Fundamenta Informaticae XIII (1990) 171-210
IOS Press

TOWARDS A NEW ALGEBRAIC FOUNDATION OF FLOWCHART SCHEME THEORY

Virgil Emil CĂZĂNESCU

Faculty of Mathematics, University of Bucharest, Str. Academiei 14, 70109 Bucharest, Romania

and

Gheorghe ŞTEFĂNESCU

Department of Mathematics, The National Institute for Scientific and Technical Creation, Bd. Păcii 220, 79622 Bucharest, Romania

Abstract. We develope a formalism for the algebraic study of flowchart schemes and their behaviours, based on a new axiomatic looping operation, called feedback.

This formalism is based on certain flownomial expressions. Such an expression is built up from two types of atomic schemes (i.e., elements in a double-ranked set X considered as unknown computation processes, and elements in a "theory" T considered as known computation processes) by using three operations: sum, composition, and feedback. Flownomial expressions are subject to certain rules of identification.

The axiomatization of flowchart schemes is based on the fact that a flowchart scheme may be identified with a class of isomorphic flownomial expressions in normal form. The corresponding algebra for flowchart schemes is called biflow.

This axiomatization is extended to certain types of behaviour. We present axiomatizations for accessible flowchart schemes, reduced flowchart schemes, minimal flowchart schemes with respect to the input behaviour, minimal flowchart schemes with respect to the input-output behaviour etc. Some results are new, others are simple translations in terms of feedback of previous results obtained by using Elgot's iteration or Kleene's repetition.

The paper also contains some historical comments.

INTRODUCTION

In the study of flowchart schemes we use a new operation called feedback (Figure 4.c) instead of the iteration to model the loops. As an identification of the return points with the inputs appears implicitly in the definition of the iteration, the use of iteration implies the use of tupling (Figure 11), therefore the algebraic theories have had a main place in the study of flowchart schemes. The use of feedback permits to leave out the tupling. Our conviction is that the symmetric strict monoidal categories (defined in [28]) are the most adequate algebraic structures to study acyclic flowchart schemes. To study flowchart schemes we use a symmetric strict monoidal category endowed with an adequate axiomatized feedback.

The aim of this paper is just to provide motivation. Proofs will be given elsewhere.

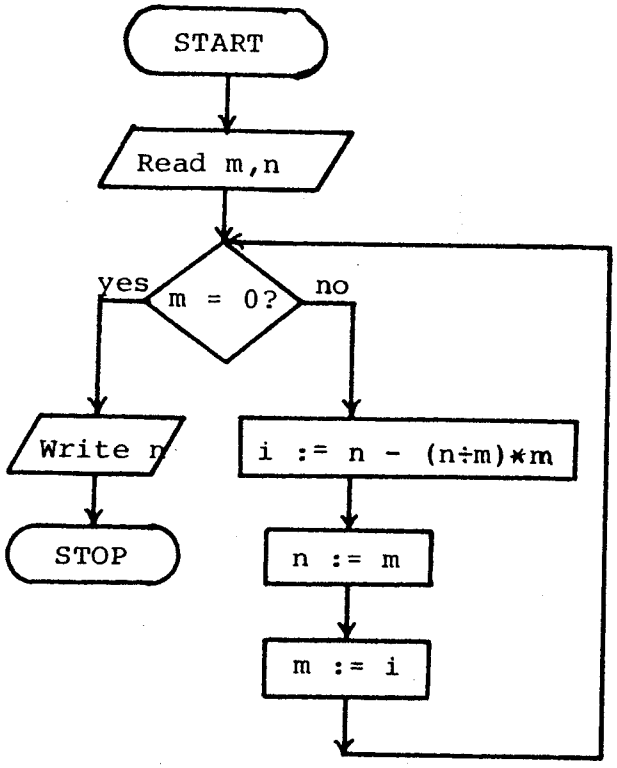


FIGURE 1
A concrete flowchart.

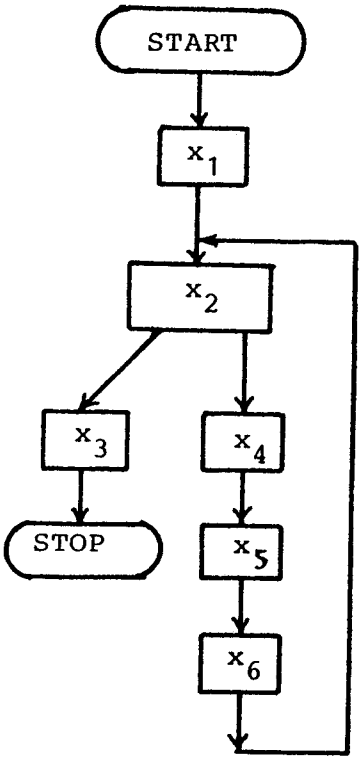


FIGURE 2
A usual abstract flowchart scheme.

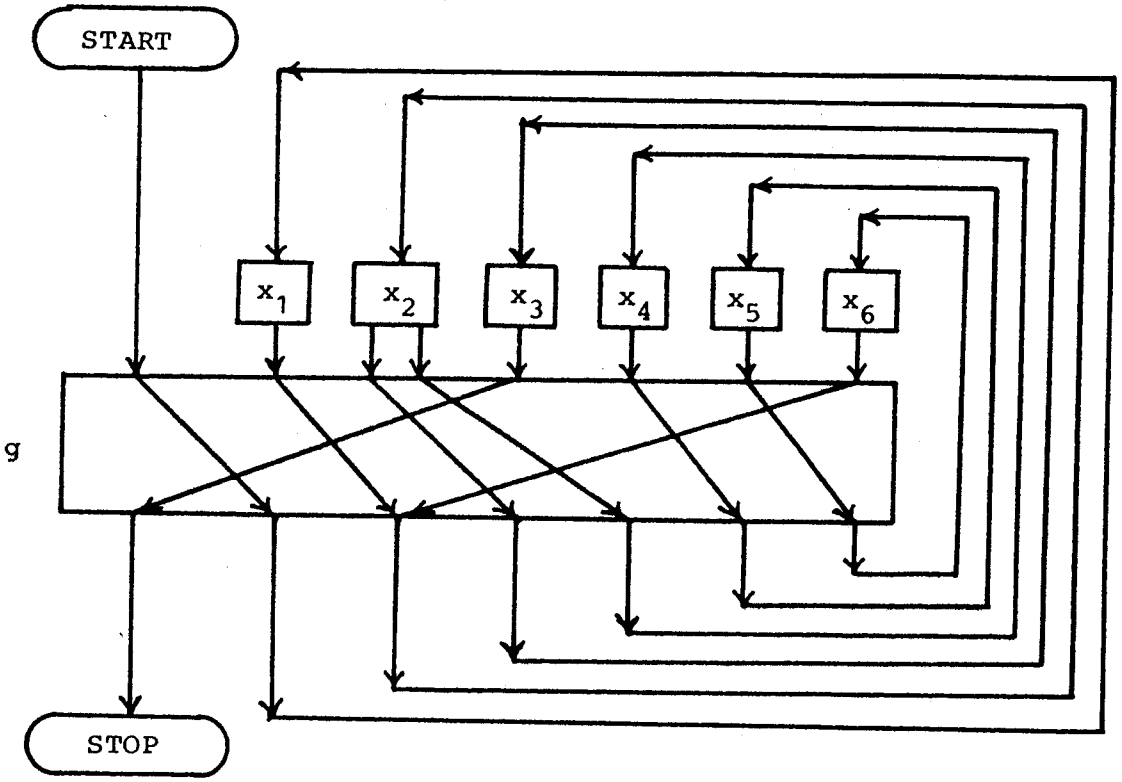


FIGURE 3
The normal form of a flowchart scheme.

1. A FORMAL REPRESENTATION OF FLOWCHART SCHEMES

1.1. A representation by pairs

The usual computation processes may be represented by flowchart pictures as in Figure 1. The meaning of the picture is the usual one: start the computation beginning with the input vertex (START) and execute the statements in the order given by arrows until an output vertex is reached; in the case a statement has more than one output arrow (exit) its execution gives at the same time the information regarding the output arrow on which the execution is continued.

The (abstract) flowchart schemes will be obtained by a double abstraction of these concrete flowchart pictures: an abstraction of statements and an abstraction of connections.

The first abstraction is easier to understand. It consists of replacing the concrete statements used to label the vertices in flowchart pictures by abstract symbols (variables). Since the statements we use may have more than one entry and one exit, the set of variables is a double-indexed set $\{X(m,n)\}_{m,n\in \mathbf{N}}$. An element $x\in X(m,n)$ is considered as an unknown computation process with m entries and n exits (a still unspecified computation process). Denote by X the disjoint union of this family of variables. Two functions $i,o : X \to \mathbf{N}$ specify the numbers of entries and of exits respectively, corresponding to a variable, i.e. $x\in X(m,n)$ iff $i(x) = m$ and $o(x) = n$.

The result of this abstraction is the usual notion of "flowchart scheme" studied in the seventies (Manna [29], Greibach [25], Kotov [27]): An X-flowchart scheme is a finite, locally ordered, oriented graph whose vertices are coherently labelled by symbols in X. Such an abstraction of the flowchart picture in Figure 1 is given in Figure 2, where $x_1, x_3, x_4, x_5, x_6 \in X(1,1)$ and $x_2 \in X(1,2)$.

The second abstraction is more complicated, and at the present stage of the presentation only a vague definition can be given. Note that every flowchart picture can be rearranged in a normal way by putting on a first level the statements of the scheme and on a second level the connections of the scheme. For example, the scheme in Figure 2 can be arranged in a normal form as in Figure 3. In this way we can image the possibility of using a "theory" for connections. (What "theory" means will be explained later.) In our concrete case, this theory is the theory of finite functions **Fn** given by the family of sets

$$\mathbf{Fn}(m,n) = \{f \mid f : [m] \to [n] \text{ function}\}, \text{ for } m,n\in \mathbf{N}$$

where $[n] = \{1,2,\ldots,n\}$. An element $f\in \mathbf{Fn}(m,n)$ used as a connection indicates the redirecting of flow of control. For the scheme in Figure 3 the connection $g\in \mathbf{Fn}(8,7)$ is given in the following table obtained using the large rectangle in Figure 3

j	1	2	3	4	5	6	7	8
g(j)	2	3	4	5	1	6	7	3

.

At the abstract level we shall use for connections a "support" theory T given by a family of sets $\{T(m,n)\}_{m,n\in \mathbf{N}}$. An element $f\in T(m,n)$ is considered as a known computation

process with m entries and n exits.

The result of this double abstraction is the concept of representation of an X-flowchart scheme over T. It can be defined as follows. For x in the free monoid X^* we denote by $|x|$ the length of the word x, and for $j \in [|x|]$ we denote by x_j the j-th letter of x. Since the operation of concatenation is denoted additive we have $x = x_1 + x_2 + \dots + x_{|x|}$. Also we use the notation: $i(x) = i(x_1) + \dots + i(x_{|x|})$ and $o(x) = o(x_1) + \dots + o(x_{|x|})$. A representation of an X-flowchart scheme over T with m entries and n exits is defined as a pair

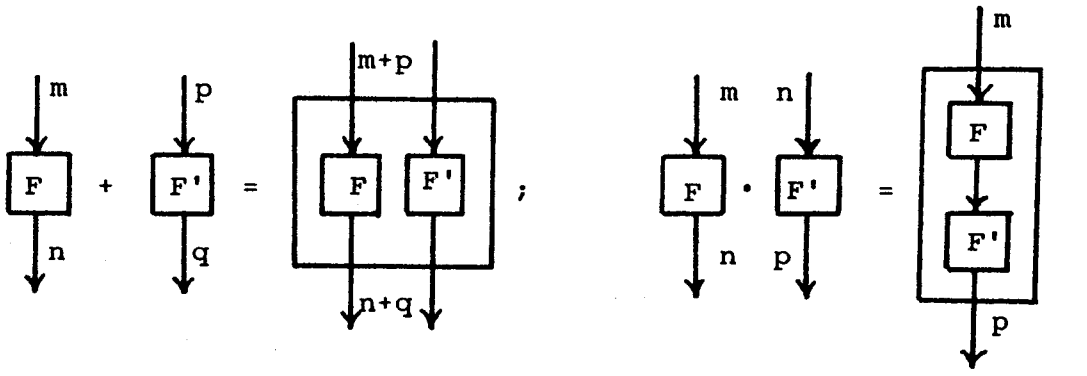
$$F = (x, f)$$

where $x = x_1 + \dots + x_{|x|} \in X^*$ specifies the vertices of the scheme, ordered in a linear way, and $f \in T(m + o(x), n + i(x))$ specifies the connection of the scheme. The scheme in Figure 3 may be represented as $(x_1 + x_2 + x_3 + x_4 + x_5 + x_6, g)$, where $g \in Fn(8,7)$ is the function defined above.

It must be emphasized that there may be more representations which correspond to a flowchart scheme. The difference between these representations is generated by the way in which the statements of the scheme are linearly ordered as a string $x \in X^*$.

We denote by $Fl_{X,T}$ the set of representations of X-flowchart schemes over T. More precisely,

$$Fl_{X,T}(m,n) = \left\{ (x,f) \,\middle|\, x \in X^*,\ f \in T(m + o(x),\ n + i(x)) \right\} .$$



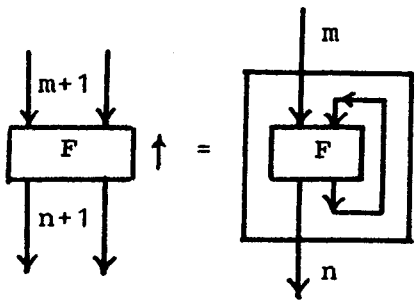


FIGURE 4
The basic operations on flowchart schemes: (a) "+" denotes sum; (b) "·" denotes composition; (c) "↑" denotes feedback.

1.2. Operations

While the above representation of schemes by pairs $F = (x,f)$ is convenient for theoretical purposes, for practical purposes it is inconvenient in the sense that it does not show how the scheme F can be obtained from the components x and f of its representation. To fill in this gap we introduce here operations on flowchart schemes.

If we look at the normal representation of schemes given in Figure 3, then we can deduce that every scheme can be obtained from the components of its representations by using the operations in Figure 4, called sum, composition and feedback. More precisely, a flowchart scheme F represented by the pair $(x,f) \in Fl_{X,T}(m,n)$ can also be represented by a formal expression

$$((1_m + x_1 + \dots + x_{|x|}) \cdot f)\uparrow^{i(x)},$$

where $\uparrow^{i(x)}$ denotes the application of the feedback by i(x) times, and $1_m \in T(m,m)$ is the scheme without (internal) vertices which directly connects the i-th entry on the i-th exit.

1.2.1. The elements of T are considered as particular schemes having only connections between entries and exits (i.e., without internal vertices). Therefore, if the operations above have sense in $Fl_{X,T}$, then they must be defined in T, too. The usual flowchart schemes have as support theory a subtheory of the theory of finite relations **Rel** defined by the family of sets

$$\mathbf{Rel}(m,n) = \{r \mid r \subseteq [m] \times [n] \text{ relation}\}, \text{ for } m,n \in \mathbf{N}.$$

Here the operations in Figure 4 have the following meaning.

