TECHNISCHE UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

The Algebra of Stream Processing Functions

Manfred Broy, Gheorghe Ştefănescu

TUM-I9620
SFB-Bericht Nr.342/11/96 A
Mai 1996

TUM–INFO–05-96-I20-350/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

The Algebra of Stream Processing Functions*

Manfred Broy[1] and Gheorghe Ştefănescu[2]

[1]Institute of Informatics, Technical University Munich
D-80290 München, Germany

Email: `broy@informatik.tu-muenchen.de`

and

[2]Institute of Mathematics, Romanian Academy
P.O. Box 1-764, RO-70700 Bucharest, Romania

Email: `ghstef@imar.ro`

Abstract. Dataflow networks are a model of concurrent computation. They consist of a collection of concurrent asynchronous processes which communicate by sending data over FIFO channels. In this paper we study the algebraic structure of the dataflow networks and base their semantics on stream processing functions.

The algebraic theory is provided by the calculus of flownomials which gives a unified presentation of regular algebra and iteration theories. The kernel of the calculus is an equational axiomatization called Basic Network Algebra (BNA) for flowgraphs modulo graph isomorphism.

We show that the algebra of stream processing functions called SPF (used for deterministic networks) and the algebra of sets of stream processing functions called $\mathcal{P}\mathsf{SPF}$ (used for nondeterministic networks) are BNA algebras. As a byproduct this shows that both semantic models are compositional. We also identify the additional axioms satisfied by the branching components that correspond to constants in these two algebraic theories.

For the deterministic case we study in addition the coarser equivalence relation on networks given by the input-output behaviour and provide a correct and complete axiomatization.

*The first author was partially supported by the DFG within Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen". The second author was partially supported by a DAAD grant.

1 Introduction

The idea of control and data flow is a classic concept that can be found in many approaches to computation, programming, and computing machinery. Often the flow is visualized by flow graphs. The idea of data flows had mainly two sources. Single assignment languages are based on the concept of a set of (nonrecursive) declarations. The order of the evaluation of the declaration is then only determined by their data dependencies. These dependencies can be shown in an acyclic graph called their data flow graph. Influenced by these ideas and by the concept of Petri-nets and their firing rules, Jack Dennis suggested data flow graphs and gave firing rule semantics for them. Quite independently, versions of data flow graphs can be found in many software engineering methods and also for the description of switching circuits.

Gilles Kahn suggested a mathematical model for asynchonously communicating agents that could be used as a model for deterministic data flow nets. The dataflow networks used in [Kah74] describe a collection of processes which work in a parallel and asynchronous way and communicate by sending values over FIFO channels. Moreover, Kahn's dataflow networks were deterministic and thus the input-output relation specified by such processes is actually a (continuous) function. The main result of Kahn in [Kah74] asserts that the function specified by a deterministic network may be obtained from the functions specified by its components using the least fixed-point construction.

It turns out that Kahn's elegant theorem cannot be extended in an easy way to the case of nondeterministic dataflow networks. In such networks, the components are capable of making arbitrary choices during computation and the input-output behaviour specified by such a network is not longer a function, but an arbitrary relation. For such networks, fundamental results by Keller [Kel78] and Brock-Ackermann [BrA81] have shown a mismatch between the operational meaning of the networks and their input-output behaviour. In other words, the input-output behaviour of its components is no longer sufficient to compute the behaviour of a network. This situation, known as merge or Brock-Ackermann anomaly, was solved by adding information to the input-output behaviour by using scenarious, traces, or oracles, etc. Extensions to the nondeterministic case were suggested for instance in [SN85, Bro87, Kok87, Jon89, Bro93].

In this paper we take the viewpoint of [Park83, Bro87] and model nondeterministic dataflow networks with the help of oracles. An oracle provides a priori global information on the choices in all the nondeterministic points and it allows to give the semantic of a nondeterministic network by a set of (stream processing) functions.

Graphs are used in many methods in computing science to represent the flow of information, data and control. To be able to use algebraic techniques for such graphs, we have to represent graphs by terms. To do this, we have to find appropriate algebraic operators for the construction of graphs. Typically, the same graphs (isomorphic graphs) can then be represented by quite different terms. Two terms that denote the same graphs are, therefore, called graph isomorphic.

Graph isomorphism of terms is an equivalence relation on terms that can be axiomatized by equations. In addition to these laws of graph isomorphism, we may use more specific laws that hold due to the semantic theories of the specific flow models.

Algebraic models for nondeterministic data flow are difficult to be obtained as an extension of those for deterministic data flow, mainly due to the unsoundness of the fixed-point equation. Our approach is to use the calculus of flownomials, see [Ste94].

The calculus of flownomials is an algebraic calculus very similar to the calculus of polynomials. Its aim is to capture the syntax and the semantics of several digraph-like models used in computer science. It was obtained as a unification of the classical regular algebras presented in [Kle56, Con71] and of the iteration theories developed starting with the study of flowchart schemes in [Elg75, BE93b, Ste87a, Ste86, CaS90] among others. The basic results of the calculus and some historical comments may be found in [Ste94].

In order to obtain an axiomatization for cyclic processes one has to use a looping operation. We use the *feedback* operator introduced in [Ste86]. The key feature of this operation is that

(1) after its application both the input and the output are hidden (they are not visible anymore).

Some other possibilities are *repetition* [Kle56, Ste87b], where

(2) after the application of this operation both the input and the output remain visible,

or *iteration* [Elg75], where

(3) after the application of this operation the input remains visible, but not the output[1].

The kernel of the flownomial calculus is given by the *$a\alpha$-flow algebra*; we also use the BNA (Basic Network Algebra) acronym of [BS94] for the corresponding equational theory. This algebra gives a complete characterization for flowgraphs/networks modulo graph isomorphism. For a detailed treatment see [Ste86, CaS88–89, Ste94].

One aim of the present paper is to show that the flownomial calculus may be applied to the study of (asynchronous) dataflow computation as well. As we said, what we study here from the various approaches to handle the semantics of nondeterministic dataflow networks are the algebraic properties of the oracle-based model presented in [Park83, Bro87, Bro93]. In this approach, the semantics of a nondeterministic dataflow network is specified as a set of stream processing functions.

The main results of our paper are as follows:

[1] In [Bro93] a "feedback" operation different from the one in this paper is used. In fact the operation in [Bro93] is a dual iteration, where after the application of the operation the output remains available, but not the input.

- We show that the algebra of stream processing functions called SPF (which we use as a semantic model for deterministic networks) and the algebra of sets of stream processing functions called $\mathcal{P}\mathsf{SPF}$ (which we use as a semantic model for nondeterministic networks) are BNA models. As a byproduct these results show that both semantics above are compositional. We also identify the additional axioms satisfied by the branching components that correspond to constants in these two algebras.

- For the deterministic case we also study the coarser equivalence on networks given by the input-output behaviour and provide a correct and complete axiomatization.

A somewhat similar approach is given by E. Stark in [Sta92]. There it is shown that an algebra with the same operators (parallel and sequential compositions and feedback) may be used to study nondeterministic dataflow networks. As branching constants Stark uses the 'copy' constant and certain sink and source constants. The main result of [Sta92] is a theorem of correctness and completeness for networks modulo "buffer bisimilarity".

The paper is organized as follows: In section 2 we give a short overview of the calculus of flownomials. Section 3 is devoted to the study of deterministic dataflow networks. We give two complete axiomatizations presented as extension of BNA (Basic Network Algebra) axioms, namely one for networks modulo graph isomorphism equivalence and one for the coarser equivalence induced on networks by the input-output behaviour. Section 4 deals with nondeterministic dataflow networks. We show that the algebra $\mathcal{P}\mathsf{SPF}$ (sets of stream processing functions) model satisfies the BNA axioms as well. Some additional sound laws are given, but the problem of a complete axiomatization for the equivalence induced on networks by the $\mathcal{P}\mathsf{SPF}$ semantics is not solved and left open. Detailed proofs of certain technical theorems are presented in section 5. Some conclusions are given in the last section 6.

2 Flownomials

The algebra of binary flownomials gives an algebraic presentation of directed flowgraphs and their behaviours. It uses three operations:

> " $+\!\!+$ " (*parallel composition*), "$\cdot$" (*sequential composition*) and "$\uparrow$" (*feedback*)

and various constants for describing the branching structure of the flowgraphs:

> "I" (*identity*), "X" (*transposition*), "$\wedge_k$" (*ramification*) and "$\vee^k$" (*identification*).

In table 3 we use some particular cases of the ramification and identification constants, namely $\wedge_0, \wedge_2, \vee^0, \vee^2$ denoted by $\perp, \wedge, \top, \vee$, respectively.

In the standard version presented in [Ste94] there are three groups of algebraic equations (see table 3):

(A) a large group of algebraic equations for flowgraphs modulo graph isomorphism
B1–B10, A1–A19, R1–R5, and F1–F5;

(S) some critical algebraic equations S1–S4 for the data flow nodes for ramification and identification;

(Z) an axiom scheme ENZ, presented as a conditional equation.

Following Milner, one may call the axioms (A) "static laws". The critical axioms S1–S4 describe the dynamic part of the model with the possibility to make copies of or to delete some components.[2] (Z) is an invariance law which allows to use S1–S4 in a cyclic environment.

The kernel of the axioms are the *BNA axioms* (the resulting algebraic structure is called *$a\alpha$-flow algebra*)

B1–B10, R1–R5, and F1–F2

which gives a complete axiomatization for flowgraphs with bijective connections modulo graph isomorphism. The remaining graph isomorphism axioms A1–A19 and F3–F5 give a complete axiomatization for the branching constants considered as *angelic* finite relations where divergence is not dominant. This standard version was designed to handle sequential flowchart algorithms. One goal of the present paper is to study axiomatizations for the branching structures of the dataflow networks, starting with the axiomatization of the angelic theory of relations.

Once the graph isomorphism axioms are considered, one has to add a few very simple axioms (as in S and Z above) in order to obtain the classical settings of algebraic theories and iteration theories or matrix theories as well as regular algebras.

One resulting algebraic structure that is of interest for the study of deterministic dataflow computation is the *$d\alpha$-flow algebra* defined by

- the graph isomorphism axioms with the branching constants $\perp, \wedge, \top$,
- the critical axioms S3–S4,
- the enzymatic axiom for converses of functions, i.e. for terms written with $+\!\!+, \cdot, \mathsf{I}, \mathsf{X}, \perp, \wedge$.

This algebraic structure is dual to the *strong iteration theory* structure of [Ste87b] and it is complete for the flowgraphs modulo unfolding equivalence (see chapter 8 of [Ste94] for more details).

Example 2.1 As a running example we use the dataflow networks as shown in Figure 1(a)–(c). They may be represented by flownomial expressions as well. For

[2]They are sometimes known in computer science as the "referential transparancy" and "garbage collection" properties.

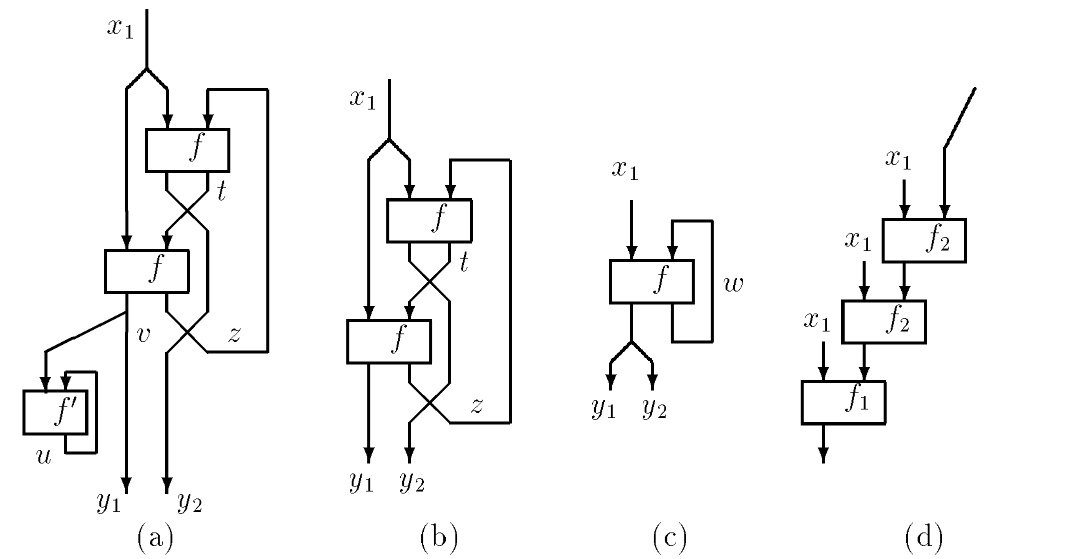

Figure 1: Dataflow networks

instance, the dataflow network shown in (a) may be represented by the following expression:

$$\wedge^1 \cdot [(\mathsf{I}_1 \mathbin{+\!\!+} f \cdot {}^1\mathsf{X}^1) \cdot (f \mathbin{+\!\!+} \mathsf{I}_1) \cdot (\wedge^1 \cdot (f' \uparrow^1 \mathbin{+\!\!+} \mathsf{I}_1) \mathbin{+\!\!+} {}^1\mathsf{X}^1)] \uparrow^1$$

We will show in the following that all the networks in figure 1 compute the same stream processing function – provided the ramification constant $\wedge$ is interpreted as the copy constant $\curlywedge$ (see below) and the cells are deterministic components – and, moreover, their equality may be proved using the $d\alpha$-flow axioms. $\square$

In the next section we adapt the algebra of flownomials to data flow nets.

