Network algebra for synchronous dataflow

Bergstra, J.A.; Middelburg, C.A.; tefánescu, G.

Publication date
2013
Document Version
Submitted manuscript

Network Algebra for Synchronous Dataflow

J.A. Bergstra[1], C.A. Middelburg[1], and Gh. Ştefănescu[2]

[1] Informatics Institute, Faculty of Science,
University of Amsterdam, Science Park 904, 1098 XH Amsterdam, the Netherlands
J.A.Bergstra@uva.nl,C.A.Middelburg@uva.nl
[2] Department of Computer Science, Faculty of Mathematics and Computer Science,
University of Bucharest, Strada Academiei 14, Bucharest, Romania
gheorghe.stefanescu@fmi.unibuc.ro

Abstract. We develop an algebraic theory of synchronous dataflow networks. First, a basic algebraic theory of networks, called BNA (Basic Network Algebra), is introduced. This theory captures the basic algebraic properties of networks. For synchronous dataflow networks, it is subsequently extended with additional constants for the branching connections that occur between the cells of synchronous dataflow networks and axioms for these additional constants. We also give two models of the resulting theory, the one based on stream transformers and the other based on processes as considered in process algebra.

Keywords: network algebra, dataflow network, synchronous dataflow, stream transformer, process algebra

1998 ACM Computing Classification: F.1.1, F.1.2

1 Introduction

In this paper we pursue an axiomatic approach to the theory of dataflow networks. Network algebra is presented as a general algebraic setting for the description and analysis of dataflow networks. A network can be any labelled directed hypergraph that represents some kind of flow between the components of a system. For example, flowcharts are networks concerning flow of control and dataflow networks are networks concerning flow of data. Assuming that the components have a fixed number of input and output ports, such networks can be built from their components and (possibly branching) connections using parallel composition ($\hbar$), sequential composition ($\circ$) and feedback ($\uparrow$). The connections needed are at least the identity (I) and transposition (X) connections, but branching connections may also be needed for specific classes of networks – e.g. the binary ramification ($\wedge$) and identification ($\vee$) connections and their nullary counterparts ($\bot$ and $\top$) for flowcharts.

An equational theory concerning networks that can be built using the abovementioned operations with only the identity and transposition constants for connections, called BNA (Basic Network Algebra), is presented. The axioms of BNA are sound and complete for such networks modulo graph isomorphism. BNA is the core of network algebra; for the specific classes of networks covered, there are

additional constants and axioms. Flowcharts constitute one such class. BNA is essentially a part of the algebra of flownomials of Căzănescu and Ştefănescu [16] which was developed for the description and analysis of flowcharts.

In addition to BNA, an extension of BNA for synchronous dataflow networks is presented. Process algebra models of BNA and this extension of BNA are given. These models provide for a very straightforward connection between network algebra and process algebra. Unlike process algebra, network algebra is used for describing systems as a network of interconnected components. A clear connection between process algebra and network algebra appears to be useful.

For the process algebra models, ACP (Algebra of Communicating Processes) of Bergstra and Klop [6] is used, with the silent step and abstraction, as well as the following additional features: renaming, conditionals, iteration, prefixing and communication free merge. Besides, a discrete-time extension of ACP is used to model synchronous dataflow networks.

There are strong connections between the work presented in this paper and other work. SCAs (Synchronous Concurrent Algorithms), introduced by Thompson and Tucker in [28], can be described in the extension of BNA for synchronous dataflow networks. In [5], Barendregt et al. present a model of computable processes which is essentially a model of BNA; but a slightly different choice of primitive operations and constants is used.

The paper starts with an outline of network algebra (Section 2) and some process algebra preliminaries (Section 3). Next the signature, the axioms and two models of BNA, including a process algebra model, are presented (Section 4). Thereafter the signature, the axioms and two models of the extension of BNA for synchronous dataflow networks, including a process algebra model, are presented (Section 5). Finally, some closing remarks are made (Section 6).

The current paper complements [8]. The latter paper is a revision of [7] in which the part on synchronous dataflow networks has been left out due to space limitations imposed by the journal. The current paper is a revision of [7] in which the part on asynchronous dataflow networks has been left out instead.

2 Overview of network algebra

This section gives an idea of what network algebra is. The meaning of its operations and constants is explained informally making use of a graphical representation of networks. Besides, dataflow networks are presented as a specific class of networks and the further subdivision into synchronous and asynchronous dataflow networks is explained in broad outline. The formal details will be treated in subsequent sections.

2.1 General

First the meaning of the operations and constants of BNA mentioned in Section 1 ($+\!\!+$, $\circ$, $\uparrow$, I and X) is explained and then the meaning of the additional constants for branching connections mentioned in Section 1 ($\wedge$, $\bot$, $\vee$ and $\top$) is explained.

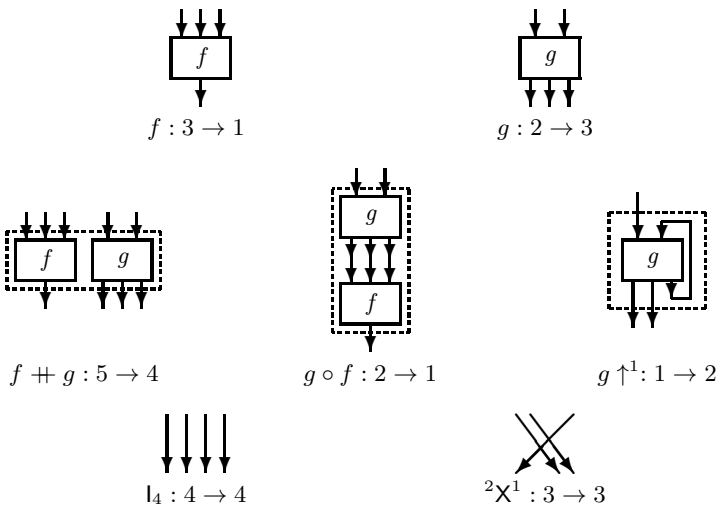

Fig. 1. Operations and constants of BNA

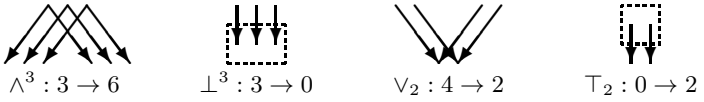

Fig. 2. Additional constants for branching connections

It is convenient to use, in addition to the operations and constants of BNA, the extensions $\uparrow^m$, I_m and ${}^m\mathsf{X}^n$ of the feedback operation and the identity and transposition constants. These extensions are defined by the equations that occur as axioms R5–R6, B6 and B8–B9, respectively, of BNA (see Section 4.1, Table 1). They are called the block extensions of the feedback operation and these constants. The block extensions of additional constants for branching connections can be defined in the same vein.

In Figure 1, the meaning of the operations and constants of BNA (including the block extensions) is illustrated by means of a graphical representation of networks. We write $f : k \to l$ to indicate that network f has k input ports and l output ports; $k \to l$ is called the sort of f. The input ports are numbered $1, \ldots, k$ and the output ports $1, \ldots, l$. In the graphical representation, they are considered to be numbered from left to right. The networks are drawn with the flow moving from top to bottom. Note that the symbols for the feedback operation and the constants fit with this graphical representation. In Figure 2, the meaning of (block extensions of) the additional constants for branching connections mentioned in Section 1 is illustrated by means of a graphical representation. The symbols for these additional constants fit with the graphical representation as well.

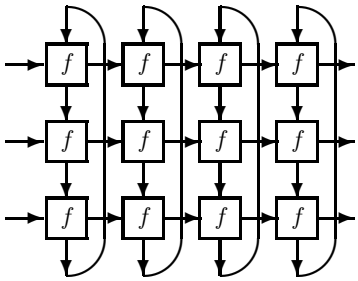

Fig. 3. A regular network

The operations and constants illustrated above allow to represent all networks (cf. [26]). For example,

$$\begin{aligned} r_{k,l} = ((&\circ_{i=1}^{k-1}(\mathsf{I}_{k-i} + \!\!+ {}^{i}f + \!\!+ \mathsf{I}_{l-i}) \circ \\ &\circ_{i=0}^{l-k}(\mathsf{I}_i + \!\!+ {}^{k}f + \!\!+ \mathsf{I}_{l-k-i}) \circ \\ &\circ_{i=k-1}^{1}(\mathsf{I}_{l-i} + \!\!+ {}^{i}f + \!\!+ \mathsf{I}_{k-i}) \circ {}^{l}\mathsf{X}^{k}) \uparrow^{l} , \end{aligned}$$

where $k < l$ and $f : 2 \to 2$, represent a regular network (some abbreviations are used here: iterated sequential composition $\circ_{i=m}^{n} f_i = f_m \circ \ldots \circ f_n$ and parallel composition to the nth $+\!\!+^n f = f + \!\!+ \ldots + \!\!+ f$ (n times)). The instance $r_{3,4}$ is illustrated in Figure 3.

The graphical illustration of the meaning of the operations and constants of BNA in Figure 1 gives intuitive grounds for the soundness of the axioms of BNA (see Section 4.1, Table 1) for the intended network model. Similarly, the illustration of the meaning of the additional constants for branching connections in Figure 2 makes most additional axioms for these constants (see Section 4.1, Table 2) plausible.

2.2 Dataflow networks

In the case of dataflow networks, the components are also called cells. The identity connections are called wires and the transposition connections are viewed as crossing wires. The cells are interpreted as processes that consume data at their input ports, compute new data, deliver the new data at their output ports, and then start over again. The wires are interpreted as queues of some kind. The classical kinds considered are firstly queues that deliver data with a neglectible delay and never contain more than one datum, and secondly unbounded, delaying queues. In this paper, they are called *minimal stream delayers* and *stream delayers*, respectively. A stream is a sequence of data consumed or produced by a component of a dataflow network. A flow (of data) is a transformation of a tuple of streams into a tuple of streams. A wire behaves as an identity flow. If the wire is a stream delayer, data pass through it with a time delay. If the wire is a minimal stream delayer, data enter and leave it with a neglectible delay –

i.e. within the same time slice in case time is divided into time slices with the length of the time unit used.