The operations in **Rel**. For $r \subseteq [m] \times [n]$ and $r' \subseteq [p] \times [q]$ the sum $r + r' \subseteq [m + p] \times [n + q]$ is defined by

$$r + r' = r \cup \{(m + j, n + j') \mid (j,j') \in r'\}.$$

For $r \subseteq [m] \times [n]$ and $r' \subseteq [n] \times [p]$ the composite $r \cdot r' \subseteq [m] \times [p]$ is the usual one defined by

$$r \cdot r' = \{(j,j') \mid \text{there exists } u \in [n] \text{ such that } (j,u) \in r \text{ and } (u,j') \in r'\}.$$

For $r \subseteq [m + 1] \times [n + 1]$ the feedback $r\uparrow \subseteq [m] \times [n]$ is defined by

$$r\uparrow = \{(j,j') \in [m] \times [n] \mid (j,j') \in r \text{ or } [(j,n + 1) \in r \text{ and } (m + 1, j') \in r]\}.$$

The meaning of $1_m \in \mathbf{Rel}(m,m)$ is clear: $1_m = \{(j,j) \mid j \in [m]\}$. In the sequel we shall use some distinguished morphisms of the support theory T, namely

$$m \leftrightarrow n \in T(m + n, n + m),\ m \vee m \in T(m + m, m),\ 0_m \in T(0,m),$$

$$\perp_n \in T(n,0) \text{ and } m \wedge m \in T(m, m + m),$$

whose meaning in **Rel** is: $m \leftrightarrow n = \{(j, n + j) \mid j \in [m]\} \cup \{(m + j, j) \mid j \in [n]\}$; $0_m = \emptyset$, $m \vee m = \{(j,j) \mid j \in [m]\} \cup \{(m + j, j) \mid j \in [m]\}$; $m \wedge m = \{(j,j) \mid j \in [m]\} \cup \{(j, m + j) \mid j \in [m]\}$; $\perp_n = \emptyset$.

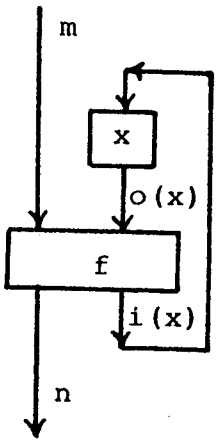


FIGURE 5
A concise picture of a schme in normal form.

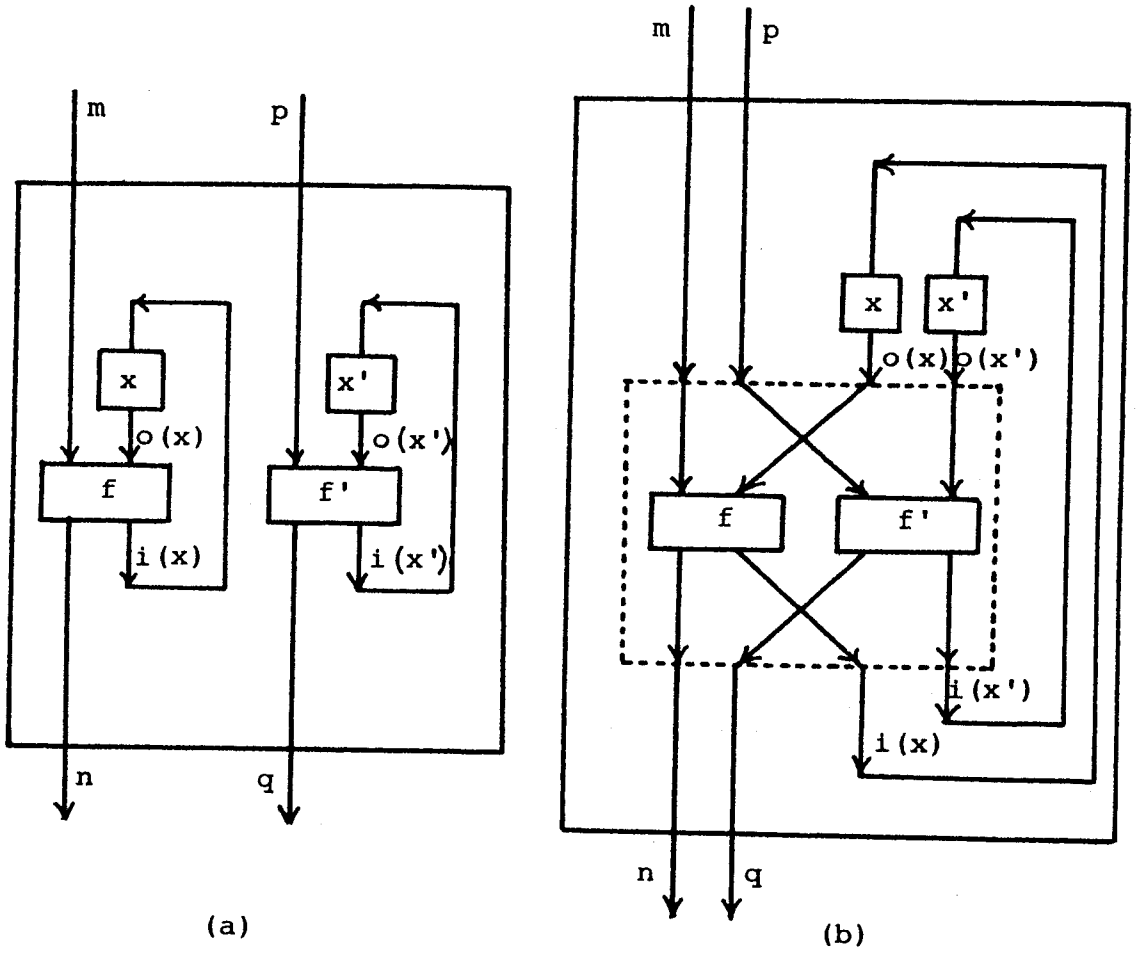


FIGURE 6
The normal form of the sum of two schemes in normal form.

Note. The subtheory of partial, finite functions in **Rel**, denoted by **Pfn** and defined by the family of sets

$$\mathbf{Pfn}(m,n) = \{f \mid f : [m] \to [n] \text{ partially defined function}\}, \text{ for } m,n \in \mathbf{N}$$

is closed under the aforementioned operations. The theory **Fn** defined in §1.1 is not closed under feedback, hence it is inconvenient to use **Fn** as a support theory for deterministic flowchart schemes (since $\mathbf{Fn}(1,0) = \emptyset$, so that for the unique function $f \in \mathbf{Fn}(2,1)$ we have $f^\dagger \notin \mathbf{Fn}(1,0)$). The use of **Pfn** as support theory in the deterministic case is equivalent to the extension of the concept of usual flowchart scheme to the concept of partial flowchart scheme. A partial flowchart scheme is obtained from a usual flowchart scheme by deleting some arrows, and one interprets such an absence of arrow as a connection to an endless loop. For the sake of contrast, sometimes the usual flowchart schemes (over **Fn**) will be called complete flowchart schemes.

1.2.2. Conversely, in the following section we shall see that it is easy to extend the operations in Figure 4 from T to $Fl_{X,T}$, supposing T "contains" bijective, finite functions.

We collect these facts as the following slogan:

> In order to define algebra $Fl_{X,T}$ we have to specify:
> - a double indexed set X;
> - a support theory T which contains finite bijections and is equipped with operations acting as in Figure 4.

1.3. The algebra of representations ($Fl_{X,T}$)

In order to extend our operations from T to $Fl_{X,T}$, T has to contain some distinguished elements $m \leftrightarrow n \in T(m+n, n+m)$ representing the "block transpositions" where $m,n \in \mathbf{N}$, i.e. ${}^{m}_{n}\times^{n}_{m}$. In the theory **Rel** the morphisms $m \leftrightarrow n$ were defined in §1.2.1.

The flowchart scheme in the normal form corresponding to a representation of an X-flowchart over T, namely $F = (x,f)$ is illustrated in Figure 5. The operations on flowchart scheme representations can be obtained by applying first the operations in Figure 4 on the pictures corresponding to the given representations, then by rearranging the obtained result in an adequate, normal form, and finally by writing the representation associated to the final picture.

The sum of two normal flowchart schemes illustrated in Figure 6.a can be rearranged in the normal form given in Figure 6.b. Hence, we can formally define the sum of two representations $(x,f) \in Fl_{X,T}(m,n)$ and $(x',f') \in Fl_{X,T}(p,q)$ by

$$(x,f) + (x',f') = (x + x',\ (1_m + p \leftrightarrow o(x) + 1_{o(x')})(f + f')(1_n + i(x) \leftrightarrow q + 1_{i(x')}))\ .$$

The composite of two normal flowchart schemes illustrated in Figure 7.a can be

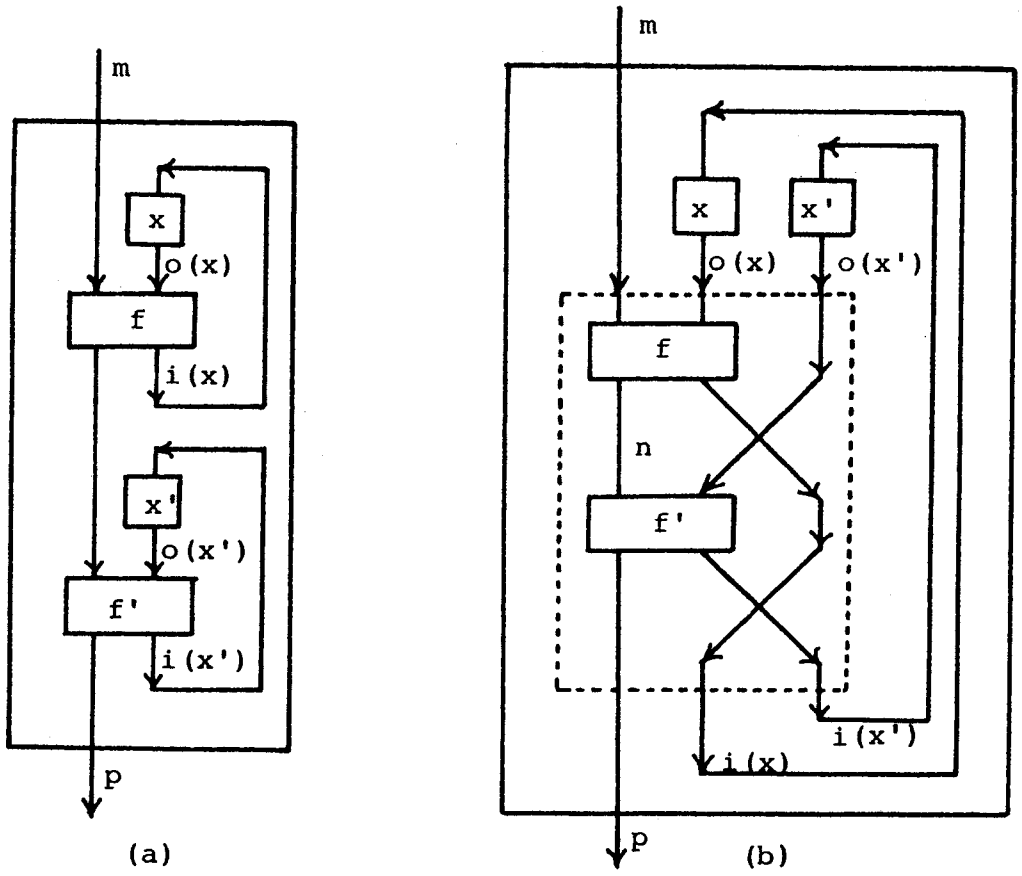


FIGURE 7
The normal form of the composite of two schemes in normal form.

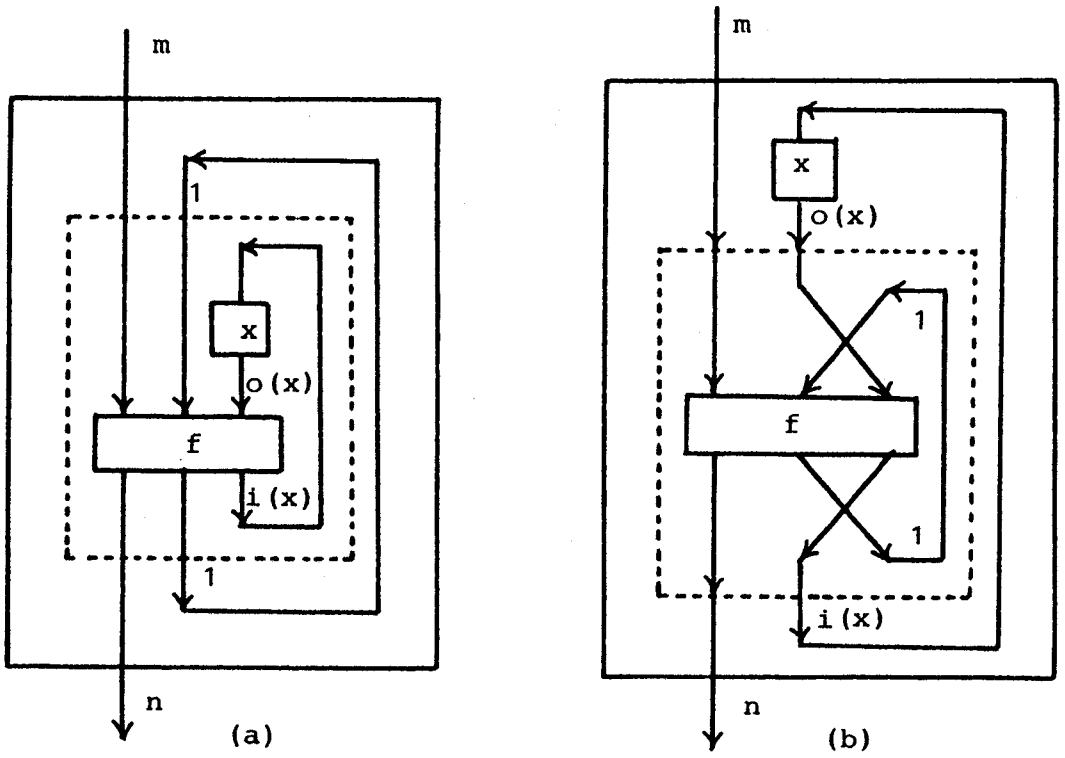


FIGURE 8
The normal form of the feedback of a scheme in normal form.

rearranged in the normal form given in Figure 7.b. Hence, we can formally define the composite of two representations $(x,f)\in Fl_{X,T}(m,n)$ and $(x',f')\in Fl_{X,T}(n,p)$ by

$$(x,f)\cdot(x',f') = (x + x',\ (f + 1_{o(x')})(1_n + i(x)\leftrightarrow o(x'))(f' + 1_{i(x)})(1_p + i(x')\leftrightarrow i(x))\ ).$$

The feedback of a normal flowchart scheme illustrated in Figure 8.a can be rearranged in the normal form given in Figure 8.b. Hence, we can formally define the feedback of a representation $(x,f)\in Fl_{X,T}(m + 1,n + 1)$ by

$$(x,f)\uparrow = (x,\ [(1_m + o(x)\leftrightarrow 1)f(1_n + 1\leftrightarrow i(x))]\uparrow\ ).$$

Let us mention that the embeddings of X and T into $Fl_{X,T}$ are given by the following applications:

$E_T(f) = (\varepsilon,f)$ for $f\in T(m,n)$, where $\varepsilon\in X^*$ is the empty word;

$E_X(x) = (x,m\leftrightarrow n)$ for $x\in X(m,n)$.

(The last equality can be extended to embed X^* into $Fl_{X,T}$: $E_X(x) = (x,i(x)\leftrightarrow o(x))$ for $x\in X^*$.)

We do not insist on the algebraic rules satisfied by the flowchart scheme representations since this study is interesting only from a technical viewpoint. We only mention that an algebraic structure, called flow [10], has been singled out, which is preserved by passing from T to $Fl_{X,T}$, and that $Fl_{X,T}$ satisfies a universal property partially similar to that satisfied by polynomials (those interpretations of X and T in a flow that satisfy a certain supplementary condition can be naturally extended in a unique way to $Fl_{X,T}$).

1.4. Flownomials, flow-calculus

As we pointed out in § 1.2 a flowchart scheme represented by a picture in a normal form may also be represented by a formal expression of the particular form $((1_m + x_1 + \ldots + x_k)\cdot f)\uparrow^{i(x_1)+\ldots+i(x_k)}$. The final form of the calculus is obtained by allowing arbitrary formal expressions written with "+", "·" and "↑".