3 Deterministic Networks

In this section we construct a semantic model $\mathsf{SPF}(M)$ for the interpretation of deterministic dataflow networks. It is based on stream processing functions.

A *stream* represents a communication history of a channel. A stream of messages over a given message set M is a finite or infinite sequence of messages. We define the set of streams M^ω by

$$M^\omega =_{def} M^* \cup M^\infty$$

By $x \frown y$ we denote the result of concatenating two streams x and y. We assume that $x \frown y = x$, if x is infinite. By $\langle\rangle$ we denote the empty stream.

If a stream x is a *prefix* of a stream y, we write $x \sqsubseteq y$. The relation $\sqsubseteq$ is called *prefix order*. It is formally specified as follows:

$$x \sqsubseteq y =_{def} \exists z \in M^\omega : x \frown z = y$$

The behavior of deterministic interactive systems with n input channels and m output channels is modeled by functions

$$f : (M^\omega)^n \to (M^\omega)^m$$

called *(m, n)-ary stream processing functions*. We often denote function application $f(x)$ by $f.x$ to avoid brackets. A stream processing function is called *prefix monotonic*, if for all tuples of streams $x, y \in (M^\omega)^n$ we have

$$x \sqsubseteq y \Rightarrow f.x \sqsubseteq f.y$$

This particular ordering is extended to tuples and functions pointwise in a straightforward way. A stream processing function f is called *continuous*, if f is monotonic and for every directed set $S \subseteq M^\omega$ we have:

$$f. \sqcup S = \sqcup \{f.x : x \in S\}$$

By $\sqcup S$ we denote the least upper bound of a set S, if it exists. A set S is called *directed*, if for any pair of elements x and y in S there exists an upper bound in S. The set of streams is complete in the sense that for every directed set of streams there exists a least upper bound.

In the following we will use an extension of this setting to the many sorted case. Let S be a set of sorts. Let $D = \{D_s\}_{s \in S}$ be an S-sorted set of messages and $M_s := D_s^* \cup D_s^\infty$ be the set of streams over D_s representing communication histories of channels of type s.

Concatenation in S^* is denoted by $+$. Let $a \in S^*$. Hence $a = a_1 + \ldots + a_{|a|}$, where a_i is the i-th letter of a. Denote by M_a the product $M_{a_1} \times \ldots \times M_{a_{|a|}}$.

Given an S-sorted set D of messages and the corresponding sets of streams M_s for $s \in S$, we define the set of stream processing functions with input sorts $a \in S^*$ and output sorts $b \in S^*$ by

$$\mathsf{SPF}(M)(a, b) = \{f : M_a \to M_b \mid f \text{ is prefix continuous }\}.$$

The BNA constants and operations are interpreted as follows:

- **Summation:** For $f \in \mathsf{SPF}(M)(a, b)$ and $g \in \mathsf{SPF}(M)(c, d)$ the parallel sum $f \mathbin{+\!\!+} g \in \mathsf{SPF}(M)(a + c, b + d)$ is defined by

$$(f \mathbin{+\!\!+} g)(x, y) = (f(x), g(y)), \quad \text{for } x \in M_a \text{ and } y \in M_c$$

- **Composition:** For $f \in \mathsf{SPF}(M)(a, b)$ and $g \in \mathsf{SPF}(M)(b, c)$ the functional composition $f \cdot g \in \mathsf{SPF}(M)(a, c)$ is the usual one defined by

$$(f \cdot g)(x) = g(f(x)), \quad \text{for } x \in M_a$$

  Note that we have used the diagramatic order.

- **Feedback:** For $f \in \mathsf{SPF}(M)(a+c, b+c)$ the feedback $f\uparrow^c \in \mathsf{SPF}(M)(a,b)$ is defined as follows: For streams $x \in M_a$, we specify

$$f\uparrow^c(x) = \bigsqcup_{k\geq 1} y_k$$

  where the streams $y_k \in M_b$ and $z_k \in M_c$ are inductively defined by:[3]

  $(y_1, z_1) = f(x, \langle\,\rangle)$, where $\langle\,\rangle$ denotes the empty stream, and

  $(y_{k+1}, z_{k+1}) = f(x, z_k)$, for $k \geq 1$.

  Since f is continuous we can equivalently define $f\uparrow^c$ by fixpoint techniques, because

$$(y, z) = \Big(\bigsqcup_{k\geq 1} y_k, \bigsqcup_{k\geq 1} z_k\Big)$$

  is the least fixpoint of the function

$$\lambda y, z : f(x, z)$$

  and, in other words, the least solution of the equation

$$(y, z) = f(x, z)$$

- **(Block) Identity:** $\mathsf{I}_a \in \mathsf{SPF}(M)(a,a)$ is defined by

$$\mathsf{I}_a(x) = x, \quad \text{for all } x \in M_a$$

- **(Block) Transposition:** ${}^a\mathsf{X}^b \in \mathsf{SPF}(M)(a+b, b+a)$ is defined by

$${}^a\mathsf{X}^b(x,y) = (y,x), \quad \text{for } x \in M_a \text{ and } y \in M_b$$

Now we look at the meaning of the various branching constants. In the case of deterministic stream processing functions the meaning of the ramification constants $\wedge$ and $\perp$ is more or less standard: $\wedge$ is the copy constant $\pitchfork$ and $\perp$ is the (rich) sink constant $\multimapinv$. They are defined as follows:

- **(Block) Copy:** $\pitchfork^a \in \mathsf{SPF}(M)(a, a+a)$ defined by

$$\pitchfork^a(x) = (x,x), \quad \text{for } x \in M_a$$

- **(Block rich) Sink:** $\multimapinv^a \in \mathsf{SPF}(M)(a, 0)$ defined by

$$\multimapinv^a(x) = (\,), \quad \text{for } x \in M_a$$

  where $(\,)$ denotes the empty tuple of streams.

The constant $\top$ may be interpreted as a dummy source $\multimapdot$, defined as follows:

[3] f is continuous, hence this definition is well formed.

- **(Block dummy) Source:** $\Uparrow_a \in \mathsf{SPF}(M)(0,a)$ defined by

$$\Uparrow_a(\,) = (\langle\,\rangle_a)$$

where $\langle\,\rangle_a$ is the a-tuple of empty streams $\langle\,\rangle$.

Finally, the constant $\vee$ is usually left *uninterpreted* in this case of deterministic dataflow networks. Its standard meaning in asynchronous dataflow is as the nondeterministic "merge" constant.[4]

3.1 Graph Isomorphism

With the operators introduced above, $\mathsf{SPF}(M)$ becomes a heterogenous algebra. This algebra fulfills the axioms of BNA.

Theorem 3.1 *(Graph isomorphism)*
$(\mathsf{SPF}(M), +\!\!+, \cdot, \uparrow, \mathsf{I}_a, {}^a\mathsf{X}^b)$ *is a BNA model.*

Theorem 3.2 *(Graph isomorphism with constants $\curlywedge, \lrcorner, \Uparrow$)*
$(\mathsf{SPF}(M), +\!\!+, \cdot, \uparrow, \mathsf{I}_a, {}^a\mathsf{X}^b, \curlywedge, \lrcorner, \Uparrow)$ *obeys the following additional axioms*

A5–A9, A12–A13, A16–A19 and F4

in table 3 with $\curlywedge, \lrcorner, \Uparrow$ instead of $\wedge, \perp, \top$, respectively.
In the terminology of [Ste94] this means that SPF *is a dβ-ssmc with feedback.*

The proof of these theorems is given in detail in section 5. The main ideas are presented below.

Sketch of proofs: It is easy to see that all the axioms apart from the feedback equations hold. Actually, $\mathsf{SPF}(M)$ is a subtheory of the algebraic theory Pow(M) of all the functions on M (defined in [TWW79], for instance), hence it is a well known fact that the axioms B1–B10, A5–A8, A16–A19, and S3–S4 in table 3 are valid. In addition, A9 and A12–A13 clearly hold.

It remains to be shown that the axioms involving the feedback operation (R1–R5 and F1–F2 and F4 in table 3) are valid. The proofs are fairly easy. The most "difficult" proof is that of axiom R5, which shows that a simultaneous multiple feedback is equivalent to repeated unary feedbacks. $\Box$

[4] However, in [BS94] an "equality test" meaning is assigned to the $\vee$ constant as a "dual" version of the copy constant.

3.2 Input-Output Behavior

The next step is to take into account the coarser equivalences on deterministic networks which identify the networks that have the same tree unfolding or compute the same input-output function. For deterministic dataflow networks both ways give the same equivalence.

First we note that the strong axioms S3–S4 of table 3 hold in SPF, hence SPF is an algebraic theory. This means that each multiple-output function is a tuple of one-output functions. Consequently, we may suppose each cell in a network and the network itself have exactly one output.

Next, a network as above (with one output and each cell with one output, too) may be unfolded towards inputs into a tree. The unfolding of a multiple output network is the tuple of the unfoldings corresponding to each output.

We say two deterministic dataflow networks F and G are *unfolding equivalent*, and write $F \equiv_{unfold} G$, iff F and G unfold into the same tuple of trees.

On the other hand, we say F and G are *input-output equivalent with respect to functional interpretations of the atoms*, and write $F \equiv_{IO} G$, iff for all interpretations of the atoms as stream processing functions the networks compute the same function.

Lemma 3.3 *(unfolding correctness)*
If $F' \equiv_{unfold} F''$, then $F' \equiv_{IO} F''$.

Proof: By a general result (see example 4.2.4 in [Ste94]), the strong axioms S3–S4 together with the graph isomorphism axioms imply the fixpoint equation. As an example for $f : a + b \to b$ we obtain:

$$(f \cdot \wedge^b) \uparrow^b = \wedge^a \cdot (\mathsf{I}_a +\!\!+ (f \cdot \wedge^b) \uparrow^b) \cdot f$$

This and the continuity assumption imply that the unfolding process is correct. $\square$

Theorem 3.4 *[Ste87b, Ste94](Axiomatizing the unfolding equivalence)*
The axioms of $d\alpha$-flow in table 3 are correct and complete for deterministic dataflow networks modulo unfolding equivalence.

(Recall that this means: the graph isomorphism axioms in t heorem 3.2, B1–B10, A5–A9, A12–A13, A16–A19, R1–R5, F1–F2, and F4, the strong axioms S3–S4, and the enzymatic axiom for converses of functions, i.e. $ENZ_{Fn^{-1}}$.)

Using this theorem we may restate lemma 3.3 above in a formal way, since unfolding correctness follows from the validity of the $d\alpha$-flow axioms in SPF. The axioms may be easily verified. The following lemma states the validity of the enzymatic rule.

Lemma 3.5 *$ENZ_{Fn^{-1}}$ holds in SPF.*

The proof is given in section 5.

Example 3.6 The proof of the above theorem may be illustrated with the help of the dataflow networks in Figure 1. One may easily see that the dataflow networks (a), (b), and (c) have the same unfolding. Actually, the unfolding is the pair (t,t), where t is the tree (d), provided that f_1 and f_2 in (d) denote the first and the second output component of f in $(a), (b)$, and (c).