In synchronous dataflow networks, the wires are minimal stream delayers. Basic to synchronous dataflow is that there is a global clock. On ticks of the clock, cells can start up the consumption of exactly one datum from each of their input ports and the production of exactly one datum at each of their output ports. A cell that started up with that completes the production of data before the next tick, and it completes the consumption of data as soon as a new datum has been delivered at all input ports. On the first tick following the completion of both, the cell concerned starts up again. In order to start the synchronous dataflow network, every cell has, for each of its output ports, an initial datum available to deliver on the initial tick. The underlying idea of synchronous dataflow is that computation takes a good deal of time, whereas storage and transport of data takes a neglectible deal of time. Phrased differently, data always pass through a wire between two consecutive ticks of the global clock. So minimal stream delayers fit in exactly with this kind of dataflow networks. The semantics of synchronous dataflow networks turns out to be rather simple and unproblematic.

In asynchronous dataflow networks, the wires are stream delayers. The underlying idea of asynchronous dataflow is that computation as well as storage and transport of data takes a good deal of time, which is sometimes more realistic for large systems. In such cases, it is favourable to have computation driven by the arrival of the data needed – instead of by clock ticks. Therefore, there is no global clock in an asynchronous dataflow network. Cells may independently consume data from their input ports, compute new data, and deliver the new data at their output ports. Because it means that there may be data produced by cells but not yet consumed by other cells, this needs wires that are able to buffer an arbitrary amount of data. So stream delayers fit in exactly with this kind of dataflow networks. However, the semantics of asynchronous dataflow networks turns out to be rather problematic. The main semantic problem is a time anomaly, known as the Brock-Ackermann anomaly. With feedback, timing differences in producing data may become important and the time anomaly actually shows that delaying queues do not perfectly fit in with that. Besides, the unbounded queues needed to keep an arbitrary amount of data are unrealistic. Note that a synchronous dataflow network can be viewed as a extreme case of an asynchronous one, where the queues never contain more than one datum.

Dataflow networks also need branching connections. Their branching structure is more complex than the branching structure of flowcharts. In case of flowcharts, there is a flow of control which is always at one point in the flowchart concerned. In consequence, the interpretation of the branching connections is rather obvious. However, in case of dataflow networks, there is a flow of data which is everywhere in the network. Hence, the interpretation of the branching connections is not immediately clear. In this paper, two kinds of interpretation are considered. For the binary branching connections, they are the *copy/equality test* interpretation and the *split/merge* interpretation. The first

kind of interpretation fits in with the idea of permanent flows of data which naturally go in all directions at branchings. Synchronous dataflow reflects this idea most closely. The second kind of interpretation fits in with the idea of intermittent flows of data which go in one direction at branchings. Asynchronous dataflow reflects this idea better. In order to distinguish between the branching constants with these different interpretations, different symbols for $\wedge^m$ and $\vee_m$ are used: $\wedge^m$ and $\vee_m$ for the copy/equality test interpretation, $\wedge^m$ and $\vee_m$ for the split/merge interpretation. Likewise, different symbols for the nullary counterparts $\bot^m$ and $\top_m$ are used: $\bot^m$ and $\top_m$ versus $\bot^m$ and $\top_m$. $\bot^m$ and $\bot^m$ are called *sink* and *dummy sink*, respectively; and $\top_m$ and $\top_m$ are called *source* and *dummy source*, respectively.

In the synchronous case, with minimal stream delayers as identity connections and the copy/equality test interpretation of the branching connections, it turns out that two axioms for $\wedge^m$ and $\vee_m$ are not valid. Fortunately the others together with two new axioms give a complete set of axioms. The asynchronous case is somewhat problematic owing to the time anomaly that occurs in the model outlined above. The asynchronous case is treated separately in [8].

Dataflow networks have been extensively studied, see e.g. [5,10,11,12,20,21,23,24].

3 Process algebra preliminaries

This section gives a brief summary of the ingredients of process algebra which make up the basis for the process algebra models presented in Sections 4 and 5. We will suppose that the reader is familiar with them. Appropriate references to the literature are included.

We will make use of ACP$^\tau$, which is an extension of ACP [6] with abstraction based on branching bisimulation [19]. In ACP$^\tau$, processes can be composed from actions, the inactive process (δ) and the silent step (τ) by sequential composition ($\cdot$), alternative composition ($+$), parallel composition ($\|$), encapsulation (∂_H), and abstraction (τ_I). For a systematic introduction to ACP$^\tau$, the reader is referred to [4]. We will use the following abbreviation. Let $(P_i)_{i\in I}$ be a indexed set of process expressions where $I = \{i_1, \ldots, i_n\}$. Then, we write $\sum_{i\in I} P_i$ for $P_{i_1} + \ldots + P_{i_n}$ if $n > 0$ and δ if $n = 0$.

We will also use some of the features added to ACP in [1]:

Renaming We will use the renaming operator ρ_f. Here f is a function that renames actions into actions, δ or τ. The expression $\rho_f(P)$ denotes the process P with every occurrence of an action a replaced by $f(a)$. So the most crucial equation from the axioms for the renaming operator is $\rho_f(a) = f(a)$.

Conditionals We will use the two-armed conditional operator $\lhd\ \rhd$. The expression $P \lhd b \rhd Q$, is to be read as $\mathsf{if}\ b\ \mathsf{then}\ P\ \mathsf{else}\ Q$. The most important equations derivable from the axioms for the two-armed conditional operator are $X \lhd \mathsf{t} \rhd Y = X$ and $X \lhd \mathsf{f} \rhd Y = Y$.

Early input prefixing We will use the early input action prefixing operators $(er_i(v)\ ;\ )$ and their generalization to a process prefixing operator $(;)$. The

most important equation derivable from the axioms for the early input action prefixing operators is $er_i(v) \,;\, X = \sum_{d \in D} r_i(d) \cdot X[d/v]$ (it is assumed that a fixed but arbitrary finite set D of data has been given).

Process prefixing We will use the process prefixing operator mainly to express parallel input: $(er_1(v_1) \parallel \ldots \parallel er_n(v_n)) \,;\, P$. We have:

$$\begin{aligned}(er_1(v_1) \parallel er_2(v_2)) \,;\, P &= \sum_{d_1 \in D} r_1(d_1) \cdot (er_2(v_2) \,;\, P[d_1/v_1]) \\ &\quad + \sum_{d_2 \in D} r_2(d_2) \cdot (er_1(v_1) \,;\, P[d_2/v_2]) \;,\end{aligned}$$

$$\begin{aligned}(er_1(v_1) \parallel er_2(v_2) \parallel er_3(v_3)) \,;\, P &= \sum_{d_1 \in D} r_1(d_1) \cdot ((er_2(v_2) \parallel er_3(v_3)) \,;\, P[d_1/v_1]) \\ &\quad + \sum_{d_2 \in D} r_2(d_2) \cdot ((er_1(v_1) \parallel er_3(v_3)) \,;\, P[d_2/v_2]) \\ &\quad + \sum_{d_3 \in D} r_3(d_3) \cdot ((er_1(v_1) \parallel er_2(v_2)) \,;\, P[d_3/v_3]) \;,\end{aligned}$$

etc.

Communication free merge We will use the communication free merge operator ($\|\!|$). This operator is in fact one of the synchronisation merge operators $\|_H$ of CSP, which are also added to ACP in [1], viz. $\|_\emptyset$. Communication free merge can also be expressed in terms of parallel composition, encapsulation and renaming. The most crucial equations from the axioms for the communication free merge operator are $P \,\|\!|\, Q = P \between Q + Q \between P$ and $a \cdot P \between Q = a \cdot (P \,\|\!|\, Q)$.

Moreover, we will make use of $\mathrm{ACP}_\tau^{\mathrm{drt}}$, which is an extension of $\mathrm{ACP}^{\mathrm{drt}}$ with abstraction based on branching bisimulation. $\mathrm{ACP}^{\mathrm{drt}}$ in turn is an extension of ACP with discrete relative timing. In $\mathrm{ACP}_\tau^{\mathrm{drt}}$, time is considered to be divided into slices indexed by natural numbers. These time slices represent time intervals of a length which corresponds to the time unit used. In $\mathrm{ACP}_\tau^{\mathrm{drt}}$, we have the additional constants $\underline{\underline{a}}$ (for each action a), $\underline{\underline{\tau}}$ and $\underline{\underline{\delta}}$, and the delay operator σ_{rel}. The process a is a performed in any time slice and $\underline{\underline{a}}$ is a performed in the current time slice. Similarly, $\underline{\underline{\tau}}$ is a silent step performed in the current time slice and $\underline{\underline{\delta}}$ is inaction in the current time slice. The expression $\sigma_{\mathsf{rel}}(P)$ denotes the process P delayed one time slice. The process a is recursively defined by the equation $X = \underline{\underline{a}} + \sigma_{\mathsf{rel}}(X)$. In a parallel composition $P_1 \parallel \ldots \parallel P_n$ the transition to the next time slice is a simultaneous transition of $P_1, \ldots, P_n$. For example, $\underline{\underline{\delta}} \parallel \sigma_{\mathsf{rel}}(\underline{\underline{b}})$ will never perform $\underline{\underline{b}}$ because $\underline{\underline{\delta}}$ can neither be delayed nor performed, so $\underline{\underline{\delta}} \parallel \sigma_{\mathsf{rel}}(\underline{\underline{b}}) = \underline{\underline{\delta}}$. However, $\underline{\underline{a}} \parallel \sigma_{\mathsf{rel}}(\underline{\underline{b}}) = \underline{\underline{a}} \cdot \sigma_{\mathsf{rel}}(\underline{\underline{b}})$. For a systematic introduction to $\mathrm{ACP}_\tau^{\mathrm{drt}}$, the reader is referred to [3].