Flownomial expressions. Let X and T be as above. Define the sets $EXP_{X,T}(m,n)$ of flownomial X-expressions over T of type $m\to n$ as follows:

(i) atomic elements $x\in X(m,n)$ and $f\in T(m,n)$ are flownomial expressions of the type $m\to n$;

(ii) compound expressions: if $F:m + 1\to n + 1$, $F^1:m\to n$, $F^2:p\to q$ and $F^3:n\to q$ are flownomial expressions of the indicated type then $F^1 + F^2 : m + p \to n + q$, $F^1\cdot F^3 : m\to q$ and $F\uparrow : m\to n$ are flownomial expressions of the indicated type.

(iii) all flownomial expressions are obtained by using rules (i) and (ii).

A flownomial expression of the particular form $((1_m + x_1 + \ldots + x_k)\cdot f)\uparrow^r$, where $r = i(x_1) + \ldots + i(x_k)$ is said to be in a normal form; in the sequel we shall use the following

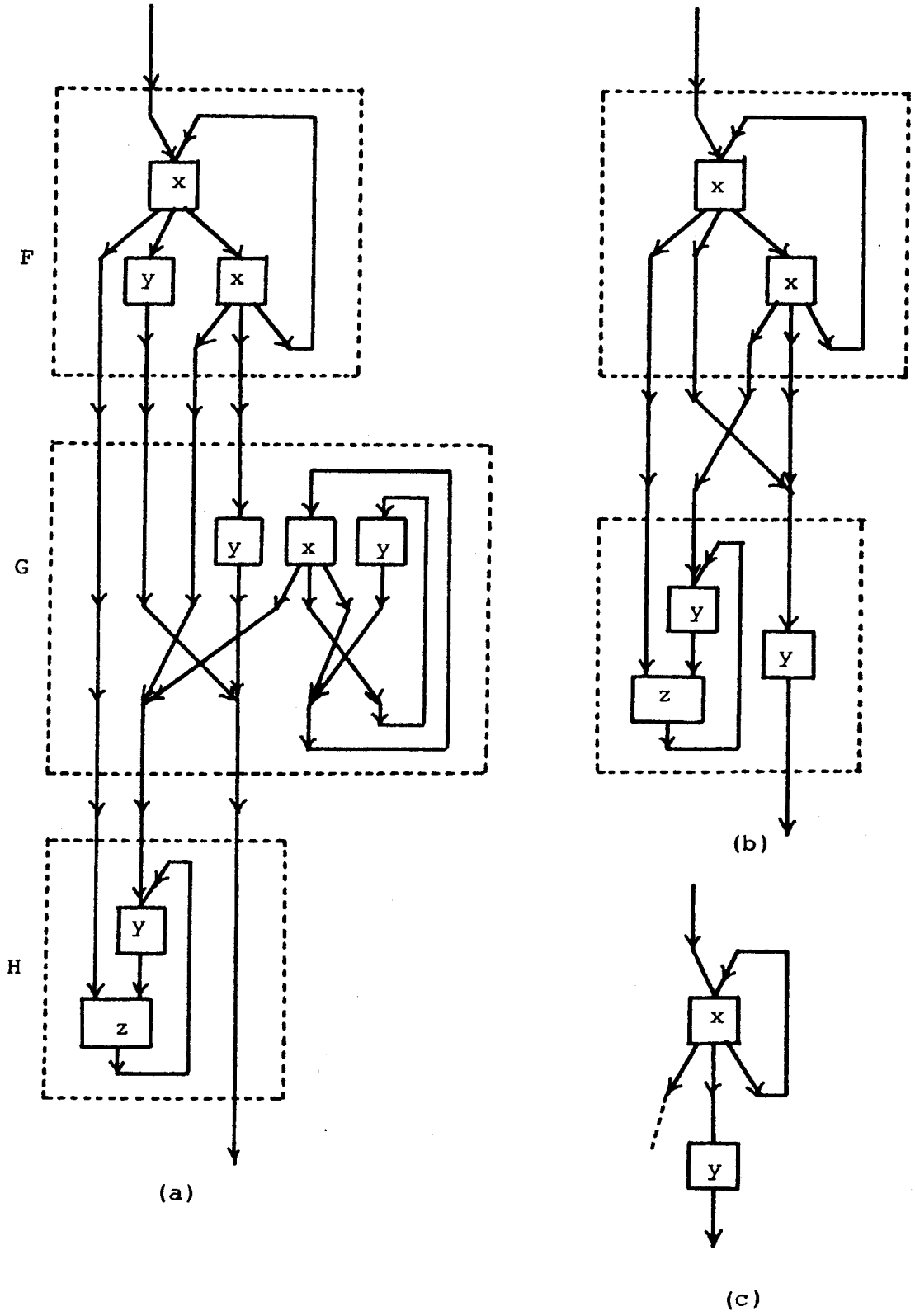


FIGURE 9
A scheme (a); its complete minimization (b); and its deterministic minimization (c).

standard notation: $\underline{x} = \sum_{j\leq k} x_j$, $i(\underline{x}) = \sum_{j\leq k} i(x_j)$, $o(\underline{x}) = \sum_{j\leq k} o(x_j)$; $\underline{x}' = \sum_{j\leq k'} x'_j$ etc. When T is closed with respect to $+, \cdot$ and $\uparrow$ and contains the block transpositions $m \leftrightarrow n$, every flownomial expression over T can be brought to a normal form by using the following rules:

(R1) replace subexpressions involving only elements in T by the corresponding value computed in T;

(R2) the normal form of $f \in T(m,n)$ is $(1_m \cdot f)\uparrow^0$ and of $x \in X(m,n)$ is $((1_m + x) \cdot m \leftrightarrow n)\uparrow^m$;

(R3) the normal form of $((1_m + \underline{x}) \cdot f)\uparrow^{i(\underline{x})} + ((1_p + \underline{x}') \cdot f')\uparrow^{i(\underline{x}')}$ is

$((1_{m+p} + \underline{x} + \underline{x}')[(1_m + p \leftrightarrow o(\underline{x}) + 1_{o(\underline{x}')})(f + f')(1_n + i(\underline{x}) \leftrightarrow q + 1_{i(\underline{x}')})])\uparrow^{i(\underline{x}+\underline{x}')}$;

(R4) the normal form of $((1_m + \underline{x}) \cdot f)\uparrow^{i(\underline{x})} \cdot ((1_n + \underline{x}') \cdot f')\uparrow^{i(\underline{x}')}$ is

$((1_m + \underline{x} + \underline{x}')[(f + 1_{o(\underline{x}')})(1_n + i(\underline{x}) \leftrightarrow o(\underline{x}'))(f' + 1_{i(\underline{x})})(1_q + i(\underline{x}') \leftrightarrow i(\underline{x}))])\uparrow^{i(\underline{x}+\underline{x}')}$;

(R5) the normal form of $((1_{m+1} + \underline{x}) \cdot f)\uparrow^{i(\underline{x})}\uparrow$ is

$((1_m + \underline{x})[(1_m + o(\underline{x}) \leftrightarrow 1)f(1_n + 1 \leftrightarrow i(\underline{x}))]\uparrow)\uparrow^{i(\underline{x})}$.

Using these rules every flownomial expression can be brought to a unique normal form, hence flownomial expressions in normal form give a complete and independent system of representations for the congruence relation R generated by the rules (R1 - 5) in the algebra of expressions $EXP_{X,T}$. In addition, it can be proved that the algebra of representations $Fl_{X,T}$ is isomorphic to the quotient algebra $EXP_{X,T}/R$. Consequently, in this enlarged frame we have the following identification:

representations by pairs = flownomial expressions in normal form.

The examples we shall give in this paper are related to the flowchart scheme in Figure 9.a. They use the variables $x \in X(1,3)$, $y \in X(1,1)$ and $z \in X(2,1)$. The support theory T is the theory of finite partial functions, i.e., T = **Pfn**. An element $f \in$ **Pfn**(m,n) is represented by the sequence of its values, i.e., $(f(1),f(2),...,f(m))_n$, where f(i) = "if f(i) = undefined then $\perp$ else f(i)", for $i \in [m]$. For instance, the function $f \in$ **Pfn**$(4,4)$ given by f(1) = 1, f(2) = f(4) = 3, f(3) = 2 is represented by $(1,3,2,3)_4$; $(1,\perp)_3$ represents the function $f \in$ **Pfn**$(2,3)$ given by f(1) = 1 and f(2) = undefined. (This representation of finite partial functions is not elegant, and is similar to the representation of natural numbers by bars, i.e., 7 = |||||| etc.)

Here we prove that the following identity holds in flow-calculus:

$$[(1 \vee 1 \cdot x)(1_2 + x)]\uparrow = [(1_1 + x + x)(5,1,2,6,3,4,5)_6]\uparrow^2,$$

i.e., the normal form of the left-hand-side expression is the right-hand-side expression. Note that the left-hand-side expression represents the top of the Figure 9.b.

Proof.

$$x = [(1_1 + x)1 \leftrightarrow 3]\uparrow = [(1_1 + x)(4,1,2,3)_4]\uparrow;$$

$$1 \vee 1 \cdot x = [1_2 \cdot (1,1)_1]\uparrow^0 \cdot [(1_1 + x)(4,1,2,3)_4]\uparrow^1 =$$

$$= \{(1_2 + x)[((1,1)_1 + 1_3)(1_1 + 0 \leftrightarrow 3)((4,1,2,3)_4 + 1_0)(1_3 + 1 \leftrightarrow 0)]\}\uparrow^{0+1} =$$

$$= [(1_2 + x)(4,4,1,2,3)_4]\uparrow;$$

$$1_2 + x = [1_2 \cdot (1,2)_2]\uparrow^0 + [(1_1 + x)(4,1,2,3)_4]\uparrow^1 =$$

$$= \{(1_3 + x)[(1_2 + 1 \leftrightarrow 0 + 1_3)((1,2)_2 + (4,1,2,3)_4)(1_2 + 0 \leftrightarrow 3 + 1_1)]\}\uparrow^{0+1} =$$

$$= [(1_3 + x)(1,2,6,3,4,5)_6]\uparrow;$$

$$(1 \vee 1 \cdot x)(1_2 + x) = [(1_2 + x)(4,4,1,2,3)_4]\uparrow^1 \cdot [(1_3 + x)(1,2,6,3,4,5)_6]\uparrow^1 =$$

$$= \{(1_2 + x + x)[((4,4,1,2,3)_4 + 1_3)(1_3 + 1 \leftrightarrow 3)((1,2,6,3,4,5)_6 + 1_1)(1_5 + 1 \leftrightarrow 1)]\}\uparrow^2 =$$

$$= [(1_2 + x + x)(6,6,1,2,7,3,4,5)_7]\uparrow^2;$$

$$[(1 \vee 1 \cdot x)(1_2 + x)]\uparrow = [(1_2 + x + x)(6,6,1,2,7,3,4,5)_7]\uparrow^2\uparrow =$$

$$= \{(1_1 + x + x)[(1_1 + 6 \leftrightarrow 1)(6,6,1,2,7,3,4,5)_7(1_4 + 1 \leftrightarrow 2)]\uparrow\}\uparrow^2 =$$

$$= [(1_1 + x + x)(5,1,2,6,3,4,5)_6]\uparrow^2.$$

2. SEMANTIC MODELS

The basic model for the study of semantics of deterministic flowchart schemes has been introduced by C.C. Elgot [15]. It consists in the following: Let S be the set of value-vectors denoting the states of memory in a computing device (the value-vectors in the registers of memory). A deterministic flowchart scheme F with m entries and n exits is interpreted via an interpretation I as a partial function $F_I : [m] \times S \to [n] \times S$ with the meaning that "$F_I(j,s)$ is defined and equal to (j',s')" iff "if the execution of the program obtained by interpreting F via I begins at entry j of the program with initial state of memory s, then the execution halts at exit j' of the program, the resulting state of memory being s'."

If we denote by

$$\mathbf{Pfn}(S)(m,n) = \{f \mid f : [m] \times S \to [n] \times S \text{ partial function}\}, \text{ for } m,n \in \mathbf{N}$$

we obtain a "theory", in a vague sense, **Pfn**(S) which is the basic semantic model in the deterministic case.

Note that in the particular case when S has exactly one element, **Pfn**(S) can be identified with **Pfn** defined above (§1.2.1). In this case the stress is laid on flow of control, whereas the memory state remains unchanged.

In a similar way the basic semantic model has been introduced in the nondeterministic case. A nondeterministic flowchart scheme F with m entries and n exits is interpreted, via

an interpretation I, as a relation $F_I \subseteq ([m] \times S) \times ([n] \times S)$ with the meaning that "$((j,s),(j',s')) \in F_I$" iff "if the execution of the program obtained by interpreting F via I begins at entry j of the program with initial state of memory s, then the execution may halt, on one variant, at exit j' of the program, the resulted state of memory being s'." If we denote by

$$\mathbf{Rel}(S)(m,n) = \{r \mid r \subseteq ([m] \times S) \times ([n] \times S)\}, \text{ for } m,n \in \mathbf{N}$$

then we obtain a theory **Rel**(S) which is the basic semantic model in the nondeterministic case.

As above, in the particular case when S has exactly one element, **Rel**(S) can be identified with **Rel** defined in §1.2.1.

In **Rel**(S) many operations and algebraic structures may be considered. The operations that interest us (sum, composition and feedback) have the following definitions.

For $r \in \mathbf{Rel}(S)(m,n)$ and $r' \in \mathbf{Rel}(S)(p,q)$ the sum $r + r' \in \mathbf{Rel}(S)(m+p, n+q)$ is defined by

$$r + r' = r \cup \{((m+j,s),(n+j',s')) \mid ((j,s),(j',s')) \in r'\}.$$

For $r \in \mathbf{Rel}(S)(m,n)$ and $r' \in \mathbf{Rel}(S)(n,p)$ the composite $r \cdot r' \in \mathbf{Rel}(S)(m,p)$ is the usual one, defined by

$$r \cdot r' = \{((j,s),(j',s')) \mid \exists (j_o,s_o) \in [n] \times S \text{ with } ((j,s),(j_o,s_o)) \in r \text{ and } ((j_o,s_o),(j',s')) \in r'\}.$$

In order to define the feedback let us note that every relation $r \in \mathbf{Rel}(S)(m,n)$ is given by a family of relations $r_{i,j} \subseteq S \times S$, for $i \in [m]$, $j \in [n]$, where $r_{i,j} = \{(s,s') \mid ((i,s),(j,s')) \in r\}$. Denote by r^* the reflexive-transitive closure of a relation $r \subseteq S \times S$, i.e., $r^* = 1_S \cup r \cup r^2 \ldots$, where $1_S = \{(s,s) \mid s \in S\}$. Using these facts, for $r \in \mathbf{Rel}(S)(m+1, n+1)$ the feedback $r\uparrow \in \mathbf{Rel}(S)(m,n)$ is defined by

$$(r\uparrow)_{i,j} = r_{i,j} \cup (r_{i,n+1} \cdot r^*_{m+1,n+1} \cdot r_{m+1,j}), \text{ for } i \in [m],\ j \in [n].$$

We finish this section by defining the natural embedding of **Rel** in **Rel**(S), given by the application

$$r \mapsto \{((i,s),(j,s)) \mid (i,j) \in r,\ s \in S\}.$$

Particularly, this application shows how various classes of finite relations in **Rel** (bijective functions, injective functions, etc.) can be thought of as being elements in an arbitrary **Rel**(S).

3. SYNTACTIC MODELS

In order to formalize some aspects regarding the study of flowchart schemes: isomorphism, accessibility, reduction, minimization with respect to the input (step-by-step) behaviour, coaccessibility, minimization with respect to the input-output (step-

-by-step) behaviour, the flow-calculus, introduced in §1.4 has to be augmented with some rules of identification for flownomials. It is an important test for this calculus whether the identifications corresponding to the natural aforementioned properties can be (easily) defined. This task can be done. The most interesting fact is that for the above properties there is a unique rule of identification (i.e., the equivalence relation generated by simulation) that has as particular cases the identification rules necessary for each property.

