarrow}} \langle (\widehat{\delta';\beta'} \otimes \widehat{\delta;\beta}) \circ s_{n,n'} \mid \cdot \rangle \\
[\mathbf{d}^{(\gamma;\delta)(\gamma';\delta')}_n \overset{\gamma\gamma'}{\underset{\beta}{\Longrightarrow}} \mathbf{d}^{(\delta;\beta)(\delta';\beta)}_n] & \\
&= \langle (\widehat{\gamma;\delta} \otimes \widehat{\gamma';\delta'}) \circ d_n \mid \cdot \rangle \overset{\gamma\gamma'}{\underset{\beta}{\Longrightarrow}} \langle (\widehat{\delta;\beta} \otimes \widehat{\delta';\beta}) \circ d_n \mid \cdot \rangle \\
[\mathbf{t}_n \underset{\beta}{\Longrightarrow} \mathbf{t}_n] &= \langle t_n \mid \cdot \rangle \underset{\beta}{\Longrightarrow} \langle t_n \mid \cdot \rangle
\end{aligned}$$

A *nonstandard computation axiom* $x \overset{\gamma}{\underset{\beta}{\Longrightarrow}} y$ is interpreted as the equivalence class of the computation sequence

$$\frac{x \Longrightarrow x}{\langle s_{m,n} \mid x \rangle \underset{\beta}{\Longrightarrow} \langle (\widehat{\epsilon} \otimes \widehat{\beta}) \circ s_{m,n} \mid x \rangle}$$

$$\frac{x \overset{\gamma}{\underset{\beta}{\Longrightarrow}} y}{\langle (\widehat{\epsilon} \otimes \widehat{\beta}) \circ s_{m,n} \mid x \rangle\rangle \Longrightarrow \langle (\widehat{\gamma} \otimes \widehat{\epsilon}) \circ s_{m,n} \mid y \rangle}$$

$$\frac{y \Longrightarrow y}{\langle (\widehat{\gamma} \otimes \widehat{\epsilon}) \circ s_{m,n} \mid y \rangle\rangle \overset{\gamma}{\Longrightarrow} \langle s_{m,n} \mid y \rangle}$$

In other words, since computation sequences for networks are equivalent if and only if the the corresponding associated buffers "below the line" and tuples of computations "above the line" are identical, a nonstandard computation axiom $x \overset{\gamma}{\underset{\beta}{\Longrightarrow}} y$ is interpreted as the equivalence class represented by the buffers β and γ and the 1-tuple $\langle x \overset{\gamma}{\underset{\beta}{\Longrightarrow}} y \rangle$.

The interpretations for the remaining types of computation expressions are analogous. For example, if the computation expression

$$\frac{\cdots}{P \overset{\gamma}{\underset{\eta}{\Longrightarrow}} Q}$$

is interpreted as an equivalence class of computation sequences from

$$\langle f \mid x_1, x_2, \ldots, x_k\rangle \qquad \text{to} \qquad \langle g \mid y_1, y_2, \ldots, y_k\rangle$$

represented by the buffers η and γ and composite k-tuple

$$\left\langle \begin{array}{ccccc} x_{01} \overset{\sigma_{11}}{\underset{\rho_{11}}{\Longrightarrow}} x_{11} & x_{11} \overset{\sigma_{21}}{\underset{\rho_{21}}{\Longrightarrow}} x_{21} & & x_{l-1,1} \overset{\sigma_{l1}}{\underset{\rho_{l1}}{\Longrightarrow}} x_{l1} \\ x_{02} \overset{\sigma_{12}}{\underset{\rho_{12}}{\Longrightarrow}} x_{12} & x_{12} \overset{\sigma_{22}}{\underset{\rho_{22}}{\Longrightarrow}} x_{22} & \cdots & x_{l-1,2} \overset{\sigma_{l2}}{\underset{\rho_{l2}}{\Longrightarrow}} x_{l2} \\ \cdots & \cdots & & \cdots \\ x_{0k} \overset{\sigma_{1k}}{\underset{\rho_{1k}}{\Longrightarrow}} x_{1k} & x_{1k} \overset{\sigma_{2k}}{\underset{\rho_{2k}}{\Longrightarrow}} x_{2k} & & x_{l-1,k} \overset{\sigma_{lk}}{\underset{\rho_{lk}}{\Longrightarrow}} x_{lk} \end{array} \right\rangle$$

then a computation expression for an input buffered network:

$$\frac{\dfrac{\cdots}{P \overset{\gamma}{\underset{\eta}{\Longrightarrow}} Q}}{P \cdot (\eta;\delta) \overset{\gamma}{\underset{\beta}{\Longrightarrow}} Q \cdot (\delta;\beta)}$$