Let us see how we may prove their equality in the axiomatic system given by the $d\alpha$-flow axioms. This axiomatic system has the graph isomorphism axioms and two new ingredients: the critical axioms S3–S4 and the invariance/enzymatic axiom $\mathrm{ENZ}_{Fn^{-1}}$, for the class E of terms ("enzymes") specified using the branching constants $\overset{\circ}{\wedge}$ and ⫰ and the BNA signature.

Of these new axioms, $\mathrm{ENZ}_{Fn^{-1}}$ is, by far, the most complicated. It may be explained using the representation of the networks as system of equations. For example, the functions computed by the network in (a), (b), (c) are given by the least fixpoint solutions corresponding to y_1 and y_2 in the following systems (S1), (S2), (S3), respectively:

> var:: $x_1 : in$; $\quad y_1, y_2 : out$; $\quad u, v, z, t : local$ in
>
> $$\begin{array}{rclcrclcrcl} v & = & y_1 & \quad & f_1(x_1,t) & = & y_1 & \quad & f_1(x_1,w) & = & y_1 \\ f_1(x_1,z) & = & y_2 & & f_1(x_1,z) & = & y_2 & & f_1(x_1,w) & = & y_2 \\ f_1(x_1,t) & = & v & & f_2(x_1,t) & = & z & & f_2(x_1,w) & = & w \\ f_2(x_1,t) & = & z & & f_2(x_1,z) & = & t & & & & \\ f_2(x_1,z) & = & t & & & & & & & & \\ f'(v,u) & = & u & & & & & & & & \\ \text{(S1)} & & & & \text{(S2)} & & & & \text{(S3)} & & \end{array}$$
>
> where f_1, f_2 are the components in which f can be decomposed such that $f_1 = f \cdot (\mathsf{I}_1 \mathbin{+\!\!+} \text{⫰}^1)$ and $f_2 = f \cdot (\text{⫰}^1 \mathbin{+\!\!+} \mathsf{I}_1)$.

In such a system, the invariance axiom applied for the relations generated by the ⫰ constant allows to delete some equations of the system, provided they define variables that are not used in the generation of the output. In the running example u and the corresponding equation may be deleted. Formally, if we write the left-hand side terms of the system as the tuple

$$F_1 := (v, f_1(x_1,z), f_1(x_1,t), f_2(x_1,t), f_2(x_1,z), f'(v,u))$$

then, by axiom S3,

$$\begin{array}{rcl} F_1 \cdot [\mathsf{I}_2 \mathbin{+\!\!+} (\mathsf{I}_3 \mathbin{+\!\!+} \text{⫰}^1)] & = & (v, f_1(x_1,z), f_1(x_1,t), f_2(x_1,t), f_2(x_1,z), \text{⫰}^1) \\ & = & [\mathsf{I}_1 \mathbin{+\!\!+} (\mathsf{I}_3 \mathbin{+\!\!+} \text{⫰}^1)] \cdot F' \end{array}$$

where F' is the tuple

$$F' := (v, f_1(x_1,z), f_1(x_1,t), f_2(x_1,t), f_2(x_1,z))$$

Due to graph isomorphism transformations the resulting system specified by F' and the variables (y_1, y_2, v, z, t) is equivalent to the system in (S2) specified by the variables (y_1, y_2, z, t) and the tuple

$$F_2 := (f_1(x_1,t), f_1(x_1,z), f_2(x_1,t), f_2(x_1,z))$$

A bit more complicated is the invariance of the relations induced by the $\curlywedge$ constant. In this case, we may identify certain variables such that the terms in the corresponding equations become equal after the identification. In the running example, we may identify z and t since after identification both terms $f_2(x_1, t)$ and $f_2(x_1, z)$ are equal. Formally, if w is a new variable, then by axiom S4 we get

$$[\mathsf{I}_1 \mathbin{+\!\!+} \curlywedge^1] \cdot F2 = (f_1(x_1, w), f_1(x_1, w), f_2(x_1, w), f_2(x_1, w)) =_{S4} F_3 \cdot [\mathsf{I}_2 \mathbin{+\!\!+} \curlywedge^1]$$

where

$$F_3 := (f_1(x_1, w), f_1(x_1, w), f_2(x_1, w))$$

The resulting system specified by F_3 and the variables (y_1, y_1, w) is shown in (S3).

Finally, we observe that each system may be minimized using such transformations and that the minimal systems are in bijective correspondence with the unfolding trees. $\square$

Next, we show that two deterministic dataflow networks unfold into the same tuple of trees if and only if they compute the same input-output function for all functional interpretations of the atomic cells.[5]

Theorem 3.7 *(unfolding equivalence = input-output equivalence)*
$F' \equiv_{unfold} F''$ iff $F' \equiv_{IO} F''$ for all functional interpretations of the atoms.

Proof: "$\Rightarrow$" Already proved (unfolding correctness).

"$\Leftarrow$" We show the validity of the equivalent statement: If F' and F'' are different trees, then there exists a functional interpretation of the atoms such that F' and F'' compute different functions.

We use a domain of data D consisting of partial Σ-terms over X ("partial" means that terms $\sigma(x_1, \ldots, x_n)$ with some undefined arguments are allowed; such undefined elements are denoted by "?"), where

- X is an infinite set of variables and
- Σ is a signature containing a symbol $\sigma_f : m \to 1$ for each atom $f : m \to 1$ which occurs either in F' or F''.

The interpretation is:

Case $m \geq 1$: A cell $f : m \to 1$ acts by:

$$f(x) = \sigma_f(?, \ldots, ?)^\frown g(x)$$
$$g(t_1^\frown x_1, \ldots, t_m^\frown x_m) = \sigma_f(t_1, \ldots, t_m)^\frown g(x_1, \ldots, x_m)$$

Case $m = 0$: A cell $f : 0 \to 1$ produces the output $(\sigma_f)^\infty$, or formally

$$f(\,) = \sigma_f(\,)^\frown f(\,)$$

[5] This is not the case for the flowchart interpretation of flowgraphs. In that case the unfolding equivalence does not coincide with the input-output equivalence and has to be combined with the reductions of the subtrees without outputs to the empty tree, see section 10 of [Ste94].

Take a distinguished variable x_i for each input i and consider as input the tuple of streams

$$((x_1)^\infty, \ldots, (x_n)^\infty)$$

The output

$$|F|((x_1)^\infty, \ldots, (x_n)^\infty)$$

produced by a tree $F : n \to 1$ is a stream of terms

$$t_1^\frown t_2^\frown \ldots$$

where t_i is the partial approximation of F up to level i.

Since F' and F'' are different, there is a level i such that they are different up to level i, hence

$$|F'|((x_1)^\infty, \ldots, (x_n)^\infty) \neq |F''|((x_1)^\infty, \ldots, (x_n)^\infty)$$

and the implication is proved. $\square$

We illustrate the proof by an example.

Example 3.8 The idea of the difficult part in the proof above may be illustrated by the tree in Figure 1(d) as follows. Under the interpretation displayed, the output computed by the tree-network is

$$g(?,?)^\frown g(x_1, h(?,?))^\frown g(x_1, h(x_1, h(?,?)))^\frown g(x_1, h(x_1, h(x_1, h(?,?))))^\frown \ldots$$

One may see that the first output gives the approximation of the tree up to level 1, the second up to level 2, and so on. $\square$

Corollary 3.9 *(Axiomatization of the input-output behavior in the deterministic case)*
The $d\alpha$-flow axioms give a correct and complete axiomatization for the stream processing functions obtained as interpretations of deterministic dataflow networks.

Following [Sta92], we say a stream processing function is a $\{\wedge, \perp, \top\}$-*buffering morphism* if it is specified by a dataflow network built up with the BNA operations and constants and

- constants the copy $\wedge$, sink $\perp$, and source $\top$,
- the trivial cells $s_d \in \mathsf{SPF}(0, s)$ for $d \in M_s,\ s \in S$ defined by $s_d(\ ) = d$.

Corollary 3.10 *The $d\alpha$-flow axioms give a complete axiomatization for the buffering functions.*

For the particular class of stream processing functions in the previous corollary it is possible to obtain a stronger axiomatization result, similar to the one in in [Sta92], where the enzymatic axiom is replaced by the following equational scheme:

$$(x^i \cdot \wedge) \uparrow = (x^j \cdot \wedge) \uparrow$$

for $i, j > 0$.

II. Axioms for the additional constants $\top, \bot, \wedge$ (without feedback)

A5	$\wedge^a \cdot (\wedge^a +\!\!+ \mathsf{I}_a) = \wedge^a \cdot (\mathsf{I}_a +\!\!+ \wedge^a)$
A6	$\wedge^a \cdot {}^a\mathsf{X}^a = \wedge^a$
A7	$\wedge^a \cdot (\bot^a +\!\!+ \mathsf{I}_a) = \mathsf{I}_a$
A8	$\top_a \cdot \wedge^a = \top_a +\!\!+ \top_a$
A9	$\top_a \cdot \bot^a = \mathsf{I}_0$

A12	$\top_0 = \mathsf{I}_0$	A13	$\top_{a+b} = \top_a +\!\!+ \top_b$
A16	$\bot^0 = \mathsf{I}_0$	A17	$\bot^{a+b} = \bot^a +\!\!+ \bot^b$
A18	$\wedge^0 = \mathsf{I}_0$	A19	$\wedge^{a+b} = (\wedge^a +\!\!+ \wedge^b) \cdot (\mathsf{I}_a +\!\!+ {}^a\mathsf{X}^b +\!\!+ \mathsf{I}_b)$

IV. Axioms for the action of feedback on the branching constants

F4 $\quad \wedge^a \uparrow^a = \top_a$

IV. The strong axioms ($f : a \to b$)

S3 $\quad f \cdot \bot^b = \bot^a$

S4 $\quad f \cdot \wedge^b = \wedge^a \cdot (f +\!\!+ f)$

VI. The enzymatic rule

$\mathrm{ENZ}_{Fn^{-1}}$: $\quad f \cdot (\mathsf{I}_b +\!\!+ y) = (\mathsf{I}_a +\!\!+ y) \cdot g$ implies $f \uparrow^c = g \uparrow^d$,

where $y : c \to d$ is a term written with
$+\!\!+\,, \cdot, \mathsf{I}, \mathsf{X}$ and some constants in $\bot, \wedge$
and $f : a + c \to b + c,\ g : a + d \to b + d$ are arbitrary

Table 1: The axiomatization of deterministic dataflow networks

4 Nondeterministic Networks

Deterministic data flow nets have more or less a canonical denotational semantics, which was used in the previous chapter. To find such semantics for nondeterministic networks is less obvious. The semantics of nondeterministic dataflow networks may be reduced to the semantics of deterministic networks using oracles. Such an oracle fixes a priori the behaviour of the network regarding the nondeterministic points. Given a fixed oracle, a nondeterministic network becomes deterministic and it computes a stream processing function. Varying the oracle we obtain the semantics of a nondeterministic network as a set of stream processing functions. Formally, we construct the model $\mathcal{P}\mathsf{SPF}(M)$ for the interpretation of nondeterministic dataflow networks as follows.

First, for streams $a, b \in S^*$ define

$$\mathcal{P}\mathsf{SPF}(M)(a,b) := \{F \mid F \subseteq \mathsf{SPF}(a,b)\}$$

Next, the operations $+\!\!+\,, \cdot, \uparrow$ are defined in an elementwise manner by

$$\begin{aligned} F +\!\!+ G &= \{f +\!\!+ g \mid f \in F, g \in G\} \\ F \cdot G &= \{f \cdot g \mid f \in F, g \in G\} \\ F\uparrow &= \{f\uparrow \mid f \in F\} \end{aligned}$$

Then, each constant $c \in \{\mathsf{I}, \mathsf{X}, \curlywedge, \curlyvee, \bullet\}$ of SPF is interpreted as a corresponding constant $\{c\}$ of $\mathcal{P}\mathsf{SPF}$.

In this model, we may give meaning to additional nondeterministic branching constants, namely:

- **(Block) Split:** for $a \in S$

  $$\curlywedge^a = \{\ \underset{\phi}{\curlywedge^a} \ \mid \phi : \omega \to \{1,2\}\}$$

  where for an oracle ϕ, $\underset{\phi}{\curlywedge^a}\ (x) =_{def} (y, z)$, with y and z obtained by splitting x according to ϕ. That is, if $\phi(i) = 1$ then the i-th input is delivered on output channel 1, otherwise on output channel 2. This definition is extended to arbitrary words $a \in S^*$ using the identities in A18–A19.[6]

- **(Block) Merge[7]:** for $a \in S$

  $$\curlyvee_a = \{\ \overset{\phi}{\curlyvee_a} \ \mid \phi : \omega \to \{1,2\}\}$$

  where for an oracle ϕ, $\overset{\phi}{\curlyvee_a}\ (x, y) = z$ with z obtained from x and y according to ϕ. With A14–A15 this definition is extended to arbitrary $a \in S^*$.