3.1. The simulation relation. Suppose we are given two flownomial expressions in normal form $F = ((1_m + x_1 + \dots + x_k) \cdot f)\uparrow^r$ and $F' = ((1_m + x'_1 + \dots + x'_{k'}) \cdot f')\uparrow^{r'}$ having the same type $m \to n$, and $u \in \mathbf{Rel}(k,k')$ (think of u as a relation between the statements $x_1,\dots,x_k$ of F and the statements $x'_1,\dots,x'_{k'}$ of F'). We say that F and F' are <u>in simulation</u> via u (in symbols $F \to_u F'$) if:

(i) $(j,j') \in u$ implies $x_j = x'_{j'}$;

(ii) the natural "block" extensions of u to the inputs of the statements, denoted i(u), and to the outputs of the statements, denoted o(u), fulfill

$$f \cdot (1_n + i(u)) = (1_m + o(u)) \cdot f' \ .$$

(this equality makes sense when T is closed under composition and the relations i(u), o(u) are "embedded" in T).

Let us explain in more details what we mean by "block" extensions and by "embedding". Suppose we are given the sequences $x_1,\dots,x_k$ and $x'_1,\dots,x'_{k'}$ and the relation $u \in \mathbf{Rel}(k,k')$ satisfying (i). Define the block extension of u to the inputs of the statements $i(u) \in \mathbf{Rel}(\sum_{j\le k} i(x_j), \sum_{j'\le k'} i(x'_{j'}))$ as follows: An $s \in [\sum_{j\le k} i(x_j)]$ can be written in a unique way as $s = \sum_{j<\alpha(s)} i(x_j) + \beta(s)$, where $\alpha(s) \in [k]$ and $\beta(s) \in [i(x_{\alpha(s)})]$ (read this: s is the input that has the number $\beta(s)$ of the statement that has the number $\alpha(s)$ in the sequence $x_1,\dots,x_k$). Similarly, every $s' \in [\sum_{j'\le k'} i(x'_{j'})]$ can be written as $s' = \sum_{j'<\alpha'(s')} i(x'_{j'}) + \beta'(s')$. Now the relation i(u) is defined by

$$i(u) = \{(s,s') \mid (\alpha(s), \alpha'(s')) \in u \text{ and } \beta(s) = \beta'(s')\} \ .$$

The block extension of u to outputs $o(u) \in \mathbf{Rel}(\sum_{j\le k} o(x_j), \sum_{j\le k'} o(x'_j))$ is defined in a similar way.

At a first stage we can translate "embedding" by "inclusion". Later on we shall give a more general meaning to "embedding" that contains, as a particular case, the embedding of **Rel** in **Rel**(S) defined in Section 2.

The meaning of $F \to_u F'$ depends on the type of u and will be given below for each particular class of relations used for u. We only mention here that this notion of simulation is the result of a historical process aiming to formalize some flowchart scheme properties. Initially we had found that isomorphism and reduction can be captured

using simulations via bijective and surjective function, respectively. Later on we found that accessibility could also be modelled by simulation, namely by simulation via injective functions, and the input (step-by-step) behaviour could be captured using simulations via functions. Coaccessibility can be modelled by simulation via converses of injective functions, and the input-output (step-by-step) behaviour, in the deterministic case, can be captured using simulations via partially defined functions.

3.2. Equivalences generated by simulations. For a subset A of **Rel** let us denote by $\rightarrow_A$ the simulation via A-relations, namely "$F \rightarrow_A F'$ iff there exists u in A such that $F \rightarrow_u F'$ ", and by $=_A$ the equivalence relation generated by $\rightarrow_A$. By the above comments it follows that the most interesting subsets A of **Rel** are: **Bi** (bijective functions), **In** (injective functions), **Sur** (surjective functions), $\mathbf{Fn}$, $\mathbf{In}^{-1}$ (converses of injective functions), $\mathbf{Pfn}$, $\mathbf{Sur}^{-1}$ (converses of surjective functions), and **Rel**.

In the case when A is closed with respect to sum and composition, $=_A$ is a congruence relation, hence the operations can be defined in the quotient structure $Fl_{X,T}/=_A$. The resulting algebraic structures $Fl_{X,T}/=_A$, for certain X,T and A, are the basic syntactic models for flowchart scheme theory.

4. FLOWCHART SCHEMES

In section § 1.1 we emphasized that more representations by pairs (or equivalently, flownomial expressions in normal form) corresponds to a flowchart scheme, the difference being generated by the way the statements of the scheme are linearly ordered. This observation suggests the identification of a flowchart scheme with the class of its representations. The mathematical formulation of the fact that two representations represent the same flowchart-picture is captured by the simulation via bijective functions.

4.1. The simulation via bijective functions (isomorphism). Suppose the support theory T "contains" bijective finite functions. The meaning of the wording "contain" will be specified later on. In the usual case T is a subtheory of **Rel**, hence the meaning is clear: $T \supseteq \mathbf{Bi}$.

The definition of simulation via bijective functions is obtained from the general definition, given in § 3.1, using for u morphisms in **Bi**.

In the particular case when T is a subtheory of **Rel** the meaning of the simulation "$F \rightarrow_u F'$ with u in **Bi**" is "F and F' represent the same flowchart scheme, the bijection u doing the connection between the linearly ordered statements of F and of F'." Therefore, the simulation via bijective functions can be named "isomorphism".

Now we turn back to the general setting. For the congruence relation $=_{\mathbf{Bi}}$, generated by $\rightarrow_{\mathbf{Bi}}$, the following equivalent characterizations can be given:

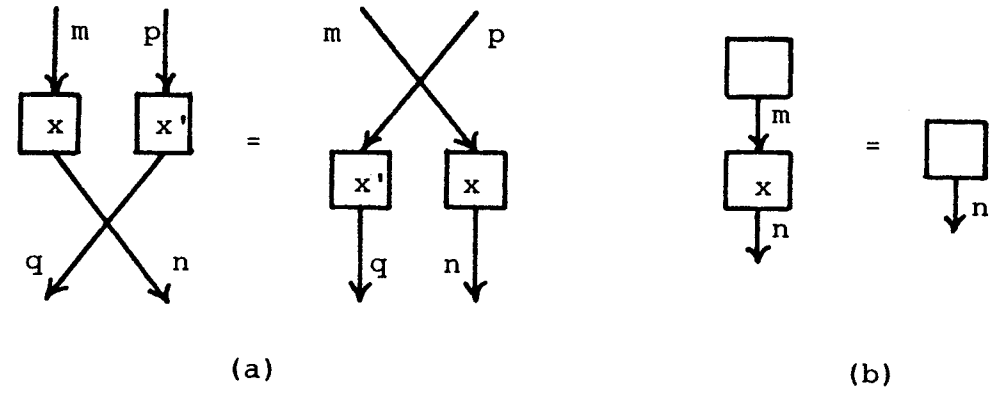


FIGURE 10
Simple generators for simulation: (a)-(e).

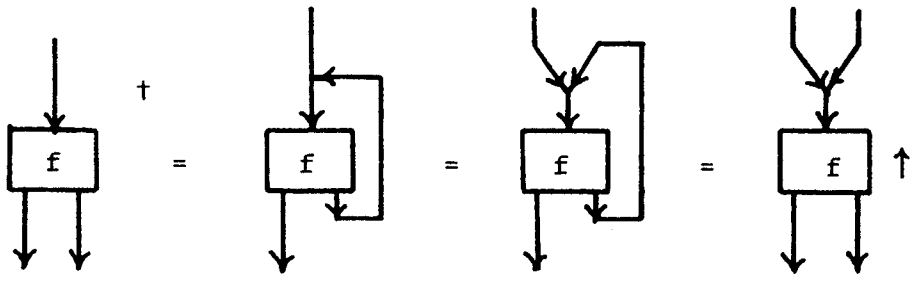


FIGURE 11
Iteration ⇒ Feedback + Tupling.

(i) $=_{Bi} = \rightarrow_{Bi}$ (hence $\rightarrow_{Bi}$ is a congruence);

(ii) $=_{Bi}$ is the congruence relation generated by the identifications

($\leftrightarrow$X) $(x + x') \cdot n \leftrightarrow q =_{Bi} m \leftrightarrow p \cdot (x' + x)$, where $x \in X(m,n)$ and $x' \in X(p,q)$

(see Figure 10.a).

4.2. The mathematical concept of flowchart schemes. The above facts show that, in the case $T \subseteq$ **Rel**, a flowchart scheme can be identified with an element in the quotient structure $Fl_{X,T}/=_{Bi}$. Generalizing, we say

> the elements in a $Fl_{X,T}/=_{Bi}$ are (abstract) flowchart schemes.

4.3. The algebra of flowchart schemes (biflow). We have selected some identities, written in terms of "+", "·", "$\uparrow$", 1_m and $m \leftrightarrow n$, and satisfied by flowchart schemes, in order to define an algebraic structure, called biflow. The identities are listed in Table I and illustrated in Figure 12. The main point is that this set of identities is complete, i.e. they suffice to prove that flownomial expressions over **Rel**, which represent the same flowchart scheme, are equal. Consequently, the identities, listed in Table I, completely characterizes flowchart schemes from the algebraic point of view.

In more details, a biflow B is an abstract structure given by:

a family of sets $\{B(m,n)\}_{m,n \geq 0}$; two types of distinguished morphisms $1_n \in B(n,n)$, $m \leftrightarrow n \in B(m + n, n + m)$; three operations: composition $\cdot : B(m,n) \times B(n,p) \rightarrow B(m,p)$, sum $+ : B(m,n) \times B(p,q) \rightarrow B(m + p, n + q)$ and feedback $\uparrow : B(m + 1, n + 1) \rightarrow B(m,n)$

and satisfying the identities listed in Table I. The axioms (B1-6) show that a biflow B is a strict monoidal category; (B7-10) show that B is a symmetric strict monoidal category, therefore the finite bijective functions are embedded in B; (B11-15) axiomatize the feedback.

Semantic models: **Rel**(S) and all of its subtheories, which contain the embedding of **Bi** in **Rel**(S) (cf. § 2), are biflows. Particularly, **Bi**, **In**, **PSur** (partial, surjective functions) **Pfn** and **Rel** are biflows.

Syntactic models: If T is a biflow, then $Fl_{X,T}/=_{Bi}$ is a biflow.

Generally, the support theory T for the flowchart schemes which interest us, has at least a structure of biflow. Since **Bi** is an initial biflow (in the sense of category theory: for every biflow B there exists a unique morphism of biflows from **Bi** to B), the initial wording "the support theory T contains bijections" gets a precise meaning, when T is a biflow.

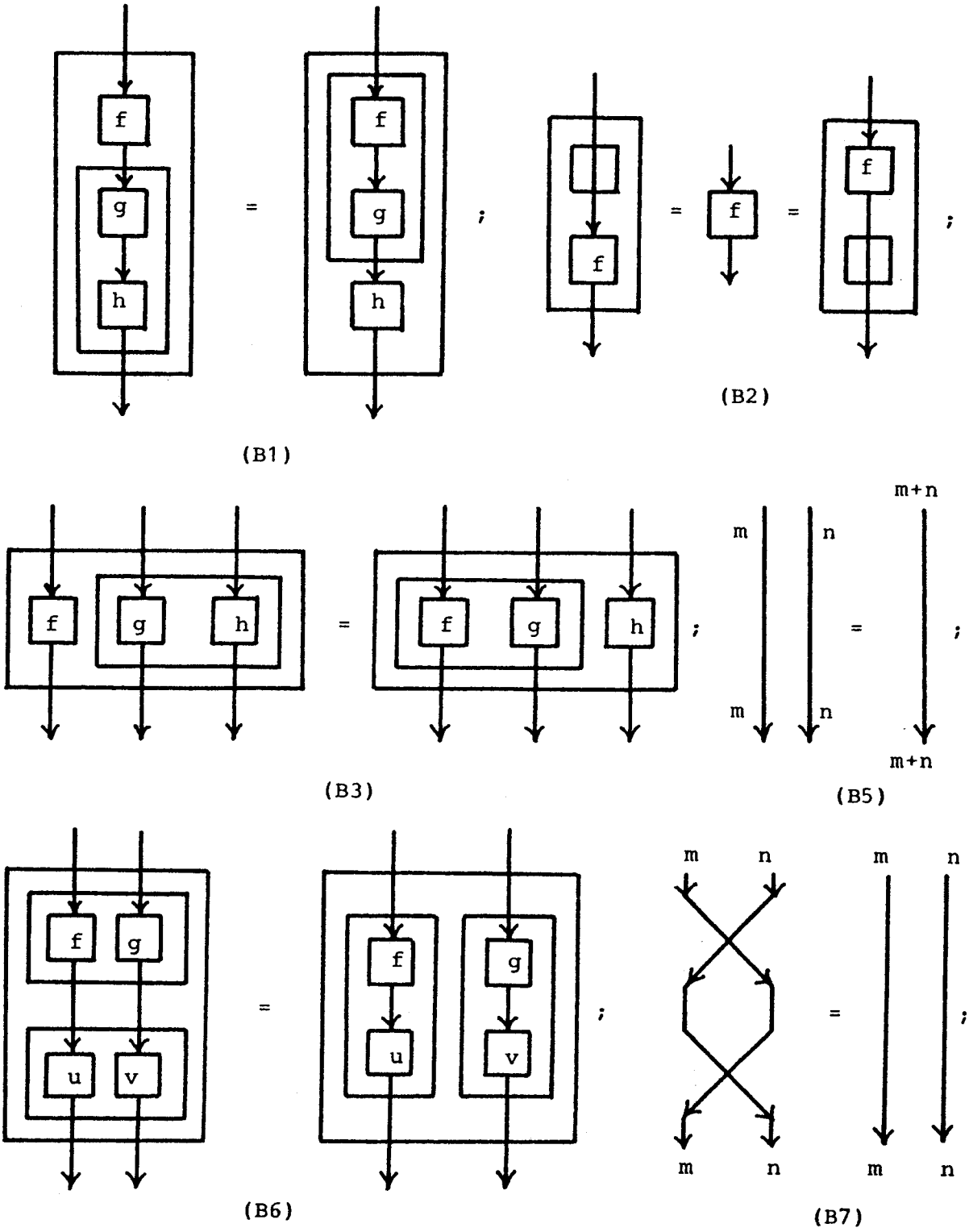
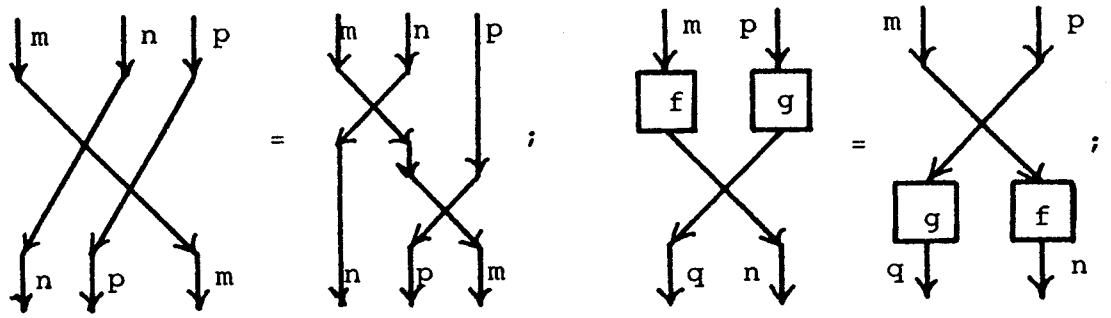
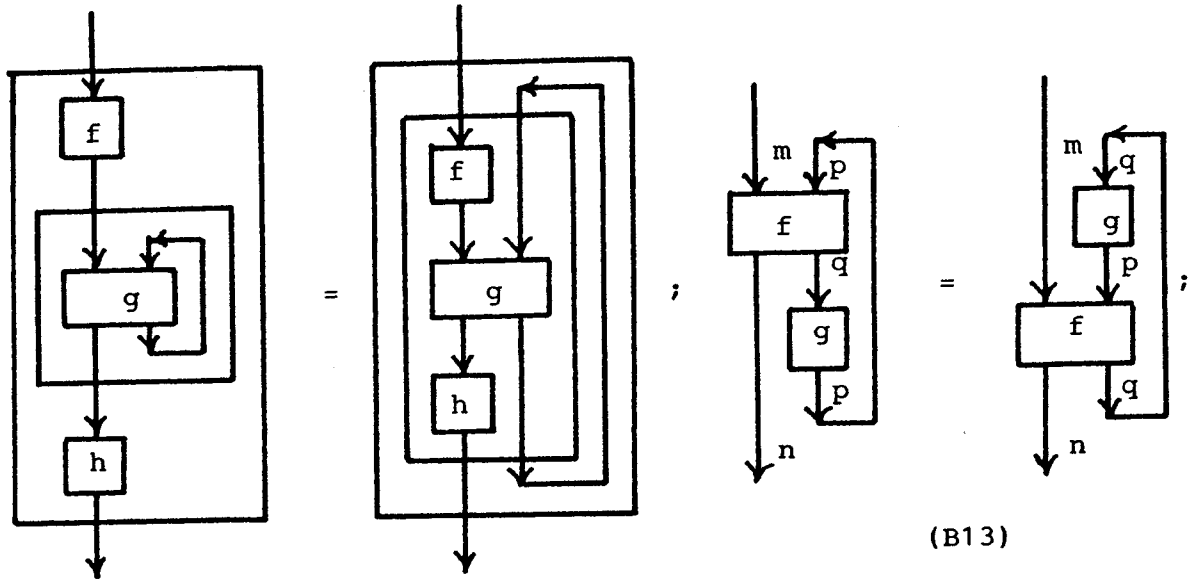


FIGURE 12
The axioms that define a biflow (axioms (B4), (B8), and (B14) cannot be illustrated).