We will also use the above-mentioned features in the setting of $\mathrm{ACP}_\tau^{\mathrm{drt}}$. The integration of renaming, conditionals, and communication free merge in the discrete time setting is obvious. The integration of early input prefixing and process prefixing may seem less clear at first sight, but the relevant equations are simply $\underline{\underline{er}}_i(v) \,;\, X = \sum_{d \in D} \underline{\underline{r}}_i(d) \cdot X[d/v]$ and $\sigma_{\mathsf{rel}}(X) \,;\, Y = \sigma_{\mathsf{rel}}(X \,;\, Y)$.

4 Basic network algebra

BNA is essentially the part of the algebra of flownomials [16] that is common to various classes of networks. In particular, it is common to flowcharts and dataflow networks. The additional constants, needed for branching connections, differ however from one class to another. In this section, BNA is presented. First of all, the signature and axioms of BNA are given. The extension of BNA to the algebra of flownomials is also addressed here. In addition, two models of BNA are described: a data transformer model and a process algebra model. In subsequent sections, an extension of BNA for synchronous dataflow networks is provided.

4.1 Signature and axioms of BNA

Signature In network algebra, networks are built from other networks – starting with atomic components and a variety of connections. Every network f has a sort $k \to l$, where $k, l \in \mathbb{N}$, associated with it. To indicate this, we use the notation $f : k \to l$. The intended meaning of the sort $k \to l$ is the set of networks with k input ports and l output ports. So $f : k \to l$ expresses that f has k input ports and l output ports.

The sorts of the networks to which an operation of network algebra is applied determine the sort of the resulting network. In addition, there are restrictions on the sorts of the networks to which an operation can be applied. For example, sequential composition can not be applied to two networks of arbitrary sorts because the number of output ports of one should agree with the number of input ports of the other.

The signature of BNA is as follows:

Name	Symbol	Arity
Operations:		
parallel composition	$\parallel$	$(k \to l) \times (m \to n) \to (k+m \to l+n)$
sequential composition	$\circ$	$(k \to l) \times (l \to m) \to (k \to m)$
feedback	$\uparrow$	$(m+1 \to n+1) \to (m \to n)$
Constants:		
identity	I	$1 \to 1$
transposition	X	$2 \to 2$

Here k, l, m, n range over $\mathbb{N}$. This means, for example, that there is an instance of the sequential composition operator for each $k, l, m \in \mathbb{N}$.

As mentioned in Section 2, we will also use the block extensions of feedback, identity and transposition. The arity of these auxiliary operations and constants is as follows:

Table 1. Axioms of BNA

B1	$f \parallel (g \parallel h) = (f \parallel g) \parallel h$	
B2	$\mathsf{I}_0 \parallel f = f = f \parallel \mathsf{I}_0$	
B3	$f \circ (g \circ h) = (f \circ g) \circ h$	
B4	$\mathsf{I}_k \circ f = f = f \circ \mathsf{I}_l$	
B5	$(f \parallel f') \circ (g \parallel g') = (f \circ g) \parallel (f' \circ g')$	
B6	$\mathsf{I}_k \parallel \mathsf{I}_l = \mathsf{I}_{k+l}$	
B7	${}^k\mathsf{X}^l \circ {}^l\mathsf{X}^k = \mathsf{I}_{k+l}$	
B8	${}^k\mathsf{X}^0 = \mathsf{I}_k$	
B9	${}^k\mathsf{X}^{l+m} = ({}^k\mathsf{X}^l \parallel \mathsf{I}_m) \circ (\mathsf{I}_l \parallel {}^k\mathsf{X}^m)$	
B10	$(f \parallel g) \circ {}^m\mathsf{X}^n = {}^k\mathsf{X}^l \circ (g \parallel f)$	for $f : k \to m,\ g : l \to n$
R1	$g \circ (f \uparrow^m) = ((g \parallel \mathsf{I}_m) \circ f) \uparrow^m$	
R2	$(f \uparrow^m) \circ g = (f \circ (g \parallel \mathsf{I}_m)) \uparrow^m$	
R3	$f \parallel (g \uparrow^m) = (f \parallel g) \uparrow^m$	
R4	$(f \circ (\mathsf{I}_l \parallel g)) \uparrow^m = ((\mathsf{I}_k \parallel g) \circ f) \uparrow^n$	for $f : k + m \to l + n,\ g : n \to m$
R5	$f \uparrow^0 = f$	
R6	$(f \uparrow^l) \uparrow^k = f \uparrow^{k+l}$	
F1	$\mathsf{I}_k \uparrow^k = \mathsf{I}_0$	
F2	${}^k\mathsf{X}^k \uparrow^k = \mathsf{I}_k$	

Symbol	Arity
$\uparrow^l$	$(m + l \to n + l) \to (m \to n)$
I_m	$m \to m$
${}^m\mathsf{X}^n$	$m + n \to n + m$

Axioms The axioms of BNA are given in Table 1. The axioms B1–B6 for $\parallel$, $\circ$ and I_m define a strict monoidal category; and together with the additional axioms B7–B10 for ${}^m\mathsf{X}^n$, they define a *symmetric* strict monoidal category (ssmc for short). The remaining axioms R1–R6 and F1–F2 characterize $\uparrow^l$. The axioms R5–R6, B6 and B8–B9 can be regarded as the defining equations of the block extensions of $\uparrow$, I and X, respectively.

The axioms of BNA are sound and complete for networks modulo graph isomorphism (cf. [26]). Using the graphical representation of Section 2.1, it is easy to see that the axioms in Table 1 are sound. By means of the axioms of BNA, each expression can be brought into a normal form

$$((\mathsf{I}_m \parallel x_1 \parallel \ldots \parallel x_k) \circ f) \uparrow^{m_1+\ldots+m_k} ,$$

where the $x_i : m_i \to n_i$ $(i \in [k])$[3] are the atomic components of the network and $f : m + n_1 + \ldots + n_k \to n + m_1 + \ldots + m_k$ is a bijective connection. A network

[3] We write $[n]$, where $n \in \mathbb{N}$, for $\{1, \ldots, n\}$.

is uniquely represented by a normal form expression up to a permutation of $x_1, \ldots, x_k$. The completeness of the axioms of BNA now follows from the fact that these permutations in a normal form expression are deducible from the axioms of BNA as well.

As a first step towards the stream transformer and process algebra models for synchronous dataflow networks described in Section 5, a data transformer model and a process algebra model of BNA are provided immediately after the connection with the algebra of flownomials has been addressed.

Extension to the algebra of flownomials The algebra of flownomials is essentially[4] a conservative extension of BNA. Recall that the algebra of flownomials was not developed for dataflow networks, but for flowcharts. The signature of the algebra of flownomials is obtained by extending the signature of BNA as follows with additional constants for branching connections:

Name	Symbol	Arity	Instances
Additional constants:			
ramification	$\wedge_k$	$1 \to k$	$\begin{cases} \wedge := \wedge_2 \\ \bot := \wedge_0 \end{cases}$
identification	$\vee^k$	$k \to 1$	$\begin{cases} \vee := \vee^2 \\ \top := \vee^0 \end{cases}$

We will restrict our attention to the instances for $k = 0$ and $k = 2$, i.e. $\wedge$, $\bot$, $\vee$ and $\top$. The other instances can be defined in terms of them:

$$\wedge_{k+1} = \wedge \circ (\wedge_k + \mathsf{I}) \,,$$
$$\vee^{k+1} = (\vee^k + \mathsf{I}) \circ \vee \,.$$

It follows from these definitions, together with the axioms A3 and A7 of the algebra of flownomials (see Table 2), that $\wedge_1 = \vee^1 = \mathsf{I}$.

We will use the block extensions of $\wedge$, $\bot$, $\vee$ and $\top$. The arity of these auxiliary constants is as follows:

Symbol	Arity
$\wedge^m$	$m \to 2m$
$\bot^m$	$m \to 0$
$\vee_m$	$2m \to m$
$\top_m$	$0 \to m$

The axioms for the additional constants of the algebra of flownomials are given in Table 2. These axioms where chosen in order to describe the branching structure

[4] For naming ports, an arbitrary monoid is used in the algebra of flownomials whereas the monoid of natural numbers is used in BNA.