[6] Notice that we have *independent oracles* for each input channel in a and *not* a unique one for all the inputs in a. With a definition that uses the later version the axiom A19 would fail.

[7] We define here a merge that is neither nonstrict nor fair. The treatment of a fair nonstrict merge needs a more sophisticated semantic model (see [Bro93]).

- **(Block rich) Source:** for $a \in S^*$

$$\mathring{\top}_a = \{g_x \mid x \in M_a\}$$

where for $x \in M_a$, $\quad g_x : 0 \to a$ is the function given by $g_x(\ ) = x$.

By split, merge, and source we have introduced three nondeterministic constants for data flow nodes.

4.1 Graph Isomorphism

As it is well-known, for nondeterministic terms certain classic algebraic equations do not hold such as the fixed-point equation. Nevertheless all equations characterizing graph isomorphismes hold, of course.

Theorem 4.1 *(Graph isomorphism)*
$(\mathcal{P}\mathsf{SPF}, +\!\!+, \cdot, \uparrow, \mathsf{I}, \mathsf{X})$ *is a BNA model.*

Proof: The proof follows directly from the corresponding result in the deterministic case (theorem 3.1). The key point is the observation that all the BNA axioms[8] are identities with both the left-hand side and the right-hand side terms containing *at most one occurrence* of a variable and each variable that occurs in one part of an identity occurs in the other part, as well. Hence the validity of the proof of a BNA axiom in $\mathcal{P}\mathsf{SPF}$ may be checked on elements and it is reduced to the validity of the corresponding axiom in SPF. $\square$

To these axioms for graph isomorphism we can add equations for the constants.

Theorem 4.2 *(Graph isomorphism with various constants)*
$(\mathcal{P}\mathsf{SPF}, +\!\!+, \cdot, \uparrow, \mathsf{I}, \mathsf{X}, \curlywedge, \dot\bot, \curlyvee, \dot\top)$ *obeys the additional axioms*

A1–A2, A4–A6, A8–A9, A12–A19 and F3–F4

in table 3 where $\curlywedge, \dot\bot, \curlyvee, \dot\top$ *replace* $\wedge, \bot, \vee, \top$*, respectively.*

Notice that, for the remaining axioms, only one inclusion holds, i.e. "$\subseteq$" for A3, A7, A11 and F5 and "$\supseteq$" for A10.

The details of the proof may be found in section 5. Since axioms A1–A2 and A5–A6 are valid, the oracle based semantics of the nondeterminism is associative and commutative. Hence we may equivalently use the extended branching constants

$\underset{\phi}{\curlywedge}{}^a_k \ : a \to ka$ and $\overset{\phi}{\curlyvee}{}^k_a \ : ka \to a$ for $k \geq 1$, where $\phi : \omega \to \{1, \ldots, k\}$ is a k-oracle. On the other hand, axioms A3 and A7 do not hold, hence we have a not fair merge and therefore a nonangelic calculus of relations.

[8] Recall, the BNA axioms are B1–B10, R1–R5 and F1–F2 in table 3.

II'. Axioms for the additional constants $\overset{\bullet}{\shortmid}, \underset{\circ}{\shortmid}, \underset{\bullet}{\vee}, \overset{\bullet}{\wedge}$ (without feedback)

A1 $(\underset{\bullet}{\vee}_a \mathbin{+\!\!+} \mathsf{I}_a) \cdot \underset{\bullet}{\vee}_a = (\mathsf{I}_a \mathbin{+\!\!+} \underset{\bullet}{\vee}_a) \cdot \underset{\bullet}{\vee}_a$ A2 ${}^{a}\mathsf{X}^{a} \cdot \underset{\bullet}{\vee}_a = \underset{\bullet}{\vee}_a$
A3^o $(\overset{\bullet}{\shortmid}_a \mathbin{+\!\!+} \mathsf{I}_a) \cdot \underset{\bullet}{\vee}_a \supset \mathsf{I}_a$ A4 $\underset{\bullet}{\vee}_a \cdot \underset{\circ}{\shortmid}^a = \underset{\circ}{\shortmid}^a \mathbin{+\!\!+} \underset{\circ}{\shortmid}^a$

A5 $\overset{\bullet}{\wedge}^a \cdot (\overset{\bullet}{\wedge}^a \mathbin{+\!\!+} \mathsf{I}_a) = \overset{\bullet}{\wedge}^a \cdot (\mathsf{I}_a \mathbin{+\!\!+} \overset{\bullet}{\wedge}^a)$ A6 $\overset{\bullet}{\wedge}^a \cdot {}^{a}\mathsf{X}^{a} = \overset{\bullet}{\wedge}^a$
A7^o $\overset{\bullet}{\wedge}^a \cdot (\underset{\circ}{\shortmid}^a \mathbin{+\!\!+} \mathsf{I}_a) \supset \mathsf{I}_a$ A8 $\overset{\bullet}{\shortmid}_a \cdot \overset{\bullet}{\wedge}^a = \overset{\bullet}{\shortmid}_a \mathbin{+\!\!+} \overset{\bullet}{\shortmid}_a$

A9 $\overset{\bullet}{\shortmid}_a \cdot \underset{\circ}{\shortmid}^a = \mathsf{I}_0$
A10^o $\underset{\bullet}{\vee}_a \cdot \overset{\bullet}{\wedge}^a \subset (\overset{\bullet}{\wedge}^a \mathbin{+\!\!+} \overset{\bullet}{\wedge}^a) \cdot (\mathsf{I}_a \mathbin{+\!\!+} {}^{a}\mathsf{X}^{a} \mathbin{+\!\!+} \mathsf{I}_a) \cdot (\underset{\bullet}{\vee}_a \mathbin{+\!\!+} \underset{\bullet}{\vee}_a)$
A11^o $\overset{\bullet}{\wedge}^a \cdot \underset{\bullet}{\vee}_a \supset \mathsf{I}_a$

A12 $\overset{\bullet}{\shortmid}_0 = \mathsf{I}_0$ A13 $\overset{\bullet}{\shortmid}_{a+b} = \overset{\bullet}{\shortmid}_a \mathbin{+\!\!+} \overset{\bullet}{\shortmid}_b$
A14 $\underset{\bullet}{\vee}_0 = \mathsf{I}_0$ A15 $\underset{\bullet}{\vee}_{a+b} = (\mathsf{I}_a \mathbin{+\!\!+} {}^{b}\mathsf{X}^{a} \mathbin{+\!\!+} \mathsf{I}_b) \cdot (\underset{\bullet}{\vee}_a \mathbin{+\!\!+} \underset{\bullet}{\vee}_b)$

A16 $\underset{\circ}{\shortmid}^0 = \mathsf{I}_0$ A17 $\underset{\circ}{\shortmid}^{a+b} = \underset{\circ}{\shortmid}^a \mathbin{+\!\!+} \underset{\circ}{\shortmid}^b$
A18 $\overset{\bullet}{\wedge}^0 = \mathsf{I}_0$ A19 $\overset{\bullet}{\wedge}^{a+b} = (\overset{\bullet}{\wedge}^a \mathbin{+\!\!+} \overset{\bullet}{\wedge}^b) \cdot (\mathsf{I}_a \mathbin{+\!\!+} {}^{a}\mathsf{X}^{b} \mathbin{+\!\!+} \mathsf{I}_b)$

IV'. Axioms for the action of feedback on the branching constants

F3 $\underset{\bullet}{\vee}_a \uparrow^a = \underset{\circ}{\shortmid}^a$ F4 $\overset{\bullet}{\wedge}^a \uparrow^a = \overset{\bullet}{\shortmid}_a$
F5^o $[(\mathsf{I}_a \mathbin{+\!\!+} \overset{\bullet}{\wedge}^a) \cdot ({}^{a}\mathsf{X}^{a} \mathbin{+\!\!+} \mathsf{I}_a) \cdot (\mathsf{I}_a \mathbin{+\!\!+} \underset{\bullet}{\vee}_a)] \uparrow^a \supset \mathsf{I}_a$

Table 2: Axioms satisfied by the split-merge interpretation of the branching constants in $\mathcal{P}\mathsf{SPF}$

4.2 The Input-Output Behaviour

In this section we look to the axiomatization problem for the input-output behaviour of nondeterministic dataflow networks.

It is easy to see that neither the strong axioms S1–S2 nor S3–S4 in table 3 hold. Similarly, due to the nondeterministic behaviour of the cells the fixpoint identity is not valid, hence *the unfolding of networks is not a correct rule.* All these comments amount to say that algebraic or iteration theories cannot be used in this setting.

These observations lead towards a counterexample to a thesis[9] of Bloom and Esik.

> Bloom and Esik's thesis (see [BE88], for instance):
>
> *Whenever an iterative process is present an iteration theory structure may be found.*

There are many examples which were studied in full detail by Bloom and Esik showing that this is the case when one tries to capture the iteration laws in combination with the algebraic theory laws. On the other hand, $\mathcal{P}\mathsf{SPF}$ provides an example of a natural iterative process which is neither an algebraic theory nor a dual algebraic theory (i.e., neither S1–S2 nor S3–S4 of table 3 hold). Since an iteration theory is an algebraic theory we get the following result.

Corollary 4.3 *Bloom's and Esik's thesis is false.*

By contrast, the flownomial calculus starts with an axiomatization of the *iteration operation combined with the monoidal category primitives* rather than with the algebraic theory primitives. This is the key reason for the successful application of the flownomial calculus to the case of nondeterministic dataflow networks, as it has been presented in the previous subsection.

One may perhaps suggest to replace Bloom and Esik thesis above by the following weaker one:

> *Whenever an iterative process is present the BNA laws hold, such that an aα-flow algebra may be found.*

Since the BNA laws are correct and complete for graphs modulo graph isomorphism, this is true whenever one correctly has a graphical description of the underlined iterative process.

The problem of axiomatizing the input-output behaviour of nondeterministic dataflow networks is still open. We do not have a result similar to corollary 3.9. To be more precise, let us define a $\{\cdot, \cdot, \cdot, \cdot, \cdot, \cdot\}$- *buffering morphism* as a set of stream processing functions specified by a dataflow network built up with

[9] Strictly speaking, this is a thesis and not a conjecture since it states that the informal notion of an iterative process is captured by the formal definition of iteration theories.

- the split $\wedge$, merge $\vee$, (rich) sink $\downarrow$, (dummy) source $\uparrow$, (rich) source $\Uparrow$ and copy $\pitchfork$ constants
- the trivial cells $s_d \in \mathsf{SPF}(0,s)$ for $d \in M_s (s \in S)$ defined by

  $s_d(\ ) = d.$

Let R be a subset of branching constants in $\{\wedge, \vee, \downarrow, \uparrow, \Uparrow, \pitchfork\}$. We are interested in the following problems for an arbitrary R and either for arbitrary networks or for acyclic networks, only.

- **Expressivness:** Characterize the R-buffering morphisms.

  (Certain invariants and/or complexity measures may be useful to classify the equivalent networks.)

- **Decidability:** Check the decidability of the equality problem for various sets R of branching constants.
- **Axiomatization:** Give complete (and correct) axiomatizations for the R-buffering morphisms.

In general, the above problem is open although we have certain partial results. (See corollary 3.10, for example.) To be more specific, the problem is open for subsets R which contains both the split and merge constants.

These problems are particularly useful for modelling communication networks like INTERNET. One may see that such a communication network is asynchronous and nondeterministic, it contains split and merge vertices. It is mainly used to broadcast data so that no computing cells are used, except for trivial buffering components.

5 Proofs of the Graph Isomorphism Theorems

In this section we give the detailed proofs of our theorems.