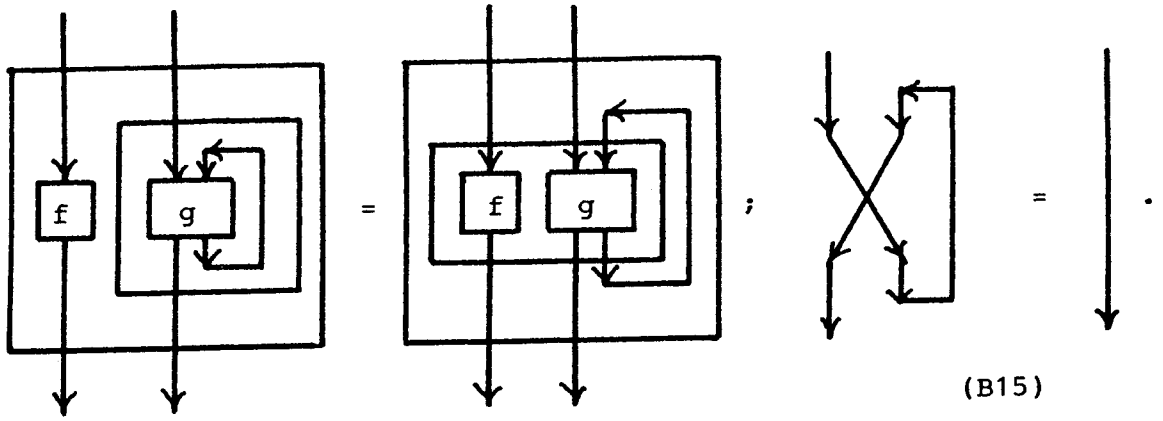


(B9) (B10)

+ + +



(B11) (B13)



(B12) (B15)

FIGURE 12 (comtinued).

TABLE I.

These axioms define a biflow	
(B1) $(fg)h = f(gh)$	(B9) $m \leftrightarrow (n + p) = (m \leftrightarrow n + 1_p)(1_n + m \leftrightarrow p)$
(B2) $1_m f = f = f\, 1_n$	(B10) $(f + g) \cdot n \leftrightarrow q = m \leftrightarrow p \cdot (g + f)$ for $f : m \to n$, $g : p \to q$
(B3) $(f + g) + h = f + (g + h)$	
(B4) $1_0 + f = f = f + 1_0$	(B11) $f(g\uparrow^p)h = ((f + 1_p)g(h + 1_p))\uparrow^p$
(B5) $1_m + 1_n = 1_{m+n}$	(B12) $(f + g)\uparrow^p = f + g\uparrow^p$
(B6) $(f + g)(u + v) = fu + gv$ for $m \xrightarrow{f} n \xrightarrow{u} p$, $m' \xrightarrow{g} n' \xrightarrow{v} p'$	(B13) $(f(1_n + g))\uparrow^p = ((1_m + g)f)\uparrow^q$ for $f : m + p \to n + q$, $g : q \to p$
(B7) $m \leftrightarrow n \cdot n \leftrightarrow m = 1_{m+n}$	(B14) $1_1\uparrow = 1_0$
(B8) $0 \leftrightarrow n = 1_n = n \leftrightarrow 0$	(B15) $1 \leftrightarrow 1\uparrow = 1_1$

4.4. The universal property. In order to get an interpretation of flownomial expressions in $EXP_{X,T}$ (or representations in $Fl_{X,T}$) in a biflow B we have to interpret the variable in X using a rank-preserving application $I_X : X \to B$ (i.e., $x \in X(m,n) \mapsto I_X(x) \in B(m,n)$) and the morphisms in T using a morphism of biflows $I_T : T \to B$ (i.e., I_T is given by a family of applications $I_T : T(m,n) \to B(m,n)$ which preserve the constants 1_m, $m \leftrightarrow n$ and the operations "+", "·", and "$\uparrow$"). Now the interpretation of a flownomial expression in normal form $F = ((1_m + x_1 + \ldots + x_k) \cdot g)\uparrow^r \in EXP_{X,T}(m,n)$ is $(I_X,I_T)^f(F) \in B(m,n)$, given by

$$(I_X,T_T)^f(F) = ((1_m + I_X(x_1) + \ldots + I_X(x_k)) \cdot I_T(g))\uparrow^r .$$

(Of course, the restriction to normal form is inessential.)

The above formula makes sense in each abstract structure B endowed with 1_m, "+", "·", and "$\uparrow$". The reasons we have taken a biflow B are twofold: (1) the interpretation $(I_X,I_T)^f$ has to commute with the operations, and (2) the $=_{Bi}$-equivalent flownomial expressions should have the same interpretation. The latter statement shows that the extension $(I_X,I_T)^f$ makes sense for $Fl_{X,T}/=_{Bi}$ too, and in that case we denote the corresponding application by $(I_X,I_T)^{bf} : Fl_{X,T}/=_{Bi} \to B$.

In the standard cases T is a subtheory of **Rel**, B is a subtheory in a **Rel**(S), I_X gives the semantics for each statement $x \in X$, and I_T is the restriction to T of the embedding of **Rel** into **Rel**(S). In this cases the interpretation $(I_X,I_T)^f(F)$ gives the behaviour of the program obtained by interpreting via I_X the flowchart scheme corresponding to the flownomial expression F.

Let (E^b_X, E^b_T) be the embedding of (X,T) into $Fl_{X,T}/=_{Bi}$ obtained using the embedding (E_X, E_T) of (X,T) in $Fl_{X,T}$, defined in §1.3, and the canonical projection from $Fl_{X,T}$ to $Fl_{X,T}/=_{Bi}$. The universal property satisfied by $Fl_{X,T}/=_{Bi}$ is similar to that satisfied by the polynomials, namely

"for every biflow B and every interpretation (I_X,I_T) of (X,T) in B there exists a unique morphism of biflows $I^{bf} : Fl_{X,T}/=_{Bi} \to B$ (namely, $(I_X,I_T)^{bf}$ defined above) such that $E^b_X \cdot I^{bf} = I_X$ and $E^b_T \cdot I^{bf} = I_T$."

In a categorical language this property shows that $Fl_{X,T}/=_{Bi}$ is the coproduct, in the category of biflows, of the biflow T and the biflow freely generated by X .

4.5. Bi-flow-calculus. The calculus with flownomials associated to flowchart schemes, called bi-flow-calculus, is obtained by adding to the rules (R1-5), that define the flow-calculus (i.e., the calculus for representations introduced in §1.4), the rule which consists in the identification of $=_{Bi}$-equivalent flownomial expressions.

Another method to define the same bi-flow-calculus is to consider flownomial X-expressions over T together with the algebraic rules that define a biflow. More precisely, the calculus is defined by the rule (R1) in §1.4 and (B1-4, B6, B10-13) in Table I. (Since T is a biflow, the other rules (B5, B7-9, B14-15) are covered by (R1).)

Example. The following identity holds in bi-flow-calculus

$$(1V1 \cdot x(1_1 + y + x))\uparrow (1_3 + y)(1,3,2,3)_3 = (1V1 \cdot x(1_2 + x))\uparrow (1,3,2,4)_4(1_2 + (y + y)1V1) .$$

(a) Proof using normal forms. As in example in §1.4 the normal form of the left-hand side expression is $NF_1 = [(1_1 + x + y + x + y)(4,1,5,6,3,2,7,4,3)_7]\uparrow^4$ and of the right-hand side expression is $NF_2 = [(1_1 + x + x + y + y)\ (4,1,6,5,2,7,4,3,3)_7]\uparrow^4$. We shall prove that $NF_1 \to_u NF_2$ for the bijection $u = (1,3,2,4)_4$. Note that u preserves the statements with respect to the sequences (x,y,x,y) and (x,x,y,y), hence condition (i) in definition §3.1 holds. The extension of u to inputs is $(1,3,2,4)_4$ and to outputs is $(1,2,3,7,4,5,6,8)_8$. Since $(4,1,5,6,3,2,7,4,3)_7(1_3 + (1,3,2,4)_4) = (4,1,6,5,3,2,7,4,3)_7 = (1_1 + (1,2,3,7,4,5,6,8)_8) \cdot (4,1,6,5,2,7,4,3,3)_7$ condition (ii) in definition §3.1 holds, too.

(b) Proof using the algebraic rules (without marking (R1) and (B1-4)).

$$\begin{aligned}
&(1V1 \cdot x(1_1 + y + x))\uparrow (1_3 + y)(1,3,2,3)_3 \\
&\quad = [1V1 \cdot x(1_2 + x)((1_1 + y + 1_2) + 1_1)]\uparrow (1_3 + y)(1,3,2,3)_3 && \text{by B6} \\
&\quad = (1V1 \cdot x(1_2 + x))\uparrow (1_1 + y + 1_2)(1_3 + y)(1,3,2,3)_3 && \text{by B11} \\
&\quad = (1V1 \cdot x(1_2 + x))\uparrow (1_1 + y + 1_1 + y)(1_1 + 1 \leftrightarrow 1 + 1_1)(1_2 + 1V1) && \text{by B6}
\end{aligned}$$

$= (1V1 \cdot x(1_2 + x))\uparrow(1_1 + 1 \leftrightarrow 1(1_1 + y) + y)(1_2 + 1V1)$ by B6, B10

$= (1V1 \cdot x(1_2 + x))\uparrow(1,3,2,4)_4(1_2 + (y + y) \cdot 1V1)$ by B6 .

5. ACCESSIBILITY

A flowchart scheme is a notation for a sequential computation process. In the process of computation only the vertices that can be reached by paths going from inputs matter; these vertices form the accessible part of the scheme. A flowchart scheme will be called accessible if it coincides with its accessible part. Here we regard as equivalent two flowchart schemes that have the same accessible part. In a formal approach accessibility is captured by simulation via injective functions.

5.1. The simulation via injective functions; the resulting congruence. Suppose that the support theory T "contains" injective finite functions. In the case $T \subseteq \mathbf{Rel}$ this means $T \supseteq \mathbf{In}$.

The definition of simulation via injective functions is obtained from the general definition, given in §3.1, using for u morphisms in **In**.

In the particular case when T is a subtheory of **Rel** the meaning of the simulation "$F \xrightarrow[u]{} F'$ with u in **In**" is "F' can be obtained from F by adding a part inaccessible from F, namely that corresponding to the vertices that are not in the image of u ". Of course, the relation $\rightarrow_{\mathbf{In}}$ is not symmetric, the meaning of the converse relation $F' \xleftarrow[u]{} F$ being "F can be obtained from F' by deleting the part corresponding to the complement of the image of u; this part is not accessible from the remaining one".

Let $\sim$ be a sorted equivalence on T (i.e. a family $\sim_{m,n}$ of equivalence relations on T(m,n)). A morphism $y \in T(p,q)$ is called $\sim$-functorial if for every $f \in T(m+p,n+p)$ and $g \in T(m+q,n+q)$ the condition $f(1_n + y) \sim (1_m + y)g$ implies $f\uparrow^p \sim g\uparrow^q$. Functorial means =-functorial for the relation of equality =.

Now we turn back to the general setting. For the congruence relation $=_{\mathbf{In}}$, generated by $\rightarrow_{\mathbf{In}}$, the following equivalent characterizations can be given:

(i) $=_{\mathbf{In}} = \mathbf{In}^{\leftarrow} \cdot \mathbf{In}^{\rightarrow}$;

(ii) $=_{\mathbf{In}}$ is the least congruence relation generated by the identifications ($\leftrightarrow$X) in §4.1 and the identifications

$$((1_m + \mathbf{x})f)\uparrow^{i(x)} = ((1_m + \mathbf{x} + \mathbf{y})g)\uparrow^{i(x+y)} \quad \text{when } f(1_n + 1_{i(x)} + O_{i(y)}) = (1_m + 1_{o(x)} + O_{o(y)})g ,$$

where **x** and **y** are finite sums of variables;

(iii) $=_{\mathbf{In}}$ is the least congruence relation generated by the identifications ($\leftrightarrow$X) and the identifications

(OX) $O_m \cdot x = O_n$, where $x \in X(m,n)$ (see Figure 10.b)

in the class of congruence relations $\sim$ satisfying: every injection is $\sim$-functorial.

Comments. By (i) two flowchart schemes are $=_{In}$-equivalent iff they can be transformed into the same scheme by deleting inaccessible parts. In (ii), by using separate simulations via bijective functions, we can suppose that the injective function u has the particular form $1_r + O_s$, and, in this case, the meaning of the formula of simulation is much clearer. Much more interesting is the characterization (iii), since it reduces the generators to $(\leftrightarrow X) + (OX)$ by restricting the class of the congruence relations used to generate $=_{In}$.

5.2. The mathematical concept of accessible flowchart scheme. The above facts show that, in the case $T \subseteq \mathbf{Rel}$, every equivalence class with respect to $=_{In}$, contains an accessible flowchart scheme, unique up to an isomorphism. Consequently we can identify an accessible flowchart scheme to its $=_{In}$-equivalence class. Generalizing we say

the elements in a $Fl_{X,T}/=_{In}$ are accessible flowchart schemes.

5.3, 5.4. We do not insist on the algebraic rules satisfied by accessible flowchart schemes. We only mention that the corresponding algebraic structure, called inflow, is a biflow, contains injections (in order to generate injections we use the distinguished morphisms $O_n : 0 \to n$), and satisfies:

(I1) $O_m \cdot f = O_n$, for $f : m \to n$;

(I2) every injection is functorial.

5.5. In flow-calculus. The calculus with flownomials associated to accessible flowchart schemes, called in-flow-calculus, is obtained by adding to the rules that define the bi-flow-calculus in §4.5 the rule which consists in the identification of $=_{In}$-equivalent expressions.

For the algebraic version, we add the rules (I1-2) above to the rules (R1, B1-4, B6, B10-13) in §4.5 that define algebraically the bi-flow-calculus.