is interpreted as the equivalence class of computation sequences from

$$\langle f \circ (\widehat{\eta;\delta} \otimes \hat{\epsilon}) \mid x_1, x_2, \ldots, x_k\rangle \qquad \text{to} \qquad \langle g \circ (\widehat{\delta;\beta} \otimes \hat{\epsilon}) \mid y_1, y_2, \ldots, y_k\rangle$$

represented by the buffers β and γ, and the same k-tuple of computations. To verify that this makes sense, it is of course necessary in each case to show that the given buffers and k-tuple really do represent a nonempty class of computation sequences, but this is simply a matter of showing how to fill in the intermediate states as we did in the case for nonstandard computation axioms. For example, the network $\langle f \circ (\widehat{\eta;\delta} \otimes \hat{\epsilon}) \mid x_1, x_2, \ldots, x_k\rangle$ can absorb the entire input β on the first transition to become the network $\langle f \circ (\eta;\widehat{\delta;\beta} \otimes \hat{\epsilon}) \mid x_1, x_2, \ldots, x_k\rangle$. The computation then proceeds *mutatis mutandis* as in a computation $\langle f \mid x_1, x_2, \ldots, x_k\rangle \overset{\gamma}{\underset{\eta}{\Longrightarrow}} \langle g \mid y_1, y_2, \ldots, y_k\rangle$. We omit the remaining details.

4 Dataflow Algebra

Recall that if S and S' are spans from A to B in a category, then an *arrow of spans* from S to S' is a morphism $F : S \to S'$ such that $d_0'F = d_0$ and $d_1'F = d_1$. If S and S' are dataflow schemes, then define a *morphism* from S to S' to be a collection

$$F = \{F_m^n : m, n \in \mathrm{Ord}\}$$

of arrows of spans, where $F_m^n : S_m^n \to (S')_m^n$. Let **DflSch** denote the category of dataflow schemes and their morphisms.

It is easy to see that the construction $S \mapsto \mathcal{N}S$ defined in the previous section is the object map of an endofunctor

$$\mathcal{N} : \mathbf{DflSch} \to \mathbf{DflSch}.$$

Moreover, we have natural transformations $\eta : 1 \stackrel{\cdot}{\longrightarrow} \mathcal{N}$ and $\mu : \mathcal{N}\mathcal{N} \stackrel{\cdot}{\longrightarrow} \mathcal{N}$, defined on states (objects) by:

$$\begin{aligned} (\eta S)^n_m :& \quad x \mapsto \langle s_{m,n} \mid x \rangle \\ (\mu S)^m_n :& \quad \langle f \mid \langle g_1 \mid X_1 \rangle, \ldots, \langle g_k \mid X_k \rangle \rangle \mapsto \langle f \Diamond (g_1 \Box \ldots \Box g_k) \mid X_1, \ldots, X_k \rangle. \end{aligned}$$

and defined on computations (arrows) by the requirement that $(\eta S)^n_m$ and $(\mu S)^n_m$ take computations represented by buffers β, γ and k-tuples $\langle x_i \overset{\eta_i}{\underset{\delta_i}{\Longrightarrow}} y_i : 1 \leq i \leq k \rangle$ to computations represented by the same buffers and k-tuples (but with appropriately different domains and codomains).

Our goal is to show that $(\mathcal{N}, \eta, \mu)$ is a monad in **DflSch**. Although this can be verified directly, things are simpler if we view $\mathcal{N}$ as a composite $\mathcal{L}\mathcal{P}$ of a monad $\mathcal{P}$ that treats the parallel composition construction alone, and a monad $\mathcal{L}$ that represents the feedback construction alone. These two constructions are related via a distributive law of feedback over parallel composition.

Thus, let $\mathcal{P} : \mathbf{DflSch} \to \mathbf{DflSch}$ be the mapping that takes a dataflow scheme S to the dataflow scheme $\mathcal{P}S$, where the states of $(\mathcal{P}S)^n_m$ are all k-tuples

$$\langle x_1, x_2, \ldots, x_k \rangle$$

with $k \geq 0$ and where the $x_i \in S^{n_i}_{m_i}$ are such that $\sum_{i=1}^k m_i = m$ and $\sum_{i=1}^k n_i = n$. The computations from $\langle x_1, x_2, \ldots, x_k \rangle$ to $\langle y_1, y_2, \ldots, y_k \rangle$ in $(\mathcal{P}S)^n_m$ are all k-tuples

$$\langle x_1 \overset{\gamma_1}{\underset{\beta_1}{\Longrightarrow}} y_1, x_2 \overset{\gamma_2}{\underset{\beta_2}{\Longrightarrow}} y_2, \ldots, x_k \overset{\gamma_k}{\underset{\beta_k}{\Longrightarrow}} y_k \rangle.$$

The functor $d_0 : (\mathcal{P}S)^n_m \to B_m$ maps such a computation to the buffer $\beta_1\beta_2\ldots\beta_k$ and the functor $d_1 : (\mathcal{P}S)^n_m \to B_n$ maps it to $\gamma_1\gamma_2\ldots\gamma_k$.