Lemma 5.1 *(Axiom R5 of table 3)*

$$(f \uparrow^{p+q}) = (f \uparrow^{q}) \uparrow^{p}$$

for $f \in \mathsf{SPF}(M)(m+p+q,\ n+p+q)$. *Hence one application of a multiple feedback may be replaced by repetitive applications of unary feedbacks.*

Proof: Let $f \in \mathsf{SPF}(M)(m+p+q, n+p+q)$. Then:

- $f \uparrow^{p+q} \in \mathsf{SPF}(M)(m,n)$ is defined by

  $$(f \uparrow^{p+q})(x) = y$$

  where $y = \sqcup_{k \geq 1} y^k$ and y^k, z^k, w^k are inductively defined by

  $$(y^k, z^k, w^k) = f(x, z^{k-1}, w^{k-1}) \quad \text{for } k \geq 1$$

where $z^0 = \langle\,\rangle, w^0 = \langle\,\rangle$.

Denote

$z := \sqcup_{k\geq 1} z^k$

$w := \sqcup_{k\geq 1} w^k$

- $(f\uparrow^q)\uparrow^p \in \mathsf{SPF}(M)(m,n)$ is defined as follows:

$$((f\uparrow^q)\uparrow^p)(x) = \overline{y}$$

where $\overline{y} = \sqcup_{i\geq 1}\overline{y}^i$ and $\overline{y}^i, \overline{z}^i$ are inductively defined by

$$(\overline{y}^i, \overline{z}^i) = (f\uparrow^q)(x, \overline{z}^{i-1}) \text{ for } i \geq 1$$

where $\overline{z}^0 = \langle\,\rangle$, hence by the definition of $f\uparrow^q$ there are elements $\tilde{y}^{i,j}, \tilde{z}^{i,j}, \tilde{w}^{i,j}$ for $i,j \geq 1$ such that for all $i \geq 1$:

$$\overline{y}^i = \sqcup_{j\geq 1}\tilde{y}^{i,j}, \quad \overline{z}^i = \sqcup_{j\geq 1}\tilde{z}^{i,j} \quad \text{and}$$

$$(\tilde{y}^{i,j}, \tilde{z}^{i,j}, \tilde{w}^{i,j}) = f(x, \overline{z}^{i-1}, \tilde{w}^{i,j-1}) \quad \text{for } j \geq 1$$

where $\tilde{w}^{i,0} = \langle\,\rangle$.

It is obvious that each sequence $(\tilde{y}^{i,j})_{i,j}, (\tilde{z}^{i,j})_{i,j}$, and $(\tilde{w}^{i,j})_{i,j}$ is increasing on both indices i,j, hence the following notation makes sense:

$\overline{z} := \sqcup_{i\geq 1}\overline{z}^i$ and

$\overline{w} := \sqcup_{i\geq 1}\overline{w}^i$, where for $i \geq 1$: $\overline{w}^i := \sqcup_{j\geq 1}\tilde{w}^{i,j}$.

Proof of $f\uparrow^{p+q} = (f\uparrow^q)\uparrow^p$:

A) $f\uparrow^{p+q} \sqsubseteq (f\uparrow^q)\uparrow^p$

First note that

$$(y^k, z^k, w^k) \sqsubseteq (\tilde{y}^{k,k}, \tilde{z}^{k,k}, \tilde{w}^{k,k}), \quad \forall k \geq 1$$

Indeed, for $k = 1$ it follows by

$$\begin{aligned}
(y^1, z^1, w^1) &= f(x, z^0, w^0) \\
&= f(x, \langle\,\rangle, \langle\,\rangle) \\
&= f(x, \overline{z}^0, \tilde{w}^{1,0}) \\
&= (\tilde{y}^{1,1}, \tilde{z}^{1,1}, \tilde{w}^{1,1})
\end{aligned}$$

and if it holds for k, then it holds for $k+1$ by:

$$\begin{aligned}
(y^{k+1}, z^{k+1}, w^{k+1}) &= f(x, z^k, w^k) \\
&\sqsubseteq f(x, \tilde{z}^{k,k}, \tilde{w}^{k,k}) \\
&\sqsubseteq f(x, \overline{z}^k, \tilde{w}^{k,k}) \\
&\sqsubseteq f(x, \overline{z}^k, \tilde{w}^{k+1,k}) \\
&= (\tilde{y}^{k+1,k+1}, \tilde{z}^{k+1,k+1}, \tilde{w}^{k+1,k+1})
\end{aligned}$$

By this we get

$$\begin{array}{rcl}(f\uparrow^{p+q})(x) & = & y \\ & = & \sqcup_{k\geq 1} y^k \\ & \sqsubseteq & \sqcup_{k\geq 1} \tilde{y}^{k,k} \\ & = & \sqcup_{i\geq 1} \sqcup_{j\geq 1} \tilde{y}^{i,j} \\ & = & \sqcup_{i\geq 1} \overline{y}^i \\ & = & \overline{y} \\ & = & ((f\uparrow^q)\uparrow^p)(x)\end{array}$$

B) $(f\uparrow^q)\uparrow^p \sqsubseteq f\uparrow^{p+q}$

We prove by a double induction that

$$(\tilde{y}^{i,j}, \tilde{z}^{i,j}, \tilde{w}^{i,j}) \sqsubseteq (y,z,w), \quad \forall i,j \geq 1$$

First note that

$$(y,z,w) = f(x,z,w)$$

Indeed,

$$\begin{array}{rcl}f(x,z,w) & = & f(x, \sqcup_{k\geq 1} z^k, \sqcup_{k'\geq 1} w^{k'}) \\ & = & \sqcup_{k\geq 1} f(x, z^k, w^k) \\ & = & \sqcup_{k\geq 1} (y^{k+1}, z^{k+1}, w^{k+1}) \\ & = & (y,z,w)\end{array}$$

For $i=1$: If $j=1$ then we have

$$\begin{array}{rcl}(\tilde{y}^{1,1}, \tilde{z}^{1,1}, \tilde{w}^{1,1}) & = & f(x, \overline{z}^0, \tilde{w}^{1,0}) \\ & = & f(x, \langle\,\rangle, \langle\,\rangle) \\ & = & (y^1, z^1, w^1) \\ & \sqsubseteq & (y,z,w)\end{array}$$

and the passing from j to $j+1$ follows by

$$\begin{array}{rcl}(\tilde{y}^{1,j+1}, \tilde{z}^{1,j+1}, \tilde{w}^{1,j+1}) & = & f(x, \overline{z}^0, \tilde{w}^{1,j}) \\ & = & f(x, \langle\,\rangle, \tilde{w}^{1,j}) \\ & \sqsubseteq & f(x,z,w) \\ & = & (y,z,w)\end{array}$$

The inductive step from i to $i+1$: If $j=1$, then

$$\begin{array}{rcl}(\tilde{y}^{i+1,1}, \tilde{z}^{i+1,1}, \tilde{w}^{i+1,1}) & = & f(x, \overline{z}^i, \tilde{w}^{i+1,0}) \\ & = & f(x, \sqcup_{j'\geq 1} \tilde{z}^{i,j'}, \langle\,\rangle) \\ & \sqsubseteq & f(x,z,w) \\ & = & (y,z,w)\end{array}$$

and the passing from j to $j+1$ is similar as in the previous case $i=1$.

$\square$

Proof: (of theorem 3.1)
The validity of the axioms without feedback B1–B10 is obvious.

R1 may be proved as follows. Let $f : a' \to a,\ g : a + c \to b + c,\ h : b \to b'$ and $x \in M_{a'}$. Then

$$[f \cdot (g \uparrow^c) \cdot h](x) = h(y)$$

where $y = \bigsqcup_k y_k$ for $y_k \in M_b,\ z_k \in M_c$ inductively defined by

$$\begin{aligned} (y_1, z_1) &= g(f(x), \langle\ \rangle_c) \\ (y_{k+1}, z_{k+1}) &= g(f(x), z_k) \quad (k \geq 1) \end{aligned}$$

On the other hand,

$$[(f +\!\!+ \mathsf{I}_c) \cdot g \cdot (h +\!\!+ \mathsf{I}_c)] \uparrow^c (x) = t$$

where $t = \bigsqcup_k t_k$ for $t_k \in M_b,\ t'_k \in M_{b'},\ w_k \in M_c$ inductively defined by

$$\begin{aligned} t_k &= h(t'_k) \quad (k \geq 1) \\ (t'_1, w_1) &= g(f(x), \langle\ \rangle_c)) \\ (t'_{k+1}, w_{k+1}) &= g(f(x), w_k)) \quad (k \geq 1) \end{aligned}$$

By induction it follows that $h(y_k) = t_k$ and $z_k = w_k$ for all k. Hence $h(y) = t$, i.e. R1 is valid. R2 may be proved in a similar way.

For R3 take $x \in M_a$. Then

$$[f \cdot (\mathsf{I}_b +\!\!+ g)] \uparrow^c (x) = y$$

where $y = \bigsqcup_k y_k$ for $y_k \in M_b,\ z'_k \in M_d,\ z_k \in M_c$ inductively defined by

$$\begin{aligned} z_k &= g(z'_k) \quad (k \geq 1) \\ (y_1, z'_1) &= f(x, \langle\ \rangle_c) \\ (y_{k+1}, z'_{k+1}) &= f(x, z_k) \quad (k \geq 1) \end{aligned}$$

and

$$[(\mathsf{I}_a +\!\!+ g) \cdot f] \uparrow^d (x) = t$$

where $t = \bigsqcup_k t_k$ for $t_k \in M_b,\ w_k \in M_d$ inductively defined by

$$\begin{aligned} (t_1, w_1) &= f(x, g(\langle\ \rangle_d)) \\ (t_{k+1}, w_{k+1}) &= f(x, g(w_k)) \quad (k \geq 1) \end{aligned}$$

Since $\langle\ \rangle_c \sqsubseteq g(\langle\ \rangle_d)$ we get $(y_1, z'_1) \sqsubseteq (t_1, w_1)$. This implies $z_1 = g(z'_1) \sqsubseteq g(w_1)$, hence $(y_2, z'_2) \sqsubseteq (t_2, w_2)$ and so on. This proves one inclusion

$$y = \bigsqcup_k y_k \sqsubseteq \bigsqcup_k t_k = t$$

For the opposite inclusion, first note that $z_1 = g(z_1') \sqsupseteq g(\langle\ \rangle_d)$, hence $(y_2, z_2') \sqsupseteq (t_1, w_1)$. This implies $z_2 = g(z_2') \sqsupseteq g(w_1)$, hence $(y_3, z_3') \sqsupseteq (t_2, w_2)$ and so on. This shows that

$$y = \bigsqcup_k y_{k+1} \sqsupseteq \bigsqcup_k t_k = t$$

and R3 is proved.

R4 is obvious, R5 has beeen proved in lemma 5.1 and F1, F2 are obviously valid. $\square$

Proof: (Theorem 3.2) The validity of the axioms A5–A9, A12–A13, A16–A19, and F4 of table 3 with $\overset{\circ}{\wedge}, \underset{\circ}{|}, \overset{\bullet}{|}$ instead of $\wedge, \bot, \top$ is obvious. $\square$

Proof: (ENZ-Correctness, lemma 3.5) We have

$$f \cdot (I_b +\!\!+\, y) = (I_a +\!\!+\, y) \cdot g$$

where $y : M_c \to M_d$ is such that for each $i \in \{1, \ldots, |c|\}$ there is a j such that

$$y(z).i = z.j$$

The results $r = (f \uparrow^c).x$ and $t = (g \uparrow^d).x$ are determined by the fixpoint iterations:

$$\begin{array}{cc} (r_0, s_0) = (\langle\rangle, \langle\rangle) & (t_0, u_0) = (\langle\rangle, \langle\rangle) \\ (r_{i+1}, s_{i+1}) = f(x, s_i) & (t_{i+1}, u_{i+1}) = g(x, u_i) \end{array}$$

We obtain

$$y(s_0) = u_0 \wedge r_0 = t_0$$

Moreover, assuming $y(s_i) = u_i \wedge r_i = t_i$ we obtain

$$\begin{array}{l} (t_{i+1}, u_{i+1}) = \\ g(x, u_i) = \\ g(x, y(s_i)) = \\ (r_{i+1}, y(s_{i+1})) \quad \text{where} \quad (r_{i+1}, s_{i+1}) = f(x, s_i) \end{array}$$

This gives us all we need for an induction proof on i that shows $y(s_i) = u_i \wedge r_i = t_i$. $\square$

Proof: (Theorem 4.2) First of all, we explain the interplay between the branching constants. The meaning of $\wedge$ and $\vee$ as the split and merge constants, respectively, is taken for granted. In order to have a theory which is closed under the feedback operation, we have to see which is the result of the application of the feedback to such constants.