Example. In in-flow-calculus the following identity holds:

$$((1_3 + y + x + y)(1,3,2,3,2,5,4,4)_5)\uparrow^2 = (1_3 + y)(1,3,2,3)_3 .$$

(a) Proof using normal forms. The normal form of the left-hand side expression (G in Figure 9) is $NF_1 = [(1_4 + y + x + y)(1,3,2,4,3,2,6,5,5)_6]\uparrow^3$ and of $(1_3 + y)(1,3,2,3)_3$ is $NF_2 = [(1_4 + y)(1,3,2,4,3)_4]\uparrow^1$. We shall prove that $NF_2 \to_u NF_1$ for the injection $u = 1_1 + O_2 = (1)_3$. Note that u preserves the statements with respect to the sequences (y) and (y,x,y), hence condition (i) in definition §3.1 holds. The extension of u to input is $(1)_3$ and the extension to outputs is $(1)_5$. Since $(1,3,2,4,3)_4(1_3 + (1)_3) = (1,3,2,4,3)_6 = (1_4 + (1)_5) \cdot (1,3,2,4,3,2,6,5,5)_6$ the condition (ii) in definition §3.1 holds, too.

(b) Proof using algebraic rules (marking the application of the new rules (I1-2) only). Note that $O_2(x + y) = (O_1 + O_1)(x + y) = O_1x + O_1y =$ (by I1) $O_3 + O_1 = O_4$, hence

$$(1_4 + \boxed{O_2})(1_3 + y + x + y)(1,3,2,3,2,5,4,4)_5 = (1_3 + y + O_4)(1,3,2,3,2,5,4,4)_5 =$$
$$= (1_3 + y)(1_4 + O_4)(1,3,2,3,2,5,4,4)_5 = (1_3 + y)(1,3,2,3)_3(1_3 + \boxed{O_2}).$$

Using (I2) we obtain

$$[(1_3 + y + x + y)(1,3,2,3,2,5,4,4)_5]\uparrow^2 = [(1_3 + y)(1,3,2,3)_3]\uparrow^0$$

hence the conclusion follows.

6. REDUCTION

We repeat: a flowchart scheme is a notation for a sequential computation process. Hence the result of the computation depends on the sequences of statements to be executed only. The (step-by-step) behaviour of a vertex in a flowchart scheme is the set of all finite and infinite sequences of statements that can be executed beginning with the given vertex. In a flowchart scheme we can identify the vertices that have the same behaviour and obtain a flowchart scheme that denotes the same computation process. A flowchart scheme will be called reduced if it has no different vertices having the same behaviour. Here we regard as equivalent two flowchart schemes that can be reduced to the same scheme by identifying vertices with the same behaviour. In a formal approach reduction is captured by simulation via surjective functions.

6.1. The simulation via surjective functions; the resulting congruence. Suppose that the support theory T "contains" surjective, finite functions. In the case $T \subseteq$ **Rel** this means $T \supseteq$ **Sur**.

The definition of simulation via surjective functions is obtained from the general definition, given in §3.1, by using for u morphisms in **Sur**.

In the particular case when T is a subtheory of **Rel** the meaning of the simulation "$F \xrightarrow[u]{} F'$ with u in **Sur**" is "F' can be obtained from F by identifying vertices which have the same label and whose output connections are equal after identification". Of course, the relation $\rightarrow_{\mathbf{Sur}}$ is not symmetric, the meaning of the converse relation $F' \xleftarrow[u]{} F$ being "F can be obtained from F' by (partially) unfolding some vertices".

Now we turn back to the general setting. For the congruence relation $=_{\mathbf{Sur}}$, generated by $\rightarrow_{\mathbf{Sur}}$, the following equivalent characterizations can be given:

(i) $=_{\mathbf{Sur}} = \rightarrow_{\mathbf{Sur}} \cdot {}_{\mathbf{Sur}}\leftarrow$;

(ii) $=_{\mathbf{Sur}}$ is the least congruence relation generated by the identifications ($\leftrightarrow$X) and the identifications

(VX) $mVm \cdot x = (x + x) \cdot nVn$, where $x \in X(m,n)$ (see Figure 10 c)

in the class of the congruence relations $\sim$ satisfying: every surjection is $\sim$-functorial.

Comments. By (i) two flowchart schemes are $=_{Sur}$-equivalent iff they can be reduced to the same scheme by identifying certain vertices. The characterization (ii) gives very simple generators for $=_{Sur}$ by restricting the class of congruence relations used to generate $=_{Sur}$.

6.2. The mathematical concept of reduced flowchart scheme. The above facts show in the case $T \subseteq Rel$ every equivalence class, with respect to $=_{Sur}$, contains a reduced flowchart scheme, unique up to an isomorphism. Consequently, we can identify a reduced flowchart scheme to its $=_{Sur}$-equivalence class. Generalizing we say:

the elements in a $Fl_{X,T}/=_{Sur}$ are reduced flowchart schemes.

6.3, 6.4. We do not exhibit all of the algebraic rules satisfied by reduced flowchart schemes. We only mention that the corresponding algebraic structure, called surflow, is a biflow, contains surjections (in order to generate surjections we use the distinguished morphisms $mVm : m + m \to m$), and satisfies:

(S1) $mVm \cdot f = (f + f) \cdot nVn$, for $f : m \to n$;

(S2) every surjection is functorial.

6.5. Sur-flow-calculus. The calculus with flownomials associated to reduced flowchart schemes, called sur-flow-calculus, is obtained by adding to the bi-flow-calculus the rule which consists in the identification of $=_{Sur}$-equivalent expressions.

For the algebraic version, we add the rules (S1-2) above to the rules that algebraically define the bi-flow-calculus in § 4.5.

Examples. In sur-flow-calculus the following identities hold:

(a) $(1V1 \cdot x(1_2 + x))\uparrow (\bot,1,\bot,1)_1 = (1V1 \cdot x)\uparrow (\bot,1)_1$;

(b) $(1V1 \cdot x(1_1 + y + x))\uparrow (1_3 + y)(1,3,2,3)_3 = (1V1 \cdot x(1_2 + x))\uparrow (1,3,2,3)_3(1_2 + y)$.

Proof of (a) using normal forms. The normal form of the left-hand side expression is $NF_1 = [(1_1 + x + x)(2,\bot,1,3,\bot,1,2)_3]\uparrow^2$ and that of the right-hand side expression is $NF_2 = [(1_1 + x)(2,\bot,1,2)_2]\uparrow^1$. We shall prove that $NF_1 \to_u NF_2$ for the surjection $u = (1,1)_1$. Note that u preserves the statements with respect to the sequences (x,x) and (x), hence the condition (i) in definition § 3.1 holds. The extension of u to inputs is $(1,1)_1$ and the extension to outputs is $(1,2,3,1,2,3)_3$. Since $(2,\bot,1,3,\bot,1,2)_3(1_1 + (1,1)_1) = (2,\bot,1,2,\bot,1,2)_2 = (1_1 + (1,2,3,1,2,3)_3)(2,\bot,1,2)_2$ the condition (ii) in definition § 3.1 holds, too.

Proof of (b) using algebraic rules. By the example in § 4.5 the left-hand side expression is equal to

$$(1V1 \cdot x(1_2 + x))\uparrow (1,3,2,4)_4(1_2 + (y + y) \cdot 1V1)$$
$$= (1V1 \cdot x(1_2 + x))\uparrow (1,3,2,4)_4(1_2 + 1V1 \cdot y) \qquad \text{by (S1)}$$
$$= (1V1 \cdot x(1_2 + x))\uparrow (1,3,2,3)_3(1_2 + y).$$

7. THE INPUT BEHAVIOUR (COMPLETE MINIMIZATION)

A flowchart scheme denotes a sequential computation process. For an input of the scheme let us consider the set of finite and infinite sequences of statements that can be executed beginning with this input. (In the case when $i(x) = 1$, $\forall x \in X$ this set can be identified with the tree obtained by completly unfolding the scheme beginning with the given input.) By (step-by-step) input behaviour of a flowchart scheme we mean the tuple of the sets obtained as above for each input. It is natural to regard as equivalent two flowchart schemes that have the same input behaviour. In the class of the flowchart schemes that have the same input behaviour there is a minimal one, unique up to an isomorphism. This minimization preserves the completeness of the scheme, namely the minimal flowchart scheme in $Fl_{X,Pfn}$ of a scheme over **Fn** is over **Fn**, too — hence the name. In a formal approach the (step-by-step) input behaviour is captured by simulation via functions.

7.1. The simulation via functions; the resulting congruence. Suppose that the support theory T "contains" functions. In the case $T \subseteq$ **Rel** this means $T \supseteq$ **Fn**.

The definition of simulation via functions is obtained from the general definition, given in §3.1, by using for u morphisms in **Fn**.

In the general case $\rightarrow_{\mathbf{Fn}} \subseteq \rightarrow_{\mathbf{Sur}} \cdot \rightarrow_{\mathbf{In}}$, hence in the case $T \subseteq$ **Rel** the meaning of the simulation "$F \rightarrow_u F'$ with u in **Fn**" is "F' can be obtained from F in two steps: first by identifying vertices with common labels and coherent continuations, and second by adding inaccessible vertices". The meaning of the equivalence relation $=_{\mathbf{Fn}}$, generated by $\rightarrow_{\mathbf{Fn}}$, is "$F =_{\mathbf{Fn}} F'$" iff "F and F' have the same (step-by-step) input behaviour (or equivalently, by completly unfolding F and F' we get the same tuple of trees)" iff "by identifying vertices and deleting inaccessible ones F and F' can be transformed into the same minimal flowchart (with respect to the input behaviour)".

For the congruence relation $=_{\mathbf{Fn}}$, generated by $\rightarrow_{\mathbf{Fn}}$, the following equivalent characterization can be given:

(i) $\quad =_{\mathbf{Fn}} = \rightarrow_{\mathbf{Sur}} \cdot \mathbf{In}^{\leftarrow} \cdot \rightarrow_{\mathbf{In}} \cdot \mathbf{Sur}^{\leftarrow}$;

(ii) $\quad =_{\mathbf{Fn}}$ is the least congruence relation generated by the identifications ($\leftrightarrow$X) in §4.1, (OX) in §5.1 and (VX) in §6.1 in the class of congruence relations $\sim$ satisfying: every function is $\sim$-functorial.

Comments. By (i) two flowchart schemes are $=_{\mathbf{Fn}}$-equivalent iff by identifying vertices and deleting inaccessible ones they can be transformed into the same scheme. Again in

statement (ii) we get very simple generators (now for $=_{Fn}$) restricting the class of congruence relations used for generation.

7.2. Computation processes (or minimal flowchart schemes with respect to the input behaviour). In the case $T \subseteq Rel$ every $=_{Fn}$-equivalence class has a minimal flowchart, unique up to an isomorphism. Since two schemes are $=_{Fn}$-equivalent iff they have the same computation sequences, we can identify such a class to a computation process that consist in finite and infinite sequences of statements. Generalizing we say

> The elements in a $Fl_{X,T}/=_{Fn}$ are:
>
> - minimal flowchart schemes with respect to the input behaviour;
>
> - computation sequential processes.

7.3. The algebra of minimal flowchart scheme (with respect to the input behaviour). We have selected some identities that are satisfied by such minimal schemes (namely, the identities listed in Table II and illustrated in Figure 13), in order to define an algebraic structure, called funflow. The main point is that the set of identities (B1-15) + (F1-5), suffices to prove that flownomial expressions over **Pfn**, which represent the same computation process, are equal.

Rigorously, a funflow (or a strong iteration algebraic theory cf. [32]) is a biflow B, with some distinguished morphisms $O_m \in B(0,m)$ and $mVm \in B(m+m,m)$, and satisfying the algebraic rules listed in Table II (it should be emphasized that (F5) is not an equation, but an implication). The axioms (B1-6) + (F1-4) give a presentation of algebraic theories — in the sense of Lawvere — in terms of sum, composition, O_m, and mVm (by definition $m \leftrightarrow n = (O_n + 1_m + 1_n + O_m) \cdot (n+m)V(n+m)$ and the axioms B7-10 follow), hence finite functions are embedded in each funflow.

TABLE II.

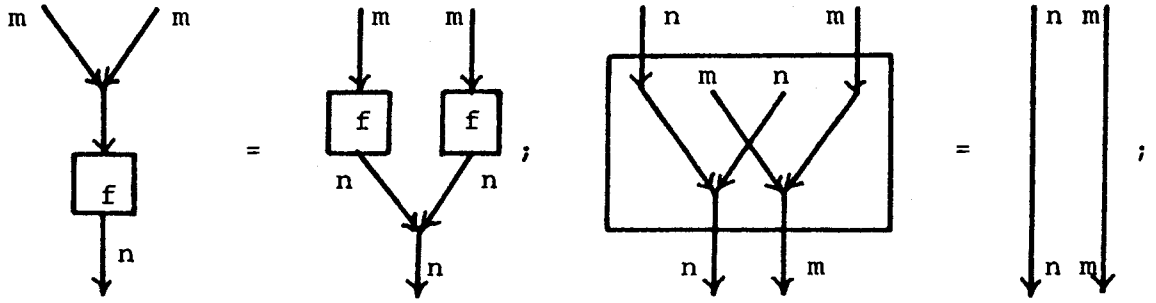
The axioms in Table I together with these ones define a funflow (= a strong iteration theory)	
(F1) $O_0 = 1_0$	(F2) $O_m f = O_n$
(F3) $mVm \cdot f = (f+f) \cdot nVn$	(F4) $(1_n + O_m + O_n + 1_m) \cdot (n+m)V(n+m) = 1_{n+m}$

(F5) every function is functorial, i.e.

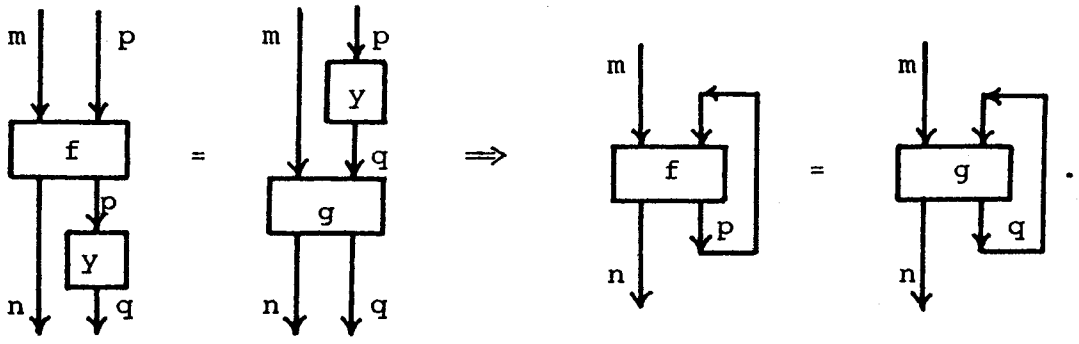
$$f(1_n + y) = (1_m + y)g \Longrightarrow f\uparrow^p = g\uparrow^q$$

for $f : m+p \to n+p$, $g : m+q \to n+q$, and $y \in Fn(p,q)$

(F1) (F2)



(F3) (F4)



(F5)

FIGURE 13
The axioms that are to be added to those in figure 12 in order to define a funflow.

Semantic models: **Rel**(S) and all of its subtheories, which contain the embedding of **Fn** in **Rel**(S), are funflows. Particularly, **Pfn** and **Rel** are funflows, **Pfn** being an initial funflow.

Syntactic models: If T is a funflow, then $Fl_{X,T}/=_{\mathbf{Fn}}$ is a funflow.

7.4. The universal property. Let (E^f_X, E^f_T) be the embedding of (X,T) into $Fl_{X,T}/=_{\mathbf{Fn}}$ obtained by using the embedding (E_X, E_T) of (X,T) into $Fl_{X,T}$, defined in § 1.3, and the canonical projection from $Fl_{X,T}$ to $Fl_{X,T}/=_{\mathbf{Fn}}$. The universal property satisfied by $Fl_{X,T}/=_{\mathbf{Fn}}$ is

"for every funflow F and every interpretation (I_X,I_T) of (X,T) in F there exists a unique morphism of funflows $I^{ff} : Fl_{X,T}/=_{\mathbf{Fn}} \to F$ such that $E^f_X \cdot I^{ff} = I_X$ and $E^f_T \cdot I^{ff} = I_T$".