Table 2. Additional axioms for flowcharts

A1	$(\vee_m \mathbin{+\!\!+} \mathsf{I}_m) \circ \vee_m = (\mathsf{I}_m \mathbin{+\!\!+} \vee_m) \circ \vee_m$
A2	${}^m\mathsf{X}^m \circ \vee_m = \vee_m$
A3	$(\top_m \mathbin{+\!\!+} \mathsf{I}_m) \circ \vee_m = \mathsf{I}_m$
A4	$\vee_m \circ \perp^m = \perp^m \mathbin{+\!\!+} \perp^m$
A5	$\wedge^m \circ (\wedge^m \mathbin{+\!\!+} \mathsf{I}_m) = \wedge^m \circ (\mathsf{I}_m \mathbin{+\!\!+} \wedge^m)$
A6	$\wedge^m \circ {}^m\mathsf{X}^m = \wedge^m$
A7	$\wedge^m \circ (\perp^m \mathbin{+\!\!+} \mathsf{I}_m) = \mathsf{I}_m$
A8	$\top_m \circ \wedge^m = \top_m \mathbin{+\!\!+} \top_m$
A9	$\top_m \circ \perp^m = \mathsf{I}_0$
A10	$\vee_m \circ \wedge^m = (\wedge^m \mathbin{+\!\!+} \wedge^m) \circ (\mathsf{I}_m \mathbin{+\!\!+} {}^m\mathsf{X}^m \mathbin{+\!\!+} \mathsf{I}_m) \circ (\vee_m \mathbin{+\!\!+} \vee_m)$
A11	$\wedge^m \circ \vee_m = \mathsf{I}_m$
A12	$\top_0 = \mathsf{I}_0$
A13	$\top_{m+n} = \top_m \mathbin{+\!\!+} \top_n$
A14	$\vee_0 = \mathsf{I}_0$
A15	$\vee_{m+n} = (\mathsf{I}_m \mathbin{+\!\!+} {}^n\mathsf{X}^m \mathbin{+\!\!+} \mathsf{I}_n) \circ (\vee_m \mathbin{+\!\!+} \vee_n)$
A16	$\perp^0 = \mathsf{I}_0$
A17	$\perp^{m+n} = \perp^m \mathbin{+\!\!+} \perp^n$
A18	$\wedge^0 = \mathsf{I}_0$
A19	$\wedge^{m+n} = (\wedge^m \mathbin{+\!\!+} \wedge^n) \circ (\mathsf{I}_m \mathbin{+\!\!+} {}^m\mathsf{X}^n \mathbin{+\!\!+} \mathsf{I}_n)$
F3	$\vee_m \uparrow^m = \perp^m$
F4	$\wedge^m \uparrow^m = \top_m$
F5	$((\mathsf{I}_m \mathbin{+\!\!+} \wedge^m) \circ ({}^m\mathsf{X}^m \mathbin{+\!\!+} \mathsf{I}_m) \circ (\mathsf{I}_m \mathbin{+\!\!+} \vee_m)) \uparrow^m = \mathsf{I}_m$

of flowcharts. The axioms A12–A19 can be regarded as the defining equations of the block extentions of $\wedge$, $\perp$, $\vee$ and $\top$.

The standard model for the interpretation of flowcharts is the model $\mathbb{R}\mathsf{el}(D)$ of relations over a set D (cf. [16,25]). All axioms of the algebra of flownomials (Tables 1 and 2) hold in this model. The algebraic structure defined by the axioms of BNA (Table 1) was introduced in [26] under the name of *biflow*. In [27] it is called *aα-ssmc with feedback*. The algebraic structure defined by the axioms of the algebra of flownomials (Tables 1 and 2) is called *dδ-ssmc with feedback* in [27].

4.2 Data transformer model of BNA

In this subsection, a data transformer model of BNA is described. A parallel data transformer $f : m \to n$ acts on an m-tuple of input data and produces an n-tuple of output data. Parallel composition, sequential composition and feedback operators as well as identity and transposition constants are defined on parallel data transformers. All axioms of BNA (Table 1) hold in the resulting model.

Definition 1. (data transformer model of BNA)
A *parallel data transforming relation* $f \in \mathsf{Rel}(S)(m,n)$ is a relation

$$f \subseteq S^m \times S^n \ ,$$

where S is a set of data. $\mathsf{Rel}(S)$ denotes the indexed family of data transforming relations $(\mathsf{Rel}(S)(m,n))_{\mathbb{N} \times \mathbb{N}}$.

The operations and constants of BNA are defined on $\mathsf{Rel}(S)$ as follows:

Notation		
$f \mathbin{+\!\!+} g$	$\in \mathsf{Rel}(S)(m+p, n+q)$	for $f \in \mathsf{Rel}(S)(m,n)$, $g \in \mathsf{Rel}(S)(p,q)$
$f \circ g$	$\in \mathsf{Rel}(S)(m,p)$	for $f \in \mathsf{Rel}(S)(m,n)$, $g \in \mathsf{Rel}(S)(n,p)$
$f \uparrow^p$	$\in \mathsf{Rel}(S)(m,n)$	for $f \in \mathsf{Rel}(S)(m+p, n+p)$
I_n	$\in \mathsf{Rel}(S)(n,n)$	
${}^m\mathsf{X}^n$	$\in \mathsf{Rel}(S)(m+n, n+m)$	

Definition[5]	
$f \mathbin{+\!\!+} g$	$= \{\langle x^\frown y, z^\frown w\rangle \mid x \in S^m \wedge y \in S^p \wedge z \in S^n \wedge w \in S^q \wedge \langle x,z\rangle \in f \wedge \langle y,w\rangle \in g\}$
$f \circ g$	$= \{\langle x,y\rangle \mid x \in S^m \wedge y \in S^p \wedge \exists z \in S^n \cdot \langle x,z\rangle \in f \wedge \langle z,y\rangle \in g\}$
$f \uparrow^p$	$= \{\langle x,y\rangle \mid x \in S^m \wedge y \in S^n \wedge \exists z \in S^p \cdot \langle x^\frown z, y^\frown z\rangle \in f\}$
I_n	$= \{\langle x,x\rangle \mid x \in S^n\}$
${}^m\mathsf{X}^n$	$= \{\langle x^\frown y, y^\frown x\rangle \mid x \in S^m \wedge y \in S^n\}$

The definitions of the operations and constants of BNA given above are very straightforward. Note that the data transformer model defined here has a global crash property: if a component of a network fails to produce output, the whole network fails to produce output.

Theorem 1. $(\mathsf{Rel}(S), \mathbin{+\!\!+}, \circ, \uparrow, \mathsf{I}, \mathsf{X})$ *is a model of BNA.*

Proof: The proof is a matter of straightforward calculation using only elementary set theory. □

Additional branching constants can be defined such that the resulting expanded model satisfies most axioms of the algebra of flownomials (Tables 1 and 2). One such set of branching constants is closely related to the one that is used in the design of (nondeterministic) SCAs [28]. The corresponding expanded model is principally the stream transformer model for synchronous dataflow networks described in Section 5 where an abstraction is made from the internals of the transformers: arbitrary data is transformed instead of streams of data. However, $\top_m$ must be interpreted as $\Qoppa_m$ in this data transformer model, to keep up relationships with SCAs, whereas it is interpreted as $\Lsh_m$ in the stream transformer model for synchronous dataflow networks.

[5] Let $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_n)$ be tuples. Then we write $x^\frown y$ for the tuple $(x_1, \ldots, x_m, y_1, \ldots, y_n)$. Moreover, we often write $\langle x_1, x_2\rangle$ instead of (x_1, x_2).

4.3 Process algebra model of BNA

Network algebra can be regarded as being built on top of process algebra.

Let D be a fixed, but arbitrary, finite set of data. D is a parameter of the model. The processes use the standard actions $r_i(d)$, $s_i(d)$ and $c_i(d)$ for $d \in D$ only. They stand for read, send and communicate, respectively, datum d at port i. On these actions, communication is defined such that $r_i(d) \mid s_i(d) = c_i(d)$ (for all $i \in \mathbb{N}$ and $d \in D$). In all other cases, it yields δ.

We write $H(i)$, where $i \in \mathbb{N}$, for the set $\{r_i(d) \mid d \in D\} \cup \{s_i(d) \mid d \in D\}$ and $I(i)$ for $\{c_i(d) \mid d \in D\}$. In addition, we write $H(i,j)$ for $H(i) \cup H(j)$, $H(i+[k])$ for $H(i+1) \cup \ldots \cup H(i+k)$ and $H(i+[k], j+[l])$ for $H(i+[k]) \cup H(j+[l])$. The abbreviations $I(i,j)$, $I(i+[k])$ and $I(i+[k], j+[l])$ are used analogously.

$in(i/j)$ denotes the renaming function defined by

$$\begin{array}{lll} in(i/j)(r_i(d)) & = r_j(d) & \text{for } d \in D \ , \\ in(i/j)(a) & = a & \text{for } a \notin \{r_i(d) \mid d \in D\} \ . \end{array}$$

So $in(i/j)$ renames port i into j in read actions. $out(i/j)$ is defined analogously, but renames send actions. We write $in(i+[k]/j+[k])$ for $in(i+1/j+1) \circ \ldots \circ in(i+k/j+k)$ and $in([k]/j+[k])$ for $in(0+[k]/j+[k])$. The abbreviations $out(i+[k]/j+[k])$ and $out([k]/j+[k])$ are used analogously.

Definition 2. (process algebra model of BNA)
A *network* $f \in \mathsf{Proc}(D)(m,n)$ is a triple

$$f = (m, n, P) \ ,$$

where P is a process with actions in $\{r_i(d) \mid i \in [m] \wedge d \in D\} \cup \{s_i(d) \mid i \in [n] \wedge d \in D\}$. $\mathsf{Proc}(D)$ denotes the indexed family of sets $(\mathsf{Proc}(D)(m,n))_{\mathbb{N} \times \mathbb{N}}$.

A *wire* is a network $\mathsf{I} = (1, 1, w_1^1)$, where w_1^1 satisfies for all networks $f = (m, n, P)$ and $u, v > \max(m, n)$:

(P1) $\tau_{I(u,v)}(\partial_{H(v,u)}(w_v^u \parallel w_u^v)) \lll P = P$,

(P2) $\tau_{I(u,v)}(\partial_{H(u,v)}((\rho_{in(i/u)}(P) \lll w_v^i) \parallel w_u^v)) = P$ for all $i \in [m]$,

(P3) $\tau_{I(u,v)}(\partial_{H(u,v)}((\rho_{out(j/v)}(P) \lll w_j^u) \parallel w_u^v)) = P$ for all $j \in [n]$,

where $w_v^u = \rho_{in(1/u)}(\rho_{out(1/v)}(w_1^1))$.