It is easy to see that

$$\underset{\phi}{\overset{\bullet}{\wedge}}{}^s \uparrow^s = \overset{\bullet}{|}_s$$

for all oracles ϕ, hence $\overset{\bullet}{\wedge}{}^s \uparrow^s = \overset{\bullet}{|}_s$. This equality reflects the fact that our feedback is the least fixed point solution.

For the other constant one may see that

$$\overset{\phi}{\curlyvee}_s\ \uparrow^s = \overset{|}{\circ}{}^s$$

for all oracles ϕ. (For each oracle ϕ, the merge function $\overset{\phi}{\curlyvee}_s$ is continuous, hence

$$\overset{\phi}{\curlyvee}_s\ \uparrow^s$$

is a well-defined function and has to be equal to the unique function $\overset{|}{\circ}{}^s : s \to 0$.)

All these amount to say that a set of branching constants including the split and merge constants and closed to the network algebra operations contains $\{\curlywedge, \curlyvee, \overset{|}{\circ}, \underset{|}{\bullet}\}$.

We use extended oracles $\phi : \omega \to \{1, \ldots, k\}$ for $k \geq 1$. For instance, the meaning of such an oracle in the case of the split constant $\underset{\phi}{\curlywedge}{}_k^s$ is to show the number of the output channel where the current token is sent to. Similarly for the merge constant.

Axioms A14–A15 and A18–A19 hold by definition. On the other hand, it is easy to see that A12–A13 and A16–A17 hold. Hence we may restrict ourself to the analysis of the remaining axioms in the case of single channels, i.e. $a = s \in S$.

For axiom A1 it is enough to see that both terms are equal to $\curlyvee_s^3$. Clearly,

$$(\overset{\phi'}{\curlyvee}_s \mathbin{+\!\!+} \mathsf{I}_s) \cdot \overset{\phi''}{\curlyvee}_s = \overset{\phi}{\curlyvee}{}_s^3$$

where ϕ is the 3-oracle obtained from ϕ' and ϕ'' according to the left-hand side formula. Similarly for the right-hand side term. The proof is finished showing that a 3-oracle may be simulated by 2-oracle in both ways corresponding to the left-hand side and right-hand side term of the identity, respectively.

A5 may be proved in a similar way.

For A2 and A6 it is enough to replace an oracle $\phi : \omega \to \{1, 2\}$ by the oracle $\overline{\phi}$ obtained interchanging numbers 1 and 2.

Axioms A4 holds since for all oracles ϕ one has $\overset{\phi}{\curlyvee}_s \cdot \overset{|}{\bullet}{}^s = \overset{|}{\bullet}{}^s \mathbin{+\!\!+} \overset{|}{\bullet}{}^s$.

Axiom A8 holds, too. (The splitting of an empty stream is a couple of empty streams.)

Clearly, $\underset{|}{\circ}{}_s \cdot \overset{|}{\bullet}{}^s = \mathsf{I}_0$, hence A9 is valid.

Finally, axioms F3 and F4 are valid, as we have already seen in the beginning part of the proof.

In the remaining part of the proof we show that the other axioms do not hold.

For A3, one may see that $[(\underset{|}{\bullet}{}_s \mathbin{+\!\!+} \mathsf{I}_s)\ \overset{\phi}{\curlyvee}_s\ ](x)$ is the prefix of x up to the maximal token k such that $\phi(1) = \ldots = \phi(k) = 2$. Hence A3 is not valid, but the "$\supseteq$" inclusion holds. On the other hand, it is interesting to note that varying ϕ and keeping fixed x we get the *prefix closure* of x.

For the dual axiom A7, one may see that

$$[\ \underset{\phi}{\curlywedge^s}\ (\multimap^s \mathbin{+\!\!+} \mathsf{I}_s)](x)$$

is the substream of x given by those positions k for which $\phi(k) = 2$. Hence A7 fails, but the inclusion "$\supseteq$" holds. In this case, varying ϕ and keeping fixed x we get the *substream closure* of x.

For A10 one may see that

$$E(\phi', \phi'', \psi', \psi'') = (\ \underset{\phi'}{\curlywedge^s}\ \mathbin{+\!\!+}\ \underset{\phi''}{\curlywedge^s}\ )(\mathsf{I}_s \mathbin{+\!\!+} {}^s\mathsf{X}^s \mathbin{+\!\!+} \mathsf{I}_s)(\ \overset{\psi'}{\curlyvee_s}\ \mathbin{+\!\!+}\ \overset{\psi''}{\curlyvee_s}\ )$$

generate a larger class of stream processing functions than

$$F(\sigma, \tau) = \overset{\sigma}{\curlyvee_s} \cdot \underset{\tau}{\curlywedge^s}$$

Indeed,

$$E(\phi', \phi'', \psi', \psi'')(1 \frown 2 \frown \ldots, a \frown b \frown \ldots) = (2 \frown \ldots, b \frown \ldots)$$

for $\phi' = 1 \frown 2 \ldots;\quad \phi'' = 1 \frown 2 \ldots;\quad \psi' = 2 \ldots;\quad \psi'' = 1 \frown 1 \frown \ldots$. On the other hand, this output is not possible for

$$F(\sigma, \tau)(1 \frown 2 \frown \ldots, a \frown b \frown \ldots)$$

since the first output on at least one channel here is in the set $\{1, a\}$.

Conversely, it may be seen that $F(\sigma, \tau)$ may be simulated by $E(\phi', \phi'', \psi', \psi'')$ if one takes ϕ' and ϕ'' as certain restrictions of τ and ψ' and ψ'' as certain restrictions of σ. More precisely, for an oracle α and a subset of natural numbers $A \subseteq \omega$ let us denote by $\alpha|_A$ the oracle obtained by restricting α to A, i.e. if A consists of the elements $a_1 < a_2 < \ldots$ then $\alpha|_A(i) = \alpha(a_i)$ for $i = 1, 2, \ldots$. Now

$$\phi' = \tau|_{\sigma^{-1}(1)}, \quad \phi'' = \tau|_{\sigma^{-1}(2)}, \quad \psi' = \sigma|_{\tau^{-1}(1)}, \quad \psi'' = \sigma|_{\tau^{-1}(2)}$$

(In case certain oracles as above are finite, we may extend them to infinite oracles in an arbitrary way and the result holds.)

With respect to A11, one may easily see that

$$E(\phi, \psi) = \underset{\phi}{\curlywedge^s} \cdot \overset{\psi}{\curlyvee_s}$$

generate a set of functions which properly includes I_s.

Finally, the left-hand side of F5 specifies a bag, hence the corresponding set of functions properly include I_s. $\square$

This concludes our proofs.

6 Conclusions

Diagrams and also flow graphs have been and still are very popular in many software engineering methods. For a foundation of such ideas the algebra of flow graphs is a very helpful basis. Its application to dataflow graphs leads to an algebraic calculus where laws of graph isomorphisms and laws of semantic characteristics of dataflow nodes are combined. Besides studying nondeterminism, another interesting area is in the field of recursively defined dataflow graphs and the related field of dynamic dataflow nets that has a close relationship to Milner's π-calculus.

Acknowledgement

It is a pleasure to thank Ch. Facchi for help in preparing the manuscript and R. Grosu for stimulating discussions on the algebra of stream processing functions.

7 Appendix: The Axioms

Table 3 lists the groups of axioms we were starting with. The adapted axioms for dataflow networks are given in the previous two tables.

I. Axioms for ssmc-ies (symmetric strict monoidal categories)

B1	$f \mathbin{+\!\!+} (g \mathbin{+\!\!+} h) = (f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h$	B2	$\mathsf{I}_0 \mathbin{+\!\!+} f = f = f \mathbin{+\!\!+} \mathsf{I}_0$
B3	$f \cdot (g \cdot h) = (f \cdot g) \cdot h$	B4	$\mathsf{I}_a \cdot f = f = f \cdot \mathsf{I}_b$
B5	$(f \mathbin{+\!\!+} f') \cdot (g \mathbin{+\!\!+} g') = f \cdot g \mathbin{+\!\!+} f' \cdot g'$	B6	$\mathsf{I}_a \mathbin{+\!\!+} \mathsf{I}_b = \mathsf{I}_{a+b}$
B7	${}^a\mathsf{X}^b \cdot {}^b\mathsf{X}^a = \mathsf{I}_{a+b}$	B8	${}^a\mathsf{X}^0 = \mathsf{I}_a$
B9	${}^a\mathsf{X}^{b+c} = ({}^a\mathsf{X}^b \mathbin{+\!\!+} \mathsf{I}_c) \cdot (\mathsf{I}_b \mathbin{+\!\!+} {}^a\mathsf{X}^c)$	B10	$(f \mathbin{+\!\!+} g) \cdot {}^c\mathsf{X}^d = {}^a\mathsf{X}^b \cdot (g \mathbin{+\!\!+} f)$ for $f : a \to c, \quad g : b \to d$

II. Axioms for the additional constants $\top, \bot, \vee, \wedge$ (without feedback)

A1	$(\vee_a \mathbin{+\!\!+} \mathsf{I}_a) \cdot \vee_a = (\mathsf{I}_a \mathbin{+\!\!+} \vee_a) \cdot \vee_a$	A2	${}^a\mathsf{X}^a \cdot \vee_a = \vee_a$
A3	$(\top_a \mathbin{+\!\!+} \mathsf{I}_a) \cdot \vee_a = \mathsf{I}_a$	A4	$\vee_a \cdot \bot^a = \bot^a \mathbin{+\!\!+} \bot^a$
A5	$\wedge^a \cdot (\wedge^a \mathbin{+\!\!+} \mathsf{I}_a) = \wedge^a \cdot (\mathsf{I}_a \mathbin{+\!\!+} \wedge^a)$	A6	$\wedge^a \cdot {}^a\mathsf{X}^a = \wedge^a$
A7	$\wedge^a \cdot (\bot^a \mathbin{+\!\!+} \mathsf{I}_a) = \mathsf{I}_a$	A8	$\top_a \cdot \wedge^a = \top_a \mathbin{+\!\!+} \top_a$

A9 $\quad \top_a \cdot \bot^a = \mathsf{I}_0$

A10 $\quad \vee_a \cdot \wedge^a = (\wedge^a \mathbin{+\!\!+} \wedge^a) \cdot (\mathsf{I}_a \mathbin{+\!\!+} {}^a\mathsf{X}^a \mathbin{+\!\!+} \mathsf{I}_a) \cdot (\vee_a \mathbin{+\!\!+} \vee_a)$

A11 $\quad \wedge^a \cdot \vee_a = \mathsf{I}_a$

A12	$\top_0 = \mathsf{I}_0$	A13	$\top_{a+b} = \top_a \mathbin{+\!\!+} \top_b$
A14	$\vee_0 = \mathsf{I}_0$	A15	$\vee_{a+b} = (\mathsf{I}_a \mathbin{+\!\!+} {}^b\mathsf{X}^a \mathbin{+\!\!+} \mathsf{I}_b) \cdot (\vee_a \mathbin{+\!\!+} \vee_b)$
A16	$\bot^0 = \mathsf{I}_0$	A17	$\bot^{a+b} = \bot^a \mathbin{+\!\!+} \bot^b$
A18	$\wedge^0 = \mathsf{I}_0$	A19	$\wedge^{a+b} = (\wedge^a \mathbin{+\!\!+} \wedge^b) \cdot (\mathsf{I}_a \mathbin{+\!\!+} {}^a\mathsf{X}^b \mathbin{+\!\!+} \mathsf{I}_b)$

III. Axioms for feedback

R1	$f \cdot (g \uparrow^c) \cdot h = ((f \mathbin{+\!\!+} \mathsf{I}_c) \cdot g \cdot (h \mathbin{+\!\!+} \mathsf{I}_c)) \uparrow^c$	(relating "$\uparrow$" and "$\cdot$")
R2	$f \mathbin{+\!\!+} g \uparrow^c = (f \mathbin{+\!\!+} g) \uparrow^c$	(relating "$\uparrow$" and "$\mathbin{+\!\!+}$")
R3	$(f \cdot (\mathsf{I}_b \mathbin{+\!\!+} g)) \uparrow^c = ((\mathsf{I}_a \mathbin{+\!\!+} g) \cdot f) \uparrow^d$ for $f : a + c \to b + d, \quad g : d \to c$	(shifting blocks on feedback)
R4	$f \uparrow^0 = f$	(no feedback)
R5	$(f \uparrow^b) \uparrow^a = f \uparrow^{a+b}$	(multiple feedbacks)

Table 3: The Algebra of Binary Flownomials

IV. Axioms for the action of feedback on constants

F1	$\mathsf{I}_a \uparrow^a = \mathsf{I}_0$	F2	${}^a\mathsf{X}^a \uparrow^a = \mathsf{I}_a$
F3	$\vee_a \uparrow^a = \perp^a$	F4	$\wedge^a \uparrow^a = \top_a$
F5	$[(\mathsf{I}_a + \wedge^a) \cdot ({}^a\mathsf{X}^a + \mathsf{I}_a) \cdot (\mathsf{I}_a + \vee_a)] \uparrow^a = \mathsf{I}_a$		

V. The strong axioms $(f : a \to b)$

S1	$\top_a \cdot f = \top_b$	S2	$\vee_a \cdot f = (f + f) \cdot \vee_b$
S3	$f \cdot \perp^b = \perp^a$	S4	$f \cdot \wedge^b = \wedge^a \cdot (f + f)$

VI. The enzymatic rule

ENZ$_{Fn^{-1}}$: $f \cdot (\mathsf{I}_b + y) = (\mathsf{I}_a + y) \cdot g$ implies $f \uparrow^c = g \uparrow^d$,

where E is a class of abstract relations (i.e., of terms written with $+, \cdot, \mathsf{I}, X$ and some constants in $\top, \perp, \vee, \wedge$), $y : c \to d$ is in E and $f : a + c \to b + c,\ g : a + d \to b + d$ are arbitrary

Table 3: The Algebra of Binary Flownomials (continued)

References

[BE88] S.L. Bloom and Z. Esik. Varieties of iteration theories. *SIAM Journal of Computing*, 17:939–966, 1988.