For a morphism $\phi S \to S'$ of dataflow schemes, define $\mathcal{P}\phi : PS \to PS'$ on states by:

$$(\mathcal{P}\phi)^n_m(\langle x_1, x_2, \ldots, x_k \rangle) = \langle \phi^{n_1}_{m_1}(x_1), \phi^{n_2}_{m_2}(x_2), \ldots, \phi^{n_k}_{m_k}(x_k) \rangle$$

and on computations by:

$$(\mathcal{P}\phi)^n_m(\langle x_i \overset{\gamma_i}{\underset{\beta_i}{\Longrightarrow}} y_i : 1 \leq i \leq k \rangle) = \langle \phi^{n_i}_{m_i}(x_i \overset{\gamma_i}{\underset{\beta_i}{\Longrightarrow}} y_i) : 1 \leq i \leq k \rangle.$$

It is easy to see that we have defined a functor $\mathcal{P} : \mathbf{DflSch} \to \mathbf{DflSch}$. For each dataflow scheme S, define the morphism $\eta^{\mathcal{P}} S : S \to \mathcal{P}S$ on states by:

$$(\eta^{\mathcal{P}})^n_m(x) = \langle x \rangle$$

and on computations by

$$(\eta^{\mathcal{P}})^n_m(x \overset{\gamma}{\underset{\beta}{\Longrightarrow}} y) \;=\; \langle x \overset{\gamma}{\underset{\beta}{\Longrightarrow}} y \rangle.$$

Define the morphism $\mu^{\mathcal{P}} S : \mathcal{P}\mathcal{P}S \to \mathcal{P}S$ on states by:

$$(\mu^{\mathcal{P}})^n_m(\langle \langle x_{11}, \ldots, x_{1,k_1} \rangle, \ldots, \langle x_{l1}, \ldots, x_{l,k_l} \rangle \rangle) = \langle x_{11}, \ldots, x_{1,k_1}, \ldots, x_{l1}, \ldots, x_{l,k_l} \rangle.$$

and similarly on computations. The following result is routine:

Lemma 3 $(\mathcal{P}, \eta^{\mathcal{P}}, \mu^{\mathcal{P}})$ *is a monad in* **DflSch**.

Another way of viewing $\mathcal{P}$ is suggested by the observation that **DflSch** has a monoidal structure given by

$$(S \otimes T)_m^n = \coprod_{\substack{p+r=m \\ q+s=n}} (S_p^q \times T_r^s).$$

The dataflow scheme I, with I_0^0 a one-object, one-arrow category and I_m^n empty for all other m and n, serves as the unit for $\otimes$. Then $\mathcal{P}$ is the functor that takes a dataflow scheme S to the "free monoid object on S generators" in **DflSch**.

We now consider the feedback construction. Let $\mathcal{L}$: **DflSch** $\to$ **DflSch** be the mapping that takes a dataflow scheme S to the dataflow scheme $\mathcal{L}S$, where the states of $(\mathcal{L}S)_m^n$ are all formal expressions

$$\langle f \mid x \rangle$$

with $f \in K_{m+q}^{n+p}$ a simple Kahn function, and $x \in S_p^q$. The computations of $(\mathcal{L}S)_m^n$ are obtained as equivalence classes of computation sequences as follows: Define the *transitions* of $(\mathcal{L}S)_m^n$ from $\langle f \mid x \rangle$ to $\langle g \mid y \rangle$ to be all expressions of the form

$$\frac{x \overset{\eta}{\underset{\delta}{\Longrightarrow}} y}{\langle f \mid x \rangle \overset{\gamma}{\underset{\beta}{\Longrightarrow}} \langle g \mid y \rangle}$$

such that for some simple Kahn function h we have $f = (\hat{\gamma} \otimes \hat{\delta}) \circ h$ and $g = h \circ (\hat{\beta} \otimes \hat{\eta})$. A *computation sequence* from $\langle f \mid x \rangle$ to $\langle g \mid y \rangle$ is a sequence of transitions of the form

$$\frac{z_0 \overset{\eta_1}{\underset{\delta_1}{\Longrightarrow}} z_1}{\langle h_0 \mid z_0 \rangle \overset{\gamma_1}{\underset{\beta_1}{\Longrightarrow}} \langle h_1 \mid z_1 \rangle} \qquad \frac{z_1 \overset{\eta_2}{\underset{\delta_2}{\Longrightarrow}} z_2}{\langle h_1 \mid z_1 \rangle \overset{\gamma_2}{\underset{\beta_2}{\Longrightarrow}} \langle h_2 \mid z_2 \rangle} \qquad \cdots \qquad \frac{z_{l-1} \overset{\eta_l}{\underset{\delta_l}{\Longrightarrow}} z_l}{\langle h_{l-1} \mid z_{l-1} \rangle \overset{\gamma_l}{\underset{\beta_l}{\Longrightarrow}} \langle h_l \mid z_l \rangle}$$

such that $f = h_0$, $x = z_0$, $h_l = g$, and $z_l = y$. As before, we define two such computation sequences to be *equivalent* if the corresponding composite buffers "below the line" and the corresponding compositions in S_p^q of the computations "above the line" are identical. The *computations* from $\langle f \mid x \rangle$ to $\langle g \mid y \rangle$ in $(\mathcal{L}S)_m^n$ are the equivalence classes of computation sequences from $\langle f \mid x \rangle$ to $\langle g \mid y \rangle$.

The functor $d_0 : (\mathcal{L}S)_m^n \to B_m$ takes a computation determined by the computation sequence

$$\frac{z_0 \overset{\eta_1}{\underset{\delta_1}{\Longrightarrow}} z_1}{\langle h_0 \mid z_0 \rangle \overset{\gamma_1}{\underset{\beta_1}{\Longrightarrow}} \langle h_1 \mid z_1 \rangle} \qquad \frac{z_1 \overset{\eta_2}{\underset{\delta_2}{\Longrightarrow}} z_2}{\langle h_1 \mid z_1 \rangle \overset{\gamma_2}{\underset{\beta_2}{\Longrightarrow}} \langle h_2 \mid z_2 \rangle} \qquad \cdots \qquad \frac{z_{l-1} \overset{\eta_l}{\underset{\delta_l}{\Longrightarrow}} z_l}{\langle h_{l-1} \mid z_{l-1} \rangle \overset{\gamma_l}{\underset{\beta_l}{\Longrightarrow}} \langle h_l \mid z_l \rangle}$$

to the composite buffer $\beta_1;\beta_2;\ldots;\beta_l$. Similarly, the functor $d_1 : (\mathcal{L}S)_m^n \to B_n$ takes such a computation to $\gamma_1;\gamma_2;\ldots;\gamma_l$. Thus, $(\mathcal{L}S)_m^n$ becomes a span from B_m to B_n in **Cat**.

For a morphism $\phi : S \to S'$ of dataflow schemes, define $(\mathcal{L}\phi)_m^n$ on states by

$$(\mathcal{L}\phi)_m^n(\langle f \mid x \rangle) = \langle f \mid \phi_p^q(x) \rangle.$$

on transitions by the condition

$$(\mathcal{L}\phi)_m^n \left(\frac{x \overset{\eta}{\underset{\delta}{\Longrightarrow}} y}{\langle f \mid x \rangle \overset{\gamma}{\underset{\beta}{\Longrightarrow}} \langle g \mid y \rangle} \right) = \frac{\phi_p^q(x) \overset{\eta}{\underset{\delta}{\Longrightarrow}} \phi_p^q(y)}{\langle f \mid \phi_p^q(x) \rangle \overset{\gamma}{\underset{\beta}{\Longrightarrow}} \langle g \mid \phi_p^q(y) \rangle}$$

and extended to all computations by the requirement of functoriality. We have thus defined a functor $\mathcal{L} : \mathbf{DflSch} \to \mathbf{DflSch}$.

For each dataflow scheme S, define $\eta^{\mathcal{L}} : S \to \mathcal{L}S$ on states by

$$(\eta^{\mathcal{L}}S)^n_m(x) = \langle s_{m,n} \mid x\rangle$$

and on computations by

$$(\eta^{\mathcal{L}}S)^n_m(x \overset{\gamma}{\underset{\beta}{\Longrightarrow}} y) \;=\; \dfrac{x \Longrightarrow x}{\langle s_{m,n} \mid x\rangle \underset{\beta}{\Longrightarrow} \langle(\hat{\epsilon}\otimes\hat{\beta})\circ s_{m,n} \mid x\rangle} \; \dfrac{x \overset{\gamma}{\underset{\beta}{\Longrightarrow}} y}{\langle(\hat{\epsilon}\otimes\hat{\beta})\circ s_{m,n} \mid x\rangle \Longrightarrow \langle(\hat{\gamma}\otimes\hat{\epsilon})\circ s_{m,n} \mid y\rangle} \; \dfrac{y \Longrightarrow y}{\langle(\hat{\gamma}\otimes\hat{\epsilon})\circ s_{m,n} \mid y\rangle \overset{\gamma}{\Longrightarrow} \langle s_{m,n} \mid y\rangle}$$

Define $\mu^{\mathcal{L}} : \mathcal{L}\mathcal{L}S \to \mathcal{L}S$ on states by

$$(\mu^{\mathcal{L}}S)^n_m(\langle f \mid \langle f' \mid x\rangle\rangle) = \langle f \Diamond f' \mid x\rangle,$$

and on computations by the condition

$$(\mu^{\mathcal{L}}S)^n_m\left(\dfrac{\dfrac{x \overset{\sigma}{\underset{\rho}{\Longrightarrow}} y}{\langle f' \mid x\rangle \overset{\eta}{\underset{\delta}{\Longrightarrow}} \langle g' \mid y\rangle}}{\langle f \mid \langle f' \mid x\rangle\rangle \overset{\gamma}{\underset{\beta}{\Longrightarrow}} \langle g \mid \langle g' \mid y\rangle\rangle}\right) \;=\; \dfrac{x \overset{\sigma}{\underset{\rho}{\Longrightarrow}} y}{\langle f\Diamond f' \mid x\rangle \overset{\gamma}{\underset{\beta}{\Longrightarrow}} \langle g\Diamond g' \mid y\rangle},$$

and the requirement of functoriality. One may verify that the morphisms $\eta^{\mathcal{L}}S$ and $\mu^{\mathcal{L}}S$ are the components of natural transformations $\eta^{\mathcal{L}} : 1 \stackrel{\cdot}{\longrightarrow} \mathcal{L}$ and $\mu^{\mathcal{L}} : \mathcal{L}\mathcal{L} \stackrel{\cdot}{\longrightarrow} \mathcal{L}$. Moreover, we have:

Lemma 4 *$(\mathcal{L}, \eta^{\mathcal{L}}, \mu^{\mathcal{L}})$ is a monad in* **DflSch**.

Proof – The proof uses Lemma 2. The proof of the associative law depends on the fact that $\Diamond$ is an associative operation on Kahn functions. The proof of the unit laws depends on the fact that the exchange maps $s_{m,n}$ are left and right units for $\Diamond$. ∎

We have defined the endofunctors $\mathcal{P}$ and $\mathcal{L}$ in such a way that $\mathcal{N} \cong \mathcal{L}\mathcal{P}$. We wish to show that the functor $\mathcal{N}$ underlies a monad $(\mathcal{N}, \eta, \mu)$ which is in some sense a kind of composite of $(\mathcal{P}, \eta^{\mathcal{P}}, \mu^{\mathcal{P}})$ and $(\mathcal{L}, \eta^{\mathcal{L}}, \mu^{\mathcal{L}})$. To do this, we use the following result about monads:

Lemma 5 *Suppose $(\mathcal{L}, \eta^{\mathcal{L}}, \mu^{\mathcal{L}})$ and $(\mathcal{P}, \eta^{\mathcal{P}}, \mu^{\mathcal{P}})$ are monads in a category C, and $\delta : \mathcal{P}\mathcal{L} \stackrel{\cdot}{\longrightarrow} \mathcal{L}\mathcal{P}$ is a natural transformation, such that the following conditions hold:*

1. $(\eta^{\mathcal{L}}\mathcal{P}) \cdot \eta^{\mathcal{P}} = (\mathcal{L}\eta^{\mathcal{P}}) \cdot \eta^{\mathcal{L}}$.
2. $\delta \cdot (\mathcal{P}\eta^{\mathcal{L}}) = \eta^{\mathcal{L}}\mathcal{P}$.

3. $\delta \cdot (\eta^{\mathcal{P}}\mathcal{L}) = \mathcal{L}\eta^{\mathcal{P}}$.

4. $(\mu^{\mathcal{L}}\mu^{\mathcal{P}}) \cdot (\mathcal{L}\delta\mathcal{P}) \cdot (\mathcal{L}\mathcal{P}\mu^{\mathcal{L}}\mu^{\mathcal{P}}) \cdot (\mathcal{L}\mathcal{P}\mathcal{L}\delta\mathcal{P}) \;=\; (\mu^{\mathcal{L}}\mu^{\mathcal{P}}) \cdot (\mathcal{L}\delta\mathcal{P}) \cdot (\mu^{\mathcal{L}}\mu^{\mathcal{P}}\mathcal{L}\mathcal{P}) \cdot (\mathcal{L}\delta\mathcal{P}\mathcal{L}\mathcal{P})$.

Let $\eta : 1 \stackrel{\cdot}{\longrightarrow} \mathcal{L}\mathcal{P}$ be $(\eta^{\mathcal{L}}\mathcal{P}) \cdot \eta^{\mathcal{P}}$ (equivalently, $(\mathcal{L}\eta^{\mathcal{P}}) \cdot \eta^{\mathcal{L}}$), and $\mu : \mathcal{L}\mathcal{P}\mathcal{L}\mathcal{P} \to \mathcal{L}\mathcal{P}$ be $(\mu^{\mathcal{L}}\mu^{\mathcal{P}}) \cdot (\mathcal{L}\delta\mathcal{P})$. Then $(\mathcal{L}\mathcal{P}, \eta, \mu)$ is also a monad in C. Moreover, the relationship $\varepsilon = \varepsilon^{\mathcal{L}} \circ (\mathcal{L}\varepsilon^{\mathcal{P}})$ determines a bijection between $\mathcal{L}\mathcal{P}$-algebra structures $\varepsilon : \mathcal{L}\mathcal{P}S \to S$ and pairs $(\varepsilon^{\mathcal{L}}, \varepsilon^{\mathcal{P}})$ consisting of an $\mathcal{L}$ algebra structure $\varepsilon^{\mathcal{L}} : \mathcal{L}S \to S$ and a $\mathcal{P}$-algebra structure $\varepsilon^{\mathcal{P}} : \mathcal{P}S \to S$ satisfying the distributive law: $\varepsilon^{\mathcal{P}} \circ (\mathcal{P}\varepsilon^{\mathcal{L}}) = \varepsilon^{\mathcal{L}} \circ (\mathcal{L}\varepsilon^{\mathcal{P}}) \circ (\delta S)$.

Proof – The proof that $(\mathcal{L}\mathcal{P}, \eta, \mu)$ is a monad is a straightforward diagram chase, using (4) to prove the associative law. The proof of the second assertion is also straightforward, once it is observed that the hypotheses imply $\mu \cdot (\mathcal{L}\eta^{\mathcal{P}}\mathcal{L}\eta^{\mathcal{P}}) = (\mathcal{L}\eta^{\mathcal{P}}) \cdot \mu^{\mathcal{L}}$, and $\mu \cdot (\eta^{\mathcal{L}}\mathcal{P}\eta^{\mathcal{L}}\mathcal{P}) = (\eta^{\mathcal{L}}\mathcal{P}) \cdot \mu^{\mathcal{P}}$. ∎

For each dataflow scheme S, define the morphisms $\delta S : \mathcal{P}\mathcal{L}S \to \mathcal{L}\mathcal{P}S$ on states by

$$(\delta S)^n_m(\langle\langle f_1 \mid x_1\rangle, \ldots, \langle f_k \mid x_k\rangle\rangle) = \langle f_1 \Box \ldots \Box f_k \mid \langle x_1, \ldots, x_k\rangle\rangle.$$

on transitions by the requirement that $(\delta S)^n_m$ map a transition

$$\frac{\dfrac{x_1 \overset{\eta_1}{\underset{\delta_1}{\Longrightarrow}} y_1}{\langle f_1 \mid x_1\rangle \overset{\gamma_1}{\underset{\beta_1}{\Longrightarrow}} \langle g_1 \mid y_1\rangle} \quad \cdots \quad \dfrac{x_k \overset{\eta_k}{\underset{\delta_k}{\Longrightarrow}} y_k}{\langle f_k \mid x_k\rangle \overset{\gamma_k}{\underset{\beta_k}{\Longrightarrow}} \langle g_k \mid y_k\rangle}}{\langle\langle f_1 \mid x_1\rangle, \ldots, \langle f_k \mid x_k\rangle\rangle \overset{\gamma}{\underset{\beta}{\Longrightarrow}} \langle\langle g_1 \mid y_1\rangle, \ldots, \langle g_k \mid y_k\rangle\rangle}$$

to the transition

$$\frac{\dfrac{x_1 \overset{\eta_1}{\underset{\delta_1}{\Longrightarrow}} y_1 \quad \cdots \quad x_k \overset{\eta_k}{\underset{\delta_k}{\Longrightarrow}} y_k}{\langle x_1, \ldots, x_k\rangle \overset{\eta}{\underset{\delta}{\Longrightarrow}} \langle y_1, \ldots, y_k\rangle}}{\langle f_1 \Box \ldots \Box f_k \mid \langle x_1, \ldots, x_k\rangle\rangle \overset{\gamma}{\underset{\beta}{\Longrightarrow}} \langle g_1 \Box \ldots \Box g_k \mid \langle y_1, \ldots, y_k\rangle\rangle}$$

and extended to computations by the condition of functoriality. Then the δS are the components of a natural transformation $\delta : \mathcal{P}\mathcal{L} \stackrel{\cdot}{\longrightarrow} \mathcal{L}\mathcal{P}$.

Theorem 2 *$(\mathcal{N}, \eta, \mu)$ is a monad in* **DflSch**.

Proof – The proof uses Lemma 2 to establish that the monad $(\mathcal{L}, \eta^{\mathcal{L}}, \mu^{\mathcal{L}})$, the monad $(\mathcal{P}, \eta^{\mathcal{P}}, \mu^{\mathcal{P}})$, and the natural transformation $\delta : \mathcal{P}\mathcal{L} \stackrel{\cdot}{\longrightarrow} \mathcal{L}\mathcal{P}$ satisfy the conditions of Lemma 5. In particular, Lemma 2 allows us to infer the law:

$$\begin{aligned} &f \diamond ((g_1 \diamond (h_{11} \Box \ldots \Box h_{1k_1})) \Box \ldots \Box (g_l \diamond (h_{l1} \Box \ldots \Box h_{lk_l}))) \\ &\quad = (f \diamond (g_1 \Box \ldots \Box g_l)) \diamond (h_{11} \Box \ldots \Box h_{lk_l}) \end{aligned}$$

(for appropriate choices of subscripts and superscripts on the $\Box$ and $\diamond$ operators), which is required in the proof of condition (4). ∎

5 Fibrations

The concept of a fibration [3] concerns a functor $F : E \to B$, having the property that the morphisms of B "act" functorially on the *fibers* $F^{-1}(x)$, which are the preimages of objects x of B. That is to say, if for each object x of B we let $J_x : F^{-1}(x) \to E$ be the inclusion of the fiber $F^{-1}(x)$, then a fibration has the property that to each morphism $b : x \to x'$ there corresponds a functor $b^* : F^{-1}(x') \to F^{-1}(x)$ and a natural transformation $\theta_b : J_x b^* \stackrel{.}{\longrightarrow} J_{x'}$ such that a certain universal mapping property is satisfied. Dually, an "opfibration" [2] has the property that to each $b : x \to x'$ there corresponds a functor $b_* : F^{-1}(x) \to F^{-1}(x')$ and a natural transformation $\psi_b : J_x \stackrel{.}{\longrightarrow} J_{x'} b^*$. The components of the natural transformations θ_b are called *cartesian morphisms* and the components of ψ_b are called *opcartesian morphisms*.

Street [12, 13] has developed an abstract theory of fibrations, so that the notion can be applied, not just in **Cat**, but more generally to any bicategory with suitable completeness properties. Here we summarize the basics of Street's theory as it applies to the 2-category **Cat**. Let $\mathrm{Spn}(A, B)$ denotes the 2-category whose objects are spans from A to B in **Cat**, whose 1-cells are arrows of spans, and whose 2-cells are natural transformations between arrows of spans. Then composition on the right with the "comma object" ΦA (the comma category A/A, viewed as a span by equipping it with the evident projections $d_0, d_1 : A/A \to A$), and composition on the left with the comma object ΦB, yield endo-2-functors:

$$\text{-} \circ \Phi A : \mathrm{Spn}(A, B) \to \mathrm{Spn}(A, B) \qquad \Phi B \circ \text{-} : \mathrm{Spn}(A, B) \to \mathrm{Spn}(A, B).$$

Street shows that these 2-functors are the underlying functors of a certain kind of 2-monad, called a "*KZ* doctrine." A span from A to B is called a "split 0-fibration" if it has a structure of $(\text{-} \circ \Phi A)$-algebra, and a "split 1-fibration" if it has a structure of $(\Phi B \circ \text{-})$-algebra. There is also an endo-2-functor

$$\Phi B \circ \text{-} \circ \Phi A : \mathrm{Spn}(A, B) \to \mathrm{Spn}(A, B),$$

which is the composite of $\text{-} \circ \Phi A$ and $\Phi B \circ \text{-}$. The 2-functor M also underlies a 2-monad (though not a KZ-doctrine). A span from A to B is called a "split bifibration" if it has a structure of $(\Phi B \circ \text{-} \circ \Phi A)$-algebra, or equivalently, if it has a $(\text{-} \circ \Phi A)$-algebra structure r and a $(\Phi B \circ \text{-})$-algebra structure l, such that $r(l \circ \Phi A) = l(\Phi B \circ r)$.

The various data mentioned in the above result have an appealing intuitive interpretation in our setting: The functors $\text{-} \circ \Phi B_m$ and $\Phi B_n \circ \text{-}$ can be thought of as operations that place buffers on the input and output, respectively, of the networks of type (m, n). These functors underlie monads, whose units can be thought of as maps that take a network state to the corresponding state of a buffered network, with an empty buffer. The multiplications of the monads can be thought of as composing two tandem buffers into a single buffer. A span S from B_m to B_n is "input buffered," "output buffered," or "buffered," if it has a structure of algebra for the input buffering, output buffering, or composite monad, respectively. If S is output buffered, then the cartesian morphisms of S can be thought of as "pure output" computations, in which data is transmitted from an output buffer onto output ports. Similarly, the opcartesian morphisms of an input buffered S are "pure input" computations. The "internal computations" of S are those computations whose images under both legs d_0 and d_1 of the span are empty. The universal mapping property satisfied by a fibration implies that "every computation has a canonical pure-input/internal/pure-output factorization."

We summarize the connections, outlined above, between dataflow algebra and fibrations in the form of the following result:

Theorem 3 *Let S be a dataflow scheme. Then, for all $m, n \in \mathrm{Ord}$, the full subspan $(\mathcal{B}S)^n_m$ of $(\mathcal{N}S)^n_m$ consisting of all states of the form $[(\gamma \cdot x) \cdot \beta]$ (equivalently, $[\gamma \cdot (x \cdot \beta)]$) is isomorphic to the span $\Phi B_n \circ S^n_m \circ \Phi B_m$. Moreover, suppose $\varepsilon : \mathcal{N}S \to S$ is an $\mathcal{N}$-algebra structure on S. Then, for all $m, n \in \mathrm{Ord}$, the restriction of ε^n_m to $(\mathcal{B}S)^n_m$ is isomorphic to a $(\Phi B_n \circ - \circ \Phi B_m)$-algebra structure on S^n_m. Thus, if S is a dataflow algebra, then each span S^n_m is a split bifibration in* **Cat**.

6 Conclusion

We have defined a "free dataflow algebra" construction on certain indexed collections of spans in **Cat**, and we have shown that this construction underlies a monad. The algebras of this monad are equipped with a monoidal structure, corresponding to a parallel composition operation on dataflow networks, and are acted on in a certain sense by the dataflow algebra of simple Kahn functions, corresponding to the formation of feedback loops. We have observed that each of the spans in a dataflow algebra is a split bifibration. We have also provided a syntax for dataflow algebra, in the form of "dataflow calculus."

There appears to be much more that can be said about dataflow algebra. One direction for future work is to find a nice system of equational axioms for dataflow calculus. For network expressions, the problem is essentially solved, since the monad and distributive laws give a reduction to normal form. Equivalence of network expressions can also be axiomatized in terms of a somewhat different set of operators (see [11]). For computation expressions, the problem amounts to a coherence theorem for the natural transformations $\eta^{\mathcal{L}}$, $\eta^{\mathcal{P}}$, $\mu^{\mathcal{L}}$, $\mu^{\mathcal{P}}$, and δ. One might also attempt to incorporate permutation of network components and deletion of unobservable components into this coherence result, to axiomatize the coarser equivalence of [11]. Other interesting directions for future work are: exploring further the relationship between the operations of dataflow algebra and constructions on fibrations, and examining various examples of dataflow algebra with the goal of understanding whether a vestige of the natural ordering on K^n_m is somehow reflected in more general dataflow algebras.

References

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

[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)*, volume 224 of *Lecture Notes in Mathematics*, 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] R. M. Keller. Denotational models for parallel programs with indeterminate operators. In E. J. Neuhold, editor, *Formal Description of Programming Concepts*, pages 337–366. North-Holland, 1978.

[7] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer Verlag, 1980.

[8] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

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

[10] E. W. Stark. Dataflow networks are fibrations. In *Category Theory and Computer Science*, volume 530 of *Lecture Notes in Computer Science*, pages 261–281. Springer-Verlag, Paris, France, 1991.

[11] E. W. Stark. A calculus of dataflow networks. In *Logic in Computer Science*, pages 125–136. IEEE Computer Society Press, 1992.

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

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