The operations and constants of BNA are defined on $\mathsf{Proc}(D)$ as follows:

Notation		
$f \leftrightarrows g$	$\in \mathsf{Proc}(D)(m+p, n+q)$	for $f \in \mathsf{Proc}(D)(m,n)$, $g \in \mathsf{Proc}(D)(p,q)$
$f \circ g$	$\in \mathsf{Proc}(D)(m,p)$	for $f \in \mathsf{Proc}(D)(m,n)$, $g \in \mathsf{Proc}(D)(n,p)$
$f \uparrow^p$	$\in \mathsf{Proc}(D)(m,n)$	for $f \in \mathsf{Proc}(D)(m+p, n+p)$
I_n	$\in \mathsf{Proc}(D)(n,n)$	
${}^m\mathsf{X}^n$	$\in \mathsf{Proc}(D)(m+n, n+m)$	

Definition

$(m,n,P) \mathbin{+\!\!+} (p,q,Q) = (m+p, n+q, R)$,
where $R = P \mathbin{|||} \rho_{in([p]/m+[p])}(\rho_{out([q]/n+[q])}(Q))$

$(m,n,P) \circ (n,p,Q) \;\; = (m,p,\tau_{I(u+[n],v+[n])}(\partial_{H(u+[n],v+[n])}(R)))$,
where $u = \max(m,p), v = u+n$, and
$R = (\rho_{out([n]/u+[n])}(P) \mathbin{|||} \rho_{in([n]/v+[n])}(Q)) \parallel w_{v+1}^{u+1} \parallel \ldots \parallel w_{v+n}^{u+n}$

$(m+p, n+p, P)\uparrow^p \;\; = (m,n,\tau_{I(u+[p],v+[p])}(\partial_{H(u+[p],v+[p])}(R)))$,
where $u = \max(m,n), v = u+p$, and
$R = \rho_{in(m+[p]/v+[p])}(\rho_{out(n+[p]/u+[p])}(P)) \parallel w_{v+1}^{u+1} \parallel \ldots \parallel w_{v+p}^{u+p}$

$$\mathsf{I}_n \;\; = \begin{cases} (n,n,w_1^1 \mathbin{|||} \ldots \mathbin{|||} w_n^n) & \text{if } n>0 \\ (0,0,\tau_{I(1,2)}(\partial_{H(1,2)}(w_2^1 \parallel w_1^2))) & \text{otherwise} \end{cases}$$

$$^m\mathsf{X}^n \;\; = \begin{cases} (m+n,n+m,w_{n+1}^1 \mathbin{|||} \ldots \mathbin{|||} w_{n+m}^m \mathbin{|||} w_1^{m+1} \mathbin{|||} \ldots \mathbin{|||} w_n^{m+n}) & \text{if } m+n>0 \\ (0,0,\tau_{I(1,2)}(\partial_{H(1,2)}(w_2^1 \parallel w_1^2))) & \text{otherwise} \end{cases}$$

The conditions (P1)–(P3) on wires given above are rather obscure at first sight, but they are equivalent to the axioms B2 and B4 of BNA: (P1) corresponds to $\mathsf{I}_0 \mathbin{+\!\!+} f = f = f \mathbin{+\!\!+} \mathsf{I}_0$, (P2) to $\mathsf{I}_m \circ f = f$, and (P3) to $f = f \circ \mathsf{I}_n$. The definitions of sequential composition and feedback illustrate clearly the differences between the mechanisms for using ports in network algebra and process algebra. In network algebra the ports that become internal after composition are hidden. In process algebra based models these ports are still visible; a special operator must be used to hide them. For typical wires, $\tau_{I(1,2)}(\partial_{H(1,2)}(w_2^1 \parallel w_1^2))$ equals δ, $\tau \cdot \delta$ or $\underline{\underline{\tau}} \cdot \delta$ (the latter only in case $\mathrm{ACP}_\tau^{\mathrm{drt}}$ is used).

In the description of a process algebra model of BNA given above, all constants and operators used are common to ACP^τ and $\mathrm{ACP}_\tau^{\mathrm{drt}}$ or belong to a few of their mutual (conservative) extensions mentioned in Section 3 (viz. renaming and communication free merge). As a result, we can specialize this general model for a specific kind of networks using either ACP^τ or $\mathrm{ACP}_\tau^{\mathrm{drt}}$; with further extensions at need. On the other hand, we can obtain general results on these process algebra models: results that only depend on properties that are common to ACP^τ and $\mathrm{ACP}_\tau^{\mathrm{drt}}$ or properties of the mutual extensions used above.

Theorem 2. $(\mathsf{Proc}(D), \mathbin{+\!\!+}, \circ, \uparrow, \mathsf{I}, \mathsf{X})$ *is a model of BNA.*

Proof: According to [27], there is an algebra equivalent to BNA (the algebra of LR-flow over $\mathbb{Bi}$), but having two renumbering operations, for (bijectively) renumbering input ports and output ports, instead of the transposition constant and the sequential composition operation of BNA. Renumbering is just renaming in the corresponding process algebra model. The crucial axioms concerning the constant I_n in the equational theory of that algebra follow immediately from the conditions (P1)–(P3) on wires in Definition 2. For quite a few axioms from this equational theory, the proof that they are satisfied by the process algebra model is a matter of simple calculation using only elementary properties of renaming,

communication free merge, or parallel composition and renaming. For the remaining axioms, reminiscent of the axioms R1–R4 of BNA, the proof is a matter of straightforward calculation using in addition properties of parallel composition and encapsulation or abstraction. All properties concerned are common to ACP^{τ} and $\mathrm{ACP}^{\mathrm{drt}}_{\tau}$ or properties of the mutual extensions used in Definition 2. □

If we select a specific wire, such as $\mathtt{msd}^1_1$ in Section 5, we have obtained a model of BNA if the conditions (P1)–(P3) are satisfied by the wire concerned.

5 Synchronous dataflow networks

In this section, an extension of BNA for synchronous dataflow networks is presented. First of all, the additional constants and axioms for synchronous dataflow are given. Next, the adaptation of the data transformer model of Section 4.2 to synchronous dataflow networks, resulting in a stream transformer model for synchronous dataflow, is described. Finally, the specialization of the process algebra model of Section 4.3 for synchronous dataflow networks is described.

5.1 Additional constants and axioms

The signature of the extension of BNA for synchronous dataflow networks is obtained by extending the signature of BNA as follows with additional constants for branching connections:

Name	Symbol	Arity
Additional constants:		
copy	copy^m	$m \to 2m$
sink	sink^m	$m \to 0$
equality test	eq_m	$2m \to m$
dummy source	dum_m	$0 \to m$

The symbols copy^m, sink^m and eq_m indicate that the copy/equality test interpretation is intended here. For technical reasons, which are explained at the end of Section 5.2, dum_m is used instead of dum_m.

The axioms for these additional constants are given in Table 3. These axioms agree with those for the additional constants of the algebra of flownomials (Table 2) with two exceptions: A3 and F5 are replaced by A3° and F5°.

In the next two subsections, the models introduced in Section 4 are specialized to describe the semantics of the synchronous dataflow networks.

5.2 Stream transformer model for synchronous dataflow

In this subsection, an adaptation of the data transformer model of BNA (Section 4.2) for synchronous dataflow is given.

Table 3. Additional axioms for synchronous dataflow networks

A1	$(\curlyvee_m + \mathsf{I}_m) \circ \curlyvee_m = (\mathsf{I}_m + \curlyvee_m) \circ \curlyvee_m$
A2	${}^m\mathsf{X}^m \circ \curlyvee_m = \curlyvee_m$
A3°	$(\mathbin{\rotatebox{180}{$\curlywedge$}}_m + \mathsf{I}_m) \circ \curlyvee_m = \mathbin{\rotatebox{180}{$\curlyvee$}}^m \circ \mathbin{\rotatebox{180}{$\curlywedge$}}_m$
A4	$\curlyvee_m \circ \mathbin{\rotatebox{180}{$\curlyvee$}}^m = \mathbin{\rotatebox{180}{$\curlyvee$}}^m + \mathbin{\rotatebox{180}{$\curlyvee$}}^m$
A5	$\curlywedge^m \circ (\curlywedge^m + \mathsf{I}_m) = \curlywedge^m \circ (\mathsf{I}_m + \curlywedge^m)$
A6	$\curlywedge^m \circ {}^m\mathsf{X}^m = \curlywedge^m$
A7	$\curlywedge^m \circ (\mathbin{\rotatebox{180}{$\curlyvee$}}^m + \mathsf{I}_m) = \mathsf{I}_m$
A8	$\mathbin{\rotatebox{180}{$\curlywedge$}}_m \circ \curlywedge^m = \mathbin{\rotatebox{180}{$\curlywedge$}}_m + \mathbin{\rotatebox{180}{$\curlywedge$}}_m$
A9	$\mathbin{\rotatebox{180}{$\curlywedge$}}_m \circ \mathbin{\rotatebox{180}{$\curlyvee$}}^m = \mathsf{I}_0$
A10	$\curlyvee_m \circ \curlywedge^m = (\curlywedge^m + \curlywedge^m) \circ (\mathsf{I}_m + {}^m\mathsf{X}^m + \mathsf{I}_m) \circ (\curlyvee_m + \curlyvee_m)$
A11	$\curlywedge^m \circ \curlyvee_m = \mathsf{I}_m$
A12	$\mathbin{\rotatebox{180}{$\curlywedge$}}_0 = \mathsf{I}_0$
A13	$\mathbin{\rotatebox{180}{$\curlywedge$}}_{m+n} = \mathbin{\rotatebox{180}{$\curlywedge$}}_m + \mathbin{\rotatebox{180}{$\curlywedge$}}_n$
A14	$\curlyvee_0 = \mathsf{I}_0$
A15	$\curlyvee_{m+n} = (\mathsf{I}_m + {}^n\mathsf{X}^m + \mathsf{I}_n) \circ (\curlyvee_m + \curlyvee_n)$
A16	$\mathbin{\rotatebox{180}{$\curlyvee$}}^0 = \mathsf{I}_0$
A17	$\mathbin{\rotatebox{180}{$\curlyvee$}}^{m+n} = \mathbin{\rotatebox{180}{$\curlyvee$}}^m + \mathbin{\rotatebox{180}{$\curlyvee$}}^n$
A18	$\curlywedge^0 = \mathsf{I}_0$
A19	$\curlywedge^{m+n} = (\curlywedge^m + \curlywedge^n) \circ (\mathsf{I}_m + {}^m\mathsf{X}^n + \mathsf{I}_n)$
F3	$\curlyvee_m \uparrow^m = \mathbin{\rotatebox{180}{$\curlyvee$}}^m$
F4	$\curlywedge^m \uparrow^m = \mathbin{\rotatebox{180}{$\curlywedge$}}_m$
F5°	$((\mathsf{I}_m + \curlywedge^m) \circ ({}^m\mathsf{X}^m + \mathsf{I}_m) \circ (\mathsf{I}_m + \curlyvee_m)) \uparrow^m = \mathbin{\rotatebox{180}{$\curlyvee$}}^m \circ \mathbin{\rotatebox{180}{$\curlywedge$}}_m$

In Section 4.2, no assumptions about the nature of the transformers were made. Here the nature of the transformers needed for synchronous dataflow networks is made precise, resulting in the definition of quasiproper stream transformers. The feedback operation is adapted to reflect a special characteristic of feedback in synchronous dataflow networks: data in the feedback loop produced in one time slice is not used to produce new data before the next time slice.

The model $\mathsf{Rel}(S)$ of Section 4.2 is a general model. In case of dataflow, streams of data are transformed. This means that

$$S = (D \cup \{\surd\})^\infty = \mathbb{N} \to (D \cup \{\surd\})$$

for some set of data D, $\surd \notin D$. For a stream $x \in S$ and $k \in \mathbb{N}$, $x(0..k)$ is the initial segment of x of length $k+1$ and $x(k)$ is the datum occurring in x on the k-th tick of the global clock if $x(k) \in D$. The absence of a datum is represented by $\surd$; so $x(k) = \surd$ indicates that no datum occurs in stream x on the k-th tick. This may happen, for example, with the equality test $\curlyvee_1$: no datum is delivered on the k-th tick unless equal data are offered at its input ports on that tick. Owing to this approach to deal with the absence of data, it is quite natural in case of synchronous dataflow to look at finite streams as infinite

ones where no datum occurs from a certain tick. This point of view has the additional advantage that the relevant definitions can be kept simple. However, it is unnatural to uphold this view-point for asynchronous dataflow.

The stream transformers used to model the cells in synchronous dataflow networks have a "dependency on the past" property which is captured by the following definition.

Definition 3. (proper stream transformer)
A stream transformer $f \in \mathsf{Rel}(S)(m,n)$ is *proper* (or *determined by the past*) if

$$\begin{array}{l}\forall x \in S^m \cdot \forall x' \in S^m \cdot \\ \quad \{y(0) \mid y \in S^n, \langle x, y\rangle \in f\} = \{y'(0) \mid y' \in S^n, \langle x', y'\rangle \in f\} \wedge \\ \quad \forall k \in \mathbb{N} \cdot x(0..k) = x'(0..k) \;\Rightarrow \\ \qquad \{y(0..k+1) \mid y \in S^n, \langle x, y\rangle \in f\} = \{y'(0..k+1) \mid y' \in S^n, \langle x', y'\rangle \in f\} \; .\end{array}$$

Note that this property reduces at the beginning to a "constant output initially" property.

The proper stream transformers fail to include constants for connections such as I, X, $\aleph$ and $\beth$, because their intended meaning is to let data pass through them with a neglectible delay. Because at least the constants I and X are necessary in order to define a network algebra, stream transformers built from proper stream transformers and stream transformers with input and output ports that are directly connected must be allowed. The resulting stream transformers are called quasiproper stream transformers. A similar notion is used in [5].

Definition 4. (direct connection)
Two ports $i \in [m]$ and $j \in [n]$ are *directly connected* via a stream transformer $f \in \mathsf{Rel}(S)(m,n)$ if

$$\begin{array}{l}\forall (x_1, \ldots, x_m) \in S^m \cdot \forall (y_1, \ldots, y_n) \in S^n \cdot \\ \quad \langle (x_1, \ldots, x_m), (y_1, \ldots, y_n)\rangle \in f \;\Rightarrow\; x_i = y_j \; .\end{array}$$

We write $dc(f)$ for the set $\{(i,j) \mid i \text{ is directly connected with } j \text{ via } f\}$.

A stream transformer $f \in \mathsf{Rel}(S)(m,n)$ is a *direct connection* if

$$\forall i \in [m] \cdot \exists j \in [n] \cdot (i,j) \in dc(f) \wedge \forall j \in [n] \cdot \exists i \in [m] \cdot (i,j) \in dc(f) \; .$$

Definition 5. (quasiproper stream transformer)
A stream transformer in $\mathsf{Rel}(S)(m,n)$ is *quasiproper* if it can be described by an expression of the form

$$h \circ (\mathsf{I}_k \parallel \aleph^{m-(k+l)} \parallel \mathsf{I}_l) \circ (f \parallel g) \circ (\mathsf{I}_{k'} \parallel \beth_{n-(k'+l')} \parallel \mathsf{I}_{l'}) \circ h' \; ,$$

where $f \in \mathsf{Rel}(S)(m-l, n-l')$ is a proper stream transformer, $g \in \mathsf{Rel}(S)(m-k, n-k')$ is a direct connection, and $h \in \mathsf{Rel}(S)(m,m)$ and $h' \in \mathsf{Rel}(S)(n,n)$ are bijective direct connections. The constants $\aleph^n \in \mathsf{Rel}(S)(n, n+n)$ and $\beth_n \in \mathsf{Rel}(S)(n+n, n)$ used here are the ones defined below in Definition 6. The restriction of $\mathsf{Rel}(S)$ to quasiproper stream transformers is denoted by $\mathsf{QRel}(S)$. The further restriction of $\mathsf{QRel}(S)$ to functions is denoted by $\mathsf{QFn}(S)$.

With $\mathsf{QFn}(S)$ only deterministic synchronous dataflow networks can be modelled, whereas $\mathsf{QRel}(S)$ covers non-deterministic synchronous dataflow as well. If S is a set of streams of data, i.e. $S = (D \cup \{\surd\})^\infty$ for some set of data D, the constants of BNA as defined on $\mathsf{Rel}(S)$ in Section 4.2 are quasiproper functions. So the identity and transposition constants are in $\mathsf{QFn}(S)$ and $\mathsf{QRel}(S)$. In addition, both $\mathsf{QFn}(S)$ and $\mathsf{QRel}(S)$ are closed under the parallel and sequential composition operations as defined on $\mathsf{Rel}(S)$. As mentioned before, the feedback operation as defined on $\mathsf{Rel}(S)$ does not model feedback in synchronous dataflow networks properly. A related problem is that $\mathsf{QFn}(S)$ is not closed under this feedback operation. All this means that only a more appropriate feedback operation and the additional constants for synchronous dataflow have to be defined.

Definition 6. (stream transformer model for synchronous dataflow)
The parallel and sequential composition operations on $\mathsf{QRel}(S)$ are the restrictions of the parallel and sequential composition operations on $\mathsf{Rel}(S)$ to $\mathsf{QRel}(S)$. The identity and transposition constants in $\mathsf{QRel}(S)$ are the ones in $\mathsf{Rel}(S)$.

The feedback operation is redefined on $\mathsf{QRel}(S)$ as follows:

Notation

$f \uparrow^p \in \mathsf{QRel}(S)(m,n)$ for $f \in \mathsf{QRel}(S)(m+p, n+p)$

Definition

$$f \uparrow^1 = \begin{array}{ll} \{\langle x, y\rangle \mid x \in S^m \wedge y \in S^n \wedge \exists z \in S \cdot \langle x^\frown z, y^\frown z\rangle \in f\} & \text{if } (m+1, n+1) \notin dc(f) \\ (\mathsf{I}_m \mathbin{+\!\!+} \mathsf{!}_1) \circ f \circ (\mathsf{I}_n \mathbin{+\!\!+} \mathsf{?}^1) & \text{otherwise} \end{array}$$

for $p \neq 1$, $\uparrow^p$ is defined by the equations occurring as axioms R5–R6 of BNA

The constants $\mathsf{!}_n \in \mathsf{QRel}(S)(0,n)$ and $\mathsf{?}^n \in \mathsf{QRel}(S)(n,0)$ used here are the ones defined right away.

The additional constants for synchronous dataflow are defined on $\mathsf{QRel}(S)$ as follows:

Notation

$\aleph^n \in \mathsf{QRel}(S)(n, n+n)$
$\mathsf{?}^n \in \mathsf{QRel}(S)(n, 0)$
$\bowtie_n \in \mathsf{QRel}(S)(n+n, n)$
$\mathsf{!}_n \in \mathsf{QRel}(S)(0, n)$

Definition

$\aleph^n = \{\langle x, x^\frown x\rangle \mid x \in S^n\}$
$\mathsf{?}^n = \{\langle x, ()\rangle \mid x \in S^n\}$
$\bowtie_n = \{\langle (x_1, \ldots, x_n, y_1, \ldots, y_n), (x_1 \& y_1, \ldots, x_n \& y_n)\rangle \mid (x_1, \ldots, x_n), (y_1, \ldots, y_n) \in S^n\}$
where $(x \& y)(k) = x(k)$ if $x(k) = y(k)$ and $(x \& y)(k) = \surd$ otherwise
$\mathsf{!}_n = \{\langle (), (\surd^\infty, \ldots, \surd^\infty)\rangle\}$

In Definition 1, the feedback operation was defined such that, for each data transformer f, the feedback loop behaves as the greatest fixpoint of f relative to the input stream of $f\uparrow$. In case of proper stream transformers, there is always a unique fixpoint provided the transformer is a function or a continuous relation (with respect to the prefixes of streams). It means that the feedback loop is also the least fixpoint. This is needed to model feedback in synchronous dataflow networks properly; for otherwise it does not agree with the operational understanding that it is iteratively feeding the network concerned with data produced by it in the previous step. The adaptation of the feedback operation given in Definition 6 is needed to get a unique fixpoint in case of quasiproper stream transformers as well. It also guarantees that $\mathsf{QFn}(S)$ is closed under feedback. Because $\Qoppa\uparrow$ now produces a dummy stream, it equals the dummy source. For this reason, $\Qoppa$ is used instead of $\Qoppa$ as constant for synchronous dataflow. Note that this stream transformer model does not have the global crash property of the data transformer model from Section 4.2: if a component of a network fails to produce output on some tick of the global clock, the effect is merely that the components connected to the port(s) concerned will fail to produce output on some future tick.

Theorem 3. $(\mathsf{QFn}(S), +\!\!+, \circ, \uparrow, \mathsf{I}, \mathsf{X})$ *is a model of BNA. The constants* $\Qoppa, \Qoppa, \Qoppa, \Qoppa$ *satisfy the additional axioms for synchronous dataflow networks (Table 3).*

Proof: For the first part, it is enough to prove R1–R4 and F1–F2. According to [14,15], it suffices to prove R1–R4 for $m = 1$, and R4 additionally for $k = l = 1$ and $g = {}^1\mathsf{X}^1$. The proofs concerned are straightforward proofs by case distinction – the cases depending on whether the ports relevant to the feedback loop are directly connected or not. The second part is a matter of tedious, but simple calculation. □

5.3 Process algebra model for synchronous dataflow

In this subsection, the specialization of the process algebra model of BNA (Section 4.3) for synchronous dataflow networks is given. In this case, we will make use of $\mathrm{ACP}_\tau^{\mathrm{drt}}$. Recall that $\mathrm{ACP}_\tau^{\mathrm{drt}}$ is $\mathrm{ACP}^{\mathrm{drt}}$ – the discrete relative time extension of ACP – extended with abstraction based on branching bisimulation.

In Section 4.3, only a few assumptions about wires and atomic cells were made. Here it is first explained how these ingredients are actualized for synchronous dataflow networks. Because of the crucial role of the time slices determined by the ticks of a global clock, discrete-time process algebra is used.

Definition 7. (wires and atomic cells in synchronous dataflow networks)
In the synchronous case, the identity constant, called the *minimal stream delayer*, is the wire $\mathsf{I}_1 = (1, 1, \mathtt{msd}_1^1)$ where $\mathtt{msd}_1^1$ is defined by

$$\mathtt{msd}_1^1 = \underline{\underline{\tau}} \cdot (er_1(x) \,;\, \underline{\underline{s}}_1(x)) \cdot \sigma_{\mathsf{rel}}(\mathtt{msd}_1^1) \;.$$

The constants I_n, for $n \neq 1$, and ${}^m\mathsf{X}^n$ are defined by the equations occurring as axioms B6 and B8–B9, respectively, of Table 1.

In the synchronous case, the deterministic cell computing a function $f : D^m \to D^n$, and having $\boldsymbol{a} = (a_1, \ldots, a_n) \in D^n$ as its initial output tuple, is the network $C_f(\boldsymbol{a}) = (m, n, P_f(\boldsymbol{a}))$ where P_f is defined by

$$P_f(\boldsymbol{a}) = \underline{\underline{\tau}} \cdot (Out(\boldsymbol{a}) \,|||\, ((er_1(x_1) \parallel \ldots \parallel er_m(x_m)) \,;\, \sigma_{\mathsf{rel}}(P_f(f(x_1, \ldots, x_m))))) \;,$$

$$\text{where } Out(\boldsymbol{a}) = \underline{\underline{s}}_1(a_1) \parallel \ldots \parallel \underline{\underline{s}}_n(a_n) \;.$$

The non-deterministic cell computing a (finitely branching) relation $R \subseteq D^m \times D^n$, and having $A \subseteq D^n$ as its set of possible initial output tuples, is the network $C_R(A) = (m, n, P_R(A))$ where P_R is defined by

$$P_R(A) = \underline{\underline{\tau}} \cdot (Out(A) \,|||\, ((er_1(x_1) \parallel \ldots \parallel er_m(x_m)) \,;\, \sigma_{\mathsf{rel}}(P_R(R(x_1, \ldots, x_m))))) \;,$$

$$\text{where } Out(A) = \underline{\underline{\tau}} \lhd A = \emptyset \rhd \sum_{(a_1,\ldots,a_n) \in A} (\underline{\underline{s}}_1(a_1) \parallel \ldots \parallel \underline{\underline{s}}_1(a_n)) \;.$$

The restriction of $\mathsf{Proc}(D)$ to the processes that can be built under this actualization is denoted by $\mathsf{SProc}(D)$.

The definition of $\mathtt{msd}_1^1$ given above expresses the following. The process $\mathtt{msd}_1^1$ waits until a datum is offered at its input port. When a datum is available at the input port, $\mathtt{msd}_1^1$ delivers the datum at its output port in the same time slice. From the next time slice, it proceeds with repeating itself.
The definition of P_f expresses the following. In the current time slice $P_f(\boldsymbol{a})$ produces the data $a_1, \ldots, a_n$ at the output ports $1, \ldots, n$, respectively. In parallel, $P_f(\boldsymbol{a})$ waits until one datum is offered at each of the input ports $1, \ldots, m$. The waiting may last into subsequent time slices. When data are available at all input ports, $P_f(\boldsymbol{a})$ proceeds with repeating itself from the next time slice with a new output tuple, viz. the value of the function f for the consumed input tuple. The non-deterministic case (P_R) is similar.

For $\mathsf{SProc}(D)$, the operations and constants of BNA as defined on $\mathsf{Proc}(D)$ can be taken with $\mathtt{msd}_1^1$ as wire. This means that only the additional constants for synchronous dataflow have to be defined.

Definition 8. (process algebra model for synchronous dataflow)
The operations $+\!\!+$, $\circ$, $\uparrow^n$ on $\mathsf{SProc}(D)$ are the instances of the ones defined on $\mathsf{Proc}(D)$ for $\mathtt{msd}_1^1$ as wire. Analogously, the constants I_n and ${}^m\mathsf{X}^n$ in $\mathsf{SProc}(D)$ are the instances of the ones defined on $\mathsf{Proc}(D)$ for $\mathtt{msd}_1^1$ as wire.

The additional constants in $\mathsf{SProc}(D)$ are defined as follows:

Notation
$\wr^1 \in \mathsf{SProc}(D)(1,2)$
$\mathfrak{z}^1 \in \mathsf{SProc}(D)(1,0)$
$\curlyvee_1 \in \mathsf{SProc}(D)(2,1)$
$\mathfrak{f}_1 \in \mathsf{SProc}(D)(0,1)$

Definition

$\Re^1 = (1, 2, copy^1)$, where $copy^1 = \underline{\underline{\tau}} \cdot (er_1(x) \,;\, (\underline{\underline{s}}_1(x) \parallel \underline{\underline{s}}_2(x))) \cdot \sigma_{\mathsf{rel}}(copy^1)$
$\mathfrak{Z}^1 = (1, 0, sink^1)$, where $sink^1 = \underline{\underline{\tau}} \cdot (er_1(x) \,;\, \underline{\underline{\tau}}) \cdot \sigma_{\mathsf{rel}}(sink^1)$
$\mathfrak{Y}_1 = (2, 1, eq_1)$, where $eq_1 = \underline{\underline{\tau}} \cdot (er_1(x_1) \,;\, P_2(x_1) + er_2(x_2) \,;\, P_1(x_2))$ and
$P_i(x) = \sigma_{\mathsf{rel}}(eq_1) + \underline{\underline{\underline{er}}}_i(y) \,;\, (\underline{\underline{s}}_1(x) \lhd x = y \rhd \underline{\underline{\tau}}) \cdot \sigma_{\mathsf{rel}}(eq_1)$ for $i \in [2]$
$\mathfrak{P}_1 = (0, 1, source_1)$, where $source_1 = \underline{\underline{\tau}} \cdot \delta$

for $n \neq 1$, these constants are defined by the equations occurring as axioms A12–A19 in Table 3

The equality test $\mathfrak{Y}_1$ does not necessarily perform one test per time slice; it does so in order not to cause a time delay. The definition of eq_1 expresses the following. The process eq_1 waits until a datum is offered at one of its input ports. When a datum is available at one input port, it waits till the end of the time slice concerned for a datum at the other port. If this happens, it tests the equality of the data, delivers either in case the test succeeds, and then proceeds with repeating itself from the next time slice. Otherwise, it skips the equality test and proceeds with repeating itself from the next time slice.

The simpler equality test $\overline{\mathfrak{Y}}_1 = (2, 1, \overline{eq}_1)$, where

$$\overline{eq}_1 = \underline{\underline{\tau}} \cdot ((er_1(x) \parallel er_2(y)) \,;\, (\underline{\underline{s}}_1(x) \lhd x = y \rhd \underline{\underline{\tau}}) \cdot \sigma_{\mathsf{rel}}(\overline{eq}_1)) \,,$$

is not appropriate. This equality test does not let data always pass through it with a neglectible delay. This means that it does not behave properly if the feedback operation is applied; $\overline{\mathfrak{Y}}_1 \uparrow^1$ is the process that deadlocks after having read one datum – it is a kind of dummy sink. This failure to consume data does not fit in with the idea of permanent flows of data which underlies synchronous dataflow.

Lemma 1. *The wire* $\mathsf{I}_1 = (1, 1, \mathtt{msd}_1^1)$ *gives an identity flow of data, i.e. for all* $f = (m, n, P)$ *in* $\mathsf{SProc}(D)$, $\mathsf{I}_m \circ f = f = f \circ \mathsf{I}_n$.

Proof: It suffices to show that these equations hold for the atomic cells and the constants. The result then follows by induction on the construction of a network in $\mathsf{SProc}(D)$. $\mathsf{I}_n \circ \mathsf{I}_n = \mathsf{I}_n$ and ${}^m\mathsf{X}^n \circ \mathsf{I}_n = {}^m\mathsf{X}^n = \mathsf{I}_m \circ {}^m\mathsf{X}^n$ follow trivially from $\mathsf{I}_1 \circ \mathsf{I}_1 = \mathsf{I}_1$. For a proof of $\mathsf{I}_1 \circ \mathsf{I}_1 = \mathsf{I}_1$, we refer to [2]. So the asserted equations hold for I_n and ${}^m\mathsf{X}^n$. The proof for the remaining constants and the atomic cells is a laborious piece of work in the same style. □

Theorem 4. $(\mathsf{SProc}(D), +\!\!+, \circ, \uparrow, \mathsf{I}, \mathsf{X})$ *is a model of BNA. The constants* $\Re$, $\mathfrak{Z}$, $\mathfrak{Y}$, $\mathfrak{P}$ *satisfy the additional axioms for synchronous dataflow networks (Table 3).*

Proof: A simple calculation shows that $\mathsf{I}_0 +\!\!+ f = f = f +\!\!+ \mathsf{I}_0$ for all $f \in \mathsf{SProc}(D)$. The first part then follows immediately from Theorem 2 and Lemma 1. The proof of the second part is a matter of tedious, but unproblematic calculation in the style of [2]. □

Theorem 5. *The axioms in Table 3 are complete for closed terms.*

Proof: For the proof of this theorem, we refer to [9]. □

Queues that deliver data with a neglectible delay and never contain more than one datum are an idealized concept; they do not occur in practice. More practical are wires that are interpreted as bounded queues. It seems that bounded queues are most easily modelled as components of asynchronous dataflow networks.

6 Closing remarks

Concerning connections with earlier work on dataflow some additional remarks are in order.

In [5] a model for synchronous dataflow networks is presented. Our Section 5.2 on a stream transformer model for synchronous dataflow can be seen as a rephrasing of this work. We consider the stream transformer model described in Section 5.2 to be more denotational and the process algebra model described in Section 5.3 to be more operational.

The model presented in [5] is essentially a BNA model, although it has some slightly different operations and constants. For example, it has "left-feedback" (*) instead of "right-feedback" (see also the table below) and "input sharing" ($^\wedge$) instead of the constants $\curlywedge$ and X. However, the constants and operations of BNA are definable in terms of the ones of this model and vice versa. The setting of [5] may be obtained from our general network algebra setting by taking BNA with the following parameters: (1) the set of data D is $\mathbb{N}$; (2) the atomic cells are "successor" and "conditional"; (3) the additional constants for branching connections are $\curlywedge$, $\curlyvee$ and $\curlyvee$. Kahn's history model [21] is also essentially a BNA model (with $\curlywedge$, $\curlyvee$ and $\curlyvee$ as additional constants) and so are Broy's oracle based models [12]. SCAs [28] require for each internal stream in a network an initial value. We have taken that viewpoint as well.

Both the left- and right-feedback can be used. The left-feedback can be defined in terms of the right-feedback as follows:

$$\uparrow^p f = ({}^p\mathsf{X}^m \circ f \circ {}^p\mathsf{X}^n) \uparrow^p,\ \ f : p+m \to p+n\ .$$

Other proposed feedback-like operators can be defined in terms of left- or right-feedback:

Name	Symbol	Network algebra definition	Ref.
feedback	*	$f^* = \uparrow^1 f,\ \ f : 1+m \to 1+n$	[5]
feedback	μ	$\mu f = (f \circ \wedge^m) \uparrow^m,\ \ f : n+m \to m$	[13]
(unary) star	*	$f^* = \wedge^1 \circ (\mathsf{I}_1 + (\vee_1 \circ f \circ \wedge^1) \uparrow^1) \circ \vee_1,\ \ f : 1 \to 1$	[17]
iteration	†	$f^\dagger = \uparrow^m (\vee_m \circ f),\ \ f : m \to m+n$	[18]
(binary) star	*	$f^* g = \wedge^1 \circ (\mathsf{I}_1 + \uparrow^1 (\vee_1 \circ f \circ \wedge^1)) \circ \vee_1 \circ g,\ \ f, g : 1 \to 1$	[22]

Acknowledgements The understanding on dataflow computation of the third author was much clarified by discussions with M. Broy and K. Stølen. The first author acknowledges discussions with J.V. Tucker on SCAs.

References

1. Baeten, J.C.M., Bergstra, J.A.: On sequential composition, action prefixes and process prefix. Formal Aspects of Computing 6, 250–268 (1994)
2. Baeten, J.C.M., Bergstra, J.A.: Some simple calculations in relative time process algebra. In: Aarts, E.H.L. et al. (eds.) Simplex Sigillum Veri: A Liber Amicorum for Prof. F.E.J. Kruseman Aretz. Department of Computer Science, Eindhoven University of Technology (1995)
3. Baeten, J.C.M., Middelburg, C.A.: Process Algebra with Timing. Monographs in Theoretical Computer Science, An EATCS Series, Springer-Verlag (2002)
4. Baeten, J.C.M., Weijland, W.P.: Process Algebra. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press (1990)
5. Barendregt, H., Wupper, H., Mulder, H.: Computable processes. Tech. Rep. CSI-R9405, Computing Science Institute, Catholic University of Nijmegen (1994)
6. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Information and Control 60, 109–137 (1984)
7. Bergstra, J.A., Middelburg, C.A., Ştefănescu, G.: Network algebra for synchronous and asynchronous dataflow. Report P9508, Programming Research Group, University of Amsterdam (1995)
8. Bergstra, J.A., Middelburg, C.A., Ştefănescu, G.: Network algebra for asynchronous dataflow. International Journal of Computer Mathematics 65, 57–88 (1997)
9. Bergstra, J.A., Ştefănescu, G.: Network algebra with demonic relation operators. Report P9509, Programming Research Group, University of Amsterdam (1995)
10. Böhm, A.P.W.: Dataflow Computation. CWI Tracts 6, Centre for Mathematics and Computer Science, Amsterdam (1984)
11. Brock, J.D., Ackermann, W.B.: Scenarios: A model of non-determinate computation. In: Diaz, J., Ramos, I. (eds.) Formalisation of Programming Concepts. pp. 252–259. LNCS 107, Springer-Verlag (1981)
12. Broy, M.: Nondeterministic dataflow programs: How to avoid the merge anomaly. Science of Computer Programming 10, 65–85 (1988)
13. Broy, M.: Functional specification of time sensitive communicating systems. ACM Transactions on Software Engineering and Methodology 2, 1–46 (1993)
14. Căzănescu, V.E., Ştefănescu, G.: A formal representation of flowchart schemes I. Analele Universităţii Bucuresti, Matematică - Informatică 37, 33–51 (1988)
15. Căzănescu, V.E., Ştefănescu, G.: A formal representation of flowchart schemes II. Studii si Cercetări Metematice 41, 151–167 (1989)
16. Căzănescu, V.E., Ştefănescu, G.: Towards a new algebraic foundation of flowchart scheme theory. Fundamenta Informaticae 13, 171–210 (1990)
17. Copy, I.M., Elgot, C.C., Wright, J.B.: Realization of events by logical nets. Journal of the ACM 5, 181–196 (1958)
18. Elgot, C.C.: Monadic computation and iterative algebraic theories. In: Rose, H.E., Sheperdson, J.C. (eds.) Logic Colloquium '73. pp. 175–230. Studies in Logic and the Foundations of Mathematics, Volume 80, North-Holland (1975)

19. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. Journal of the ACM 43(3), 555–600 (1996)
20. Jonsson, B.: A fully abstract trace model for dataflow and asynchronous networks. Distributed Computing 7, 197–212 (1994)
21. Kahn, G.: The semantics of a simple language for parallel processing. In: Rosenfeld, J.L. (ed.) Information Processing '74. pp. 471–475 (1974)
22. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies. pp. 3–41. Annals of Mathematical Studies, Volume 34, Princeton University Press (1956)
23. Kok, J.: A fully abstract semantics for data flow nets. In: de Bakker, J.W., Nijman, A.J., Treleaven, P.C. (eds.) PARLE '87. pp. 351–368. LNCS 259, Springer-Verlag (1987)
24. Russell, J.: Full abstraction for nondeterministic dataflow networks. In: FoCS '89. IEEE Computer Science Press (1989)
25. Ştefănescu, G.: On flowchart theories: Part II. The nondeterministic case. Theoretical Computer Science 52, 307–340 (1987)
26. Ştefănescu, G.: Feedback theories (a calculus for isomorphism classes of flowchart schemes). Revue Roumaine de Mathematiques Pures et Applique 35, 73–79 (1990)
27. Ştefănescu, G.: Algebra of flownomials. Part 1: Binary flownomials, basic theory. Report TUM I9437, Department of Computer Science, Technical University Munich (1994)
28. Thompson, B.C., Tucker, J.V.: Algebraic specification of synchronous concurrent algorithms and architecture. Tech. Rep. 10-91, Department of Mathematics and Computer Science, University College of Swansea (1991)