[BE93b] S.L. Bloom and Z. Esik. *Iteration theories. The equational logic of iterative processes*. Springer–Verlag, 1993.

[Böh85] A.P.W. Böhm. Dataflow computation. CWI Tracts, Vol. 6. Centre for Mathematics and Computer Science, Amsterdam, 1984.

[BrA81] J.D. Brock and W.B. Ackermann. Scenarios: A model of non-determinate computation. *Proc. Peniscola Colloquium*, 252-259. LNCS 107, Springer Verlag, 1988.

[Bro83] M. Broy. Fixed point theory for communication and concurrency. In: *Formal Description of Programming Concepts - II*, D. Bjorner (ed.), pages 125–147. North-Holland, 1983.

[Bro87] M. Broy. Semantics of finite or infinite networks of communicating agents. *Distributed Computing*, 2:13–31, 1987.

[Bro88] M. Broy. Nondeterministic dataflow programs: how to avoid the merge anomaly. *Science of Computer Programming*, 10:65–85, 1988.

[Bro93] M. Broy. Functional specification of time sensitive communicating systems. *ACM Transactions on Software Engineering and Methodology* 2:1–46, 1993.

[BS94] J.A. Bergstra and Gh. Ştefănescu. Network algebra for synchronous and asynchronous dataflow. Technical Report No. 122, Logic Group Preprint Series, Utrecht University, 1994.

[CaS88–89] V.E. Căzănescu and Gh. Ştefănescu. A formal representation of flowchart schemes I, II. *Analele Universitătii Bucuresti, Matematică - Informatică*, 37(2):33–51, 1988 and *Studii si Cercetări Metematice*, 41:151–167, 1989.

[CaS90] V.E. Căzănescu and Gh. Ştefănescu. Towards a new algebraic foundation of flowchart scheme theory. *Fundamenta Informaticae*, 13:171–210, 1990.

[Con71] J. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.

[Elg75] C.C. Elgot. Monadic computation and iterative algebraic theories. In *Proceedings Logic Colloquium'73*, pages 175–230. North-Holland, 1975. Studies in Logic and the Foundations of Mathematics, Volume 80.

[Gro94] R. Grosu. A formal foundation for concurrent object-oriented programming. Dissertation, Fakultät für Informatik, Technische Universität München, December 94.

[Jon89] B. Jonsson. A fully abstract trace model for dataflow networks. In: *Prooc. 16th ACM Symp. POPL'89*, 155–165, 1989.

[JoK91] B. Jonsson and J. Kok. Towards a complete hierarchy of compositional dataflow models. In: *Proc. TACS'91*, 204-225. LNCS 526, Springer Verlag, 1991.

[Kah74] G. Kahn. The semantics of a simple language for parallel processing. In: *Proc. IFIP Congress'74*, 471–475, 1974.

[Kel78] R. Keller. Denotational models for parallel programs with indeterminate operators. In: *Formal description of Programming Concepts*, 337–366. North–Holland, 1986.

[KeP86] R. Keller and P. Panangaden. Semantics of networks with nondeterminate operators. *Distributed Computing*, 1:235–245, 1986.

[Kle56] S.C. Kleene. Representation of events in nerve nets and finite automata. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956. Annals of Mathematical Studies, Volume 34.

[Kok87] J. Kok. A fully abstract semantics for data flow nets. In: *Proc. PARLE*, pages 351–368. LNCS 259, Springer Verlag, 1987.

[Par80] D. Park. On the semantics of fair parallelism. In *Proc. Copenhagen Winter School*, 504–526. LNCS 82, Springer Verlag, 1980.

[Park83] D. Park. The fairness problem and nondeterministic computing networks. In *Foundations of Computer Scince IV: Part 2. Semantics and Logic*, 133–161. Mathematical Centre Tracts 159, Amsterdam, 1983.

[Rus89] J. Russell. Full abstraction for nondeterministic dataflow networks. In: *Proc. FoCS'89*, IEEE Computer Science Press, 1989.

[Sta92] E.W. Stark. A calculus of dataflow networks. In: *Proceedings LICS'92*, 125–136. IEEE Computer Science Press, 1992.

[SN85] J. Staples and V. Nguyen. A fixpoint semantics for nondeterministic data flow. *Journal of the Association for Computing Machinery*, 32:411–444, 1985.

[Ste87a] Gh. Ştefănescu. On flowchart theories I: The deterministic case. *Journal of Computer and System Sciences*, 35:163–191, 1987.

[Ste87b] Gh. Ştefănescu. On flowchart theories II: The nondeterministic case. *Theoretical Computer Science*, 52:307–340, 1987.

[Ste86] Gh. Ştefănescu. Feedback theories (a calculus for isomorphism classes of flowchart schemes. *Preprint Series in Mathematics*, The National Institute for Scientific and Technical Creation, Bucharest, No. 24/April 1986. Also in: *Revue Roumaine de Mathematiques Pures et Applique*, 35:73–79, 1990.

[Ste94] Gh. Ştefănescu. Algebra of flownomials. Part 1: Binary flownomials, basic theory. Technical report I9437 and SFB-Bericht Nr. 342/16/94 A, Dept. of Computer Science, Technical University Munich, 1994.

[TWW79] J.W. Thatcher, E.G. Wagner, and J.B. Wright. Notes on algebraic fundamentals for theoretical computer science. In *Foundation of Computer Science III: Part 2. Language, logic, semantics*, pages 83–164. Mathematical Centrum, Amsterdam, 1979.

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

342/1/90 A	Robert Gold, Walter Vogler: Quality Criteria for Partial Order Semantics of Place/Transition-Nets, Januar 1990
342/2/90 A	Reinhard Fößmeier: Die Rolle der Lastverteilung bei der numerischen Parallelprogrammierung, Februar 1990
342/3/90 A	Klaus-Jörn Lange, Peter Rossmanith: Two Results on Unambiguous Circuits, Februar 1990
342/4/90 A	Michael Griebel: Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen Transformations-Mehrgitter-Methode
342/5/90 A	Reinhold Letz, Johann Schumann, Stephan Bayerl, Wolfgang Bibel: SETHEO: A High-Performance Theorem Prover
342/6/90 A	Johann Schumann, Reinhold Letz: PARTHEO: A High Performance Parallel Theorem Prover
342/7/90 A	Johann Schumann, Norbert Trapp, Martin van der Koelen: SETHEO/PARTHEO Users Manual
342/8/90 A	Christian Suttner, Wolfgang Ertel: Using Connectionist Networks for Guiding the Search of a Theorem Prover
342/9/90 A	Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Jaroslav Kremenek, Robert Lindhof, Thomas Ludwig, Peter Luksch, Thomas Treml: TOPSYS, Tools for Parallel Systems (Artikelsammlung)
342/10/90 A	Walter Vogler: Bisimulation and Action Refinement
342/11/90 A	Jörg Desel, Javier Esparza: Reachability in Reversible Free- Choice Systems
342/12/90 A	Rob van Glabbeek, Ursula Goltz: Equivalences and Refinement
342/13/90 A	Rob van Glabbeek: The Linear Time - Branching Time Spectrum
342/14/90 A	Johannes Bauer, Thomas Bemmerl, Thomas Treml: Leistungsanalyse von verteilten Beobachtungs- und Bewertungswerkzeugen
342/15/90 A	Peter Rossmanith: The Owner Concept for PRAMs
342/16/90 A	G. Böckle, S. Trosch: A Simulator for VLIW-Architectures
342/17/90 A	P. Slavkovsky, U. Rüde: Schnellere Berechnung klassischer Matrix-Multiplikationen
342/18/90 A	Christoph Zenger: SPARSE GRIDS

Reihe A

342/19/90 A	Michael Griebel, Michael Schneider, Christoph Zenger: A combination technique for the solution of sparse grid problems
342/20/90 A	Michael Griebel: A Parallelizable and Vectorizable Multi- Level-Algorithm on Sparse Grids
342/21/90 A	V. Diekert, E. Ochmanski, K. Reinhardt: On confluent semi-commutations-decidability and complexity results
342/22/90 A	Manfred Broy, Claus Dendorfer: Functional Modelling of Operating System Structures by Timed Higher Order Stream Processing Functions
342/23/90 A	Rob van Glabbeek, Ursula Goltz: A Deadlock-sensitive Congruence for Action Refinement
342/24/90 A	Manfred Broy: On the Design and Verification of a Simple Distributed Spanning Tree Algorithm
342/25/90 A	Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Peter Luksch, Roland Wismüller: TOPSYS - Tools for Parallel Systems (User's Overview and User's Manuals)
342/26/90 A	Thomas Bemmerl, Arndt Bode, Thomas Ludwig, Stefan Tritscher: MMK - Multiprocessor Multitasking Kernel (User's Guide and User's Reference Manual)
342/27/90 A	Wolfgang Ertel: Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems
342/28/90 A	Rob van Glabbeek, Frits Vaandrager: Modular Specification of Process Algebras
342/29/90 A	Rob van Glabbeek, Peter Weijland: Branching Time and Abstraction in Bisimulation Semantics
342/30/90 A	Michael Griebel: Parallel Multigrid Methods on Sparse Grids
342/31/90 A	Rolf Niedermeier, Peter Rossmanith: Unambiguous Simulations of Auxiliary Pushdown Automata and Circuits
342/32/90 A	Inga Niepel, Peter Rossmanith: Uniform Circuits and Exclusive Read PRAMs
342/33/90 A	Dr. Hermann Hellwagner: A Survey of Virtually Shared Memory Schemes
342/1/91 A	Walter Vogler: Is Partial Order Semantics Necessary for Action Refinement?
342/2/91 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Rainer Weber: Characterizing the Behaviour of Reactive Systems by Trace Sets
342/3/91 A	Ulrich Furbach, Christian Suttner, Bertram Fronhöfer: Massively Parallel Inference Systems
342/4/91 A	Rudolf Bayer: Non-deterministic Computing, Transactions and Recursive Atomicity
342/5/91 A	Robert Gold: Dataflow semantics for Petri nets
342/6/91 A	A. Heise; C. Dimitrovici: Transformation und Komposition von P/T-Netzen unter Erhaltung wesentlicher Eigenschaften
342/7/91 A	Walter Vogler: Asynchronous Communication of Petri Nets and the Refinement of Transitions
342/8/91 A	Walter Vogler: Generalized OM-Bisimulation
342/9/91 A	Christoph Zenger, Klaus Hallatschek: Fouriertransformation auf dünnen Gittern mit hierarchischen Basen