The axiom (F5) ensures that the interpretation $(I_X,I_T)^f$, defined in § 4.4, identifies $=_{\mathbf{Fn}}$-equivalent flownomial expressions. Note that the morphism I^{ff} above is the morphism that is induced by $(I_X,I_T)^f$ in the quotient structure $Fl_{X,T}/=_{\mathbf{Fn}}$.

7.5. Fn-flow-calculus. The calculus with flownomials associated to computation processes (or to minimal flowchart schemes, with respect to the input behaviour) called fn-flow-calculus, is obtained by adding to the bi-flow-calculus the rule which consists in the identification of $=_{\mathbf{Fn}}$-equivalent expressions.

For the algebraic version, we add the rules (F2, F3, F5) in Table II to the rules that algebraically define the bi-flow-calculus.

Examples. In fn-flow-calculus the following identity holds

$$F \cdot G \cdot H = [1V1 \cdot x(1_2 + x)]\uparrow(1,3,2,3)_3[((1_1 + 1V1 \cdot y)z)\uparrow + y],$$

where $F := [1V1 \cdot x(1_1 + y + x)]\uparrow$, $G := [(1_3 + y + x + y)(1,3,2,3,2,5,4,4)_5]\uparrow^2$ and $H := ((1_1 + 1V1 \cdot y)z)\uparrow + 1_1$. Moreover, the right-hand side expression is $=_{\mathbf{Fn}}$-minimal.

Indeed, by examples in §5.5 $G = (1_3 + y)(1,3,2,3)_3 =: G'$ and by example in §6.5.b $F \cdot G' = [1V1 \cdot x(1_2 + x)]\ (1,3,2,3)_3(1_2 + y)$. Hence the identity holds. The right-hand side expression is $=_{\mathbf{Fn}}$-minimal, because the associated flowchart scheme is reduced and accessible.

Comment (iteration theories, cf. [5,21]; see also [6,7,22]). The theorem in § 7.4 shows that a correct interpretation of the input behaviour of schemes may be given in strong iteration theories. If we restrict the class of schemes to those over **Pfn** or, more generally, over a strong iteration theory T satisfying

(p) for every $f \in T(n,p+r)$ there exists $f' \in T(n,p+r+k)$ such that:

(i) there exists $y \in \mathbf{Sur}(k,r)$ with $f = f'(1_p + \langle 1_r, y \rangle)$;

(ii) for every $g \in T(n,p+r+k)$, $z \in \mathbf{Sur}(r+k,s)$ such that $f'(1_p+z) = g(1_p+z)$ there exists $z' \in \mathbf{Fn}(r+k,r+k)$ with $z'z = z$ such that $g = f'(1_p+z')$.

then the theorem in § 7.4 may be made a bit stronger: a correct interpretation may be given in all iteration theories. More exactly, the new theorem is:

"if T satisfies (p), then for every iteration theory F and every interpretation (I_X, I_T) of (X,T) in F there exists a unique morphism of iteration theories $I^{ff} : Fl_{X,T}/=_{\mathbf{Fn}} \to F$ such that $E^f_X \cdot I^{ff} = I_X$ and $E^f_T \cdot I^{ff} = I_T$".

Iteration theories are defined only by equations and are weaker than strong iteration theories. They may be defined by the axioms that define strong iteration theories B1-15 + F1-5 in which the implication F5 must be replaced with the equation

if $h:n \to m+p$, $y \in \mathbf{Fn}(m,n)$ and $y_1,\dots,y_m \in \mathbf{Fn}(m,m)$ are such that $y_i y = y$, $\forall i \in [m]$, then $(\langle x^m_1 yh(y_1+1_p), \dots, x^m_m yh(y_m+1_p)\rangle)^\dagger = y(h(y+1_p))^\dagger$.

Here the tupling of $f:m \to p$ and $g:n \to p$ is $(f+g)pVp$, while the iterate of $f:m \to m+n$ is $(mVm \cdot f \cdot m \leftrightarrow n)\uparrow^m$; x^m_i is $O_{i-1}+1_1+O_{m-i}$.

8. COACCESSIBILITY

Sometimes in a computation process we are interested in successful computation sequences only (i.e., computation paths that finish normally by reaching an output). In that case, in the execution process only the vertices that belong to paths going to outputs matter; these vertices form the coaccessible part of the scheme. Here we regard as equivalent two flowchart schemes that have the same coaccessible part. In a formal approach coaccessibility is captured by simulation via relations whose converses are injective functions.

The study of coaccessibility can be reduced to the study of accessibility, made in Section 5, by using a principle of duality: The dual flowchart scheme associated to a scheme F with m inputs and n outputs, is the scheme F^o, with n inputs and m outputs, obtained by reversing arrows of F (in the abstract case this method consists in taking the composition in the dual category). In this way the coaccessible part of a scheme F is the dual of the accessible part of the dual scheme F^o.

For this reason we omit any details here.

9. THE INPUT-OUTPUT BEHAVIOUR (DETERMINISTIC MINIMIZATION)

The input-output behaviour of a scheme is the restriction of the (step-by-step) input behaviour to the successful (terminal) paths. Here we regard as equivalent two flowchart schemes that have the same input-output behaviour. In the class of the schemes that have

the same input-output behaviour there is a minimal one, unique up to an isomorphism. The minimization with respect to the input-output behaviour does not preserve the completeness of a scheme, i.e., the minimal scheme associated to a scheme over **Fn** may be over **Pfn** $\setminus$ **Fn**. However, this minimization preserves the determinism of a scheme, i.e., the minimal scheme in $Fl_{X,\mathbf{Rel}}$ of a scheme over **Pfn** is over **Pfn**, too — hence the name. Formally the input-output behaviour is captured by simulation via partially defined functions.

9.1. The simulation via partial functions; the resulting congruence. Suppose that the support theory T "contains" partial functions. In the case $T \subseteq \mathbf{Rel}$ this means $T \supseteq \mathbf{Pfn}$.

The definition of simulation via partial functions is obtained from the general definition, given in §3.1, by using for u morphisms in **Pfn**.

In the general case $\rightarrow_{\mathbf{Pfn}} \subseteq \rightarrow_{\mathbf{In}^{-1}} \cdot \rightarrow_{\mathbf{Sur}} \cdot \rightarrow_{\mathbf{In}'}$, hence in the case $T \subseteq \mathbf{Rel}$ the meaning of the simulation "$F \rightarrow_u F'$ with u in **Pfn**" is "F' can be obtained from F in three steps: first by deleting noncoaccessible vertices, second by identifying vertices with common labels and coherent continuations, and finally by adding inaccessible vertices".

The meaning of the equivalence relation $=_{\mathbf{Pfn}}$, generated by $\rightarrow_{\mathbf{Pfn}}$, is "$F =_{\mathbf{Pfn}} F'$ " iff "F and F' have the same input-output behaviour" iff "by deleting noncoaccessible vertices, identifying vertices with common labels and coherent continuations, and deleting inaccessible vertices F and F' can be transformed into the same minimal scheme (with respect to the input-output behaviour)".

For the congruence relation $=_{\mathbf{Pfn}}$, generated by $\rightarrow_{\mathbf{Pfn}}$, the following equivalent characterization can be given:

(i) $\quad =_{\mathbf{Pfn}} = \rightarrow_{\mathbf{In}^{-1}} \cdot \rightarrow_{\mathbf{Sur}} \cdot \mathbf{In}^{\leftarrow} \cdot \rightarrow_{\mathbf{In}} \cdot \mathbf{Sur}^{\leftarrow} \cdot \mathbf{In}^{-1\rightarrow}$;

(ii) $\quad =_{\mathbf{Pfn}}$ is the least congruence relation generated by the identifications ($\leftrightarrow$X) in §4.1, (OX) in §5.1, (VX) in §6.1 and

($\perp$X) $x \cdot \perp_n = \perp_m$, where $x \in X(m,n)$ (see figure 10 d)

in the class of the congruence relations $\sim$ satisfying: every partial function is $\sim$-functorial.

Comments. By (i) two flowchart schemes are $=_{\mathbf{Pfn}}$-equivalent iff by deleting noncoaccessible vertices, identifying vertices and deleting inaccessible ones they can be transformed into the same scheme. In (ii) we get very simple generators for $=_{\mathbf{Pfn}}$, by restricting the class of congruence relations used to generate $=_{\mathbf{Pfn}}$.

9.2. Successful computation processes (or minimal flowchart schemes with respect to the input-output behaviour). In the case $T \subseteq \mathbf{Rel}$ every $=_{\mathbf{Pfn}}$-equivalence class has a minimal scheme, unique up to an isomorphism. Since two schemes are $=_{\mathbf{Pfn}}$-equivalent iff they have the same successful computation processes, we can identify such a class to a

successful computation process that consists of finite terminal sequences of statements. Generalizing we say:

> The elements in a $Fl_{X,T}/=_{Pfn}$ are:
>
> - minimal flowchart schemes with respect to the input-output behaviour;
>
> - successful sequential computation processes.

9.3, 9.4. We only mention that the algebraic structure corresponding to successful computation processes, called parfunflow, is a funflow, contains partial functions (in order to generate partial functions we use the distinguished morphisms $\perp_m : m \to 0$), and satisfies:

(P1) $f \cdot \perp_n = \perp_m$, for $f : m \to n$;

(P2) every partial function is functorial.

9.5. Pfn-flow-calculus. The calculus with flownomials associated to successful computation processes (or, to minimal flowchart schemes, with respect to the input-output behaviour), called pfn-flow-calculus, is obtained by adding to the bi-flow-calculus the rule which consists in the identification of $=_{Pfn}$-equivalent expressions.

For the algebraic version, we add the rules (P1-2) above to the rules that algebraically define the fn-flow-calculus in §7.5.

Example. In pfn-flow-calculus the following identity holds

$$F \cdot G \cdot H = (1V1 \cdot x)\uparrow (\perp_1 + y),$$

where F,G,H are those defined in example §7.5. Moreover, the left-hand side expression is $=_{Pfn}$-minimal.

Indeed $1_0 = \perp_0$, hence $[(1_1 + 1V1 \cdot y)z]\uparrow = [(1_1 + 1V1 \cdot y)z]\uparrow \cdot 1_0 = [(1_1 + 1V1 \cdot y)z]\uparrow \cdot \perp_0 =$ (by P1) $\perp_2$. Consequently, using the example §7.5 we obtain

$$F \cdot G \cdot H = [1V1 \cdot x(1_2 + x)]\uparrow (1,3,2,3)(\perp_2 + y) = (1V1 \cdot x(1_2 + x))\uparrow (\perp,1,\perp,1)_1 y .$$

Finally, using the example §6.5.a the desired identity follows easily. The right-hand side expression is $=_{Pfn}$-minimal since the associated flowchart scheme is coaccessible, reduced and accessible.

10. COREDUCTION

In the case we are interested in the study of input-output behaviours we can successfully use the duality defined in Section 8. The input behaviour is not preserved by duality, while the input-output behaviour is preserved. More exactly, the input-output behaviour of a dual scheme contains the same computation sequences as the given

scheme, but having the statements concatenated in the reversed order. Let us call (step-by-step) cobehaviour of a vertex in a scheme, the (step-by-step) behaviour of the corresponding vertex in the dual scheme, defined as in Section 6. In a scheme we can identify vertices that have the same cobehaviour, without changing the input-output behaviour of the scheme. A flowchart scheme will be called coreduced if it has no different vertices having the same cobehaviour. In a formal approach coreduction is captured by simulation via relations whose converses are surjective functions.

This coreduction cannot be used properly in the context of deterministic flowchart schemes. The reason is the following. By reduction we identify vertices provided after identification they have the same output arrows and bring together the corresponding input arrows. By coreduction we have to identify vertices provided after identification they have the same input arrows and bring together the corresponding output arrows, hence a nondeterministic choice between different output arrows of continuation occurs.

More details can be obtained from Section 6 by duality.

11. NONDETERMINISTIC FLOWCHART SCHEMES

The feature of flowchart schemes we take now into account is "nondeterministic choice", i.e., the possibility in a point of a scheme (input, or continuation after a statement) to have more arrows of continuation for the flow of control, from which the execution process chooses one variant in a random way. Consequently, in the context of usual flowchart schemes, represented as in § 1.1, the basic support theory for this nondeterministic case is **Rel**, while in the deterministic case the basic support theory was **Pfn**. In the presence of the nondeterministic choice we are interested in considering the input-output behaviour, rather than the input behaviour. For modeling the input-output behaviour, in this nondeterministic case we can try to apply simulation via relations. This syntactic transformation of flowchart schemes is again useful: it is correct, in the sense it preserves the input-output behaviour, but at the present time we do not know whether it is complete, i.e. we do not know whether two usual nondeterministic schemes, having the same input-output behaviour, can be connected by a chain of simulations.

11.1. The simulation via relations; the resulting congruence. Suppose that the support theory T "contains" finite relations. In the case of usual flowchart schemes this means T = **Rel**. The definition of the simulation via relations was given in § 3.1.

In the general case $\rightarrow_{\mathbf{Rel}} \subseteq \rightarrow_{\mathbf{In}^{-1}} \cdot \rightarrow_{\mathbf{Sur}^{-1}} \cdot \rightarrow_{\mathbf{Sur}} \cdot \rightarrow_{\mathbf{In}}$, hence in the case T = **Rel** the meaning of the simulation "$F \rightarrow F'$" is "F' can be obtained from F in four steps: first by deleting noncoaccessible vertices, second by multiplying vertices keeping fixed the inputs and sharing the outputs, then by identifying vertices that give the same outputs and bring together the corresponding inputs, and finally by adding inaccessible vertices". The meaning of the equivalence relation $=_{\mathbf{Rel}}$, generated by $\rightarrow_{\mathbf{Rel}}$, is still unclear. We conjecture that "$F =_{\mathbf{Rel}} F'$", iff "F and F' have the same input-output behaviour".

For the congruence relation $=_{Rel}$, generated by $\rightarrow_{Rel}$, the following equivalent characterization can be given:

(i) $$=_{Rel} = In^{\leftarrow} \cdot Sur^{\leftarrow} \cdot \rightarrow In^{-1} \cdot \rightarrow Sur^{-1} \cdot \rightarrow Sur \cdot \rightarrow In \cdot Sur^{-1\leftarrow} \cdot In^{-1\leftarrow}$$

(ii) $=_{Rel}$ is the least congruence relation generated by the identifications $(\leftrightarrow X)$ in §4.1, (OX) in §5.1, (VX) in §6.1, $(\perp X)$ in §9.1 and

$(\wedge X)\ x \cdot n \wedge n = m \wedge m \cdot (x + x)$, where $x \in X(m,n)$ (see Figure 10 e)

in the class of congruence relations $\sim$ satisfying: every relation is $\sim$-functorial.

As we do not know the semantic meaning of $=_{Rel}$ we do not insist on this syntactic study which has been done as a natural extension of the above ones.

12. SOME HISTORICAL COMMENTS

Our comments concern the last ten years of algebraic theory of monadic computation. This mathematical model of computation was introduced by Ianov [26] and its algebrization was initiated by Elgot in the early seventies [14,15,16]. The comments only present our works and some other closely related papers and should by no means be considered a complete survey of the area.

In order to get an algebraic theory of computation one needs an axiomatic looping operation. This may be Kleene's repetition (cf. [13,33], for example), Elgot's iteration [15] or feedback [30,31].

Nondeterministic computation. The proper acyclic context for repetition seems to be a matrix theory (such a theory is equivalent with the theory of matrices over a semiring [17]). The equational axiom for the looping operation are not easily codified. A regular algebra cf. Conway [13] is a structure which satisfies all the identities (written in terms of union, composition, repetition and constants 0, 1) which are valid in the algebras of regular events. The theory of matrices over a regular algebra is a matrix theory, but the axioms for repetition are yet unknown (by authors' knowledge). This algebra is intended as a model for the input-output behaviour of nondeterministic computation.

This nondeterministic case is more difficult and was considered before the deterministic case. The latter works on deterministic schemes put it in a new light.

Our result on nondeterministic computation in Section 11 are the translation in terms of feedback of the results in [33]. Arbib and Manes [2] deals with nondeterminism in the setting of partially additive categories. Some other algebras for nondeterministic computation are presented by Bloom and Ésik in [7, the extended abstract].

Much work must be done to put some order on all of these algebraic approaches.

Deterministic computation. The proper acyclic context for iteration seems to be an algebraic theory in the sense of Lawvere [14]. Hence iteration may be used in a more general context than repetition.

It is well known that the operations of "structured programming", i.e. **composition, if--then-else** and **while-do** are not enough for representing all flowchart scheme behaviours, essentially since they use one-input/one-exit schemes. However they suffice, provided additional memory is permitted. Elgot's idea in [16] is to use many-input/many-exit flowchart schemes, having **composition, tupling** and **scalar iteration** as basic operations. These operations suffice for representing all flowchart scheme behaviours. More precisely, from Theorem 4.1 in [16] it follows that every flowchart scheme is equivalent with respect to the input behaviour (in Elgot's terms: is "strongly equivalent") to a scheme built up from atomic schemes and trivial ones by means of these operations. (This is done without adding memory.) However, for representing all flowchart schemes (pictures) in this setting Elgot ([16], Theorem 3.1) used a **vector iteration,** which is not obtained by a repeated application of the scalar iteration.

Elgot's algebra for the input behaviours of acyclic (resp. cyclic) schemes was an algebraic theory [14] (resp. an iterative algebraic theory [15] — note that this algebra is not equationally presented). The fact that a class of trees forms the free iterative theory was conjectured by Goguen and Thatcher [24] and proved in [23] and [19].

An iteration theory cf. Bloom, Elgot and Wright [5] is a structure which satisfies all the identities (written in terms of tupling, composition, iteration and constants 1_a, O_a, x_i^a) which are valid in the theories of the input behaviours of flowchart schemes over **Fn** or **Pfn** (i.e. in the theories of the regular trees). The axiomatization for iteration theories was found by Ésik [21]. An iteration theory is an algebraic theory in which an iteration operation is given fulfilling some axioms. This algebra is intended as a model for the input behaviour of deterministic computation. More exactly, the schemes over **Pfn** which have the same input behaviour have the same interpretation in all iteration theories.

However, iteration theories seems to be too weak to study minimization. The process of complete minimization is captured by the simulation via functions [32]. In the case when the support theory is **Fn** this fact was shown by Elgot [18]. (Note that the schemes over **Fn** are not closed with respect to iteration.) In the abstract case the appropriate algebraic structure for the support theory in order to deal with complete minimization seems to be a strong iteration theory, i.e. an iteration theory in which every function is functorial, cf. [32]. Strong iteration theories are not equationally presented. If T is a strong iteration theory, then the schemes over T which have the same input behaviour have the same interpretation in all strong iteration theories; if moreover T satisfies the condition (p) in the Comment in §7.5, then they have the same interpretation in all iteration theories. Condition (p) is adopted from [21]. So if we consider schemes over an arbitrary strong iteration theory, not only over **Pfn**, then we have to restrict the class where an appropriate interpretation may be given from iteration to strong iteration

theories.

The results of Section 7 are the translation in terms of feedback of the results in [32]. In the translation of these results the use of the new set of operations (composition-sum-feedback) allows to separate the study of accessibility, given in Section 5, from the study of reduction, given in Section 6. The results of Sections 5 and 6 are new and cannot be properly done using algebraic theories and iteration.

There is a close connection between simulation and the functoriality axiom (partially suggested by the results presented in the present paper) which we hope to make clear in our further papers. The functoriality axiom in its full generality was considered by Arbib and Manes [1]. The particular functoriality axiom that "every function is functorial" also appear in [21].

Strong iteration theories are already implicit in [21]. The existence of an iteration theory which is not a strong iteration theory is pointed out in [22] where it is also noted that most known iteration theories are strong iteration theories.

In Section 9 we have given some details for the extension of the calculus to the input-output behaviour of deterministic flowchart schemes announced in [32, Section 7.a]. The results appear here for the first time but a weaker variant directly follows from the results in the nondeterministic case in [33].

An equational presentation of the variety of iteration theories generated by **Pfn**(S) appeared in Bloom and Ésik [7]. This algebra is intended as an algebra for the input-output behaviour of deterministic schemes. An interesting result proved by Bloom and Ésik in [7] shows that **Pfn**(S) with the standard algebraic theory structure become biflow in a unique way. (Of course, their result is in terms of iteration theories, but the proof use only the biflow rules.)

Flowchart schemes. The proper acyclic context for feedback is a symmetric strict monoidal category in the sense of MacLane [28]. (An analoguous structure has been introduced by Arnold and Dauchet in [3] under the name of "magmoïd".) Hence feedback may be used in a more general context than repetition or iteration.

The feedback operation was introduced in [30,31]. It is a "scalar" operation and it seems that this operation is more adequate to study (cyclic) flowchart schemes than scalar iteration. One reason is the following: All flowchart schemes (pictures) can be built up from atomic schemes and trivial ones by means of **composition**, (separated) **sum** and (scalar) **feedback**.

A biflow is a structure which satisfies all the identities (written in terms of separated sum, composition, feedback and constants 1_a, $a \leftrightarrow b$) which are valid in the algebras of flowchart schemes. An axiomatization for biflows is given in [31]. A biflow is a symmetric strict monoidal category in which a feedback operation is given fulfilling some axioms. This model is more related with the algorithms themselves than with their behaviours.

Normal forms, in essence similar to particular flownomial expressions in normal form as given here, have been used elsewhere, cf. [15, 16, 21, 6]. The representation of flowchart schemes by triples (i.e. the connection morphism is split into its "input" part and its "transfer" part) is due to Elgot [15,16]. The schemes in [16] are over **Fn**, in [20] over **Sur** and in [6] over **Pfn** (although it was not thought of connections as being morphisms in a "theory"), and the operations on flowchart schemes are verbally defined.

In [12] the schemes are extended from schemes over **Fn** to schemes over **Pfn** in order to obtain a good definition for scheme iteration. Based on this extension a general extension of connections to morphisms in an "algebraic theory with iterate" is given leading to the concept of "schemes over a theory." The operations on flowchart schemes were defined formally by extending those of the theory of connections. (The motivation for the extension of connections from simple relations to more complicated known computation processes is given in [32] and some example of schemes connected with such more complicated computation processes are given in [32,33].)

The representation of flowchart schemes by flownomial expressions in normal form was introduced in [30,31]. The extension to arbitrary flownomial expressions was given in [11]. It should be emphasized that our concept of flowchart scheme is a purely syntactic one. Indeed, for an appropriate choice of the connection biflow (e.g. partial functions) there is an isomorphism between our schemes and the schemes defined as a kind of directed graph.

The results of Sections 1 and 4 are new. The details for Section 1 are given in [10]. Without axiomatizing finite bijections the result of Section 4 is presented in [31].

The results in Sections 1 and 4 seem to be much more natural than those in [6,9,12]. Actually a theory with iterate, as introduced in [12], may be defined as a biflow over an algebraic theory; see [10]. (Later this algebraic structure was used as a frame to study context-free trees in [8].) The main result in [12] characterizes the representations in $Fl_{X,T}$, where T is a theory with iterate, as being the "T-module with iterate" freely generated by X. The extension to schemes in $Fl_{X,T}/=_{Bi}$ was given in [9]. The main obstacle in obtaining a natural result regarding the algebraic characterization of flowchart schemes in [12,9] was the use of an algebraic theory as support theory. Indeed, the flowchart schemes do not have a structure of an algebraic theory but only of a symmetric strict monoidal category.

Another algebra for flowchart schemes is given by Bloom and Ésik in [6]. Their schemes are over **Pfn**, the algebra may be presented as a biflow over a symmetric strict monoidal category which extend **Pfn** and the main result is a particular case of the Theorem in § 4.4.

Also there are some technical advantages in working in the setting of sum-composition-feedback. For example the axiomatization for flowchart schemes as in Table I is easier than the corresponding axiomatizations in terms of iteration in [6,12].

Finally, let us compare in more details the underlying equational algebras. As we already pointed out for the acyclic context there are some natural inclusions

matrix theories $\subseteq$ algebraic theories $\subseteq$ (symmetric) strict monoidal categories

and the inclusions are strict. In the cyclic context the following inclusions holds

matrix theories of regular algebras $\subseteq$ iteration theories over matrix theories $\subseteq$ biflows over matrix theories

and

iteration theories $\subseteq$ biflows over algebraic theories.

It seems likely that one can prove that the above inclusions are also strict — this was proved by Ésik in [22] for the latter one.

ACKNOWLEDGEMENTS.

We should like to thank Professor Sergiu Rudeanu and the anonymous referees for their help in improving the presentation and the content of the section with historical comments, as well as Camelia Minculescu for a rapid typing.

REFERENCES

[1] M.A. Arbib and E.G. Manes, Partially additive categories and flow diagram semantics, J. Algebra **62** (1980), 203-227.

[2] M.A. Arbib and E.G. Manes, Algebraic approaches to program semantics, Springer-Verlag, 1986.

[3] A. Arnold and M. Dauchet, Théorie des magmoïdes, RAIRO Inform. Theor. **12** (1978), 235-257 and **13** (1979), 135-154.

[4] S.L. Bloom, Calvin C. Elgot, Selected Papers, Springer-Verlag, 1982.

[5] S.L. Bloom, C.C. Elgot and J.B. Wright, Vector iteration in pointed iterative theories, SIAM J. Comput. **9** (1980), 525-540.

[6] S.L. Bloom and Z. Ésik, Axiomatizing schemes and their behaviours, J. Comput. System Sci. **31** (1985), 375-393.

[7] S.L. Bloom and Z. Ésik, Some varieties of iteration theories, extended abstract, Bull. EATCS, Oct. 1984; full version to appear in SIAM J. Comput.

[8] V.E. Căzănescu, On context-free trees, Theoret. Comput. Sci. **41** (1985), 33-50.

[9] V.E. Căzănescu and Ş. Grama, "On the Definition of M-Flowcharts," Preprint Series in Mathematics, No.56/1984, INCREST, Bucharest; also in: An. Univ. "Al.I.Cuza" Iaşi, Mat. **XXXIII**, 4 (1987), 311-320.

[10] V.E. Căzănescu and Gh. Ştefănescu, "A formal representation of flowchart schemes," Preprint Series in Mathematics, No.22/1987, INCREST, Bucharest; also in: An. Univ. Bucureşti, Mat.-Inf. **XXXVII**, 2 (1988), 33-51.

[11] V.E. Căzănescu and Gh. Ştefănescu, A calculus for flowchart schemes, in: Abstracts 8th International Congress of Logic, Methodology and Philosophy of Science Vol.1, Nauka, Moscow, 1987, 124-127.

[12] V.E. Căzănescu and C. Ungureanu, "Again on Advice on Structuring Compilers and Proving them Correct", Preprint Series in Mathematics, No. 75/1982, INCREST, Bucharest; revised version: The free algebraic structure of flowcharts, Rev. Roumaine Mat. Pures Appl. **34** (1989), 281-302.

[13] J.H. Conway, Regular algebra and finite machines, Chapman and Hall, London, 1971.

[14] C.C. Elgot, Algebraic theories and program schemes, in: Semantics of algorithmic languages, Lecture Notes in Mathematics **188**, Springer-Verlag, New York /Berlin, 1971.

[15] C.C. Elgot, Monadic computation and iterative algebraic theories, in: Logic Colloquium '73, Studies in Logic and the Foundations of Mathematics, Vol. **80**, North-Holland, Amsterdam, 1975, 175-230.

[16] C.C. Elgot, Structured programming with and without GOTO statements, IEEE Trans. Software Eng. **SE-2** (1976), 41-53.

[17] C.C. Elgot, Matricial theories, J. Algebra **42** (1976), 391-421.

[18] C.C. Elgot, Some geometrical categories associated with flowchart schemes, in: Proceedings, Fundamentals of Computation Theory, Poznan, 1977, Lecture Notes in Computer Science **56**, Springer-Verlag, Berlin / New York, 1977, 256-259.

[19] C.C. Elgot, S.L. Bloom and R. Tindell, On the algebraic structure of rooted trees, J. Comput. System Sci. **16** (1978), 362-399.

[20] C.C. Elgot and J.C. Shepherdson, "An Equational Axiomatization of the Algebra of Reducible Flowchart Schemes," Research Report, RC-8221, IBM, 1980.

[21] Z. Ésik, Identities in iterative and rational algebraic theories, Comput. Linguistics Comput. Languages, **XIV** (1980), 183-207.

[22] Z. Ésik, Independence of the equational axioms of iteration theories, J. Comput. System Sci. **36** (1988), 66-76.

[23] S. Ginali, "Iterative Algebraic Theories, Infinite Trees, and Program Schemata", Dissertation, University of Chicago, June 1976; see also: Regular trees and free iterative theory, J.Comput. System Sci. **18** (1979), 228-242.

[24] J.A. Goguen and J.W. Thatcher, Initial algebra semantics, in: Proceedings 15th IEEE Symposium on Switching and Automata Theory, 1974, 63-77; also the paper with E.G. Wagner and J.B. Wright, Initial algebra semantics and continuous algebras, J. Assoc. Comput. Mach. **24** (1977), 68-95.

[25] S. Greibach, Theory of program structures: schemes, semantics, verification, Springer-Verlag, Berlin / New York, 1975.

[26] Iu.I. Ianov, On logical schemata of algorithms, in: Problemi kibernetiki **1**, Fizmatgiz, Moskva, 1958 [Russian].

[27] V.E. Kotov, Introduction to the theory of program schemes, Nauka, Novosibirsk, 1978 [Russian].

[28] S. MacLane, Categories for the working mathematician, Springer-Verlag, Berlin / New York, 1971.

[29] Z. Manna, Mathematical theory of computation, McGraw-Hill, New York, 1974.

[30] Gh. Ştefănescu, An algebraic theory of flowchart schemes (extended abstract), in: Proceedings CAAP'86, Lecture Notes in Computer Science **214**, Springer-Verlag, Berlin / New York, 1986, 60-73.

[31] Gh. Ştefănescu, "Feedback Theories (A Calculus for Isomorphism Classes of Flowchart Schemes)," Preprint Series in Mathematics, No. 24/1986, INCREST, Bucharest.

[32] Gh. Ştefănescu, On flowchart theories. Part I. The deterministic case, J. Comput. System Sci. 35 (1987), 163-191; preliminary versions: Preprint Series in Mathematics Nos. 39/1984, 7/1985, and 52/1986, INCREST, Bucharest.

[33] Gh. Ştefănescu, On flowchart theories. Part II: The nondeterministic case, Theoret. Comput. Sci. 52 (1987), 307-340; preliminary version: Preprint Series in Mathematics No.32/1985, INCREST, Bucharest.

[34] J.W. Thatcher, E.G. Wagner and J.B. Wright, Notes on algebraic fundamentals for theoretical computer science, in: Foundations of computer science III, Part 2: Language, logic, semantics (J.W. de Bakker and J. van Leeuwen, Eds.), Mathematical Centre Tracts 109, Amsterdam, 1979, 83-164.

ADDITIONAL REFERENCES

Some details for Section 4 may be found in [35] below, while a comparation between feedback, iteration, and repetition may be found in [36].

[35] V.E. Căzănescu and Gh. Ştefănescu, "A Formal Representation of Flowchart Schemes II," Preprint Series in Mathematics No.60/1988, INCREST, Bucharest; Stud. Cerc. Mat. Vol. 41, No.3 (1989), in press.

[36] V.E. Căzănescu and Gh. Ştefănescu, "Feedback, Iteration and Repetition," Preprint Series in Mathematics No.42/1988, INCREST, Bucharest.