Reihe A

342/10/91 A	Erwin Loibl, Hans Obermaier, Markus Pawlowski: Towards Parallelism in a Relational Database System
342/11/91 A	Michael Werner: Implementierung von Algorithmen zur Kompaktifizierung von Programmen für VLIW-Architekturen
342/12/91 A	Reiner Müller: Implementierung von Algorithmen zur Optimierung von Schleifen mit Hilfe von Software-Pipelining Techniken
342/13/91 A	Sally Baker, Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Udo Graf, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Rainer Knödlseder, Jaroslav Kremenek, Siegfried Langenbuch, Robert Lindhof, Thomas Ludwig, Peter Luksch, Roy Milner, Bernhard Ries, Thomas Treml: TOPSYS - Tools for Parallel Systems (Artikelsammlung); 2., erweiterte Auflage
342/14/91 A	Michael Griebel: The combination technique for the sparse grid solution of PDE's on multiprocessor machines
342/15/91 A	Thomas F. Gritzner, Manfred Broy: A Link Between Process Algebras and Abstract Relation Algebras?
342/16/91 A	Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Thomas Treml, Roland Wismüller: The Design and Implementation of TOPSYS
342/17/91 A	Ulrich Furbach: Answers for disjunctive logic programs
342/18/91 A	Ulrich Furbach: Splitting as a source of parallelism in disjunctive logic programs
342/19/91 A	Gerhard W. Zumbusch: Adaptive parallele Multilevel-Methoden zur Lösung elliptischer Randwertprobleme
342/20/91 A	M. Jobmann, J. Schumann: Modelling and Performance Analysis of a Parallel Theorem Prover
342/21/91 A	Hans-Joachim Bungartz: An Adaptive Poisson Solver Using Hierarchical Bases and Sparse Grids
342/22/91 A	Wolfgang Ertel, Theodor Gemenis, Johann M. Ph. Schumann, Christian B. Suttner, Rainer Weber, Zongyan Qiu: Formalisms and Languages for Specifying Parallel Inference Systems
342/23/91 A	Astrid Kiehn: Local and Global Causes
342/24/91 A	Johann M.Ph. Schumann: Parallelization of Inference Systems by using an Abstract Machine
342/25/91 A	Eike Jessen: Speedup Analysis by Hierarchical Load Decomposition
342/26/91 A	Thomas F. Gritzner: A Simple Toy Example of a Distributed System: On the Design of a Connecting Switch
342/27/91 A	Thomas Schnekenburger, Andreas Weininger, Michael Friedrich: Introduction to the Parallel and Distributed Programming Language ParMod-C
342/28/91 A	Claus Dendorfer: Funktionale Modellierung eines Postsystems
342/29/91 A	Michael Griebel: Multilevel algorithms considered as iterative methods on indefinite systems
342/30/91 A	W. Reisig: Parallel Composition of Liveness
342/31/91 A	Thomas Bemmerl, Christian Kasperbauer, Martin Mairandres, Bernhard Ries: Programming Tools for Distributed Multiprocessor Computing Environments

Reihe A

342/32/91 A	Frank Leßke: On constructive specifications of abstract data types using temporal logic
342/1/92 A	L. Kanal, C.B. Suttner (Editors): Informal Proceedings of the Workshop on Parallel Processing for AI
342/2/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS
342/2-2/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS - Revised Version (erschienen im Januar 1993)
342/3/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/4/92 A	Claus Dendorfer, Rainer Weber: Development and Implementation of a Communication Protocol - An Exercise in FOCUS
342/5/92 A	Michael Friedrich: Sprachmittel und Werkzeuge zur Unterstüt- zung paralleler und verteilter Programmierung
342/6/92 A	Thomas F. Gritzner: The Action Graph Model as a Link between Abstract Relation Algebras and Process-Algebraic Specifications
342/7/92 A	Sergei Gorlatch: Parallel Program Development for a Recursive Numerical Algorithm: a Case Study
342/8/92 A	Henning Spruth, Georg Sigl, Frank Johannes: Parallel Algorithms for Slicing Based Final Placement
342/9/92 A	Herbert Bauer, Christian Sporrer, Thomas Krodel: On Distributed Logic Simulation Using Time Warp
342/10/92 A	H. Bungartz, M. Griebel, U. Rüde: Extrapolation, Combination and Sparse Grid Techniques for Elliptic Boundary Value Problems
342/11/92 A	M. Griebel, W. Huber, U. Rüde, T. Störtkuhl: The Combination Technique for Parallel Sparse-Grid-Preconditioning and -Solution of PDEs on Multiprocessor Machines and Workstation Networks
342/12/92 A	Rolf Niedermeier, Peter Rossmanith: Optimal Parallel Algorithms for Computing Recursively Defined Functions
342/13/92 A	Rainer Weber: Eine Methodik für die formale Anforderungsspezifkation verteilter Systeme
342/14/92 A	Michael Griebel: Grid– and point–oriented multilevel algorithms
342/15/92 A	M. Griebel, C. Zenger, S. Zimmer: Improved multilevel algorithms for full and sparse grid problems
342/16/92 A	J. Desel, D. Gomm, E. Kindler, B. Paech, R. Walter: Bausteine eines kompositionalen Beweiskalküls für netzmodellierte Systeme
342/17/92 A	Frank Dederichs: Transformation verteilter Systeme: Von applikativen zu prozeduralen Darstellungen
342/18/92 A	Andreas Listl, Markus Pawlowski: Parallel Cache Management of a RDBMS
342/19/92 A	Erwin Loibl, Markus Pawlowski, Christian Roth: PART: A Parallel Relational Toolbox as Basis for the Optimization and Interpretation of Parallel Queries

Reihe A

342/20/92 A Jörg Desel, Wolfgang Reisig: The Synthesis Problem of Petri Nets
342/21/92 A Robert Balder, Christoph Zenger: The d-dimensional Helmholtz equation on sparse Grids
342/22/92 A Ilko Michler: Neuronale Netzwerk-Paradigmen zum Erlernen von Heuristiken
342/23/92 A Wolfgang Reisig: Elements of a Temporal Logic. Coping with Concurrency
342/24/92 A T. Störtkuhl, Chr. Zenger, S. Zimmer: An asymptotic solution for the singularity at the angular point of the lid driven cavity
342/25/92 A Ekkart Kindler: Invariants, Compositionality and Substitution
342/26/92 A Thomas Bonk, Ulrich Rüde: Performance Analysis and Optimization of Numerically Intensive Programs
342/1/93 A M. Griebel, V. Thurner: The Efficient Solution of Fluid Dynamics Problems by the Combination Technique
342/2/93 A Ketil Stølen, Frank Dederichs, Rainer Weber: Assumption / Commitment Rules for Networks of Asynchronously Communicating Agents
342/3/93 A Thomas Schnekenburger: A Definition of Efficiency of Parallel Programs in Multi-Tasking Environments
342/4/93 A Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: A Proof of Convergence for the Combination Technique for the Laplace Equation Using Tools of Symbolic Computation
342/5/93 A Manfred Kunde, Rolf Niedermeier, Peter Rossmanith: Faster Sorting and Routing on Grids with Diagonals
342/6/93 A Michael Griebel, Peter Oswald: Remarks on the Abstract Theory of Additive and Multiplicative Schwarz Algorithms
342/7/93 A Christian Sporrer, Herbert Bauer: Corolla Partitioning for Distributed Logic Simulation of VLSI Circuits
342/8/93 A Herbert Bauer, Christian Sporrer: Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving
342/9/93 A Peter Slavkovsky: The Visibility Problem for Single-Valued Surface (z = f(x,y)): The Analysis and the Parallelization of Algorithms
342/10/93 A Ulrich Rüde: Multilevel, Extrapolation, and Sparse Grid Methods
342/11/93 A Hans Regler, Ulrich Rüde: Layout Optimization with Algebraic Multigrid Methods
342/12/93 A Dieter Barnard, Angelika Mader: Model Checking for the Modal Mu-Calculus using Gauß Elimination
342/13/93 A Christoph Pflaum, Ulrich Rüde: Gauß' Adaptive Relaxation for the Multilevel Solution of Partial Differential Equations on Sparse Grids
342/14/93 A Christoph Pflaum: Convergence of the Combination Technique for the Finite Element Solution of Poisson's Equation
342/15/93 A Michael Luby, Wolfgang Ertel: Optimal Parallelization of Las Vegas Algorithms
342/16/93 A Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: Pointwise Convergence of the Combination Technique for Laplace's Equation

Reihe A

342/17/93 A	Georg Stellner, Matthias Schumann, Stefan Lamberts, Thomas Ludwig, Arndt Bode, Martin Kiehl und Rainer Mehlhorn: Developing Multicomputer Applications on Networks of Workstations Using NXLib
342/18/93 A	Max Fuchs, Ketil Stølen: Development of a Distributed Min/Max Component
342/19/93 A	Johann K. Obermaier: Recovery and Transaction Management in Write-optimized Database Systems
342/20/93 A	Sergej Gorlatch: Deriving Efficient Parallel Programs by Systemating Coarsing Specification Parallelism
342/01/94 A	Reiner Hüttl, Michael Schneider: Parallel Adaptive Numerical Simulation
342/02/94 A	Henning Spruth, Frank Johannes: Parallel Routing of VLSI Circuits Based on Net Independency
342/03/94 A	Henning Spruth, Frank Johannes, Kurt Antreich: PHIroute: A Parallel Hierarchical Sea-of-Gates Router
342/04/94 A	Martin Kiehl, Rainer Mehlhorn, Matthias Schumann: Parallel Multiple Shooting for Optimal Control Problems Under NX/2
342/05/94 A	Christian Suttner, Christoph Goller, Peter Krauss, Klaus-Jörn Lange, Ludwig Thomas, Thomas Schnekenburger: Heuristic Optimization of Parallel Computations
342/06/94 A	Andreas Listl: Using Subpages for Cache Coherency Control in Parallel Database Systems
342/07/94 A	Manfred Broy, Ketil Stølen: Specification and Refinement of Finite Dataflow Networks - a Relational Approach
342/08/94 A	Katharina Spies: Funktionale Spezifikation eines Kommunikationsprotokolls
342/09/94 A	Peter A. Krauss: Applying a New Search Space Partitioning Method to Parallel Test Generation for Sequential Circuits
342/10/94 A	Manfred Broy: A Functional Rephrasing of the Assumption/Commitment Specification Style
342/11/94 A	Eckhardt Holz, Ketil Stølen: An Attempt to Embed a Restricted Version of SDL as a Target Language in Focus
342/12/94 A	Christoph Pflaum: A Multi-Level-Algorithm for the Finite-Element-Solution of General Second Order Elliptic Differential Equations on Adaptive Sparse Grids
342/13/94 A	Manfred Broy, Max Fuchs, Thomas F. Gritzner, Bernhard Schätz, Katharina Spies, Ketil Stølen: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/14/94 A	Maximilian Fuchs: Technologieabhängigkeit von Spezifikationen digitaler Hardware
342/15/94 A	M. Griebel, P. Oswald: Tensor Product Type Subspace Splittings And Multilevel Iterative Methods For Anisotropic Problems
342/16/94 A	Gheorghe Ştefănescu: Algebra of Flownomials
342/17/94 A	Ketil Stølen: A Refinement Relation Supporting the Transition from Unbounded to Bounded Communication Buffers
342/18/94 A	Michael Griebel, Tilman Neuhoeffer: A Domain-Oriented Multilevel Algorithm-Implementation and Parallelization

Reihe A

342/19/94 A Michael Griebel, Walter Huber: Turbulence Simulation on Sparse Grids Using the Combination Method

342/20/94 A Johann Schumann: Using the Theorem Prover SETHEO for verifying the development of a Communication Protocol in FOCUS - A Case Study -

342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids

342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law

342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space

342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes

342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems

342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter

342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology

342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies

342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness

342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design

342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System

342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm

342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming

342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks

342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs

342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik

342/17/95 A Georg Stellner: Using CoCheck on a Network of Workstations

342/18/95 A Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications

342/19/95 A Thomas Schnekenburger: Integration of Load Distribution into ParMod-C

342/20/95 A Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication

342/21/95 A Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control

342/22/95 A Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes

Reihe A

342/23/95 A	Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation
342/24/95 A	M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems
342/01/96 A	Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries
342/02/96 A	Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs
342/03/96 A	Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes
342/04/96 A	Thomas Huckle: Efficient Computation of Sparse Approximate Inverses
342/05/96 A	Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification
342/06/96 A	Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components
342/07/96 A	Richard Mayr: Some Results on Basic Parallel Processes
342/08/96 A	Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht
342/09/96 A	P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme
342/10/96 A	Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFS-Lib – A File System for Parallel Programming Environments
342/11/96 A	Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox - Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS