IBM Research

RC 5908
(#25545)
3/18/76
Computer
Science

176 pages

A JUNCTION BETWEEN COMPUTER SCIENCE AND CATEGORY THEORY, I: BASIC CONCEPTS AND EXAMPLES (PART 2)

J. A. Goguen,* J. W. Thatcher, E. G. Wagner, and J. B. Wright
Mathematical Sciences Department

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Yorktown Heights, New York

San Jose, California

Zurich, Switzerland

RC 5908
(#25545)
3/18/76
Computer
Science

176 pages

A JUNCTION BETWEEN COMPUTER SCIENCE AND CATEGORY THEORY, I: BASIC CONCEPTS AND EXAMPLES (PART 2)

J. A. Goguen,* J. W. Thatcher, E. G. Wagner, and J. B. Wright
Mathematical Sciences Department

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Typed by Joan Petrosillo and Martha Pierce on MTST

ABSTRACT: This is the second part of the first report in a series devoted to exploring the interface or "junction" between computer science and category theory. Both benefit from this exploration: computer science by a powerful set of tools and a general methodology providing a rigorous and uniform approach to many of its basic concepts, methods, and questions; and category theory by a nontrivial collection of practical applications and illustrations, plus a number of new problems and results. Our present general purposes are to provide a clear, leisurely, and well-illustrated introduction to the basic language of category theory, and to give introductory formulations of some of the computer science topics, including programs, machines, automata, and languages.

This Part covers graphs and diagrams, and introduces the third key categorial concept, natural transformation. An extended example covering correctness and termination of flow-diagram programs illustrates many of the concepts covered so far in the series.

* Current address: Computer Science Department, UCLA, Los Angeles, California 90024.

PREFACE

This is the second part (Section 3) of the first report in the series, "A Junction between Computer Science and Category Theory." In this first report, we are introducing the basic categorical concepts with numerous computer science-related examples. The first part included the definitions of category and functor. Here in the second part we introduce the third basic definition, that of natural transformation.

Again it is our hope that the applications and examples will be of particular interest for the computer scientist, because that was our intention in choosing them. While this report is still of rather expository character, some of the applications are now more substantial, and can stand on their own as support for our contentions about the usefulness of category theory in computer science. This is particularly true of the considerations of program correctness and termination in Sections 3.3 and 3.4.

This second part of ADJ I is considered strictly a continuation of Part 1 and is paginated accordingly. Thus, Section 3 begins on page 72 where Section 2 of the first part left off. In this way, we combine the index (and bibliography) of the two parts of the first ADJ report on basic concepts and examples.

The general outline and purpose of the ADJ series has been discussed in the first part, as were our general notational and typographical conventions. A reminder may be helpful: ADJ I/1 means the first part of the first report, and, Section II. 5.2 means the second subsection of the fifth section of the second ADJ report.

ERRATA, ADJ I, Part 1

Page 20, Line -3: "$n \mapsto n \cup \{n\}$".

Page 22, Line 9: "$R \cap (A_0 \times B)$".

Page 25, Line -7: "$A_0 \restriction f \restriction B_0$".

Page 33, Line -2: "say $S=1=\{0\}$".

Page 34, Line -7: "$w' : n' \to X$".

Page 36, Line 10: "$(x_0 \cdot x_2)+(x_1 \cdot x_2)$".

Page 37, Line 10: "$\top$ (top)".

Line 12: "$\bot$ (bottom)".

Line -12: "$x_0 \wedge x_0' = \bot$".

Line -10: "A lattice with zero and unit is complemented iff every element of the lattice has a complement."

Page 39, Line -3: "morphisms;".

Page 41, Line 7: "morphisms from".

Page 45, Line -10: "category. A small discrete category $\underline{\underline{C}}$ and its underlying set $|\underline{\underline{C}}|$ determine each other in some sense; see Example 4.27 of Section 4.6."

Line -5: "$\underline{\underline{M}}$ with".

Page 50, Line -9: "$\underline{\underline{Sem}}_*$ (Example 2.6),".

Page 51, Line -11: "generated or determined by X."

Page 56, Line 3: "then the image $\underline{\underline{C}}$ of F has".

Line -12: "U under f".

Page 58, Line 11: "Then any functor".

Page 59, Line 3: "See Example 3.9 (in Part 2)."

ERRATA, ADJ I, Part 1, Continued.

Page 60, Line 12: "each monoid".

Line -6 to -1: "This functor is clearly injective, and its image is the full subcategory $\underline{\underline{\underline{Mon}}}_C$ of $\underline{\underline{\underline{Cat}}}$ whose objects are all small categories with exactly one object and $1_e=e$ for that object. In fact, when corestricted to this subcategory of $\underline{\underline{\underline{Cat}}}$, F is bijective, thus giving an isomorphism of categories. We say that F embeds Mon into Cat.□"

Page 61, Line 6: "Example 2.4".

Page 62, Line -1: "that (as a subcategory of $\underline{\underline{\underline{Set}}}$)".

Page 64, Line -3: "$r:n \to R$".

Page 65, Line 8: "*Proposition 2.3."

Page 69, Line -9, before *2.4, insert:

"The following result is later used to some advantage for the construction of bifunctors.

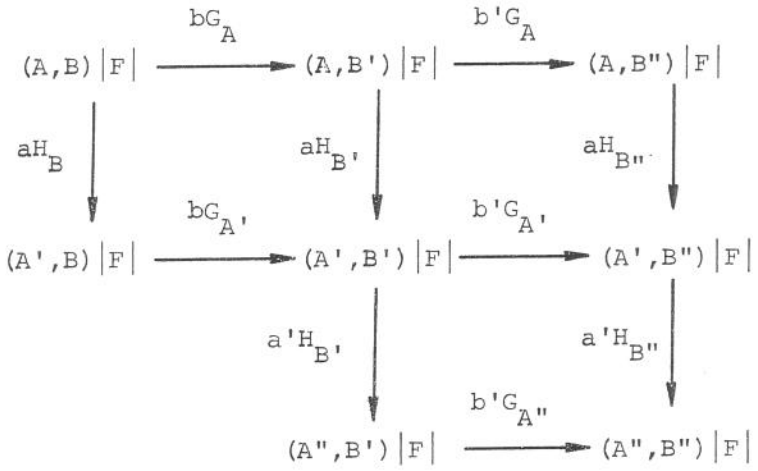
Proposition 2.4. Let $\underline{\underline{A}}$, $\underline{\underline{B}}$, $\underline{\underline{C}}$ be categories; for each $A\epsilon|\underline{\underline{A}}|$ let $G_A:\underline{\underline{B}} \to \underline{\underline{C}}$ be a functor; for each $B\epsilon|\underline{\underline{B}}|$ let $H_B:\underline{\underline{A}} \to \underline{\underline{C}}$ be a functor; assume that $B|G_A| = A|H_B|$ for each $A\epsilon|\underline{\underline{A}}|$, $B\epsilon|\underline{\underline{B}}|$, and let this common value be denoted $(A,B)|F|$. Assume that for each $a:A \to A'$ in $\underline{\underline{A}}$ and $b:B \to B'$ in $\underline{\underline{B}}$ the diagram

$$\begin{array}{ccc} (A,B)|F| & \xrightarrow{bG_A} & (A,B')|F| \\ {\scriptstyle aG_B}\downarrow & & \downarrow{\scriptstyle aG_{B'}} \\ (A',B)|F| & \xrightarrow[bG_{A'}]{} & (A',B')|F| \end{array}$$

commutes in $\underline{\underline{C}}$. Then there is a unique functor $F:\underline{\underline{A}}\times\underline{\underline{B}} \to \underline{\underline{C}}$ with object part $|F|$ such that $(1_A,b)F=bG_A$ and $(a,1_B)F=aH_B$.

ERRATA, ADJ I, Part 1, Continued

Proof. The conditions above give $|F|$, and F on certain subsets of $\underline{A}\times\underline{B}$. In fact, we define $(a,b)F$ to be the diagonal of the above square. We must check functorality. Assuming $a':A' \to A''$ and $b':B' \to B''$, consider the diagram



whose long diagonal is $(aa',bb')F=(a,b)F(a',b')F$.

If F' is another functor satisfying the given conditions, then $(a,b)F'=(1_A,b)F'(a,1_{B'})F'=bG_AaH_{B'}=(a,b)F$. □"

Page 69, Line -4: "Proposition 2.5" instead of "Proposition 2.4".

Page 70, Line-11: "Proposition 2.6" instead of "Fact 2.5".

Line -6: "Proposition 2.7" instead of "Fact 2.6".

CONTENTS

Part 1

Part 2

3. GRAPHS, DIAGRAMS AND PROGRAMS

Some persons claim that graph theory is a (or even the) basic methodology of computer science. We would not particularly dispute this claim because the relationship between graphs and categories is very intimate. Moreover, we shall be using graphs throughout this series of reports, for example in connection with diagrams. On the other hand, there are several respects in which the study of graphs is facilitated by categorical methods. We shall see this in our study of a special kind of graph, the tree, and in this section we obtain a useful characterization of the set of all paths in a graph -- in fact, this set has the structure of a category, freely generated by the graph. The category of paths in a graph plays a major role in our approach in this section to the semantics (or meaning) and correctness of flow diagram programs (we will see other approaches to these problems in later reports).

This section also introduces our third key categorical concept, the natural transformation. We will see that it is closely related to certain notions of diagram and graph homomorphism, and also to automaton homomorphisms. Moreover, natural transformations enable us to give the structure of a category to the class of functors between two categories.

3.1 GRAPHS

Definition 3.1 A graph G consists of:

(1) a set V of nodes or verticies (sometimes denoted $|G|$);

(2) a set E of edges (sometimes denoted by the symbol G again);

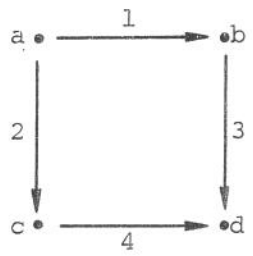
(3) a pair of monadic functions, $\partial_0, \partial_1 : E \to V$, called source (or origin) and target (or terminus), respectively.

(This data is subject to no conditions whatsoever.) □

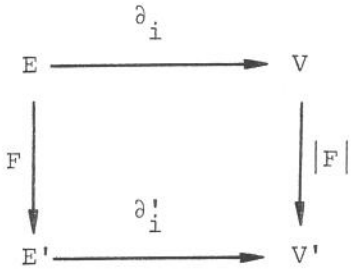
We may think of E as a basketfull of flexible, stretchable arrows; of V as a basketfull of dots; and of ∂_0, ∂_1 telling us for each arrow, which dots it must run from and to. That is, a graph in the sense above, is a description of how to construct a picture, which is a graph in the usual intuitive sense. For example, say E = {1,2,3,4}, V = {a,b,c,d}, and that ∂_0, ∂_1 are given by the table

	∂_0	∂_1
1	a	b
2	a	c
3	b	d
4	c	d

Then we must draw 1 from a to b, 2 from a to c, etc., obtaining the following picture.



Note that a graph is a two-sorted algebra $G = \langle V, E, \partial_0, \partial_1 \rangle$. A <u>graph morphism</u> $G \to G'$ is defined accordingly (see Section 1.5) to be a quadruple $\langle G, |F|, F, G' \rangle$ such that $|F| : V \to V'$, $F : E \to E'$, and $e\partial_i |F| = eF\partial_i'$ for all $i \in \{0,1\}$ and $e \in E$, i.e., such that



commutes for $i\in\{0,1\}$. Parallel to the notational conventions for functors, we write a graph morphism $F:G \to G'$, knowing that F consists of maps $F:E \to E'$ and $|F|:V \to V'$, preserving source and target. Note also that $v\in|G|$ will mean $v\in V$, and sometimes $e\in G$ will mean $e\in E$. Moreover, $e:v_0 \to v_1$ will be shorthand for $e\in E$ and $e\partial_0 = v_0$ and $e\partial_1 = v_1$. Thus we have a category $\underline{\underline{Gph}}$ of graphs, which is really only $\underline{\underline{Alg}}_\Sigma$ for the appropriate 2-sorted operator set Σ (see Examples 2.7 and 2.8).

The similarity between the definitions of graph (3.1) and of category (2.1) is intentional and significant. Corresponding to any small category $\underline{\underline{C}}$ is a graph $\underline{\underline{C}}U = \langle|\underline{\underline{C}}|,\underline{C},\partial_0,\partial_1\rangle$, obtained by "forgetting" the composition and identity operations of $\underline{\underline{C}}$. Furthermore if $F:\underline{\underline{C}} \to \underline{\underline{C}}'$ is a functor, ($F = \langle\underline{\underline{C}},|F|,F,\underline{\underline{C}}'\rangle$) then $FU:\underline{\underline{C}}U \to \underline{\underline{C}}'U$ is a graph morphism ($FU = \langle\underline{\underline{C}}U,|F|,F,\underline{\underline{C}}'U\rangle$). Thus, there is a forgetful functor $U:\underline{\underline{Cat}}_S \to \underline{\underline{Gph}}$, and we call $\underline{\underline{C}}U$ the underlying graph of $\underline{\underline{C}}$. We now consider a sort of "unforgetful" or "reconstruction" functor, from graphs to small categories.

Let $G = \langle V,E,\partial_0,\partial_1\rangle$ be a graph. Then a path p in G is a string (i.e., a finite sequence) $e_0e_1\cdots e_{n-1}$ of edges in G such that $e_{i-1}\partial_1 = e_i\partial_0$ for $0 < i < n$. We say that n is the length of p. For $n > 0$, we say

$p = e_0 \ldots e_{n-1}$ is a path from source $p\partial_0 = e_0\partial_0$ to target $p\partial_1 = e_{n-1}\partial_1$. For any $v \in V$, we shall always take λ, the empty string of edges, to be a path, from v to v.

The following basic fact follows immediately from these definitions.

Fact 3.1. If p is a path from v_0 to v_1 and p' is a path from v_1 to v_2, then their concatenation pp' is a path from v_0 to v_2. □

The collection of all paths in a graph is a category which is rather similar to the collection of all strings in a set, and we employ a notation which reflects this similarity.

Proposition 3.2. Let G be any graph, and construct $G^{\circledast}$ as follows:

(1) $|G^{\circledast}| = |G|$;

(2) $G^{\circledast} = \{\langle v_0, p, v_1\rangle \mid p \text{ is a path from } v_0 \text{ to } v_1\}$;

(3) $\langle v_0, p, v_1\rangle \partial_i = v_i$, for $i=0,1$;

(4) $1_v = \langle v, \lambda, v\rangle$; and

(5) $\langle v_0, p, v_1\rangle \circ \langle v_1, p', v_2\rangle = \langle v_0, pp', v_2\rangle$.

Then $G^{\circledast}$ is a category, which we call the path category of G, or the free category generated by G.

Proof. By Fact 3.1, pp' is a path from v_0 to v_2, so that composition ((5) above) makes sense (i.e., keeps us in $G^{\circledast}$). Associativity of concatenation (of strings) gives associativity of composition of paths, and the fact that λ is an identity for concatenation implies that 1_v satisfies the identity equations. The other axioms of Definition 2.1 are straightforward. □

Example 3.1 (See Example 2.2.) Transition Graphs. Let $A = \langle X, S, \delta\rangle$ be an automaton (i.e., a deterministic transition system). Let G_A be the graph $\langle S, '\delta', \partial_0, \partial_1\rangle$, where by '$\delta$' we mean the set of all triples $\langle s_0, x, s_1\rangle$ such

that $(s_0,x)\delta = s_1$, and where $\langle s_0,x,s_1\rangle\partial_i = s_i$ for $i = 0,1$. Then G_A is called the transition graph of the automaton A. (There is of course a functor G: Aut $\to$ Gph corresponding to this construction, but we now have sufficiently many examples of functors to leave this for the reader who may be interested to work it out by himself.) The category $G_A^{\circledast}$ of paths in G_A is essentially the category $\underline{\underline{Tr}}_A$ of extended state transitions developed in Example 2.2. The only difference in the two categories is that whereas $\underline{\underline{Tr}}_A$ was constructed directly to have morphisms $\langle s,w,s'\rangle$ for $w\in X^*$, $G_A^{\circledast}$ has morphisms $\langle s,p,s'\rangle$ where p is a sequence $\langle s,x_0,s_1\rangle\langle s_1,x_1,s_2\rangle\ldots$ of edges in G_A. We let the reader prove that under this correspondence $G_A^{\circledast}$ and $\underline{\underline{Tr}}_A$ are isomorphic categories. □

The second "doctrine" of Section 0.2 indicates that this path category construction, of categories from graphs, should be represented by a functor.

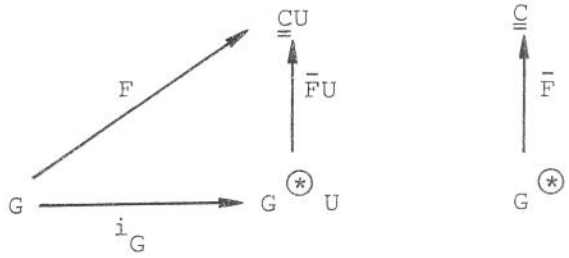
Proposition 3.3. Define $^{\circledast}:\underline{\underline{Gph}} \to \underline{\underline{Cat}}_S$ by letting $G|^{\circledast}| = G^{\circledast}$ and for $F:G \to G'$ in Gph, letting $F^{\circledast}:G^{\circledast} \to G'^{\circledast}$ be defined by $|F^{\circledast}| = |F|:V \to V'$ and $\langle v_0,p,v_1\rangle F^{\circledast} = \langle v_0|F|,pF^*,v_1|F|\rangle$, where $F^*:E^*: \to E'^*$ is the (letter-wise) homomorphic extension (See Example 2.6) of $F:E \to E'$. Then $^{\circledast}$ is a functor.

Proof. We must first show that $F^{\circledast}:G^{\circledast} \to G'^{\circledast}$ is indeed a functor, and then that $^{\circledast}$ itself is. We first check preservation of composition for $F^{\circledast}$: $(\langle v_0,p,v_1\rangle\circ\langle v_1,p',v_2\rangle)F^{\circledast} = \langle v_0,pp',v_2\rangle F^{\circledast} = \langle v_0|F|,(pp')F^*,v_2|F|\rangle = \langle v_0|F|,pF^*p'F^*,v_2|F|\rangle = \langle v_0|F|,pF^*,v_1|F|\rangle\circ\langle v_1|F|,p'F^*,v_2|F|\rangle = \langle v_0,p,v_1\rangle F^{\circledast}\circ\langle v_1,p',v_2\rangle F^{\circledast}$, using the fact that F^* preserves concatenation. Next $1_v F^{\circledast} = \langle v,\lambda,v\rangle F^{\circledast} = \langle v|F|,\lambda,v|F|\rangle = 1_{v|F|^{\circledast}|}$, as desired.

Now we show $\circledast$: $\underline{\underline{\text{Gph}}} \to \underline{\underline{\text{Cat}}}_S$ is a functor. Let $F:G \to G'$ and $F':G' \to G''$ in $\underline{\underline{\text{Gph}}}$, and let $\langle v_0,p,v_1\rangle \epsilon G^{\circledast}$. Then $\langle v_0,p,v_1\rangle (FF')^{\circledast} = \langle v_0|FF'|,p(FF')^*,v_1|FF'|\rangle = \langle v_0|F||F'|,pF^*F'^*, v_1|F||F'|\rangle = \langle v_0,p,v_1\rangle (F^{\circledast})(F'^{\circledast})$, as desired. We omit verification that $1_{G^{\circledast}} = (1_G)^{\circledast}$. □

Not only does the path category construction employ the free monoid construction, but it has a quite similar "free" or "universal" property. Let $i_G:G \to G^{\circledast}U$ be the natural injection of a graph into the underlying graph of its path category, i.e., $v|i_G| = v$ for $v\epsilon|G|$ and $ei_G = \langle e\partial_0,e,e\partial_1\rangle$ (one often thinks of this as an "inclusion", even though it really isn't). The universal property is given by the following:

Proposition 3.4. Let $\underline{\underline{C}}$ be a category and $\underline{\underline{C}}U$ its underlying graph. Then for every graph morphism $F:G \to \underline{\underline{C}}U$ there is a unique functor $\bar{F}:G^{\circledast} \to \underline{\underline{C}}$ which "extends F" in the sense that the diagram on the left



commutes in $\underline{\underline{\text{Gph}}}$.

Proof. We must have $v|F| = v|i_G\bar{F}U| = v|\bar{F}U| = v|\bar{F}|$ for $v\epsilon|G|$, and $eF = ei_G\bar{F}U = \langle e\partial_0,e,e\partial_1\rangle\bar{F}U = \langle e\partial_0,e,e\partial_1\rangle\bar{F}$ for $e\epsilon G$. Moreover $\bar{F}$ must be a functor, so we must have $1_v\bar{F} = 1_{v|\bar{F}|} = 1_{v|F|}$, and for $p = e_0\ldots e_k:v \to v'$, we must have $(\langle v,e_0,e_0\partial_1\rangle\langle e_1\partial_0,e_1,e_1\partial_1\rangle\ldots\langle e_k\partial_0,e_k,v'\rangle)\bar{F} =$

$\langle v, e_0, e_0\partial_1 \rangle \bar{F} \circ \langle e_1\partial_0, e_1, e_1\partial_1 \rangle \bar{F} \circ \cdots \circ \langle e_k\partial_0, e_k, v' \rangle \bar{F} = e_0F \circ e_1F \circ \cdots \circ e_kF$. One now easily checks that these necessary conditions in fact constitute the definition of a functor $\bar{F}$. The necessity argument gives the desired uniqueness. □

Not only are the statement and proof of 3.4 above suspiciously like those for the free monoid construction, but the latter can actually be obtained as a special case.

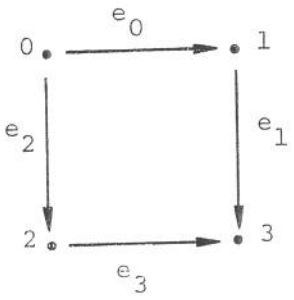
Example 3.2 (See Example 1.3.). Free Monoids. Given a set X, construct a graph E_X with node set $|E_X| = \{0\}$, edge set X, and $x\partial_0 = x\partial_1 = 0$ for all $x \in X$. Then $E_X^{\circledast}$ is a one object category with sequences from X as morphisms and concatenation as composition; i.e., $E_X^{\circledast}$ is X^* viewed as a one object category (see Example 2.5). If we now restrict Proposition 3.4 to one object categories, we in fact obtain Proposition 1.4, the freeness of X^* over X. □

3.2 DIAGRAM AND FUNCTOR CATEGORIES

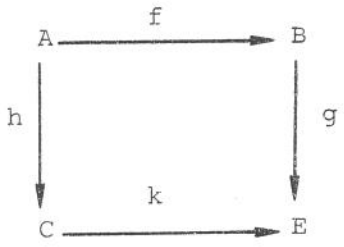
Up to this point we have used diagrams in a number of special categories (Set, Gph, etc.) as convenient and graphic abbreviations for relations holding among morphisms in these categories. We now give the general definition of a diagram, and of a commutative diagram, in an arbitrary category. We show that the collection of all diagrams with the same shape in a category $\underline{C}$ is itself a category with a natural notion of morphism. When diagrams are extended to functors, their morphisms turn out to be natural transformations (see Doctrine 3 of Section I.0.2). Using natural transformations as morphisms between functors leads to the more general notion of a functor category.

Definition 3.2. Let $\underline{\underline{C}}$ be a category and let $\underline{\underline{C}}U$ denote its underlying graph. Let G be any graph. Then a diagram in $\underline{\underline{C}}$ of shape G is a graph morphism $D:G \to \underline{\underline{C}}U$. We call G the shape or underlying graph of D. □

Example 3.3. Let A,B,C,E be sets, and let $f:A \to B$, $g:B \to E$, $h:A \to C$, $k:C \to E$ be set maps. Let G be the graph with $|G| = \{0,1,2,3\}$, and edges $e_0:0 \to 1$, $e_1:1 \to 3$, $e_2:0 \to 2$, $e_3:2 \to 3$. G is the graph normally drawn as follows:

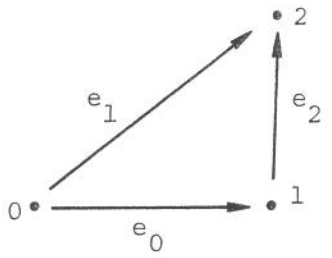


Now let $\underline{\underline{C}} = \underline{\underline{Set}}$ and define a graph morphism $D:G \to \underline{\underline{C}}U$ by $0|D| = A$, $1|D| = B$, $2|D| = C$, $3|D| = E$, $e_0D = f$, $e_1D = g$, $e_2D = h$, $e_3D = k$. The conventional representation for this is as below. □



Example 3.4. Let G_0 be the graph with $|G_0| = \{0,1,2\}$ and with edges $e_0:0 \to 1$, $e_1:0 \to 2$, and $e_2:1 \to 2$. This graph is conventionally

pictured by



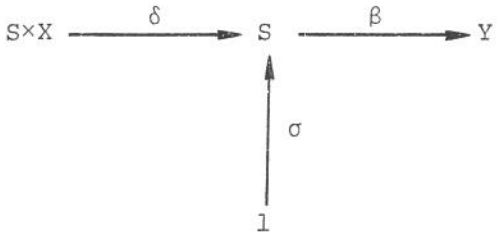
Now let $\underline{\underline{C}}$ be a category, G a graph, and $F:G \to \underline{\underline{C}}U$ a graph morphism. Define a diagram $D:G_0 \to \underline{\underline{C}}U$ by $0|D| = G$, $1|D| = G^{\circledast}U$, $2|D| = \underline{\underline{C}}U$, $e_0D = i_G$, $e_1D = F$, and $e_2D = \bar{F}U$. Then D is the diagram of Proposition 3.4 in the preceding subsection. □

Recall (from Proposition 3.3) that a graph morphism $D:G \to \underline{\underline{C}}U$ determines a unique functor $\bar{D}:G^{\circledast} \to \underline{\underline{C}}$ extending D in the sense that $D = i_G \circ \bar{D}U$.

Definition 3.3. A diagram $D:G \to \underline{\underline{C}}U$ is commutative iff for all $v,v' \in |G|$ and all $p,p':v \to v'$ in $G^{\circledast}$, $p\bar{D} = p'\bar{D}$. □

This merely says that the composites along any two paths between the same vertices are equal. Commutativity of the diagrams of Examples 3.3 and 3.4 would mean that $fg = hk$ and $F = i_G \circ \bar{F}U$, respectively.

Example 3.5. (See Example 2.9 and 2.2.) We can, if we wish, view a machine as a diagram in Set of the special form

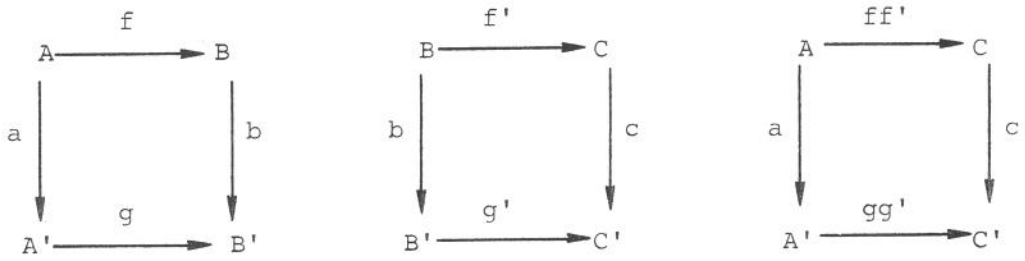


or more simply, view an automation as a diagram

$$S\times X \xrightarrow{\ \delta\ } S$$

in Set. (These pictures do not specify the underlying graphs of the diagrams; suitable ones could easily be chosen. Note that they trivially commute, and only two extra nontrivial paths occur, both in the first diagram, namely, $\delta\beta$ and $\sigma\beta$.) □

Proposition 3.5. In a category C, if the first two diagrams below commute, so does the third.



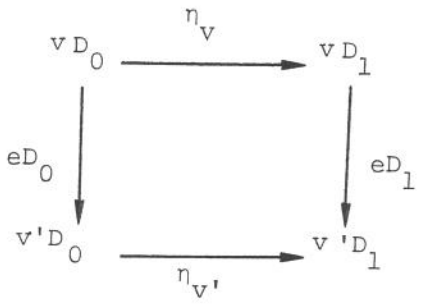
Proof. This result is proved exactly like Proposition 1.1. □

Uses of this result, and its numerous cousins of greater complexity, are refered to as instances of "diagram chasing". A general result of this form is something like, if all inner polygons of a diagram commute, then so does its outside; but this result is probably not worth stating explicitly and exactly; in any case, the reader can supply proofs of the same form as that of Proposition 1.1 for any special cases that may arise.

Since we already know what morphisms of machines and automata are, namely families of functions indexed by the objects X,S,Y such that certain diagrams commute, this example might suggest that morphisms of diagrams should be

families of morphisms satisfying appropriate commutativity conditions.

Definition 3.4. Let $D_0 : G \to \underline{\underline{C}}U$ and $D_1 : G \to \underline{\underline{C}}U$ be diagrams in $\underline{\underline{C}}$, each with shape G. Then a morphism of diagrams, $\eta : D_0 \Longrightarrow D_1$, from D_0 to D_1, is a triple $\langle D_0, \eta, D_1 \rangle$, where η is a $|G|$ indexed family $\langle \eta_v : vD_0 \Longrightarrow vD_1 \mid v \in |G| \rangle$[†] of morphisms in $\underline{\underline{C}}$ such that for all $e : v \to v'$ in G, the diagram



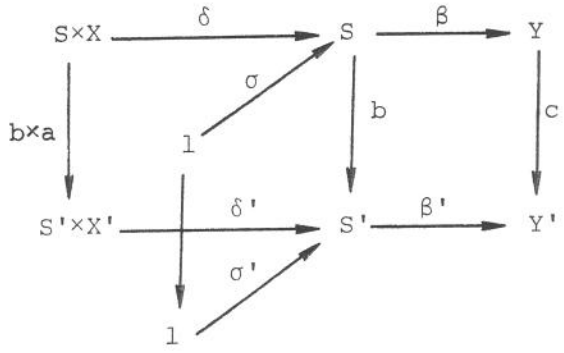
commutes in $\underline{\underline{C}}$. (Note that we use a "double arrow" notation for diagram morphisms. This is to avoid the confusion of having ordinary single arrows denote all possible kinds of morphisms.) □

Example 3.6. (See Example 3.5.) Using the shapes in Example 3.5, the machine and automaton morphisms of Example 2.9 are in fact diagram morphisms. (But note that not all the diagram morphisms are machine morphisms!) In fact, letting D and D' be the diagramatic forms of machines M and M', a morphism $D \Longrightarrow D'$ is a family of four set maps, $\eta_0 : S \times X \to S' \times X'$, $\eta_1 : S \to S'$, $\eta_2 : Y \to Y'$, and $\eta_3 : 1 \to 1$. Let b denote η_1, and note that η_3 must be the identity on the one point set $1 = \{0\}$. Since the objects attached to η_0 are

---

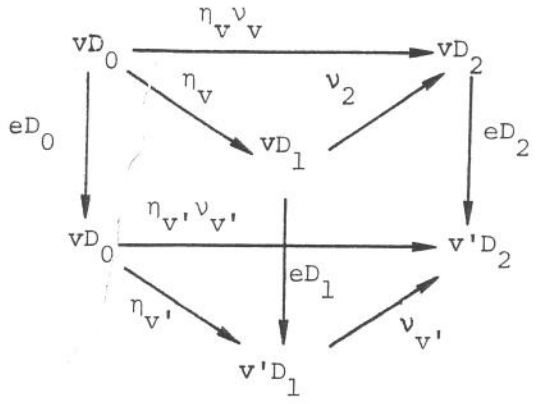
† Here and below we drop the cumbersome notation $|D_i|$ for the vertex part of graph morphism; it will be clear from context whether we are talking about the vertex (object) part or the edge (morphism) part.

products, it is reasonable to impose the additional requirement, that η_0 be a product morphism, one component of which agrees with the already given map $b:S \to S'$, i.e., that $\eta_0 = b \times a:S \times X \to S' \times X'$, for some $a:X \to X'$. Finally, letting c denote η_2, one can verify that the commutativity conditions for η to be a diagram morphism now give exactly our earlier notion of a machine homomorphism. The following commutative diagram should help visualize this.



The same considerations apply to automata. □

Example 3.7. Diagram Categories. We know that machine morphisms can be composed, yielding a category Mach of machines, and this suggests that we should also be able to compose morphisms of diagrams. In fact, if $\eta:D_0 \Longrightarrow D_1$ and $\nu:D_1 \Longrightarrow D_2$ are each morphisms of diagrams with shape G in $\underline{C}$, then we define the composite $\eta\nu:D_0 \Longrightarrow D_2$ by $(\eta\nu)_v = \eta_v \nu_v$ for each v in $|G|$. Clearly, $\eta_v \nu_v:vD_0 \to vD_2$, and to show $\eta\nu$ is a diagram morphism, it remains to show the commutativity conditions hold. This can be seen from the following commutative diagram in $\underline{C}$,



in which commutativity in the back face follows from the commutativity of all other faces. One can now verify that composition is associative and that the family $\langle 1_{vD} : vD \to vD \mid v \in |G| \rangle$ is in fact an identity morphism $1_D : D \Longrightarrow D$. Thus we have a category $\underline{\underline{\text{Diag}}}\,[G, \underline{\underline{C}}]$ of diagrams in $\underline{\underline{C}}$ with shape G. □

We have essentially shown in Example 3.6 that $\underline{\underline{\text{Mach}}}$ and $\underline{\underline{\text{Aut}}}$ are subcategories (with functions between product sets being product functions) of diagram categories $\underline{\underline{\text{Diag}}}\,[G, \underline{\underline{\text{Set}}}]$ for suitable G. This leads to the very fruitful idea of replacing $\underline{\underline{\text{Set}}}$ by some other $\underline{\underline{C}}$. For example, using the category $\underline{\underline{\text{Lin}}}_K$ of vector spaces over a field K (such as the reals R) leads to linear sequential machines, also called sampled data systems; see Goguen (1971).

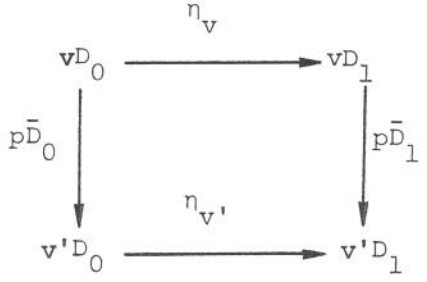
Diagrams can be viewed as functors. By Proposition 3.4, $D : G \to \underline{\underline{C}}U$ (in $\underline{\text{Gph}}$) determines a unique functor $\bar{D} : G \overset{\circledast}{\to} \underline{\underline{C}}$ such that $i_G^\circ(\bar{D}U) = D$; moreover, this same equation tells us how to recover D from $\bar{D}$ (compose i_G with U of $\bar{D}$). Thus, D and $\bar{D}$ determine each other; otherwise put, there is an isomorphism of sets

$$\underline{\underline{\mathrm{Gph}}}\ (G, \underline{\underline{C}}U) \overset{\sim}{=} \underline{\underline{\mathrm{Cat}}}\ (G^{\circledast}, \underline{\underline{C}})$$

(explicitly: send $D:G \to \underline{\underline{C}}U$ to $\bar{D}:G^{\circledast} \to \underline{\underline{C}}$). It may even be convenient to speak of a functor $G^{\circledast} \to \underline{\underline{C}}$ as a diagram in $\underline{\underline{C}}$, as is often done in the literature; c.f. Mac Lane (1971a).

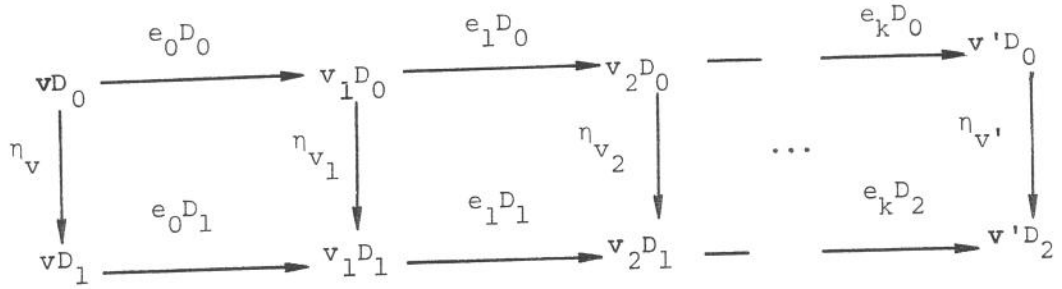
When diagrams are viewed as functors, their morphisms ought to be some special kind of "morphism of functors". These are called natural transformations. As preparation for the definition of natural transformations, we show that diagram morphisms have the appropriate property.

Proposition 3.6. Let $\eta:D_0 \Longrightarrow D_1$ be a morphism of diagrams in $\underline{\underline{C}}$ with shape G. Then for each path $p:v \to v'$ in G, the diagram

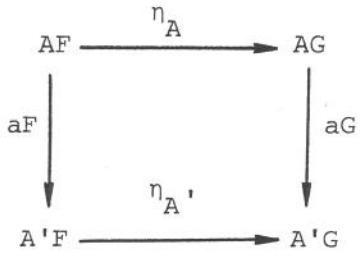


commutes in $\underline{\underline{C}}$.

Proof. Say $p = e_0 e_1 \cdots e_k$. Then (turning things sideways to save space) commutativity of each subsquare below, along with the functorality of $\bar{D}_0$ and $\bar{D}_1$, i.e., that $p\bar{D}_i = e_0 D_i e_1 D_i \cdots e_k D_i$, gives the desired square. □



Definition 3.5. Let $F, G:\underline{\underline{A}} \to \underline{\underline{B}}$ be functors. Then a natural transformation $\eta:F \Longrightarrow G$ from F to G, is a triple $\langle F,\eta,G\rangle$, where η is a $|A|$-indexed family $\langle \eta_A:AF \to AG\rangle_{A\in|\underline{\underline{A}}|}$ of morphisms in $\underline{\underline{B}}$ such that for each $a:A \to A'$ in $\underline{\underline{A}}$, the diagram



commutes in $\underline{\underline{B}}$. This commutativity condition is often called "the naturality condition." □

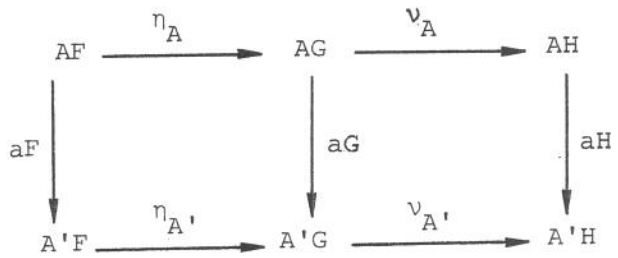
Thus, Proposition 3.6 says that a morphism of diagrams is a natural transformation of their extensions to functors. Moreover, we can compose natural transformations in exactly the same way we did diagram morphisms; namely:

Definition 3.5 continued. Given $\eta:F \Longrightarrow G$ and $\nu:G \Longrightarrow H$ where $H:\underline{\underline{A}} \to \underline{\underline{B}}$ also, define $\eta\nu:F \Longrightarrow H$ by $(\eta\nu)_A = \eta_A{}^{\circ}\nu_A:AF \to AH$; this is called the composite or composition of η with ν, or sometimes the vertical composite. Given $F:\underline{\underline{A}} \to \underline{\underline{B}}$, define $1_F:F \Longrightarrow F$ by $(1_F)_A = 1_{A|F|}:AF \to AF$ in $\underline{\underline{B}}$. It is easy to see that this is a natural transformation; we call it the identify transformation at (or for) F. □

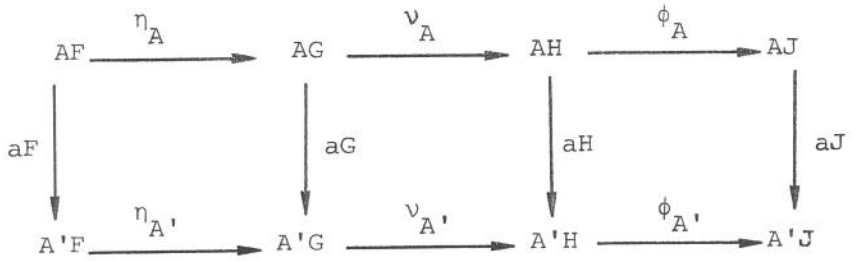
Proposition 3.7. Given $\eta:F \Longrightarrow G$ and $\nu:G \Longrightarrow H$ where $F,G,H:\underline{\underline{A}} \to \underline{\underline{B}}$, then $\eta\nu:F \Longrightarrow H$ (as defined above) is a natural transformation, and so is $1_F:F \Longrightarrow F$ (as defined above). Moreover, letting $\underline{\underline{B}}^{\underline{\underline{A}}}$ or $\underline{\underline{Nat}}(\underline{\underline{A}},\underline{\underline{B}})$ have objects all functors $\underline{\underline{A}} \to \underline{\underline{B}}$ and morphisms natural transformations

between such functors, we get a category, called a functor category or category of functors from $\underline{\underline{A}}$ to $\underline{\underline{B}}$; its identities are the identity transformations.

Proof. That $\eta\nu$ is natural is proved by concatenating the squares for naturality of η and ν, plus Proposition 3.5,



and $1_F: F \Longrightarrow F$ is trivially natural and satisfies the identify laws for composition. The associativity axiom is verified with the diagram



using Proposition 3.5 again. □

Other notations found in the literature for the functor category are $\underline{\underline{\text{Cat}}}(\underline{\underline{A}},\underline{\underline{B}})$, $\underline{\underline{\text{Fun}}}(\underline{\underline{A}},\underline{\underline{B}})$, and $[\underline{\underline{A}},\underline{\underline{B}}]$.

Proposition 3.8. For a graph G and category $\underline{\underline{C}}$, we have an isomorphism of categories

$$\underline{\underline{Diag}}(G,\underline{\underline{C}}) \overset{\sim}{=} \underline{\underline{Nat}}(G^{\circledast},\underline{\underline{C}}).$$

Proof. We have already noted the bijective correspondence between the objects in these categories, by $D \longmapsto \bar{D}$, and Proposition 3.6 gives the natural translation of diagram morphisms to natural transformations. We omit the straightforward verifications of functorality and bijectivity. □

Example 3.8. (See Example 2.18). Let $\underline{\underline{M}}$ be a monoid M viewed as a category. Then the category of all M-automata (i.e., deterministic M-transition systems) is just $\underline{\underline{Set}}^{\underline{\underline{M}}}$. For $M = X^*$, it is easily seen that this category is isomorphic to the category $\underline{\underline{Aut}}^X$ of Example 2.11. If one wants the category of M-automata, with a fixed starting state preserved by morphisms, replace $\underline{\underline{Set}}$ be the cateogry $\underline{\underline{Set}}_*$ of pointed sets (Example 2.12); i.e., consider $\underline{\underline{Set}}_*^{\underline{\underline{M}}}$. One can pass to finite state systems by replacing $\underline{\underline{Set}}$ by $\underline{\underline{Fin}}$ (or $\underline{\underline{Fin}}_*$), or to linear systems by replacing $\underline{\underline{Set}}$ by $\underline{\underline{Lin}}_K$. □

We devote the remainder of this subsection to additional examples of natural transformations. One very important class of these is the so-called natural isomorphisms. In order to consider this, we must anticipate the general notion of isomorphism which is discussed in greater detail in Section 4. Let $\underline{\underline{C}}$ be a category. Then $f:A \to B$ in $\underline{\underline{C}}$ is an <u>isomorphism</u> iff there is some $g:B \to A$ in $\underline{\underline{C}}$ such that $fg = 1_A$ and $gf = 1_B$. We also say that the objects A and B are <u>isomorphic</u> (written $A \overset{\sim}{=} B$) in this case. The reader might want to check that this notion agrees with the usual notion of bijective homomorphism at least in the categories $\underline{\underline{Alg}}_\Sigma$ of Σ-algebras (this example and others are taken up in Section 4).

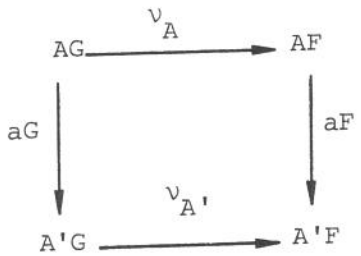
Definition 3.6. A natural equivalence or natural isomorphism of functors $F, G:\underline{\underline{A}} \to \underline{\underline{B}}$ is a natural transformation $\eta:F \Longrightarrow G$ such that for all objects $A \in |\underline{\underline{A}}|$, η_A is an isomorphism in $\underline{\underline{B}}$. □

Example 3.9. Consider functors F and $G:\underline{\underline{Set}} \times \underline{\underline{Set}} \to \underline{\underline{Set}}$ defined by $|F|:\langle A,B\rangle \mapsto A\times B$, $F:\langle f,g\rangle \mapsto f\times g$; and $|G|:\langle A,B\rangle \mapsto B\times A$, $G:\langle f,g\rangle \mapsto g\times f$; where for $f:A \to A'$, $g:B \to B'$; $f\times g:A\times B \to A'\times B'$ is defined by $\langle a,b\rangle \mapsto \langle af,bg\rangle$. For $\langle A,B\rangle \in |\underline{\underline{Set}}\times\underline{\underline{Set}}|$, let $\eta_{A,B}:A\times B \to B\times A$ be defined by $\langle a,b\rangle \mapsto \langle b,a\rangle$. Then each $\eta_{A,B}$ is an isomorphism in $\underline{\underline{Set}}$; and $\eta:F \Longrightarrow G$ is a natural transformation, which expresses the commutativity of Cartesian product up to isomorphism. One often writes, somewhat loosely, $\eta:A\times B \Longrightarrow B\times A$, naming the functors by their values. Similarly, one has a natural equivalence $(A\times B)\times C \Longrightarrow A\times(B\times C)$ expressing the natural associativity of Cartesian product up to isomorphism; and another pair, $1\times A \Longrightarrow A$ and $A\times 1 \Longrightarrow A$, expressing a natural identity law for the one point set 1. □

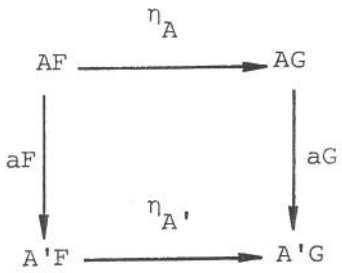
These examples, and many others of the same type, point toward the intuitive content of the word "natural": if we had chosen some "unnatural" family of isomorphisms $\eta_{A,B}:A\times B \to B\times A$, the commutativity condition would have failed, but by choosing them all uniformly or "coherently" and "naturally", it does hold. See the discussion of Doctrine 3 in Section 0.2. The following result is often helpful in dealing with natural isomorphisms, and shows that the two ways of thinking of them are equivalent.

Proposition 3.9. A natural transformation $\eta:F \Longrightarrow G$ of functors $F,G:\underline{\underline{A}} \to \underline{\underline{B}}$, is a natural isomorphism iff η is an isomorphism in the functor category $\underline{\underline{Nat}}(\underline{\underline{A}},\underline{\underline{B}})$.

Proof. If η is a natural isomorphism, then each $\eta_A : AF \to AG$ has an inverse, say $\nu_A : AG \to AF$, such that $\eta_A \nu_A = 1_{AF}$ and $\nu_A \eta_A = 1_{AG}$. Then the family of ν_A's constitutes a natural transformation $\nu : G \Longrightarrow F$. For given $a : A \to A'$ in $\underline{\underline{A}}$, the square



commutes in $\underline{\underline{B}}$, because its inverse square

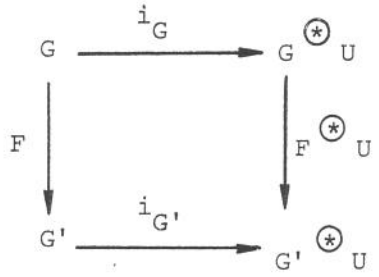


does (in equations, $aF \circ \eta_{A'} = \eta_A \circ aG$ implies $\eta_A^{-1} \circ aF = aG \circ \eta_{A'}^{-1}$). Now clearly $\nu\eta = 1_G$ and $\eta\nu = 1_F$ that is, F and G are inverses in $\underline{\underline{B}}^{\underline{\underline{A}}}$.

Conversely, if there is an $\nu : G \Longrightarrow F$ such that $\nu\eta = 1_G$ and $\eta\nu = 1_F$, then for all $A \in |\underline{\underline{A}}|$ $\nu_A \eta_A = 1_{AG}$ and $\eta_A \nu_A = 1_{AF}$, so η_A is an isomorphism. □

Example 3.10. The family $i_G : G \to G \circledast U$ of the graph morphisms used in

Proposition 3.4 constitute a natural transformation $i:1_{\underline{Gph}} \Longrightarrow \circledast U$, where $1_{\underline{Gph}}:\underline{Gph} \to \underline{Gph}$ denotes the identity functor, noting that the composite $\circledast U$ is also a endofunctor of $\underline{Gph}$. Now let $F:G \to G'$ be a graph morphism. We have to check that



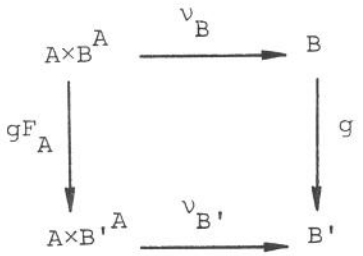
commutes in $\underline{Gph}$. Let $e:v \to v'$ in G. Then $ei_G \circ F \circledast U = \langle v,e,v'\rangle F*U = \langle v|F|,eF,v'|F|\rangle$ (see Proposition 3.3) while $eFi_{G'}$ comes out the same. □

Example 3.11. Exactly the same considerations apply to the free monoid construction. Let X be a set, and let $i_X:X \to X*U$ denote the injection of X into X*U as strings of length one, where X*U is the underlying set of the monoid X*. Then $i:1_{\underline{Set}} \Longrightarrow \_*U$ is a natural transformation, where $\_*U$ denotes the composite of the functors $\_*:\underline{Set} \to \underline{Mon}$ and $U:\underline{Mon} \to \underline{Set}$. □

Example 3.12. Evaluation. Fix a set A. Then for any set B, there is an "evaluation function" $\nu_B:A\times B^A \to B$, defined by $\langle a,f\rangle\nu_B = af$, the evaluation of $f:A \to B$ at the argument $a\in A$. The family of ν_B's is in fact a natural transformation, to the identity functor on $\underline{Set}$ from the functor $F_A:\underline{Set} \to \underline{Set}$ defined as follows: $B|F_A| = A\times\underline{Set}(A,B)$ $(= A\times B^A)$; and for $g:B \to B'$, $gF_A:A\times B^A \to A\times B'^A$ is given by $\langle a,f\rangle(gF_A) = \langle a,fg\rangle$.

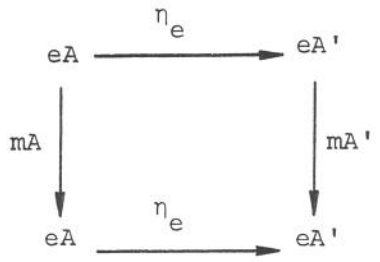
F_A is a functor, because it is the composite of functors $\underline{\underline{Set}}(A, \_)$ (Example 2.30) and $A \times \_$ (Example 2.31).

Naturality of $\langle \nu_B : BF_A \to B \rangle_{B \in |\underline{\underline{Set}}|}$ is commutativity of



for each $g : B \to B'$, which follows from the calculation $\langle a,f \rangle \nu_B \; g = (af)g = a(fg) = \langle a,fg \rangle \nu_{B'} = \langle a,f \rangle (gF_A) \nu_{B'}$. This can be interpreted as saying that function evaluation is a natural process in the category of sets. □

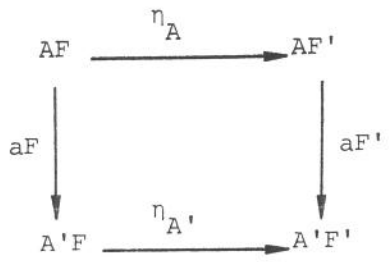
Example 3.13. (See Example 3.8). Let $\underline{\underline{M}}$ be a monoid M viewed as a category, and recall (Example 2.18) that $\underline{\underline{Set}}^{\underline{\underline{M}}}$ is the category of M-automata. Then there is an interesting bifunctor (i.e., functor of two arguments) which evaluates an M-automaton's behavior on its inputs. It is defined using Proposition 2.4 and the following two functors: $G_e : \underline{\underline{Set}}^{\underline{\underline{M}}} \to \underline{\underline{Set}}$ sending A to eA and $\eta : A \to A'$ to $\eta_e : eA \to eA'$; and $H_A : \underline{\underline{M}} \to \underline{\underline{Set}}$ sending e to eA and m to mA. Note that $e|H_A| = A|G_e| = eA$, and that for $\eta : A \to A'$ in $\underline{\underline{Set}}^{\underline{\underline{M}}}$ and $m \in \underline{\underline{M}}$, the diagram



commutes (in $\underline{\underline{Set}}$). In fact this is precisely the naturality condition for η (it looks kind of degenerate because $\underline{\underline{M}}$ has only one object, e). Therefore (Proposition 2.4) there is a unique functor $\nu:\underline{\underline{M}}\times\underline{\underline{Set}}^{\underline{\underline{M}}} \to \underline{\underline{Set}}$ such that $\langle e,A\rangle\ \nu = eA$, $\langle m,A\rangle\nu = mA$, $\langle e,\eta\rangle\ \nu = \eta_e$. □

*Example 3.14. Evaluation of functors on arguments makes fine sense when the arguments come from arbitrary category $\underline{\underline{A}}$ rather than just a one object category. Thus, fix $\underline{\underline{A}}$, and define $F_{\underline{\underline{A}}}:\underline{\underline{Cat}} \to \underline{\underline{Cat}}$ as follows: $\underline{\underline{B}}\left|F_{\underline{\underline{A}}}\right| = \underline{\underline{A}}\times\underline{\underline{B}}^{\underline{\underline{A}}}$; and for $G:\underline{\underline{B}} \to \underline{\underline{B}}'$ in $\underline{\underline{Cat}}$, $GF_{\underline{\underline{A}}}:\underline{\underline{A}}\times\underline{\underline{B}}^{\underline{\underline{A}}} \to \underline{\underline{A}}\times\underline{\underline{B}}'^{\underline{\underline{A}}}$ sends $\langle A,F\rangle$ to $\langle A,FG\rangle$ (object part), and sends $\langle a,\eta:F \to B'\rangle$ to $\langle a,\eta F\rangle$ (morphism part), where $(\eta F)_A = \eta_A F$.

Now for $A\epsilon\left|\underline{\underline{A}}\right|$, define $G_{\underline{\underline{A}}}:\underline{\underline{B}}^{\underline{\underline{A}}} \to \underline{\underline{B}}$ by $F\left|G_{\underline{\underline{A}}}\right| = AF$ and $\eta G_{\underline{\underline{A}}} = \eta_A$. And for $F:\underline{\underline{A}} \to \underline{\underline{B}}$, define $H_F:\underline{\underline{A}} \to \underline{\underline{B}}$ by $A\left|H_F\right| = A\left|F\right|$, and $aH_F = aF$. Note that $F\left|G_{\underline{\underline{A}}}\right| = A\left|H_F\right| = AF$, and that for $\eta:F \to F'$ and $a:A \to A'$, the diagram

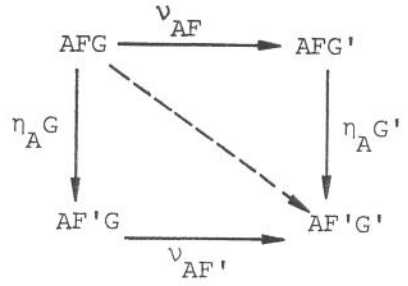


commutes, exactly by naturality of η! Thus (by Proposition 2.4) there is a unique bifunctor $\nu_{\underline{\underline{A}}}:\underline{\underline{A}}\times\underline{\underline{B}}^{\underline{\underline{A}}} = \underline{\underline{B}}\ F_{\underline{\underline{A}}} \to \underline{\underline{B}}$.

In fact, this family of functors is a natural transformation $\nu:F_{\underline{\underline{A}}} \to 1_{\underline{\underline{Cat}}}$, as can be verified more or less as in Example 3.12. □

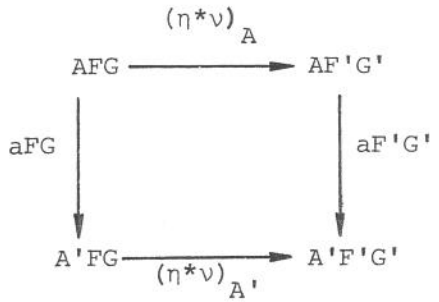
Not only can natural transformations be composed "vertically" as in Proposition 3.7 $((\eta\eta')_A = \eta_A\eta'_A)$ but also "horizontally", with one another and with functors. This gives rise to quite a rich structure on $\underline{\underline{Cat}}$ which is used in a number of situations, including composition in the category of flow diagram programs in the following section. The reader is hereby forewarned that the proofs below are somewhat tedious, and that the motivation is deferred. Before beginning, recall that for $F:\underline{\underline{A}} \to \underline{\underline{B}}$, $1_F:F \Rightarrow F$ is the natural transformation with $(1_F)_A = 1_{AF}:AF \to AF$.

Proposition 3.10. Given functors $F,F':\underline{\underline{A}} \to \underline{\underline{B}}$ and $G,G':\underline{\underline{B}} \to \underline{\underline{C}}$ and natural transformations $\eta:F \Rightarrow F'$ and $\nu:G \Rightarrow G'$, then define $(\eta*\nu)_A$ to be the diagonal of the following commutative square

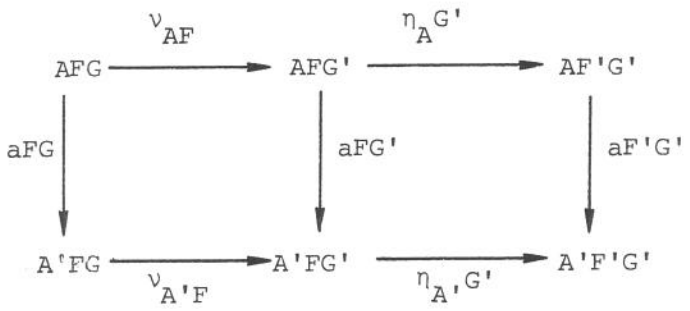


giving a natural transformation $\eta*\nu:FG \Rightarrow F'G'$. This horizontal composition operation on transformations is associative, in the sense that $\eta*(\nu*\phi) = (\eta*\nu)*\phi$ (when everything is defined). Moreover, if $I_{\underline{\underline{B}}}$ denotes the identity transformation on the functor $1_{\underline{\underline{B}}}:\underline{\underline{B}} \to \underline{\underline{B}}$, then $\eta*I_{\underline{\underline{B}}} = \eta$ and $I_{\underline{\underline{A}}}*\eta = \eta$. Thus, there is a category whose objects are categories and whose morphisms are natural transformations, $\eta:F \Rightarrow G:\underline{\underline{A}} \to \underline{\underline{B}}$, with horizontal composition, and identity $I_{\underline{\underline{A}}}:1_{\underline{\underline{A}}} \Rightarrow 1_{\underline{\underline{A}}}:\underline{\underline{A}} \to \underline{\underline{A}}$ at $\underline{\underline{A}}$. This category is denoted $\underline{\underline{Nat}}$.

Proof. We first check that the above square does commute; this is by naturality of ν with respect to the morphism $\eta_A:AF \to AF'$. Next we have to check that $\eta*\nu$ is itself natural. So letting $a:A \to A'$ in $\underline{\underline{A}}$, we need commutativity of

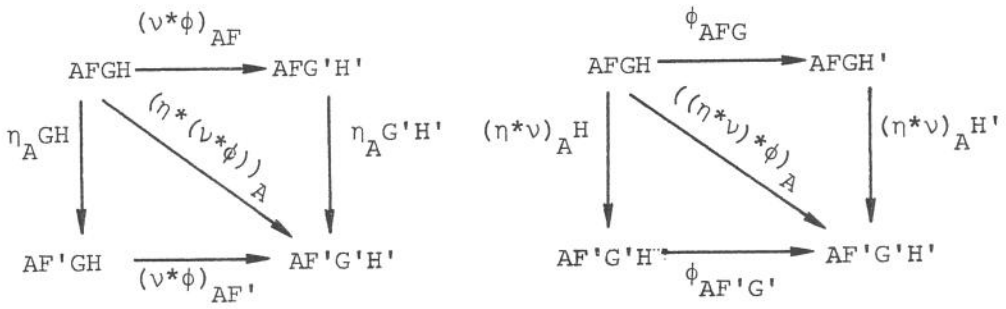


which follows (using Proposition 3.7 as usual) from commutativity of

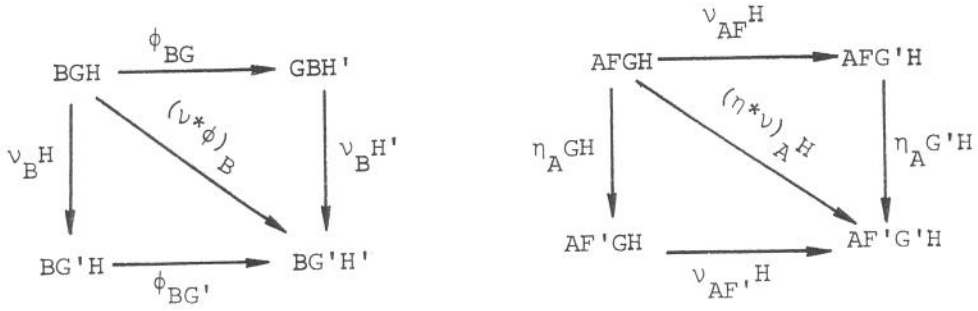


the left square of which commutes by naturality of ν with respect to $aF:AF \to A'F$, and the right square of which commutes by applying G' to the square for naturality of η with respect to $a:A \to A'$. So $\eta*\nu$ is a natural transformation $FG \Rightarrow F'G'$.

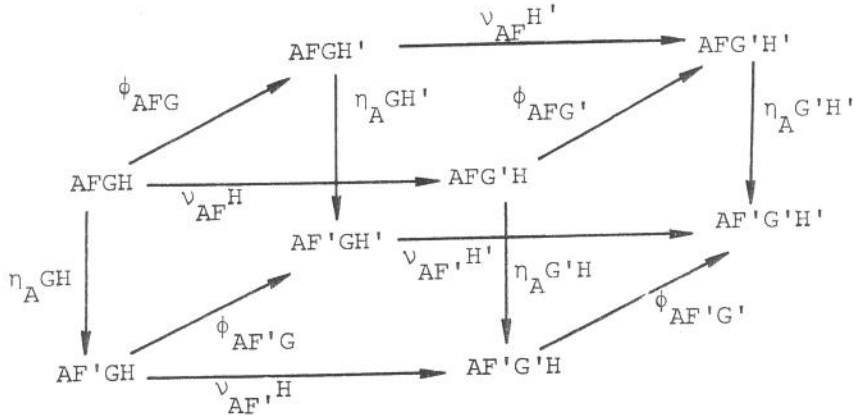
Now assume in addition $H,H':\underline{\underline{C}} \to \underline{\underline{D}}$ and $\phi:H \Rightarrow H'$. We want to show that $\eta*(\nu*\phi) = (\eta*\nu)*\phi$; that is, we want to show that the diagonals of the following two squares are equal



where $(\nu*\phi)_B$ and $(\eta*\nu)_A H$ are given by

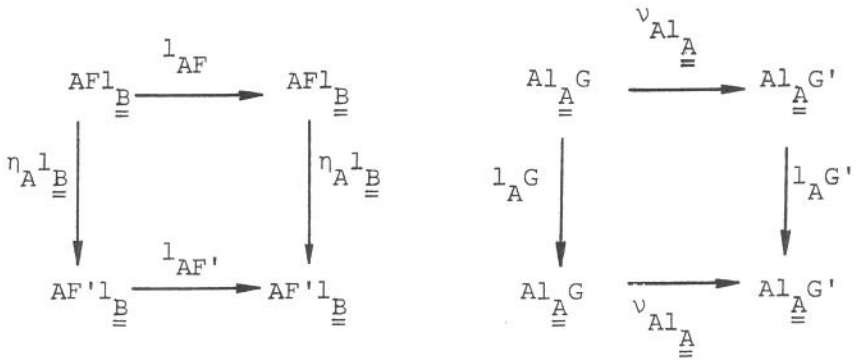


and of course $(\eta*\nu)_A H'$ is the same with H replaced by H'. The desired result follows from contemplation of the diagram



possibly labelling the various diagonals as above; with B each AF and AF'; the main diagonal from AFGH to AF'G'H' is both $(\eta*(\nu*\phi))_A$ and $((\eta*\nu)*\phi)_A$.

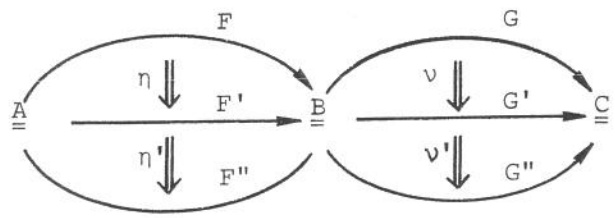
The facts about identities are much simpler, and follow from commutativity of the squares below



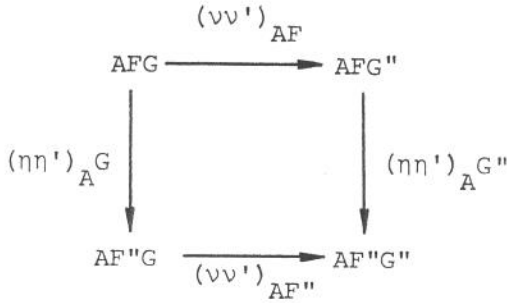
the first of which arises by substituting $\nu = I_{\underline{\underline{B}}}$ and $G = G' = 1_{\underline{\underline{B}}}$ in the definition of $(\eta*\nu)_A$, and the second of which arises by substituting $\eta = I_{\underline{\underline{A}}}$ and $F = F' = 1_{\underline{\underline{A}}}$. □

A special case frequently of interest, for example in the next section, is when one of the transformations is an identity. When $\eta = 1_F$ (and $F' = F$), i.e., η is the identity transformation on the functor F; then it is convenient to use the same symbol for the functor and the identity transformation (c.f. Mac Lane (1971a) p. 43). Here $\eta*\nu$ is written $F*\nu:FG \Longrightarrow FG'$, and when $\nu = 1_G$ (and $G' = G$), $\eta*\nu$ is written $\eta*G:FG \Longrightarrow F'G$. The associative and identity laws proved above also apply to this case. A further result, of perhaps less general interest, is the following.

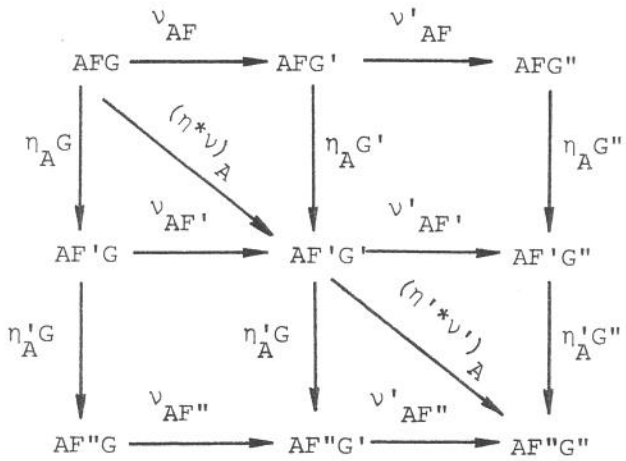
*Proposition 3.11. Given $\eta:F \Rightarrow F':\underline{\underline{A}} \to \underline{\underline{B}}$, $\nu:G \Rightarrow G':\underline{\underline{B}} \to \underline{\underline{C}}$, $\eta':F' \Rightarrow F'':\underline{\underline{A}} \to \underline{\underline{B}}$ and $\nu':G' \Rightarrow G'':\underline{\underline{B}} \to \underline{\underline{C}}$, then the double law is satisfied, $(\eta\eta')*(\nu\nu') = (\eta*\nu)(\eta'*\nu')$.



Proof. $((\eta\eta')*(\nu\nu'))_A$ is given by the diagonal of the square



while $((\eta*\nu)(\eta'*\nu'))_A$ is given by the composite of the diagonals of



all subsquares of which commute by naturality, and whose outside edges composed yield the top square above, whose diagonal is therefore equal to that of the lower square. □

There is a kind of distributivity law which follows immediately from the double law

*Corollary 3.12. Given $\eta:F \Rightarrow F':\underline{\underline{A}} \to \underline{\underline{B}}$, $\nu:G \Rightarrow G':\underline{\underline{B}} \to \underline{\underline{C}}$ and $\nu':G' \Rightarrow G":\underline{\underline{B}} \to \underline{\underline{C}}$, then $\eta*(\nu\nu') = (\eta*\nu)(F'*\nu') = (F*\nu)(\eta*\nu')$.

Proof. Recalling that with the notation described above F denotes the identity transformation on the functor F, we can apply 3.11 with $\eta = \eta F' = F\eta$. □

These last results, together with Proposition 3.7, can be summarized by saying that the class of natural transformations can be viewed as a category in two ways: taking categories as objects, i.e., $(\eta:F_0 \Rightarrow F_1: \underline{\underline{A}}_0 \to \underline{\underline{A}}_1)\partial_i = \underline{\underline{A}}_i$, with horizontal composition (*), or taking functors as objects, i.e., $(\eta:F_0 \Rightarrow F_1:\underline{\underline{A}}_0 \to \underline{\underline{A}}_1)\partial_i = F_i$, with vertical composition. Such a class of morphisms with two compositions satisfying the appropriate double law (Proposition *3.11) is called a "double category" (by Ehresmann (1965); see also Kelly and Street (1974) for a detailed discussion of these concepts.) In the special case where every horizontal identity is also a vertical identity, a double category is called a "2-category," and that is the case for natural transformations because $I_{\underline{\underline{A}}}:1_{\underline{A}} \Rightarrow 1_{\underline{\underline{A}}}:\underline{\underline{A}} \to \underline{\underline{A}}$ is the identity transformation on the identity functor and the vertical identities consist of all identity transformations (on arbitrary functors). It might seem that these structures are of such generality and abstractness as to be of no possible interest for computer science; yet in Section 4.6 we shall indicate (but not prove) that the program homomorphisms of the following subsection form a double category which is not a 2-category.

*3.3 FLOW DIAGRAM PROGRAMS

In this section we take some steps toward a unified theory of flow diagram semantics, including correctness and termination, following ideas of Burstall (1972) (where the categorical approach to program proofs first

appeared) and Goguen (1972a, 1974) (who introduced homomorphisms). This material is interesting as an illustration of the concepts and techniques of the preceeding sections, and has been claimed (by Goguen (1974); see also Milner (1972)) to have significant practical applications, in giving more compact (by factors of three to ten) proofs of correctness and termination. Here we consider only non-recursive programs in flow diagram form (see Burstall and Thatcher (1974) for hints on how to handle recursion) and in particular, we avoid all questions of programming language syntax (see Morris (1973) for a translation from a syntactially specified language to a flow diagram language.)

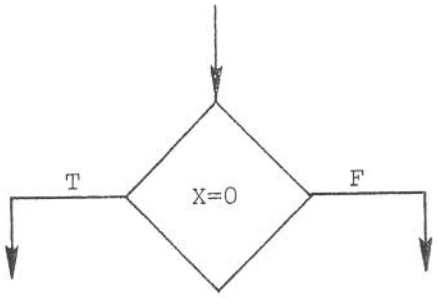
Semantic theories are for describing what programs are supposed to do, and then for proving that they do it. Floyd (1967) and Naur (1966) seem to have taken the first steps toward giving precise formulations of correctness methods, based on earlier work of McCarthy (1963, 1963a). †

---

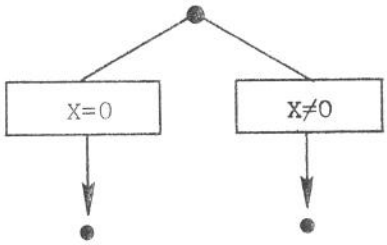
† Knuth (1968) says, "The idea of inductive assertions actually appeared in embryonic form in 1946, at the same time the concept of flow charts was introduced by H. H. Goldstein and J. von Neumann." [See von Neumann (1963), Volume 5, pp 91-99.] These original flow charts included "assertion boxes" closely related to Floyd's inductive assertions. A. Turing (1949) in "Checking a large routine," wrote, "How can one check a routine in the sense of making sure that it is right? In order that the man who checks may not have too difficult a task, the programmer should make a number of definite assertions which can be checked individually and from which the correctness of the whole programme easily follows." This is the essence of the idea we are exploring here. Manna (1969) clarifies and expands some of Floyd's work using the predicate calculus, while Milner (1971) puts some aspects in a set-theoretic and algebraic setting. More recently, Burstall (1972) has emphasized the problem of validating the correctness methods themselves. This seems to be quite a significant issue in semantics of programming languages, requiring a firm mathematical formulation such as is presented here.

From the practical point of view, the methods for proving properties of programs suggested in this section are not category theoretic, but are rather set-theoretic. The categorical framework is expeditions for the theoretical development, for validating the correctness methods; but the methods themselves are much more elementary. Indeed, this simplicity is their advantage over (for example) the logical approach, which is characterized by tedious manipulations of highly complex forms. In a set-theoretic approach one is free to choose whatever mathematical representation one finds convenient, to introduce new notation, to take advantage of the whole tradition of informal mathematical communication with its well-known advantages of efficiency; whereas in a logical approach, one is bound to a specific language with a specific restricted syntax, usually an applied first order calculus (see Manna (1969)). These points will be illustrated in examples in this section.

Flow diagrams are usually viewed as graphs with operations and tests at nodes and with edges determining flow of control. As Karp (1959) observed, it is convenient (for some purposes) to have all operations appear as labels on edges, with tests then represented by partial functions. For example the test



occurs in an "edge-labeled" program as



where in the latter picture, X = 0 (respectively X≠0) denotes the partial identity function defined exactly when X = 0 (respectively X≠0).

So we consider flow diagram programs as graphs (Definition 3.1) with edges "labeled" by partial functions. Recall, from 1.3 that a partial function from set A to set B is a triple $\langle A,f,B\rangle$ where $f \subseteq A\times B$ such that for each $a\in A$ there is at most one $b\in B$ such that $\langle a,b\rangle\in f$, and that we often write $f:A \multimap B$ for $\langle A,f,B\rangle$. By the domain (or set) of definition of $f:A \multimap B$ we mean the set $def(f) = \{a \mid \exists\, b\ \langle a,b\rangle\in f\}$ Our approach relies on some special structural properties of the category $\underline{\underline{\underline{Pfn}}}$ (Example 2.12) whose objects are sets and whose morphisms are partial functions: each hom set $\underline{\underline{\underline{Pfn}}}(A,B)$ is a poset under the usual "inclusion" ordering of partial functions ($\langle A,f,B\rangle \sqsubseteq \langle A,f',B\rangle$ iff $f \subseteq f'$); least upper bounds of bounded sets and of directed sets exist ($S \subseteq \underline{\underline{\underline{Pfn}}}(A,B)$ is directed, iff every finite subset of S has an upper bound in S; S is bounded if S has an upper bound; the least upper bound of S is written

$\bigsqcup$S); and composition preserves all least upper bounds that exist, i.e., composition is continuous by components: for $S \subseteq \underline{\underline{\underline{Pfn}}}(A,B)$ and $S' \subseteq \underline{\underline{\underline{Pfn}}}(B,C)$ if $\bigsqcup S$ and $\bigsqcup S'$ both exist then

$$(\bigsqcup S)\circ(\bigsqcup S') = \bigsqcup\{s\circ s' \mid s\in S \text{ and } s'\in S'\}.$$

Definition 3.7. A $\underline{\underline{\underline{Pfn}}}$-flow-diagram is a diagram $P:G \to \underline{\underline{\underline{Pfn}}}U$ in $\underline{\underline{\underline{Pfn}}}$ such that if e and e' are two edges in G with common source, then eP and eP' have disjoint domains of definition. □

Commonly, the same set is assigned to each node, with elements of that set corresponding to what McCarthy (1963) called state vectors, while the partial functions assigned to edges represent computational steps, including conditionals or branches via the representation described above.

Example 3.15. Let G be the graph with node set {a,b,c,d,e}, with edge set {<a,b>,<b,c>,<c,d>,<d,b>,<b,e>}, and with the source and target functions, as indicated in Figure 3.1.

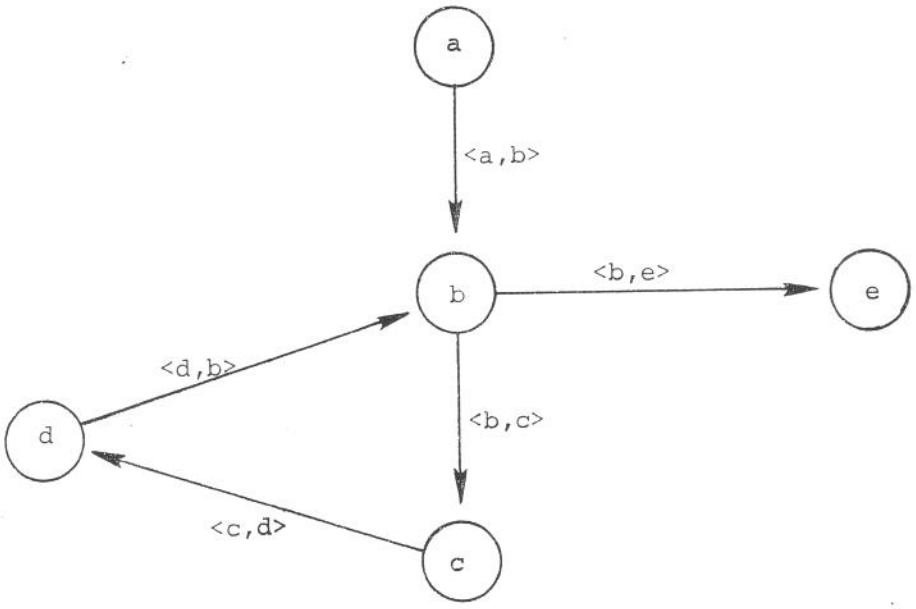


Figure 3.1

Now let $\underline{Id}$ be the set $\{X,Y,Z\}$ of "identifiers" and $S = \omega^{\underline{Id}}$ the set of "environments". A particular flow diagram P of shape G (Figure 3.1) is defined by taking $vP = S$ for all nodes v of G and by assigning partial functions to the edges of G as indicated in Figure 3.2. The expressions in boxes denote functions from

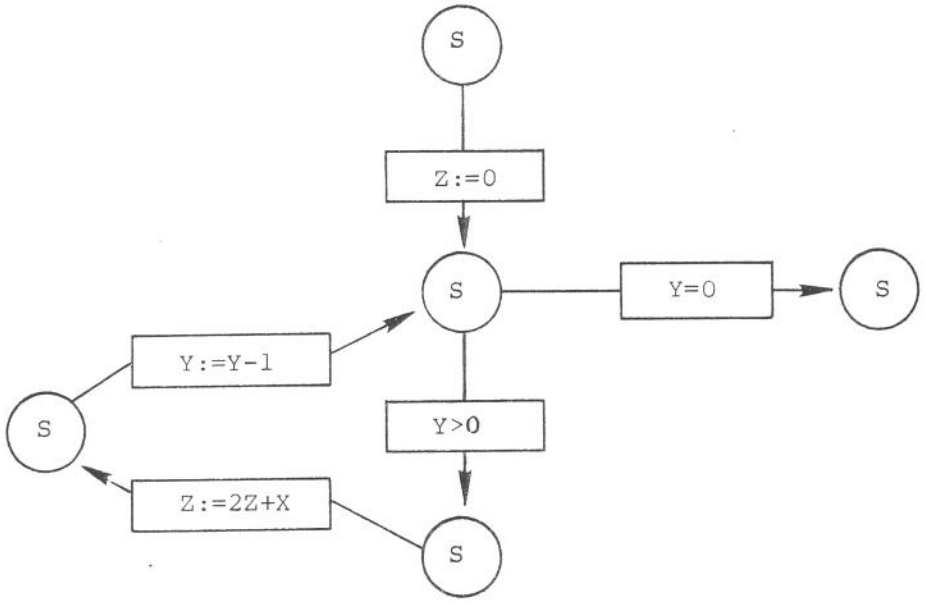


Figure 3.2

environments to environments. For example $\langle a,b\rangle P:S \to S$ is the function (denoted $Z:=0$ in Figure 3.2) taking $s \in S$ to s' where $Zs' = 0$, $Xs' = Xs$ and $Ys' = Ys$; $\langle a,e\rangle P:S \to S$ (denoted $Y:=0$ in the figure) is the partial identity function on S defined at $s \in S$ iff $Ys = 0$. Hereafter we will feel free to abbreviate function definitions like the above in the form "$\langle X,Y,Z\rangle \mapsto \langle X,Y,0\rangle$"; and "$\langle X,Y,Z\rangle \mapsto \langle X,Y,Z\rangle$ iff $Y = 0$". □

The partial function computed along some path of a flow diagram is simply the composition of the partial functions labeling its edges.

Proposition 3.4 puts this concisely: for any $\underline{\underline{\underline{Pfn}}}$-flow-diagram $P:G \to \underline{\underline{\underline{Pfn}}}U$, there exists a (unique) functor $\bar{P}:G^{\circledast} \to \underline{\underline{\underline{Pfn}}}$ which extends P in the sense that $i_G(\bar{P}U) = P$. Moreover for each path $p:v \to v'$ in G, $p\bar{P}:vP \to v'P$ is (as can be seen in the proof of Proposition 3.4) the partial function computed along that path. The isomorphism $\underline{\underline{\underline{Gph}}}(G, \underline{\underline{\underline{Pfn}}}U) \tilde{=} \underline{\underline{\underline{Cat}}}(G^{\circledast}, \underline{\underline{\underline{Pfn}}})$ (see the discussion following Example 3.7) says we could equivalently define a flow diagram to be a functor $\bar{P}:G^{\circledast} \to \underline{\underline{\underline{Pfn}}}$ as in Goguen (1972a); this is convenient for some proofs, but here we have taken the diagram version (as in Burstall (1972)) for the basic definition.

Example 3.15 continued. In Figure 3.2, the function computed along the path <a,b><b,c><c,d><d,b><b,c><c,d><d,b><b,e>, from node a to node e going twice around the loop, is the partial function $\langle X,Y,Z\rangle \longmapsto \langle X,0,3X\rangle$ iff $y = 2$. □

To describe the computation from one node in a flow diagram to another, we must join together computations along alternative paths: intuitively this should be the union of partial functions computed along the paths. To show that this union is actually a function, we need

Proposition 3.13. If $P:G \to \underline{\underline{\underline{Pfn}}}U$ is a $\underline{\underline{\underline{Pfn}}}$-flow-diagram, if v is a node of G, and if f,f' are paths with source v such that neither is

an initial segment of the other[†], then $f\bar{P}$ and $f'\bar{P}$ have disjoint domains of definition.

Proof. Write $f = geh$ and $f' = ge'h'$ with edges $e \neq e'$ (and g possibly the identity path for v). Then the domains of definition, $def(f\bar{P})$ and $def(f'\bar{P})$, of $f\bar{P}$ and $f'\bar{P}$ are contained in $(def(eP))(g\bar{P})^{-1}$ and $(def(e'P))(g\bar{P})^{-1}$ respectively, which are disjoint because $def(eP)$ and $def(e'P)$ are disjoint. □

Definition 3.8. A Pfn-program is a triple $\langle P, v_0, v_1 \rangle$ such that $P:G \to \underline{\underline{\underline{Pfn}}}U$ is a Pfn-flow-diagram, and v_0, v_1 are "distinguished" nodes of G called entry and exit respectively, with respect to which G is (fully) protected in the sense that $v_0\partial_1^{-1} = v_1\partial_0^{-1} = \emptyset$, (i.e., no edges go out of v_1 or into v_0) and is reachable in the sense that every node of G lies on some path from v_0 to v_1. □

The requirement that the shape G of P be fully protected and reachable means that the entry and exit nodes are uniquely determined. Because of this we shall generally speak of just a Pfn-program P, assuming that the entry and exit nodes in the shape of P have been specified; and sometimes we will drop the prefix "Pfn" since the context of the section makes it clear that we are considering only one rather special model of flow diagram programs. Note, however, that at the end of the section we do consider the replacement of Pfn by other categories $\underline{\underline{C}}$ and introduce at that time an appropriate notation.

---

† In $G^{\circledast}$, $f:v \to v'$ is an initial segment of $f':v \to v''$ iff there exists $g:v' \to v''$ such that $fg = f'$.

Example 3.16. The flow diagram P of Example 3.15 gives rise to a Pfn-program <P,a,e>. Here is another with Id = {U,V,W,X,Y,Z}, with $S=\omega^{Id}$, and with the uppermost node as entry and rightmost node as exit.

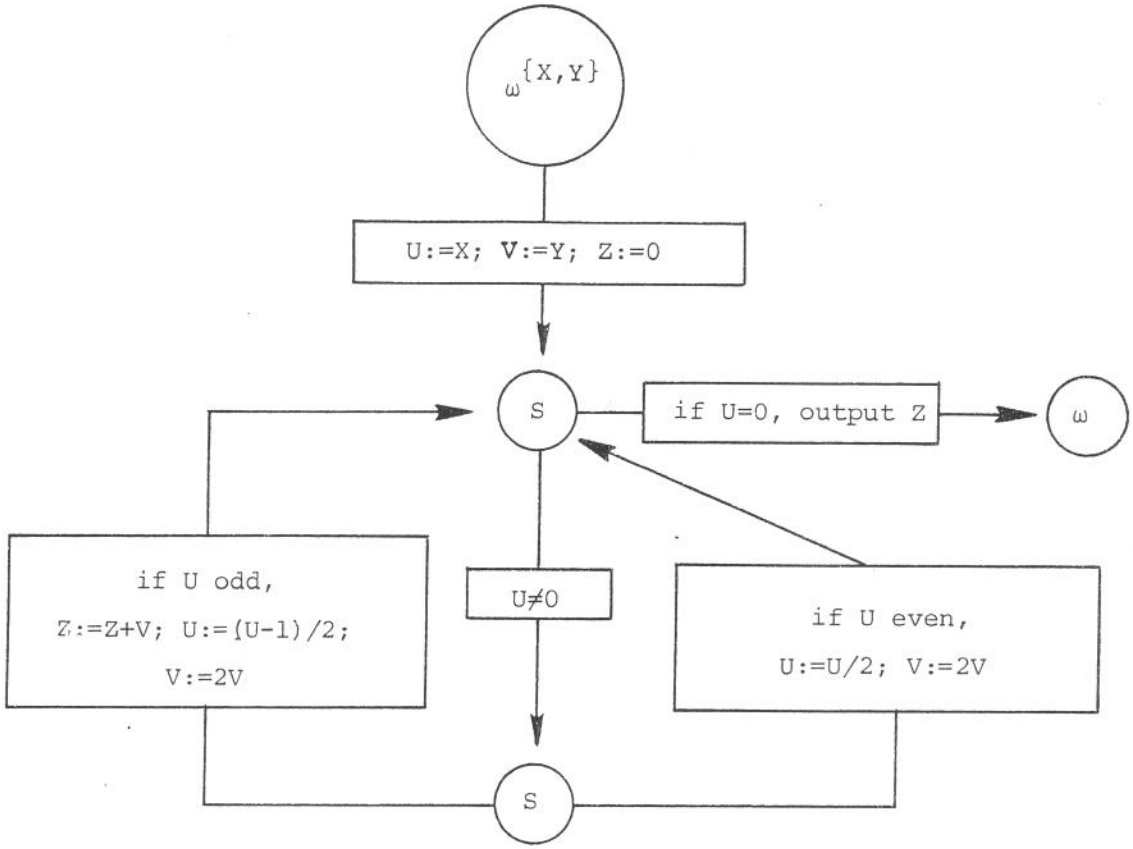


Figure 3.3

This isn't such an easy program to guess the behavior of, and even when one has a good guess, one ought to have some way of verifying it. This requires a rigorous definition of "behavior", which we now supply. □

Definition 3.9. If $\langle P,v_0,v_1\rangle$ is a $\underline{\underline{\underline{\text{Pfn}}}}$-program, let

$$P\mathcal{B} = \bigsqcup\{f\bar{P} \mid f\in G \circledast (v_0,v_1)\}$$

be called the behavior of P. □

Corollary 3.14. If P is a $\underline{\underline{\underline{\text{Pfn}}}}$-program, then $P\mathcal{B}:v_0P \to v_1P$ is a partial function.

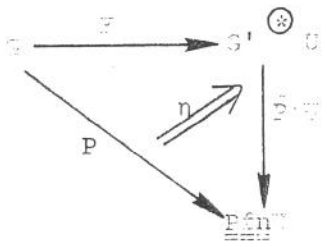
Proof. By protection, no path in $G \circledast (v_0,v_1)$ is an initial segment of another. Therefore, by Proposition 3.13, the join in the definition of $P\mathcal{B}$ is a join of partial functions with disjoint domains of definition, and is therefore itself a partial function. □

Example 3.15 continued. Letting $\langle P,a,e\rangle$ be the program of Figure 3.2 again, we claim that $P\mathcal{B}$ is the total function $\langle X,Y,Z\rangle \mapsto \langle X,0,(2^Y-1)X\rangle$ from "state vectors" to "state vectors". (Of course, in practice, P would probably be "used" to compute the function $\langle X,Y\rangle \mapsto (2^Y-1)X$.) □

To formulate and justify proofs of these kinds of assertions about programs we introduce the following concept.

Definition 3.10. Let $\langle P:G \to \underline{\underline{\underline{\text{Pfn}}}}U,v_0,v_1\rangle$ and $\langle P':G' \to \underline{\underline{\underline{\text{Pfn}}}}U,v_0',v_1'\rangle$ be two programs. Then a program homomorphism from $\langle P,v_0,v_1\rangle$ to $\langle P',v_0',v_1'\rangle$ is a pair $\langle F,\eta\rangle$ where $F:G \to G' \circledast U$ is a graph morphism called the shape part of the homomorphism, and $\eta:P \Longrightarrow F(\bar{P}'U)$ is a diagram morphism, called the operation part of the homomorphism, such that $v_iF = v_i'$ for $i = 0,1$. □

The following diagram may help visualize this situation,



The intuitive meaning is that $\langle F,\eta\rangle : P \to P'$ maps edges of P (consistently) to paths in P', via F; and maps operations performed along these edges correspondingly via η. Of course η doesn't map operations at all, but its naturality (Definition 3.4 and Proposition 3.6) does impose a consistency constraint upon operations along paths in G' relative to the corresponding operations along paths in G. In particular, if each η_v is an inclusion, the operation $eF(\bar{P}'U)$ must equal eP on the domain of definition of eP.

If the shape of P is finite (has a finite number of edges) then it is a finite process to verify whether or not $\langle F,\eta\rangle : P \to P'$ is a program homomorphism, or, for that matter, to construct a program homomorphism. These finiteness considerations are important for the examples and applications; in particular, this observation underlies the finite character of the method for proving correctness described by Floyd (1967).

Before making a connection between morphisms of programs and their behavior, we give names for some special kinds of program homomorphisms:

Definition 3.10 continued. A program homomorphism $\langle F,\eta\rangle:P \to P'$ is a restriction iff $\bar{F}_{v_0v_1}:G \circledast (v_0,v_1) \to G' \circledast (v_0',v_1')$ is surjective and each η_v is an inclusion; P is said to be a restriction† of P'. $\langle F,\eta\rangle$ is a simulation iff, again, $\bar{F}_{v_0v_1}$ is surjective and each η_v is injective; in this case we say that P' simulates P. $\langle F,\eta\rangle$ is a projection iff $\bar{F}_{v_0v_1}$ is surjective and η_{v_0} is a total function. Finally, $\langle F,\eta\rangle$ is an equivalence iff it is a simulation in which each η_v is an identity. □

Note that $\bar{F}_{v_0v_1}$ being surjective means that each path in G' from v_0' to v_1' can be obtained from some path in G from v_0 to v_1. This might suggest that when $G' \circledast (v_0',v_1')$ is infinite (i.e., when there are infinitely many paths from v_0' to v_1' in G') that there are an infinite number of conditions to be checked. Fortunately, as long as G' is finite the condition for surjectivity reduces to a finitely verifiable condition, even when $G' \circledast$ is infinite. In fact it need only be checked that each simple path†† which contributes to paths in G' can be obtained from some path in $G \circledast (v_0,v_1)$.

---

† Burstall (1972) would say that P' is a "conservative restriction" of P, but in effect F is limited to be the identity (actually the injection $i_G:G \to G \circledast U$). Goguen (1972) says that $\langle F,\eta\rangle$ is a "simulation" but comments that the more general definition of "simulation" that we give here might be of interest. Milner's notion of "verification" despite his restriction to programs the shape of Figure 3.3 comes closest in a way to our "restriction", as Proposition 3.13 below indicates.

†† A path $p:v \to v'$ is simple iff it is not an identity $\langle v,\lambda,v\rangle$ and has no repeated edges, i.e., if $p = p_0e_0p_1e_1p_2$ then $e_0 \neq e_1$.

Example 3.17. We now construct a program homomorphism which is later used to prove correctness of the program P of Example 3.15. For $x,y\in\omega$, let P_{xy} be the program indicated by Figure 3.4,

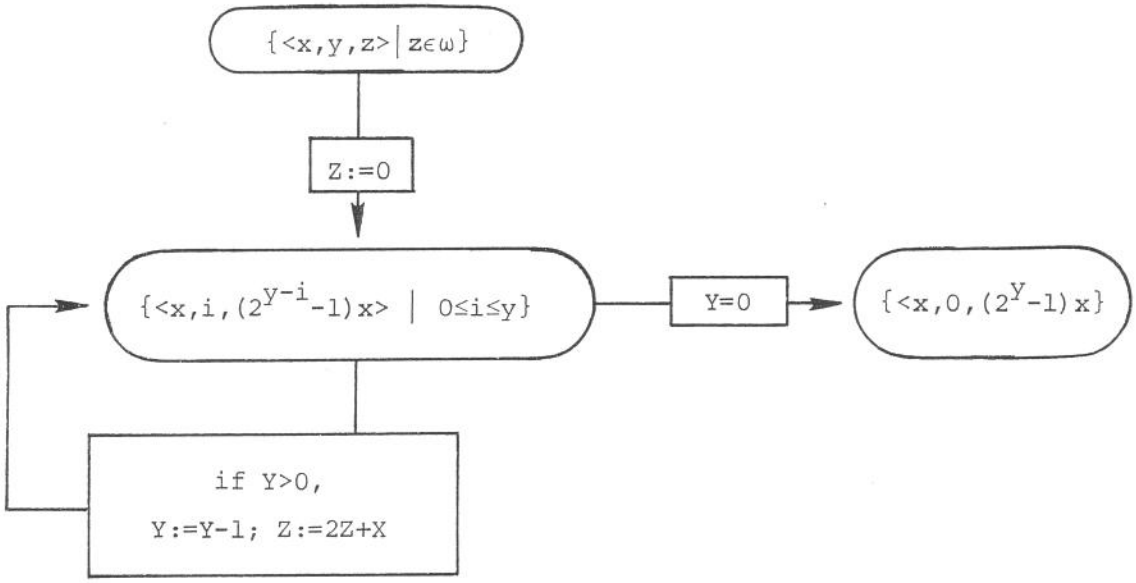


Figure 3.4

and let the nodes of its shape G_0 be a,b,e, in order from top to right, so that its three edges are <a,b>,<b,b>, and <b,e>, computing respectively the (partial) functions $\langle X,Y,Z\rangle \mapsto \langle X,Y,0\rangle$, $\langle X,Y,Z\rangle \mapsto \langle X,Y-1,2Z+X\rangle$ iff $Y > 0$, and $\langle X,Y,Z\rangle \mapsto \langle X,0,Z\rangle$ iff $Y = 0$; note we are using X,Y,Z as variables in these definitions. We leave the reader the task of checking that these three functions each actually take elements of their source sets to elements of their target sets.

Now let $F:G_0 \to G \circledast U$ (this is the G of Figure 3.1) send each a,b,e, and $\langle a,b\rangle,\langle b,e\rangle$ to themselves, and send $\langle b,b\rangle$ to the path $\langle b,c\rangle\langle c,d\rangle\langle d,b\rangle$. This is certainly a graph morphism. Now for $v\in\{a,b,e\}$, let η_v be the appropriate inclusion†, e.g., $\eta_b : s \longmapsto \langle Xs,Ys,Zs\rangle$. For $\langle F,\eta\rangle$ to be a program homomorphism $P_{xy} \to P$, three commulative squares must be verified, one for each edge in G_0. These squares amount to saying that the functions on edges of P_{xy} are birestrictions of their correspondents from P (see Proposition 1.2), and their commutativity is easily verified.

Note in particular that the source and target of this homomorphism have rather different shapes, that the sets on nodes of P_{xy} are quite different from one another, and that $\langle F,\eta\rangle$ is what we called a restriction in Definition 3.10, since F_{ae} is surjective (each path from the entrance, around the loop $n\geq 0$ times and out to the exit in P can be obtained from the corresponding one in P_{xy}). □

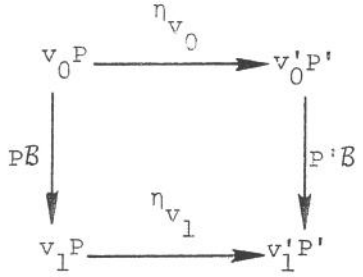
In order to apply homomorphisms to correctness and termination proofs, we need explicit relationships between the behaviors of programs P and P' connected by a program homomorphism $\langle F,\eta\rangle : P \to P'$. Think of $P\mathcal{B}$ as the partial function from the "input state set" v_0P to the "output state set" v_1P, and consider $\eta_{v_1} : v_1P \to v_1'P'$ as an "output decoding" from the output state set of P to that of P'. Also think of $\eta_{v_0} : v_0P \to v_0'P'$ as

---

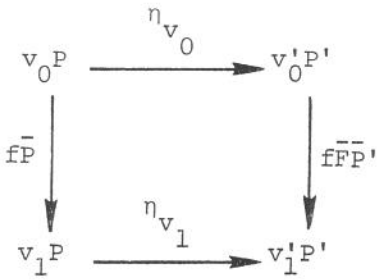
† As explained in Example 3.15, this is only a change of notation, since we are using $\langle x,y,z\rangle$ to denote $s:\{X,Y,Z\} \to \omega$ with $Xs = x$, $Ys = y$, $Zs = z$.

an "input encoding" (recall $v_iF = v'_i$ for $i=0,1$) mapping input states of P to those of P'. The relationship is that computation of P followed by the output decoding is a subfunction of input encoding followed by the computation of P'. We show below that this result permits the extension of $\mathcal{B}$ to a functor from programs to behaviors; it is also the basis of our approach to the so called "Floyd's method".

Theorem 3.15. Let $\langle F,\eta\rangle : P \to P'$ be a program homomorphism. Then $(P\mathcal{B})\eta_{v_1} \sqsubseteq \eta_{v_0}(P'\mathcal{B})$; and if $\bar{F}_{v_0v_1} : G^{\circledast}(v_0,v_1) \to G'^{\circledast}(v'_0,v'_1)$ is surjective, then $(P\mathcal{B})\eta_{v_1} = \eta_{v_0}(P'\mathcal{B})$.



Proof. Proposition 3.5 implies that the square



commutes in $\underline{\underline{\underline{\text{Pfn}}}}$ for all paths $f:v_0 \to v_1$ in $G^{\circledast}$. Then by taking least upper bounds on both sides, we have $\bigsqcup_{f\in G^{\circledast}(v_0,v_1)} (f\bar{P})\eta_{v_1} = \bigsqcup_{f\in G^{\circledast}(v_0,v_1)} \eta_{v_0}(f\bar{\bar{F}}\bar{\bar{P}}')$. By continuity of composition in $\underline{\underline{\underline{\text{Pfn}}}}$, $(P\mathcal{B})\eta_{v_1} = (\bigsqcup_{f\in G^{\circledast}(v_0,v_1)} f\bar{P})\eta_{v_1} = \eta_{v_0}(\bigsqcup_{f\in G^{\circledast}(v_0,v_1)} f\bar{\bar{F}}\bar{\bar{P}}') \sqsubseteq \eta_{v_0}(P'\mathcal{B})$; this last step because $\{f\bar{F} \mid f\in G^{\circledast}(v_0,v_1)\} \subseteq G'^{\circledast}(v_0', v_1')$. Moreover, if $\bar{F}_{v_0v_1}$ is surjective then this set inclusion is equality, and so is the partial function inclusion above. □

Corollary 3.16. If $\langle F,\eta\rangle:P \to P'$ is a restriction, then $P\mathcal{B} \sqsubseteq P'\mathcal{B}$; moreover, if $\langle F,\eta\rangle:P \to P'$ is a simulation and η_{v_0}, η_{v_1} are identities, then $P\mathcal{B} = P'\mathcal{B}$. □

An intuitive description of "Floyd's (1967) method" is that it proves a program correct by attaching assertions at each node of a program and verifying that each assertion (except the one at the entry point) follows from the preceeding assertion or assertions. Floyd's "Verification theorem" says that if these verifications exist and if the program is started in a state satisfying the initial assertion, then the program, if it halts, halts in a state satisfying the output assertion; this is called partial correctness. We employ a set theoretic definition *a la* Milner (1971), rather than a propositional definition like that suggested by Floyd (1967) and Manna (1969).

Definition 3.11. A program P is partially correct with respect to $\langle S_0, S_1 \rangle \subseteq \langle v_0P, v_1P \rangle$ iff $S_0(P\mathcal{B}) \subseteq S_1$, i.e., iff whenever $\alpha \in S_0$ and $\alpha P\mathcal{B}$ is defined, then $\alpha P\mathcal{B} \in S_1$. □

To connect with the logical approach, one can consider the sets involved to be the extensions of the propositional assertions, i.e., the sets of states satisfying the assertions. Attaching assertions to a program edges (nodes for us) corresponds to our having another program with the assertion sets on its nodes; this program may even have a different shape.

Proposition 3.17 (Verification theorem). If P is a restriction of P', then P' is partially correct with respect to $\langle v_0P, v_1P \rangle$.

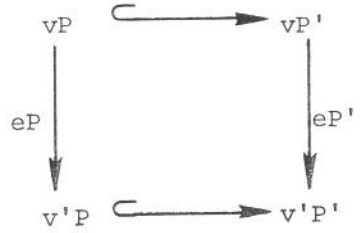
Proof. By Theorem 3.15, a restriction implies the commutativity of diagram (3.11), and since η_{v_0} and η_{v_1} are inclusions, Proposition 1.2 says that for $\alpha \in v_0P$ and $\alpha(P'\mathcal{B})$ defined, then $\alpha(P'\mathcal{B}) \in v_1P$. □

One consequence of the verification theorem is that if the shape of P' is finite (so that it is a finite process to check that P is a restriction of P') then it is a finite process to verify correctness. The process is to show that the inclusions $\eta_v : vP \to vP'$ satisfy the naturality square, which in turn is the same as showing that if $x \in vP$ (x satisfies the assertion vP), if $e : v \to v'$ is an edge of the shape of P, and if $x(eP)$ is defined, then $x(eP) \in v'P$ ($x(eP)$ satisfies the assertion $v'P$) after executing edge e.

Both Milner (1971) and Manna (1969) state Proposition 3.15 as an "iff", but we prefer to separate the converse as a completeness result for the assertion method, to wit:

Proposition 3.18 (Completeness Theorem). If P' is partially correct with respect to $\langle S_0, S_1\rangle$ then there exists a program P which is a restriction of P' such that $\langle v_0P, v_1P\rangle = \langle S_0, S_2\rangle$.†

Proof. Define P to have the same shape as P' with $v_iP = S_i$ $(i = 0,1)$ and for all nodes v such that $v \neq v_0$ and $v \neq v_1$, $vP = \{s(w\bar{P}') \mid w \in G^{\circledast}(v_0, v) \text{ and } s \in S_0\}$. The edges $e: v \to v'$ of this (minimal) restriction are assigned partial functions by $eP = vP \restriction eP' \upharpoonright v'P$ (birestriction). With this definition P is a Pfn-program, and to get a homomorphism with graph part F the identity, we must check that the inclusions $\eta_v: vP \subseteq vP'$ work, i.e., that for all $e: v \to v'$ in the shape of P'



commutes. First if $v' \neq v_1$, assume $x \in vP$, i.e., there is some $w: v_0 \to v$ with $s(w\bar{P}') = x$ for some $s \in S_0 = v_0P$; then $s(we\bar{P}') = s(w\bar{P}')(eP') \in v'P$ by the definition of $v'P$, so $x(eP') = x(vP \restriction eP' \upharpoonright v'P) = x(eP)$. If $v' = v_1$, then by the hypothesis, for all $w: v_0 \to v_1$ and $s \in S_0$, $s(w\bar{P}') = s(P\mathcal{B}) \in S_1$. □

† Proposition 3.16 is Burstall's (1972) corollary to Theorem 2 which in turn corresponds to a special case of our Theorem 3.14. Together, Propositions 3.17 and 3.18 compare with Milner's (1971) Theorem 3.1 (noting that his programs are of very restricted shape) and Manna's (1969) Theorem 1.

Example 3.17 continued. We prove correctness of the program of Example 3.15 using the homomorphism of Example 3.17 and Proposition 3.16. In fact, we get that if $s \in \{\langle x,y,z\rangle \mid z \in \omega\}$ then $s(P_{xy}\mathcal{B}) \in \{\langle x,0,(2^y-1)x\rangle\}$ provided $s(P_{xy}\mathcal{B})$ is defined; i.e., partial correctness with respect to each pair $\langle \{\langle x,y,z\rangle \mid z \in \omega\}\ ,\ \{\langle x,0,(2^y-1)x\rangle\}\rangle$ for $x,y \in \omega$; i.e., P in effect computes the function $\langle x,y\rangle \mapsto (2^y-1)x$ whenever it halts. □

We now give a form of Proposition 3.17 which is purely set theoretic (does not mention homomorphisms) and is phrased to be particularly convenient for direct application.

Corollary 3.19. Let $\langle P, v_0, v_1\rangle$ be a $\underline{\underline{\underline{\text{Pfn}}}}$-program with shape G, and for each $v \in |G|$ let vS be a subset of vP such that whenever $e:v \to v'$ is an edge of G, then

$$vS(eP) \subseteq v'S$$

(the image of vS under eP is contained in v'S). Then P is partially correct with respect to $\langle v_0S, v_1S\rangle$.

Proof. Define a $\underline{\underline{\underline{\text{Pfn}}}}$-program $S:G \to \underline{\underline{\underline{\text{Pfn}}}}U$ with node v labelled vS and edge $e:v \to v'$ labelled by the birestriction $vS \restriction eP \upharpoonright v'S$. Then S is a restriction of P, so by Proposition 3.17 we are done. □

It remains to show how this result can be wielded efficaciously in proving programs.

Example 3.18. Here we prove partial correctness of the program of Example 3.16. Let the entry node be labelled a, the exit node d, and the two internal nodes b,c, with c the bottom one. Let $aS = \{\langle x,y\rangle\}$

for some fixed $x,y,\in\omega$; let $bS = \{\langle u,v,x,y,xy-uv\rangle \mid u\leq x\}$, $cS = \{\langle u,v,x,y,xy-uv\rangle \mid 0 < u \leq x\}$, $dS = \{x\cdot y\}$. Let the two edges from c to b be denoted even, odd; and let the other edges be denoted as usual by their juxtaposed source-target word. Then $aS(abP) = \{\langle x,y,x,y,0\rangle\} \subseteq bS$; $bS(bdP) = \{xy-uv\} = \{xy\}$ since $u = 0$; $bS(bcP) = cS$ since $u \neq 0$; $cS(\text{even } P) = \{\langle u/2,2v,x,y,xy-uv\rangle \mid 0 < u \leq x\} \subseteq bS$; and $cS(\text{odd } P) = \{\langle (u-1)/2,2v,x,y,xy-uv\rangle \mid 0 < u \leq x\} \subseteq bS$ since $xy - \frac{u-1}{2}\cdot 2v+v = xy-uv$. Thus, this program multiplies (if it terminates). □

Example 3.19. Warshall's Algorithm. This is a somewhat more complicated correctness proof, although our approach seems to be simpler than any other we know in the literature. (Warshall (1962) gives a fairly informal proof in two printed pages, but he may have an incorrect assertion; Aho-Ullman (1972) prove a (generalized) version also in two printed pages, but an error was pointed out by Wegner (1974), who gives a three type-written page proof; Martin-Cleveland (1974) give a rather detailed version which runs five typed pages.) The purpose of this algorithm is to compute the transitive closure of a Boolean matrix.

Fix $n>0$, let $2 = \{0,1\}$ as usual, and let $M = 2^{n^2}$; then elements of M are n-by-n Boolean matrices, and will be indicated with an underline, e.g., $\underline{a}\in M$. In Figure 3.5 the variables J and A refer to ω and M respectively.

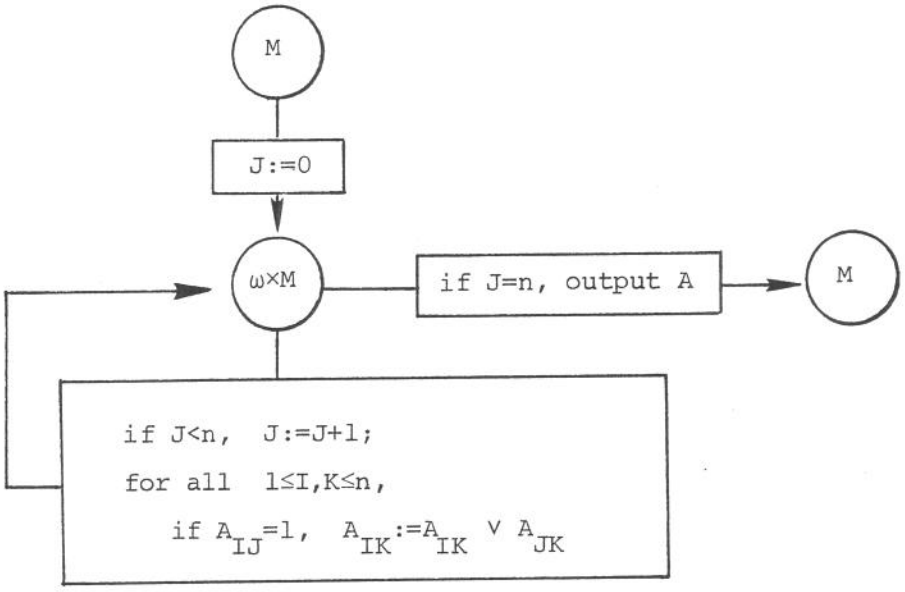


Figure 3.5

For $j<n$ the function on the loop, sends $\langle j,\underline{a}\rangle \in \omega\times M$ to $\langle j+1,\underline{a}'\rangle$

where $\underline{a}'_{ik} = \begin{cases} \underline{a}_{ik} \vee \underline{a}_{(j+1)k} & \text{if } a_{i(j+1)}=1 \\ \underline{a}_{ik} & \text{if } a_{i(j+1)}=0, \end{cases}$

i.e., $\underline{a}'_{ik} = \underline{a}_{ik} \vee (\underline{a}_{i(j+1)} \wedge \underline{a}_{(j+1)k})$. We can express this in terms of a matrix operations as $\underline{a}' = \underline{a} \vee (\underline{a}_{*(j+1)} \circ \underline{a}_{(j+1)*})$ where $\underline{a}_{*(j+1)}, \underline{a}_{(j+1)*}$ are the $(j+1)$st column and row (respectively) of $\underline{a}$, $\vee$ is componentwise, and $\circ$ is Boolean matrix multiplication. This description is similar to that used by Martin-Cleveland (1974).

Now for the (partial) correctness. Let the entry, internal and exit nodes be denoted v_0, c, v_1 respectively. Appealing to Corollary 3.9, let $v_0S=\{\underline{a}\}$, for some $\underline{a}\in M$; let $\underline{b}(j)$ denote the matrix $\underline{a}\circ(\bigvee_{m=0}^{j}(\underline{a}/j)^m)$ for $0\leq j\leq n$, where $(\underline{a}/j)\in M$ has the first j rows of $\underline{a}$ and is elsewhere 0 (note that $(\underline{a}/0)$ is the zero matrix $\underline{0}$, and $\underline{0}^0=\underline{1}$, the identity matrix); let $cS = \{\langle j,\underline{b}(j)\rangle \mid 0\leq j\leq n\}$; and finally, let $v_1S = \underline{a}^+ = \bigvee_{j=1}^{n+1}\underline{a}^j$, the transitive closure of $\underline{a}$. Checking the hypotheses of 3.9, we compute: $v_0S(v_0cP) = \langle 0,\underline{a}\rangle \in cS$ since $\bigvee_{m=0}^{0}(\underline{a}/0)^m = (\underline{a}/0)^0 = \underline{1}$, and $\underline{a}\circ\underline{1} = \underline{a}$. Next, $\langle j,\underline{a}\circ\bigvee_{m=0}^{j}(\underline{a}/j)^m\rangle(cv_1P)$ is defined only if $j=n$, in which case we have $\langle n,\underline{a}\circ\bigvee_{m=0}^{n}(\underline{a}/n)^m\rangle(cv_1P) = \underline{a}\circ\bigvee_{m=0}^{n}(\underline{a}/n)^m = \underline{a}\circ\bigvee_{m=0}^{n}\underline{a}^m = a^+$.

Finally $\langle j,\underline{b}(j)\rangle(ccP) = \langle j+1,\ \underline{b}(j)\vee(\underline{b}(j)_{*j+1}\circ\underline{b}(j)_{j+1*})\rangle$, for $0\leq j<n$, and we must prove that this equals $\langle j+1,\ \underline{b}(j+1)\rangle$, i.e., for all $i,k, 0\leq i,k\leq n$

$$(*)\qquad \underline{b}(j+1)_{ik} = \underline{b}(j)_{ik} \vee (b(j)_{*j+1}\circ\underline{b}(j)_{j+1*})_{ik}.$$

The following argument illustrates the convenience of not being tied down to any particular logical formalism. Let $G_{\underline{a}}$ be the graph with nodes $\{1,2,\ldots,n\}$ and an edge $i\to k$ iff $\underline{a}_{ik} = 1$. Call a path $p:i\to k$ j-<u>suitable</u> if it has $\leq j+1$ edges and every internal node is $\leq j$. From $\underline{b}(j) = \underline{a}\circ(\bigvee_{m=0}^{j}(\underline{a}/j)^m)$ we see $\underline{b}(j)_{ik} = 1$ iff there is a j-suitable edge $p:i\to k$[†]. Also, $(\underline{b}(j)_{*(j+1)}\circ\underline{b}(j)_{(j+1)*})_{ik} = 1$ iff $\underline{b}(j)_{i(j+1)} = \underline{b}(j)_{(j+1)k} = 1$ i.e., iff there exist j-suitable paths $p:i\to j+1$ and

---

[†] A consideration of the meaning of Boolean matrix multiplication proves this: $(\underline{a}\circ\underline{b})_{ik} = 1$ iff there is an h such that $\underline{a}_{ih} = 1$ and $\underline{b}_{hk} = 1$.

$p':j+1 \to k$. But then since $pp':i \to k$ containing only interior nodes $\leq j+1$ it can be shortened to a $j+1$-suitable path by deleting subpaths which begin and end at the same node. Thus if $\underline{b}(j)_{ik} \vee (\underline{b}(j)_{*(j+1)} \circ \underline{b}(j)_{(j+1)*})_{ik} = 1$ then there is a $j+1$-suitable path $q:i \to k$ and so $\underline{b}(j+1)_{ik} = 1$.

Conversely, if $\underline{b}(j+1)_{ik} = 1$ then there must exist a $j+1$-suitable path $q:i \to k$. If q does not hit node $j+1$ then since all its internal nodes are $\leq j$, we can shorten q to a j-suitable path $p:i \to k$ so that $\underline{b}(j)_{ik} = 1$. While if q hits node $j+1$ then, since all its internal j-suitable paths $p:i \to j+1$ and $p':j+1 \to k$ so that $(\underline{b}(j)_{*(j+1)} \circ \underline{b}(j)_{(j+1)*}) = 1$. Indeed let p be the initial subpath of q from i to the first occurrence of $j+1$ and let p' be the final subpath of q from the last occurrence of $j+1$ to k, then p and p' have fewer than $j+1$ edges and hit only nodes $\leq j$.

Thus (*) holds and so the algorithm is partially correct with respect to $\langle\{\underline{a}\},\{\underline{a}^+\}\rangle$ by Corollary 3.19. □

It must be noted that we are considering "abstract programs" in these examples, using abstract mathematical data structures and operations. We could show equivalence of a concrete program to an abstract program already proved correct in order to establish its correctness. For example, a version of Warshall's algorithm coded in ALGOL 60 could be proved equivalent to the above abstract program.†

---

† For such an approach, see the forthcoming Masters thesis of L. Tai at UCLA which treats the automatic translation of ALGOL 60 programs into set-theoretic semantic models.

We now turn to questions of termination. A program terminates iff $P\mathcal{B}$ is a total function. Theorem 3.14 gives a termination proof method in:

Corollary 3.20. If $\langle F,\eta\rangle:P \to P_1$ is a projection (η_{v_0} is total, $\bar{F}_{v_0v_1}$ is surjective) and if P_1 terminates, then P terminates.

Proof. With $\bar{F}_{v_0v_1}$ surjective, Theorem 3.15 gives $(P\mathcal{B})\eta_{v_1} = \eta_{v_0}(P_1\mathcal{B})$ and the right-hand side is a total function, thus so is $P\mathcal{B}$. □

The method of proving termination justified by Corollary 3.20 is to construct a (simpler) program P_1 and a projection $\langle F,\eta\rangle:P \to P_1$. In any case we need to know that P_1 terminates and Floyd (1967) suggests a large class of such programs which can be shown to terminate by "well ordering". We make this precise in

Proposition 3.21. Let $\langle W,\leq\rangle$ be a well ordered set[†] and $P:G \to \underline{\underline{\text{Pfn}}}U$ with G finite such that:

(1) $vP \subseteq W$ for all vertices v of G;

(2) For each vertex $v \neq v_1$

$$\bigsqcup\{\text{def}(eP) \mid e\partial_0 = v\} = vP$$

(3) For each $x\in W$, $v\in|G|$ and simple path $p:v \to v$,

$$\text{if } x\in\text{def}(p\bar{P}) \text{ then } x > x(p\bar{P}).$$

Then P terminates for all inputs $x\in v_0P$.

[†] A well ordering is a linear ordering (i.e., for all x and y, $x \leq y$ or $y \leq x$) such that each nonempty subset has a least element.

Proof. By a "computation" we shall mean a sequence

$$\begin{array}{ccccccccc} x_0 & & x_1 & & & & x_{n-1} & & x_n \\ & e_0 & & e_1 & \cdots & & & e_n & \\ v_0 & & v_1 & & & & v_{n-1} & & v_n \end{array}$$

such that $e_i : v_i \to v_{i+1}$ and $x_{i+1} = x_i(e_i P)$. By condition (2), if v_n is not the exit of P then there exists $e : v_n \to v_{n+1}$ and $x_n \in \mathrm{def}(eP)$. Thus a non-halting computation must be infinite and some edge occurs infinitely often, say, $e = (e_{i_1} : v_{i_1} \to v_{i_1+1}) = e_{i_2} = e_{i_3} \ldots$ with $i_1 < i_2 < i_3 \ldots$. By condition (3) $x_{i_1} > x_{i_2} > x_{i_3} > \ldots$ which is impossible because W is well ordered. Therefore P terminates for all inputs. □

The methodology in Corollary 3.20 and Proposition 3.21 is that we can "project out" control variables to prove termination.

Example 3.20. For the simple program P of Figure 3.2 we obtain the program P_1 shown schematically in Figure 3.6. P_1 satisfies the conditions of Proposition 3.21 and therefore terminates. Using $F = i_G : G \to G \circledast U$ and

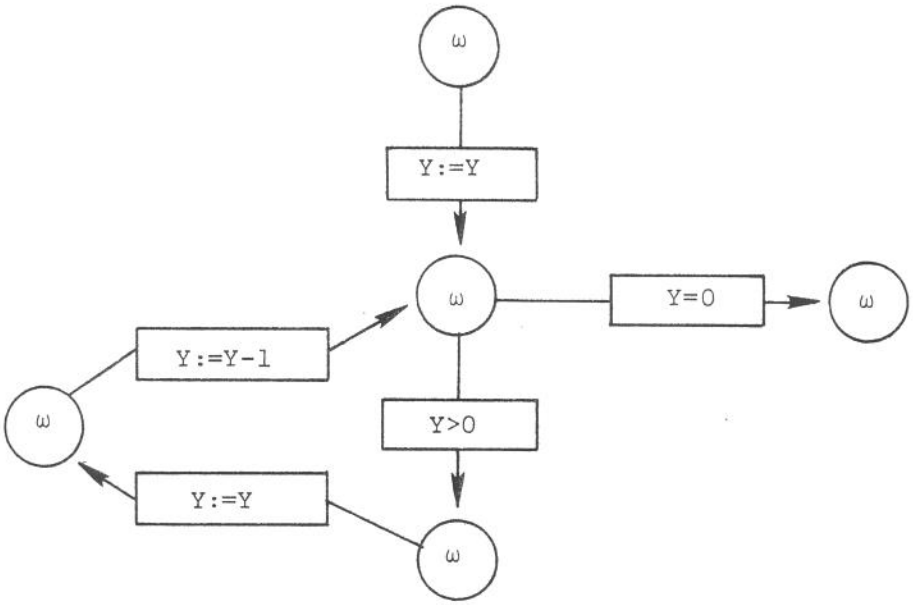


Figure 3.6

η, the projection of S on the Y-component, $\langle F,\eta\rangle : P \to P_1$ is a projection and thus P of Figure 3.2 terminates. □

There is a trivial completeness to this termination method: as Floyd (1967) points out, if P terminates, then for each node v there is a partial function $\eta_v : vP \to \omega$ defined on reachable elements of vP yielding the number of operations to be performed before termination. Then this family η with the identity graph morphism F is a projection to the program P_1 of the same shape as P which just counts down a single

variable. Of course P_1 terminates by Proposition 3.21 and then P terminates by Corollary 3.20.

Example 3.20 continued. Just to clarify this point, look again at the program P of Figure 3.2. The counting down program is just that in Figure 3.7

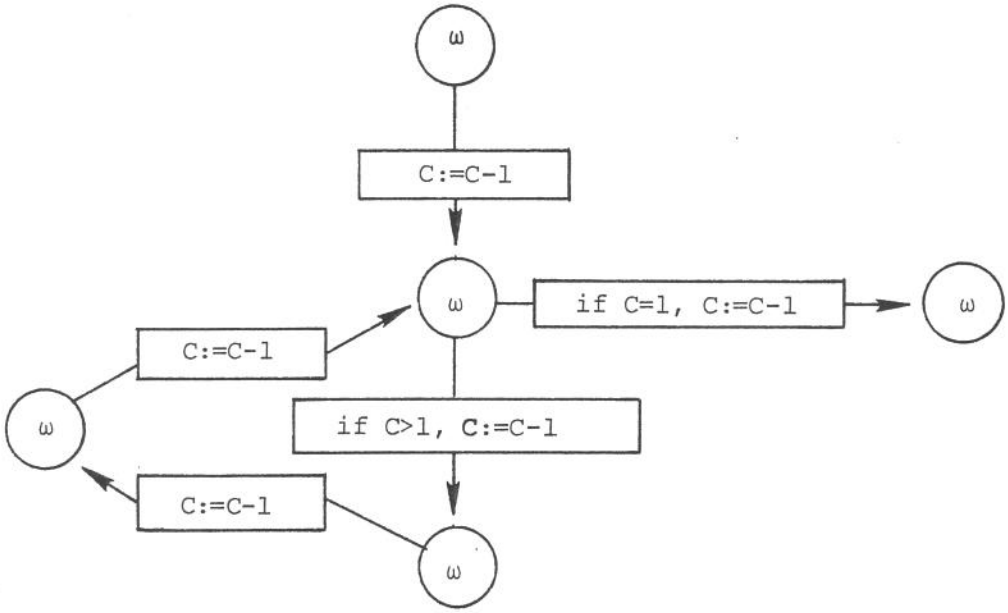


Figure 3.7

and the values for η_v can easily be calculated from the Y component of S for each node; e.g., $(x,y,z)\eta_b = 3y+1$ and $(x,y,z)\eta_d = 3y-1$, viewing S as ω^3. □

Just for fun, let's prove the termination of some other programs already introduced.

Example 3.21. We prove termination of the program P of Example 3.16. Let P_1 be the program of Figure 3.8,

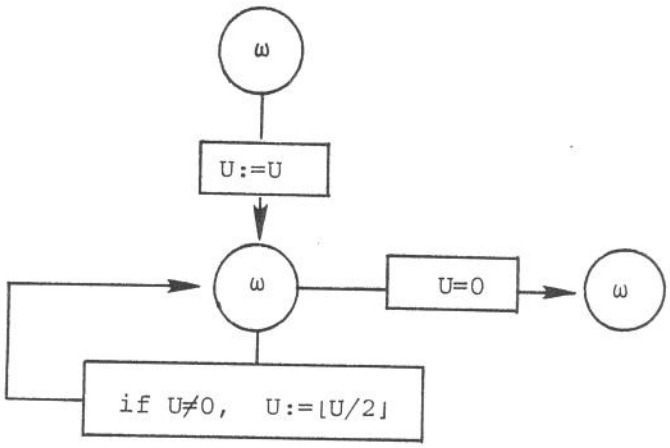


Figure 3.8

with single variable U, where $\lfloor U/2 \rfloor$ denotes the greatest integer $\leq U/2$. P_1 certainly terminates by Proposition 3.21. Let the entry, internal and exist nodes of (the shape of) P_1 be 1, 2 and 3 respectively, and let shape of P be as given in Example 3.18. Then the desired projection $\langle F,\eta\rangle : P \to P_1$ is such that $|F| : a \mapsto 1;\ b,c \mapsto 2;\ d \mapsto 3;$ $F : \langle a,b\rangle \mapsto \langle 1,2\rangle, \langle b,c\rangle \mapsto \langle 2,\lambda,2\rangle,$ odd, even $\mapsto \langle 2,2\rangle, \langle b,d\rangle \mapsto \langle 2,3\rangle$ and $\eta_a : \langle x,y\rangle \mapsto x,\ \eta_b : \langle u,v,z\rangle \mapsto u,\ \eta_c : \langle u,v,z\rangle \mapsto u$ and $\eta_d : z \mapsto 0.$ □

Example 3.22. (see Example 3.18.) The termination proof for Warshall's algorithm is far easier than the partial correctness. Let P_1 be the program of Figure 3.9

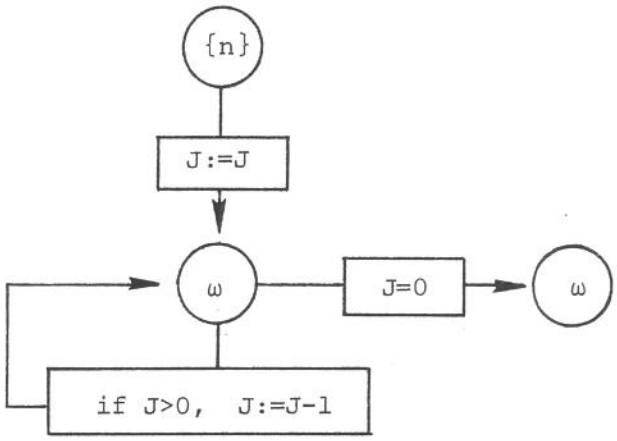


Figure 3.9

which terminates by Proposition 3.21 and again gives termination of P by Corollary 3.20 using projection $\langle F,\eta\rangle : P \to P_1$ where F is the identity and $\eta_a : \underline{a} \mapsto n$ (recall $\underline{a}$ is $n \times n$), $\eta_b : \langle j, \underline{a}\rangle \mapsto n-j$ and $\eta_c : \underline{a} \mapsto 0$. □

We are often interested in termination for only a subset of the input state set $v_0 P$. For example if, in Figure 3.2, we had taken all the integers instead of natural numbers as values of the variables then the program there would terminate only for states s such that $Ys \geq 0$. Corollary

3.20 can be modified as below to take care of this case; the proof is the same.

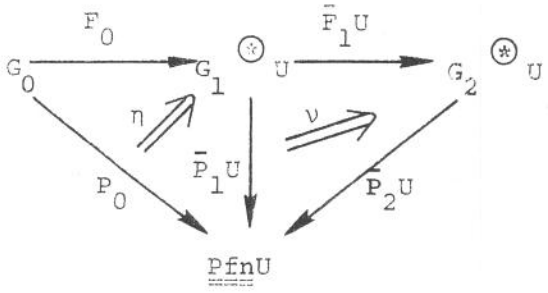
Corollary 3.22. If $\langle F,\eta\rangle:P \to P_1$ is a program homomorphism such that $\bar{F}_{v_0 v_1}$ is surjective and P_1 terminates, then P terminates on $def(\eta_{v_0})$. □

*3.4 CATEGORIES OF FLOW DIAGRAM PROGRAMS

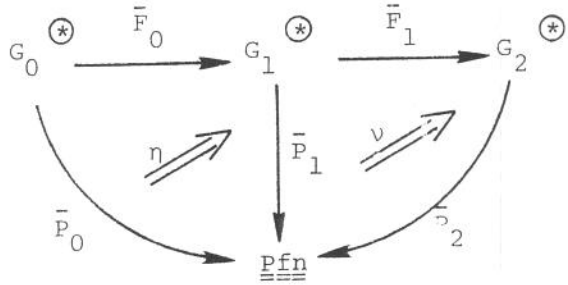
Consistent with Doctrine 1 (Section 0.2), there is a category $\underline{\underline{Prog}}(\underline{\underline{Pfn}})$, or $\underline{\underline{Prog}}$ for short, whose objects are $\underline{\underline{Pfn}}$-programs and whose morphisms are the program homomorphisms of Definition 3.10. We will see below that part of the construction of a functor from $\underline{\underline{Prog}}$ to a category of behaviors of programs is actually an alternative proof of the Verification Theorem (Proposition 3.16).

The identity for a program $\langle P:G \to \underline{\underline{Pfn}}U, v_0, v_1\rangle$ is $\langle i_G, l_P\rangle$ where $i_G:G \to G \circledast U$ is the injection of G into the underlying graph of its path category and $l_P:P \Rightarrow i_G(\bar{P}U) = P$ (Proposition 3.4) is the identity diagram morphism for P (c.f. Example 3.7).

Now for composition of program homomorphisms: given $\underline{\underline{Pfn}}$-programs $P_i:G_i \to \underline{\underline{Pfn}}U$ for $i = 0,1,2$, and program homomorphisms $\langle F_0,\eta\rangle:P_0 \to P_1$ and $\langle F_1,\nu\rangle:P_1 \to P_2$, their composite is $\langle F_0(\bar{F}_1 U),\xi\rangle$ where $\xi_v = \eta_v \nu_{vF_0}$ for each $v \in |G_0|$.



We have many things to check: that the composite is a program homomorphism (ξ is natural); that composition is associative; and, that we have appropriate identities. For this discussion it is convenient and illustrative to employ the equivalent program and homomorphism concepts based on functors and natural transformations (c.f. the discussion after Example 3.15) so as to be able to apply the results of Section 3.2. Thus we will consider programs $\bar{P}_i : G_i^{\circledast} \to \underline{\underline{\underline{Pfn}}}$, $i = 0,1,2$, and a program homomorphism $\langle \bar{F}_0, \eta \rangle$ from P_0 to P_1 is a functor $\bar{F}_0 : G_0^{\circledast} \to G_1^{\circledast}$ and a natural transformation $\eta : \bar{P}_0 \Rightarrow \bar{F}_0 \bar{P}_1$; given also $\langle \bar{F}_1, \nu \rangle : \bar{P}_1 \to \bar{P}_2$, the diagram



summarizes the situation. Notice particularly that the same family $\langle \eta_v : vP_0 \to vF_0P_1 \rangle_{v \in |G_0|}$ is a diagram morphism $P_0 \Rightarrow F_0(\bar{P}_1 U)$ and a natural transformation $\bar{P}_0 \Rightarrow \bar{F}_0\bar{P}_1$, by Proposition 3.6 and the discussion which follows it. In this functorial and natural transformation framework, the composite $\langle \bar{F}_0, \eta \rangle \langle \bar{F}_1, \nu \rangle$ is $\langle \bar{F}_0\bar{F}_1, \eta \circ (\bar{F}_0 * \nu) \rangle$.† We must check that this coincides with the definition given above. Indeed $\bar{F}_0\bar{F}_1$ is a functor from $G_0^{\circledast}$ to $G_2^{\circledast}$ and $i_{G_0}(\bar{F}_0\bar{F}_1)U = i_{G_0}(\bar{F}_0 U)(\bar{F}_1 U) = F_0(\bar{F}_1 U)$, so by Proposition 3.4, $\bar{F}_0\bar{F}_1 = \overline{F_0(\bar{F}_1 U)}$, which checks out the first component of the homomorphism. For the natural transformation part, first observe from Proposition 3.10 (with its diagram adapted to this special case),

$$\begin{array}{ccc} v\bar{F}_0\bar{P}_1 & \xrightarrow{\nu_{vF_0}} & v\bar{F}_0\bar{F}_1\bar{P}_2 \\ \downarrow{\scriptstyle 1_{vF_0}\bar{P}_1} & & \downarrow{\scriptstyle 1_{vF_0}\bar{F}_1\bar{P}_2} \\ v\bar{F}_0\bar{P}_1 & \xrightarrow{\nu_{vF_0}} & v\bar{F}_0\bar{F}_1\bar{P}_2 \end{array}$$

that $(\bar{F}_0 * \nu)_v = (1_{vF_0}\bar{P}_1)(\nu_{vF_0}) = 1_{vF_0P_1}\nu_{vF_0} = \nu_{vF_0}$, so that $(\nu \circ (\bar{F}_0 * \nu))_v = \eta_v \nu_{vF_0}$ as in the original definition.

† We are using $\circ$ for vertical composition (Definition 3.5 and Proposition 3.7, page 87) and $*$ (as before) for horizontal composition, page 94.

Now with these connections made, we can, as promised, use the results of Section 3.2 to obtain the needed facts about the category of Pfn-programs.

First we have that the composite of $\langle \bar{F}_0, \eta \rangle$ and $\langle \bar{F}_1, \nu \rangle$ is the right kind of object, i.e., $\bar{F}_0\bar{F}_1 : G_0^{\circledast} \to G_2^{\circledast}$ and $\eta \circ (\bar{F}_0 * \nu) : \bar{P}_0 \Rightarrow \bar{F}_0\bar{F}_1\bar{P}_2$ is natural by Proposition 3.7 (for $\circ$) and Proposition 3.10 (for $*$). Now for associativity, assume in addition we have $\langle \bar{F}_2, \phi \rangle$ from P_2 to P_3. Then $(\langle \bar{F}_0, \eta \rangle \langle \bar{F}_1, \nu \rangle) \langle \bar{F}_2, \phi \rangle = \langle \bar{F}_0\bar{F}_1, \eta \circ (\bar{F}_0 * \nu) \rangle \langle \bar{F}_2, \phi \rangle =$ $\langle (\bar{F}_0\bar{F}_1)\bar{F}_2, (\eta \circ (\bar{F}_0 * \nu)) \circ (\bar{F}_0\bar{F}_1 * \phi) \rangle$; and the other way, $\langle \bar{F}_0, \eta \rangle (\langle \bar{F}_1, \nu \rangle \langle \bar{F}_2, \phi \rangle) =$ $\langle \bar{F}_0, \eta \rangle \langle \bar{F}_1\bar{F}_2, \nu \circ (\bar{F}_1 * \phi) \rangle = \langle \bar{F}_0(\bar{F}_1\bar{F}_2), \eta \circ (\bar{F}_0 * (\nu \circ (\bar{F}_1 * \phi))) \rangle$. The first components of the two associations are equal by associativity of composition of functors (Proposition 2.2). For the second components we use Corollary 3.12 to the double law to get

$$\eta \circ (\bar{F}_0 * (\nu \circ (\bar{F}_1 * \phi))) = \eta \circ (\bar{F}_0 * \nu) \circ (\bar{F}_0 * (\bar{F}_1 * \phi)),$$

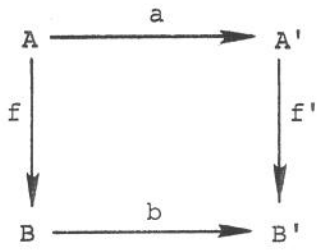
so that the two second components are equal since $\bar{F}_0 * \bar{F}_1 = \bar{F}_0\bar{F}_1$ as is easily verified.

Finally we leave it to the reader to verify that $\langle 1_{G^{\circledast}}, \bar{P} \rangle$ is the functorial version of the identity for the program $\bar{P} : G^{\circledast} \to$ Pfn described earlier in the diagram.

Summing up now we have proved

Proposition 3.23. Prog is a category. □

The computation of a $\underline{\underline{\underline{\text{Pfn}}}}$-program is a partial function. We are now ready to define the target category of the computation functor, $\mathcal{B}$. The objects of the categories $\square\underline{\underline{\underline{\text{Pfn}}}}$ and $\circledcirc\underline{\underline{\underline{\text{Pfn}}}}$ are partial functions and for both, morphisms from $\langle A,f,B\rangle$ to $\langle A',f',B'\rangle$ are pairs, $\langle a:A \to A',\ b:B \to B'\rangle$ in $\underline{\underline{\underline{\text{Pfn}}}}$ such that



$fb = af'$ for $\square\underline{\underline{\underline{\text{Pfn}}}}$ and $fb \subseteq af'$ for $\circledcirc\underline{\underline{\underline{\text{Pfn}}}}$. (For $\square\underline{\underline{\underline{\text{Pfn}}}}$ the square commutes and as Milner (1971) says, the square semi-commutes when $fb \subseteq af'$.)

Let $\underline{\underline{\underline{\text{Prog}}}}^{+}$ be the strict subcategory of $\underline{\underline{\underline{\text{Prog}}}}$ with morphisms $\langle F,\eta\rangle$ where $\bar{F}_{V_0V_1}$ is surjective, and let $\mathcal{B}^{+}$ be the restriction of $\mathcal{B}$ to $\underline{\underline{\underline{\text{Prog}}}}^{+}$.

Milner's (1971) motivation for studying program simulation (and one of our motivations for the more general program homomorphism) was that correctness techniques can be made more practical by constructing a secondary (simpler and more natural) program that simulates a given program, and then proving that the constructed program is correct. Because of this common motivation, it is interesting and illuminating (of both

correctness and categorical methodology) to explore the relationships between Milner's notions and those presented here. It would be nice if the fit were perfect but several choices in formulation rather encumber comparison, and so we point out some differences as well as similarities.

We begin by considering a strict subcategory of Prog, denoted GProg (the G stands for Graph morphism) determined by morphisms $\langle F,\eta\rangle:P_0 \to P_1$ where $F:G_0 \to G_1 \circledast U$ takes edges to edges only, i.e., the shape part of program homomorphism is obtained from a graph morphism $F':G_0 \to G_1$ composed with the injection $i_{G_1}:G_1 \to G_1 \circledast U$. We will see that Milner's simulations (homomorphisms) can not permit the interpretation of edges in one program as paths in another (a fact Milner was well aware of, c.f. Burstal (1972), p.13).

Now consider the full subcategory MProg (M for Milner) of GProg, determined by objects which are Milner programs. These programs are rather sparse in control structure, having the shape, denoted M, of Figure 3.10

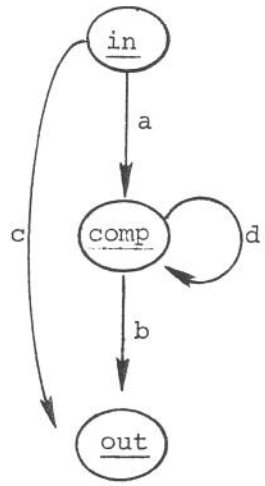


Figure 3.10

with only one loop. Already we have generalized the Milner idea because in a Pfn-program $D:M \to \underline{Pfn}U$, $aD \sqcup cD$ need not be total whereas Milner requires totality. Also, he requires the sets $D_{\underline{in}}$, $D_{\underline{comp}}$ and $D_{\underline{out}}$ (using his notation $D_{\underline{in}}$ for $(\underline{in})D$) to be disjoint; this is unnecessary in the current formulation and simply a convenience in his, since he views the program itself to be a (total) function $F:D_{\underline{in}} \cup D_{\underline{comp}} \cup D_{\underline{out}} \to D_{\underline{comp}} \cup D_{\underline{out}}$ subject to certain natural constraints. It should be clear that a program in MProg is obtained by suitable birestrictions of Milner's total function; e.g., $aD = D_{\underline{in}} \lceil F \rceil D_{\underline{comp}}$ and $dD = D_{\underline{comp}} \lceil F \rceil D_{\underline{comp}}$.

As we have just indicated, the objects in MProg are slightly more general than Milner's programs but we shall see that Milner's morphisms (simulations) are more general than the program homomorphisms (morphisms in MProg). It is easy to check that the shape part of a program morphism between D and D' in MProg must in fact be the identity (actually the injection $i_M:M \to M \circledast U$)[†] so that the morphism reduces to the three components of the operation part, $\eta_{\underline{in}}:D_{\underline{in}} \to D'_{\underline{in}}$, $\eta_{\underline{comp}}:D_{\underline{comp}} \to D'_{\underline{comp}}$ and $\eta_{\underline{out}}:D_{\underline{out}} \to D'_{\underline{out}}$, subject to the naturality conditions (depicted by the diagram in Figure 3.11), e.g., $\eta_{\underline{in}}(aD') = (aD)\eta_{\underline{comp}}$. Now

---

[†] The reader should note that this is not the case in Prog although it is in GProg.

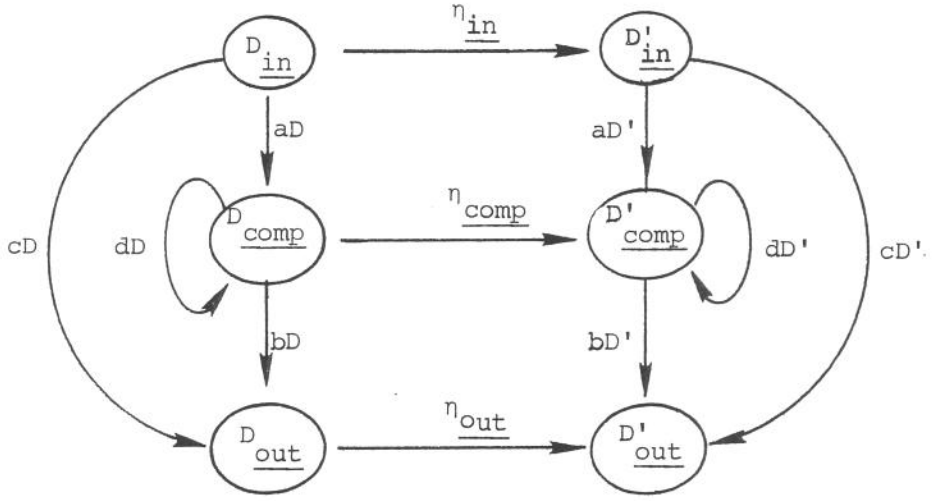


Figure 3.11

every such program homomorphism is a weak simulation (Milner's Definition, page 5), but curiously, he allows the η's to be relations even though his programs have the totality restriction described above. Further, our conventional naturality condition is replaced by a semicommuting naturality condition: for $e:v \to v'$ (in M) $\eta_v(eD') \sqsubseteq (eD)\eta_{v'}$. In MProg, the shape part is always surjective, so that Theorem 3.5 gives $(D\mathcal{B})\eta_{out} = \eta_{in}(D'\mathcal{B})$ analogous to Milner's Theorem 3.4. In fact, a check of the proof of Theorem 3.5 shows that the semicommuting naturality condition (remembering that the shape part is surjective) yields the conclusion $(D\mathcal{B})\eta_{out} \sqsubseteq \eta_{in}(D'\mathcal{B})$ which compares with Milner's Theorem 3.3.

It remains a question as to what is gained with Milner's relations and semicommuting naturality and whether this relates to the Rel-programs described at the end of this subsection.

Having examined the extent to which Milner programs fit into the category $\underline{\underline{\text{Prog}}}$, we now consider the reverse, that is, a functor $M:\underline{\underline{\text{GProg}}} \to \underline{\underline{\text{MProg}}}$, extracting a Milner program from every program P in $|\underline{\underline{\text{GProg}}}| = |\underline{\underline{\text{Prog}}}|$. For $P:G \to \underline{\underline{\text{Pfn}}}U$ define $PM:M \to \underline{\underline{\text{Pfn}}}U$ in the following way

$$\underline{\text{in}}(PM) = v_0P$$

$$\underline{\text{out}}(PM) = v_1P$$

$$\underline{\text{comp}}(PM) = \{\langle v,x\rangle \mid v \in |G|,\ v \neq v_0, v_1;\ \text{and}\ x \in vP\}$$

Note that the set associated with the node $\underline{\text{comp}}$ in PM is the disjoint union (or coproduct, see Section 5.2) of all the sets associated with the internal nodes of P; because of this, the partial functions associated with the edges of PM are uniquely determined by the corresponding functions of P. Let $i_v:vP \to \underline{\text{comp}}(PM)$ be the natural injection $(i_v:x \mapsto \langle v,x\rangle)$, and let π_v be the (partial function) "projection" of $\underline{\text{comp}}(PM)$ to vP, that is, $\pi_v:\langle v,x\rangle \mapsto x$. Then,

$$a(PM) = \bigsqcup \{(eP)i_v \mid e:v_0 \to v \neq v_1\}$$

$$b(PM) = \bigsqcup \{\pi_v(eP) \mid e:v \to v_1 \text{ and } v \neq v_0\}$$

$$c(PM) = \bigsqcup \{eP \mid e:v_0 \to v_1\}$$

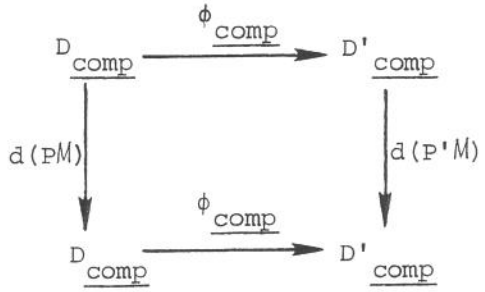
$$d(PM) = \bigsqcup \{\pi_v(eP)i_{v'} \mid e:v \to v',\ v \neq v_0 \text{ and } v' \neq v_1\}.$$

All these least upper bounds exist in $\underline{\underline{\text{Pfn}}}$ because of the determinateness condition on $\underline{\text{Pfn}}$-programs (Definition 3.7) and because all the π_v, for $v \in |G| - \{v_0,v_1\}$, have disjoint domains of definition.

This describes the object part of the functor M; the morphism part is also naturally determined (again by the coproduct character (see 5.2) of the "computation node"). In particular, if $P':G' \to \underline{\underline{Pfn}}U$ is another $\underline{\underline{Pfn}}$-program and $\langle F,\eta\rangle:P \to P'$ is a program homomorphism of the restricted kind ($F = F'i_{G'}$ where $F':G \to G'$ is a graph morphism), define $\langle F,\eta\rangle M$ to be $\langle i_M,\phi\rangle$ (recall the shape part has to be the injection) where $\phi_{\underline{in}} = \eta_0$, $\phi_{\underline{out}} = \eta_1$ and $\phi_{\underline{comp}}:(PM)_{\underline{comp}} \to (P'M)_{\underline{comp}}$ is defined by

$$(v,x)\ \phi_{\underline{comp}} = \langle vF,\ x\eta_v\rangle \in (P'M)_{\underline{comp}}.$$

Naturality of ϕ takes a little checking (four edges involved); it follows for edge c from the naturality of η; and we here just check it out for edge $d:\underline{comp} \to \underline{comp}$.



For each edge $e:v \to v'$ in P (v,v' not v_0 or v_1) and $x \in vP$, we have $\langle v,x\rangle \in D_{\underline{comp}}$ and from the definitions above

$$\langle v,x\rangle(d(PM))\phi_{\underline{comp}} = \langle v',\ x(eP)\rangle\phi_{\underline{comp}} = \langle v'F,\ x(eP)\eta_v\rangle .$$

On the other hand:

$$\langle v,x\rangle\phi_{\underline{comp}}(d(P'M)) = \langle vF,\ x\eta_v\rangle(d(P'M))$$

and $eF:vF \to v'F$ (by the restriction on the morphisms) so that $\langle vF,x\eta_v\rangle(d\ (P'M)) = \langle v'F,x\eta_v(eFP')\rangle$. Naturality of η gives $eP\eta_v = \eta_v eFP'$, so the square above commutes and ϕ is natural (assuming the other edges are checked likewise). Furthermore M is functorial and preserves identities; we leave this to the reader and now have:

Proposition 3.25. "Milnerization" M is an endofunctor on the category $\underline{\underline{GProg}}$ of programs with morphisms whose shape parts are in fact graph morphisms. □

Not only is there a Milnerization functor on a strict subcategory of $\underline{\underline{Prog}}$ (thus applying to all $\underline{\underline{Pfn}}$-programs), but the Milnerization (almost) simulates (is behaviorly equivalent to) the original program. Define the program homomorphism $\langle J,\alpha\rangle:P \to PM$ (in $\underline{\underline{GProg}}$) where J is defined as follows: $v_0 \mapsto \underline{in}$, $v_1 \mapsto \underline{out}$ and $v \mapsto \underline{comp}$ for $v \neq v_0,v_1$; and for edges $e:v \to v'$ in the shape of P:

$$\text{if } v = v_0 \text{ and } v' \neq v_1, \quad eJ = a',$$

$$\text{if } v \neq v_0 \text{ and } v' = v_1, \quad eJ = b',$$

$$\text{if } v \neq v_0 \text{ and } v' \neq v_1, \quad eJ = c',$$

$$\text{if } v = v_0 \text{ and } v' = v_1, \quad eJ = d'.$$

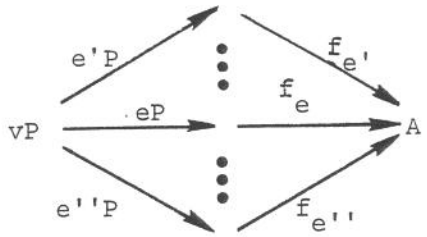
For the operation part of the homomorphism, α_{v_0} and α_{v_1} are the identity functions on v_0P and v_1P respectively, and for $v \neq v_0, v_1$ $\alpha_v = i_v : vP \to P\underline{M}_{\underline{comp}}$, i.e., α_v is the injection of vP into the disjoint union of all the sets associated with internal nodes of P. Naturality of α follows from the definitions of the operations in $P\underline{M}$.

From Theorem 3.5 we have $(P\underline{B})\alpha_{v_1} = P\underline{B} \subseteq (P\underline{M})\underline{B} = \alpha_{v_0}((P\underline{M})\underline{B})$ and we would have equality were J surjective. In fact we do have equality, $P\underline{B} = (P\underline{M})\underline{B}$, because any edge not in the image of J has the empty function associated with it (for example, with edge d, when there is no edge in P directly from v_0 to v_1) and thus contributes nothing to the computation of $P\underline{M}$. As can be seen in the proof of Theorem 3.5, the inequality arises just in case there are paths in the image along which computations can be performed.

We close this subsection by generalizing from $\underline{\underline{\underline{Pfn}}}$-programs to $\underline{\underline{C}}$-programs, where $\underline{\underline{C}}$ is a continuous category, meaning that (like $\underline{\underline{\underline{Pfn}}}$); each hom set $\underline{\underline{C}}(A,B)$ is a complete poset (having partial order relation $\subseteq$, least element $\perp$, and least upper bounds of bounded and of directed subsets); composition is left strict ($\perp\alpha = \perp$); and composition is continuous by components,[†] i.e., $(\bigsqcup_I \alpha_i)(\bigsqcup_J \beta_j) = \bigsqcup_{I\times J} \alpha_i\beta_j$ when the least upper bounds on the left exist. Now the generalization of Definition 3.7 is:

[†] It is important to note that this does not say that composition as a function from $\underline{\underline{C}}(A,B)\times\underline{\underline{C}}(B,C) \to \underline{\underline{C}}(A,C)$, is continuous. This stronger condition says that $(\bigsqcup_I \alpha_i)(\bigsqcup_I \beta_i) = \bigsqcup_I \alpha_i\beta_i$ and is false in $\underline{\underline{\underline{Pfn}}}$ and all other interesting categories we have considered.

Definition 3.12. Let $\underline{\underline{C}}$ be a continuous category and $P:G \to \underline{\underline{C}}U$, a diagram in $\underline{\underline{C}}$. P is a $\underline{\underline{C}}$-flow-diagram iff for every $v \in |G|$, $A \in |\underline{\underline{C}}|$ and family $\langle f_e : (e\partial_1)P \to A \rangle_{e\partial_0 = v}$ (which might be called "output morphisms"), the least upper bound, $\bigsqcup_{e\partial_0=v} (e\bar{P}) f_e$ exists in $\underline{\underline{C}}(vP, A)$. □



The reader can check that Definitions 3.7 and 3.12 are in fact equivalent for $\underline{\underline{C}} = \underline{\underline{Pfn}}$. One instance of this generalization has $\underline{\underline{C}} = \underline{\underline{Rel}}$; there every diagram satisfies the condition of 3.12, and so every diagram is a $\underline{\underline{Rel}}$-flow-diagram; these flow diagrams might be called non-deterministic in the sense of Floyd (1967a).

The general definition is designed so that the behavior $P\mathcal{B}$ of a $\underline{\underline{C}}$-program P is a well defined morphism of $\underline{\underline{C}}$. Analogous to Proposition 3.12 we have

Proposition 3.26. If $P:G \to \underline{\underline{C}}U$ is a $\underline{\underline{C}}$-flow-diagram, $v \in |G|$, S is any incomparable set of paths with common source v, and $\langle f_s : s\partial_1 P \to A \rangle_{s \in S}$ is any family of "output morphisms" with common target A, then $\bigsqcup_{s \in S} (s\bar{P}) f_s$ exists in $\underline{\underline{C}}(vP, A)$.

Proof. Assume that the Proposition is true for sets of paths of bounded length (we will prove this by induction in a moment). $S = \bigcup_{k\in\omega} S_k$ where S_k is the set of paths of length less than or equal to k: $S_k = \{s\in S \mid s\ell g \leq k\}$. Define $g_k = \bigsqcup_{s\in S_k} (s\bar{P})f_s$ which exists by our assumption. Since $S_k \subseteq S_{k+1}$, $g_k \sqsubseteq g_{k+1}$, so the g's form a chain in $\underline{\underline{C}}(vP,A)$ and their least upper bound $g = \bigsqcup_{k\in\omega} g_k = \bigsqcup_{k\in\omega}(\bigsqcup_{s\in S_k}(s\bar{P})f_s) = \bigsqcup_{s\in S}(s\bar{P})f_s$ exists in $\underline{\underline{C}}(vP,A)$.

Now for the bounded length path sets. If $S = \emptyset$, the least upper bound of the empty family exists and is $\perp \in \underline{\underline{C}}(vP,A)$. Assuming $S \neq \emptyset$, let $S\ell g = \max\{s\ell g \mid s\in S\}$ which is well defined because the paths in S are now assumed to be of bounded length. We prove the Proposition by induction on $S\ell g$. For $S\ell g = 0$, $S = \{\lambda : v \to v\}$ and for any $f:vP \to A$, $\bigsqcup(\lambda\bar{P})f = \bigsqcup 1_{vP} f = \bigsqcup f = f$ exists in $\underline{\underline{C}}(vP,A)$. Assuming our result is true for sets S with $S\ell g < k$, consider S such that $S\ell g = k \geq 1$. For each e with $e\partial_0 = v$, define $e\backslash S = \{s \mid es\in S\}$. Each $e\backslash S$ is incomparable since S was ($s \leq s'$ in $e\backslash S$ implies $es \leq es'$ in S), $(e\backslash S)\ell g < k$, and $\bigcup_{e\partial_0=v} e(e\backslash S) \subseteq S$, immediately from the definition. On the other hand $\lambda \notin S$ because S is incomparable and $S\ell g \geq 1$ so everything in S is of the form es and thus is in some $e(e\backslash S)$, i.e., $\bigcup_{e\partial_0=v} e(e\backslash S) = S$. Now for each e with $e\partial_0 = v$ define

$$g_e = (\bigsqcup_{s\in e\backslash S}(s\bar{P})f_{es}) : e\partial_1 P \to A$$

which exists by inductive hypothesis and

$$g = \bigsqcup_{e\partial_0=v} (e\bar{P})\, g_e$$

exists in $\underline{\underline{C}}(vP,A)$ by definition $\underline{\underline{C}}$-flow-diagram. And g is the desired morphism because $g = \bigsqcup_{e\partial_0=v}((e\bar{P})(\bigsqcup_{s\in e\backslash S}(s\bar{P})f_{es})) =$ [by continuity of composition] $\bigsqcup_{e\partial_0=v}(\bigsqcup_{s\in e\backslash S}(e\bar{P})(s\bar{P})f_{es}) =$ [by functorality of $\bar{P}$] $\bigsqcup_{e\partial_0=v}\bigsqcup_{s\in e\backslash S}(es\bar{P})f_{es} =$ [by $S = e(e\backslash S)$ from above] $\bigsqcup_{s\in S}(s\bar{P})f_s$. This completes the inductive step and thus the proof. □

This result allows us to define the computation of $\underline{\underline{C}}$-programs because, as with $\underline{\underline{\underline{Pfn}}}$-programs, paths from initial to terminal nodes are incomparable.

Definition 3.13. A $\underline{\underline{C}}$-program is a triple $\langle P, v_0, v_1\rangle$ such that $P: G \to \underline{\underline{C}}U$ is a $\underline{\underline{C}}$-flow-diagram, $v_0, v_1 \in |G|$ are called entry and exit respectively, and P is fully protected, i.e., $v_0\partial_1^{-1} = \emptyset = v_1\partial_0^{-1}$. The behavior of P is $PB = \bigsqcup_{p\in G} \circledast_{(v_0,v_1)} (p\bar{P}) \in \underline{\underline{C}}(v_0P, v_1P)$. □

The other aspects of our treatment of $\underline{\underline{\underline{Pfn}}}$-programs carry over to $\underline{\underline{C}}$-programs, including program homomorphisms, the category of $\underline{\underline{C}}$-programs, functorality of "behavior" and correctness and termination methodologies.

BIBLIOGRAPHY

This is a collection of work related to the junction between computer science and category theory. It therefore contains material from each partner, as well as the fruit of their union. We cannot hope to include all possible applicable algebra or algebracizable computer science, nor even all applicable categorical algebra and categoricizable computer science. We do hope for completeness in the area of the junction itself, but know we are unlikely to be able to achieve it, in part because of the rapid growth, and the intellectual and geographical diversity of the subject. Suggestions from readers for additions would be greatly appreciated.

In the belief that categorical algebra is only algebra in a more extreme or pure form, so that most algebra is categoricizable, we have included here material which is algebraic in character, style, or form, but not explicitly categorical. This includes both pure algebra and its applications to computer science. Moreover, we have also included applications to certain fields closely connected to computer science, particularly system science. Finally, we have included a fair amount of applied material which is not explicitly algebraic in character, but which seems to us to be particularly "algebraicizable" or which is necessary background for applications which are algebraic.

Works are listed alphabetically by author, and then by year. The order of publications with the same author(s) and year is arbitrary, but fixed by a small letter of the alphabet.

ADJ

(19XX) See Goguen, Thatcher, Wagner and Wright (19XX).

Aho, A. V.

(1973) (Ed.) Currents in Computing, Prentice Hall, New Jersey (1973).

Aho, A. V. and Ullman, J. D.

(1968) "Translating on a context-free grammar," Proceedings ACM Symposium on Theory of Computing (1968) 93-112; also, Information and Control 19 (1971) 439-475.

(1969) Formal Languages and their Relation to Automata, Addison-Wesley, Reading, Mass. (1969).

(1969a) "Syntax directed translations and the push-down assembler," J. Comp. and Sys. Sci. 3 (1969) 27-36.

(1969b) "Properties of syntax directed translations," J. Comp. and Sys. Sci. 3 (1969) 319-334.

(1972) The Theory of Parsing, Translation, and Compiling, Vol. I, Prentice-Hall, Englewood Cliffs, New Jersey (1972).

Alagic, S.

(1973) "Natural state transformations," Report 73B-2, Computer and Information Sciences Department, Univ. of Mass. (1973).

(1973a) "Algebraic aspects of ALGOL 68," Computer and Information Science Technical Report 73B-5, Univ. of Mass. (1973).

(1974) "Categorical theory of tree transformations," Category Theory Applied to Computation and Control, Univ. of Mass. (1974); Lecture Notes in Computer Science 25 (1975) 65-72.

Arbib, M. A.

(1966) "Categories of (M,R)-systems," Bull. of Math. Bio-physics 28 (1966).

(1967) See Kalman, Falb and Arbib.

(1968) See Give'on and Arbib.

(1968a) (Ed.) Algebraic Theory of Machines, Languages and Semigroups. Academic Press, New York (1968).

(1969) Theories of Abstract Automata, Prentice Hall, New Jersey (1969).

Arbib, M.A. and Give'on, Y.

(1968) "Algebra automata I: Parallel programming as a prolegoma to the categorical approach," Information Control 12 (1968) 331-345.

Arbib, M. A. and Manes, E.G.

(1972) "Decomposable machines and simple recursion," Computer and Information Sciences Technical Report 72B-2, Univ. of Mass. (1972).

Arbib, M. A. and Manes, E.G. (cont'd)

(1974) "Foundations of systems theory: decomposable machines," Automatica 10 (1974) 285-302.

(1974a) "Machines in a category: An expository introduction," SIAM Review 16 (1974) 163-192.

(1975) "Adjoint machines, state behavior machines, and duality," J. Pure and Appl. Alg., to appear, 1975.

(1975a) Arrows, Structures, and Functors, Academic Press (1975).

Arbib, M.A. and Zeiger, H.P.

(1969) "On the relevance of abstract algebra to control theory," Automatica 5 (1969) 589-606.

Arden, D.N.

(1961) "Delayed-logic and finite-state machines," Course Notes 6.531, E.E. Dept., MIT, Summer, 1961.

Baianu, I.

(1969) See Comorasan and Baianu.

Baianu, I. and Marinescu, M.

(1968) "Organismic supercategories: I. Proposals for a general unitary theory of systems," Bull. of Math. Bio-physics 30 (1968) 625-635.

Bainbridge, E.S.

(1972) "A unified minimal realization theory with duality," Ph.D. Dissertation, Dept. of Computer and Communication Sciences, Univ. of Michigan, November (1972).

(1974) "Addressed machines and duality," Category Theory Applied to Computation and Control, Univ. of Mass., 1974, Lecture Notes in Computer Science 25 (1975) 93-98.

Baker, B. S.

(1973) "Tree transductions and families of tree languages," Harvard Univ., Center for Research in Computing Technology Technical Report TR9-73 (1973).

Baker, J.L.

(1974) "Factorization of Scott-style automata," Category Theory Applied to Computation and Control, Univ. of Mass., 1974. Springer Lecture Notes in Computer Science 25 (1975) 99-105.

Bar-Hillel, Y., Perles, M. and Shamir, E.

(1961) "On formal properties of phrase structure grammars," Z. Phonetik, Sprach. Kummunikationsforsch 14 (1961) 143-172.

Beckman, F.S.

(1970) "Categorical notions and duality in automata theory," IBM T. J. Watson Res. Ctr. Report RC 2977, July (1970).

Bekic, H.

(1969) "Definable operations in general algebra, and the theory of automata and flowcharts," Report IBM Laboratory Vienna, (1969).

Benson, D.B.

(1970) "Syntax and semantics: A categorical view," Information and Control 17 (1970) 145-160.

(1974) "Semantic preserving translations," Math. Sys. Theory 8 (1974) 105-125.

(1974a) "An abstract machine theory for formal language parsers," Acta Informatica, 3 (1974) 187-202.

(1975) "The basic algebraic structures in categories of derivations," Information and Control 28 (1975) 1-29.

Birkhoff, G.

(1935) "On the structure of abstract algebras," Proceedings Cambridge Phil. Soc. 31 (1935) 433-454.

(1967) Lattice Theory, Amer. Math. Soc. Colloq. Pub. 25 New York (1948). (revised edition) (1967).

(1967a) See Mac Lane and Birkhoff.

Birkhoff, G. and Lipson, D.

(1970) "Hetrogeneous algebras," J. Combinatorial Theory 8 (1970) 115-133.

Bloom, S.L. and Elgot, C.C.

(1974) "The existence and construction of free iterative theories," IBM Research Report RC-4937 (1974). To appear JCSS.

Brainerd, W.S.

(1967) "Tree generating systems and tree automata," Ph.D. Dissertation, Purdue University, June (1967).

(1968) "The minimization of tree automata," Information and Control 13 (1968) 484-491.

(1969) "Tree generation regular systems," Information and Control 14 (1969) 217-231.

Brockett, R.W. and Willsky, A.S.

(1973) "Finite-state homomorphic sequential machines," IEEE Trans. Aut. Cont. AC-17 (1973) 483-490.

(1974) "Some structural properties of automata defined on groups," Category Theory Applied to Computation and Control, Univ. of Mass., 1974. Springer Lecture Notes in Computer Science 25 (1975) 112-118.

Büchi, J.R.

(1959) "Weak second-order arithmetic and finite automata," Univ. of Michigan, Logic of Computer Group Technical Report, September (1959); Z. Math. Logik Grundlagen Math. 6 (1960) 66-92.

Büchi, J.R.

(1962) "Mathematische theorie des verhaltens endlicher automaten," Z. Angewandte Math. und Mech. 42 (1962) 9-16.

(1964) "Regular canonical systems," Arch. Math. Logik Grundlagenforschung 6 (1964) 91-111.

Büchi, J.R. and Elgot, C.C.

(1958) "Decision problems of weak second-order arithmetics and finite automata," Abstract 553-112, Notices Amer. Math. Soc. 5 (1958) 834.

Büchi, J.R. and Wright, J.B.

(1960) "Mathematical theory of automata," Notes on material presented by J.R. Büchi and J.B. Wright, Communication Sciences 403 Fall (1960) The Univ. of Michigan.

Bucur, J. and Deleanu, A.

(1969) Theory of Categories, Academic Press, New York (1969).

Burks, A.W. and Wright, J.B.

(1962) "Sequence generators and digital computers," in Recursive Function Theory, (Ed. J.C.E. Dekker) Proceedings Symposia in Pure Math. V, Amer. Math. Soc., Providence, R. I. (1962) 139-199.

Burstall, R.M.

(1969) "Proving properties of programs by structural induction," Computer Journal, February (1969).

(1972) "An algebraic description of programs with assertions, verification and simulation," Proceedings ACM Conference on Proving Assertions about Programs, Las Cruces, New Mexico (1972) 7-14.

(1972a) "Some techniques for proving correctness of programs which alter data structures," Machine Intelligence 7 (Eds. B. Meltzer and D. Michie) Edinburgh University Press (1972) 23-50.

Burstall, R.M. and Landin, P.J.

(1969) "Programs and their proofs: An algebraic approach," Machine Intelligence 4 (Eds. B. Meltzer and D. Michie), Edinburgh University Press (1969) 17-43.

Burstall, R. M. and Thatcher, J. W.

(1974) "The algebraic theory of recursive program schemes," Proceedings AAAS Symposium on Category Theory Applied to Computation and Control, University of Massachusetts Press, Amherst (1974); Lecture Notes in Computer Science 25 (1975) 126-131.

Cadiou, J.M.

(1972) See Manna and Cadiou

Chandra, A. K.

(1974) "Degrees of translatability and canonical forms of program schemes," Proceedings, 6th Annual ACM Symposium on Theory of Computing (1974).

Chomsky, N.

(1961) "On the notion of 'rule of grammar'," Structures of Language and its Mathematical Aspects, Proceedings Symposia Appl. Math, XII, Amer.Math. Soc., Providence, R.I. (1961) 6-24.

(1965) Aspects of the Theory of Syntax, The M.I.T. Press, Cambridge, Massachusetts (1965).

Chomsky, N. and Schutzenberger, M.P.

(1963) "The algebraic theory of context-free languages," Computer Programming and Formal Systems (Ed. P. Braffort and D. Hirschberg), North-Holland Co., Amsterdam (1963) 118-161.

Claus, V.

(1971) See Hotz and Claus.

Cleveland, J. C.

(1974) See Martin and Cleveland

Cohn, P. M.

(1965) Universal Algebra, Harper and Row, New York (1965).

Cohen, P.M.

(1966) Set Theory and the Continuum Hypothesis, W. A. Benjamin, New York (1966).

Cole, J.C.

(1971) "Categories of sets and models of set theory," Aarhus University, Matematisk Institut, Preprint Series, 52 (1971).

Comorasan, S. and Baianu, I.

(1969) "Abstract representations of biological systems in supercategories," Bull. of Math. Bio-physics 31 (1969) 59-70.

Culik, K.

(1967) "On some transformations in context-free grammars and languages," Czechoslovak Math. J. 17 (1967) 278-311.

Davis, R.

(1969) "Universal coalgebra and categories of transition systems," Math. Sys. Th. 4 (1969) 91-95.

de Bakker, J.W.

(1969) See Scott and de Bakker.

(1971) "Recursive procedures," Mathematical Centre Tracts 24, Mathematical Centre, Amsterdam (1971).

(1971a) "Recursion induction and symbol manipulation," Proceedings MC-25 Informatica Symposium, Mathematical Centre Tracts 37, Mathematical Centre, Amsterdam (1971).

de Bakker, J.W. and de Roever, W.P.

(1973) "A calculus for recursive program schemes," to appear in Proceedings, IRIA Symposium on Automata, Formal Languages and Programming, North-Holland, Amsterdam (1973).

de Bakker, J. W. and Meertens, L.G.L. th.

(1972) "Simple recursive program schemes and inductive assertions," Mathematical Centre Tracts 142 Amsterdam (1972).

Deleanu, A.

(1969) See Bucur and Deleanu.

Desoer, C.A.

(1963) See Zadeh and Desoer.

de Roever, W.P.

(1973) See de Bakker and de Roever.

(1974) "Operational, mathematical and axiomatized semantics for recursive procedures and data structures," Mathematical Centre Report ID 1/74 (1974).

Dieudoune, J.

(1971) See Grothendieck and Dieudoune.

Doner, J.E.

(1965) "Decidability of the weak second-order theory of two successors," Abstract No. 6ST-468, Notices American Math. Soc. 12 (1965) 819.

(1970) "Tree acceptors and some of their applications," J. Comp. and Sys. Sci. 4 (1970) 406-451.

Ehresmann, Ch.

(1965) Categories et Structures, Dunod, Paris (1965).

Ehrig, H.

(1972) "Automata theory in monoidal categories," Proceedings, Tagung über Kategorien, Mathematisches Forschungsinstitut Oberwolfach (1972) 12-15.

(1973) "Axiomatic theory of systems and systematics," Report 73-05, Technische Universitat Berlin, Kybernetic Fachbereich, March (1973).

Ehrig, H. and Pfender, M.

(1972) Kategorien und Automaten, de Gruytes, Berlin (1972).

Ehrig, H. and Kreowski, H.-J.

(1974) "Power and initial automata in pseudoclosed categories," Category Theory Applied to Computation and Control, Univ. of Mass.(1974); Springer Lecture Notes in Computer Science 25 (1975) 144-150).

Ehrig, H., Kreowski, H.-J. and Pfender, M.

(1975) "Kategorielle Theorie der Reduktion, Minimerung und Aquivalenz von Automaten," Math. Nachr., to appear (1975).

Ehrig, H., Pfender, M. and Schneider, H.J.

(1973) "Graph grammars: An algebraic approach," Proceedings, 14th Annual Symposium on Switching and Automata Theory (1973) 167-180.

(1973a) "Applications of category theory to higher dimensional formal languages," Proceedings, Tagung über Kategorien, Mathematisches Forschungsinstitut Oberwolfach (1973) 10-12.

Eilenberg, S.

(1969) "The algebraicization of mathematics," The Mathematical Sciences Essays for COSRIMS, MIT Press, Cambridge, Mass. (1969) 153-160.

(1974) Automata, Languages and Machines, Vol. A, Academic Press (1974).

Eilenberg, S. and Elgot, C.C.

(1969) "Iteration and recursion," Proceedings Nat. Acad. of Sci. 61 (1968) 378-379.

(1970) Recursiveness, Academic Press, New York

Eilenberg, S. and Kelly, G.M.

(1966) "Closed categories," Proceedings, Conference on Categorical Algebra, Springer-Verlag (1966) 421-562.

Eilenberg, S. and Mac Lane, S.

(1942) "Group extensions and homology," Ann. Math. 43 (1942) 757-831.

(1945) "General theory of natural equivalence," Trans. Amer. Math. Soc. 58 (1945) 231-294.

Eilenberg, S. and Schutzenberger, M.

(1969) "Rational sets in commutative monoids," Journal of Algebra 13 (1969) 173-191.

Eilenberg, S. and Wright, J.

(1967) "Automata in general algebras," Information and Control 11 (1967) 52-70.

Elgot, C.C.

(1958) See Büchi and Elgot.

(1961) "Decision problems of finite automaton design and related arithmetics," Trans. Amer. Math. Soc. 98 (1961) 21-51.

(1968) See Eilenberg and Elgot.

(1970) See Eilenberg and Elgot.

(1970a) "The common algebraic structure of exit-automata and machines," Computing 6 (1970) 349-370.

(1971) "Algebraic theories and program schemes," Symp. on Semantics of Algorithmic Languages, (Ed. E. Engeler), Springer-Verlag (1971) 71-88.

(1972) "Remarks on one-argument program schemes," Formal Semantics of Programming Languages, (Ed. R. Rustin), Prentice-Hall, New Jersey (1972) 59-64.

Elgot, C.C. (cont'd)

(1973) "Monadic computation and iterative algebraic theories," IBM Research Report RC-4564, October 1973. Proceedings, Logic Colloquium '73, North Holland Publishing Company (1975) 175-230.

(1974) See Bloom and Elgot (1974).

(1975) "Structured programming with and without GO TO statements," IBM Research Report RC 5626 (1975).

Engeler, E.

(1968) Formal Languages: Automata and Structures, Markham, Chicago (1968).

(1971) (Ed.) "Symposium on semantics of algorithmic languages," Lecture Notes in Mathematics 188 (1971).

Engelfriet, J.

(1971) "Bottomup and topdown tree transformations," Memorandum 19, Technische Hogeschool Turente, July (1971).

Engelfriet, J. and Schmidt, E.M.

(1975) "IO and OI," Datalogisk Afdeling Report DAIMI PB-417, Aarhus University, Denmark, July, 1975.

Falb, D.L.

(1967) See Kalman, Falb and Arbib.

Feferman, S.

(1969) "Set-theoretical foundations of category theory," Reports of the Midwest Category Seminar III (ed. by S. Mac Lane), Lecture Notes in Mathematics, 106 Springer-Verlag (1969) 201-247.

Fischer, J.

(1968) "Grammars with macro-like productions," Proceedings, 9th Ann. Symp. on Switching and Automata Theory, (1968).

Floyd, R.W.

(1967) "Assigning meanings to programs," Proceedings, Symp. on Appl. Math, 19, Amer. Math. Soc., Providence, Rhode Island (1967) 19-32.

(1967a) "Nondeterministic algorithms," JACM 14 (1967) 636-644.

Freyd, P.

(1964) Abelian Categories, Harper and Row (1964).

(1972) "Aspects of topoi," Bull. Austral. Math. Soc. 7 (1972) 1-76.

Gabriel, P. and Zisman, M.

(1967) Calculus of Fractions and Homotopy Theory, Springer-Verlag (1967).

Gallaire, H.

(1969) "Decomposition of linear sequential machines II," Math. Sys. Th. 4 (1969) 168-190.

Gallaire, H. and Harrison, M.A.

(1969) "Decomposition of linear sequential machines," Math. Sys. Th. 3 (1969) 246-287.

Ginali, S.M.

(1975) "A concrete introduction to algebraic automata theory," Quarterly Report #44, Institute for Computer Research, University of Chicago, February 7, 1975.

Ginsburg, S.

(1966) The Mathematical Theory of Context-Free Languages, McGraw-Hill, New York (1966).

Ginsburg, S. and Rice, H.G.

(1962) "Two families of languages related to ALGOL," J.ACM 9 (1962) 350-371.

Ginzberg, A.

(1968) Algebraic Theory of Automata, Academic Press, New York.

Give'on, Y.

(1966) "On some categorical algebra aspects of automata theory: The categorical properties of transition systems," Ph.D. Thesis, Communication Sciences Program, University of Michigan. February (1966).

(1967) "Categories of semimodules: The categorical structural properties of transition systems," Math. Sys. Th. 1 (1967) 67-78.

(1967a) "On some properties of the free monoids with applications to automata theory," J. Comp. Sys. Sci. 1 (1967) 137-154.

(1968) See Arbib and Give'on.

(1970) "A categorical review of algebra automata and system theories," Symposia Mathematica 4 Instituto Nazionale di Alta Matematica, Academic Press, New York (1970).

Give'on, Y. and Arbib, M.A.

(1968) "Algebra automata II: The categorical framework for dynamic analysis," Information and Control 12 (1968) 346-370.

Give'on, Y. and Zalcstein, Y.

(1970) "Algebraic structures in linear systems theory," J. of Comp. and Sys. Sci. 4 (1970) 539-556.

Goguen, J.A.

(1967) "L-fuzzy sets," J. Math. Anal. and Appl. 18 (1967) 145-174.

(1968) "Categories of fuzzy sets," Ph.D. Thesis, Dept. of Math., Univ. of Cal. at Berkeley, May (1968).

Goguen, J.A. (cont'd)

(1969) "Categories of L-sets," Bull. Amer. Math. Soc. 75 (1969) 622-624.

(1969a) "The logic of inexact concepts," Synthese 19 (1969) 325-373.

(1970) "Mathematical representation of hierarchically organized systems," Global Systems Dynamics, (Ed. E.D. Attinger), S. Karger (1970) 111-129.

(1971) "Systems and minimal realization," Proceedings, 1971 IEEE Conference on Decision and Control, Miami Beach (1971) 42-46.

(1971a) "Discrete-time machines in closed monoidal categories, I," Institute for Computer Research, Quarterly Report No. 30, August (1971) The Univ. of Chicago; also J. of Comp. and Sys. Sci. 10 (1975) 1-43.

(1972) "Minimal realization of machines in closed categories," Bull.Amer. Math. Soc. (1972) 777-783.

(1972a) "On homomorphisms, simulations, correctness, subroutines, and termination for programs and program schemes," Proceedings, 13th IEEE Symp. on Switching and Automata Theory (1972) 52-60. See also Goguen (1974) for a revised version of this paper.

(1972b) "Hierarchical inexact data structures in artificial intelligence problems," Proceedings, Fifth Hawaii Int'l Conference on System Sciences (1972) 343-347.

(1972c) "On mathematics in computer science education," IBM T. J. Watson Res. Ctr. Technical Report RC 3899 (1972).

(1972d) See Goguen and Yacobellis.

(1973) "Realization is universal," Math. Sys. Th. 6 (1973) 359-374.

(1973a) "System theory concepts in computer science," Proceedings, Sixth Hawaii Int'l Conference on System Sciences (1973) 77-80.

(1974) "On homomorphisms, correctness, termination, unfoldments, and equivalence of flow diagram programs," J. of Comp. and Sys. Sci. 8 (1974) 333-365.

(1974a) "Set theoretic correctness proofs," UCLA Computer Science Technical Report No. 1, September (1974).

(1974b) "Semantics of computation, "Category Theory Applied to Computation and Control, (M. Arbib and E. Manes, Eds.) U. Mass. Press, Amherst (1974) 234-239. Also, Lecture Notes in Computer Science 25 (1975) 151-163.

Goguen, J.A., Thatcher, J.W., Wagner, E.G. and Wright, J.B.

(1973) "A junction between computer science and category theory: I, Basic Definitions and Examples, Part 1," IBM Research Report RC 4526 (September, 1973).

(1974) "Factorizations, congruences and the decomposition of automata and systems," IBM Research Report RC 4934, July 1974. Proceedings, Second Annual Symposium on Mathematical Foundations of Computer Science, Warsaw, Poland, Lecture Notes in Computer Science 28 (1975) 33-45.

Goguen, J.A., Thatcher, J.W., Wagner, E.G. and Wright, J.B. (cont'd)

(1974a) "Initial algebra semantics," Extended Abstract, IBM Research Report RC 4865, May, 1974. Proceedings, 15th IEEE Symposium on Switching and Automata Theory (1974) 63-77. Full paper, IBM Research Report, RC 5243, January, 1975. (See 1975 listing)

(1975) "Initial algebra semantics and continuous algebras." Expanded and revised version of (1974a), IBM Research Report RC 5701, November 3, 1975. To appear, JACM.

(1975a) "Parallel realization of systems, using factorizations and quotients in categories." IBM Research Report RC 5668, October 1975; also to appear, J. Franklin Institute.

(1975b) "Introduction to categories, algebraic theories and algebras." IBM Research Report RC 5369, April, 1975.

(1975c) "Abstract data types as initial algebras and correctness of data representations," Proceedings, Conference on Computer Graphics, Pattern Recognition, and Data Structures, May, 1975, 89-93.

(1975d) "A junction between computer science and category theory: I, Basic definitions and examples, Parts 2 and 3; II, Universal Constructions; III, IV, Algebraic theories and structures; V, Initial factorizations and dummy variables." IBM Research Reports to appear.

Goguen, J..A. and Yacobellis, R.

(1972d) "The Myhill functor, input-reduced machines, and generalized Krohn-Rhodes theory," Proceedings, Fifth Princeton Conference on Information Sciences and Systems (1972) 574-478.

Griffiths, T.V.

(1968) "Some remarks on derivations in general rewriting systems," Information and Control 12 (1968) 27-54.

Gratzer, G.

(1967) Universal Algebra. Van Nostrand (1967).

Gray, J.

(1965) "Sheaves with values in a category," Topology 3 (1965) 1-18.

Grothendieck, A. and Dieudoune, J.

(1971) Elements de Geometrie Algebrique, Springer-Verlag (1971).

Halmos, P.R.

(1958) Finite Dimensional Vector Spaces, Van Nostrand (1958).

(1960) Naive Set Theory, Van Nostrand, Princeton, New Jersey (1960).

Harrison, M.A.

(1965) Introduction to Switching and Automata Theory, McGraw-Hill (1965).

(1969) See Gallaire and Harrison.

Hartmanis, J. and Stearns, R.E.

(1966) Algebraic Structure Theory of Sequential Machines, Prentice-Hall, New Jersey (1966).

Hautus, M.L.J.

(1971) See Kalman and Hautus.

Heller, A.

(1967) "Probabilistic automata and stochastic transformations," Math. Sys. Th. 1 (1967) 197-208.

Herrlich, H. and Strecker, C.E.

(1973) Category Theory, Allyn and Bacon (1973)

Herman, H.T. and Kotelly, J.C.

(1967) "An approach to formal psychiatry," Perspectives in Biology and Medicine 10 (1967).

Higgins, P.J.

(1963) "Algebras with a scheme of operators," Math. Nachr. 27 (1963) 115-132.

Hitchcock, P. and Park, D.M.R.

(1972) "Induction rules and proofs of termination," in Automata Languages and Programming (M. Nivat, Ed.) North-Holland, Amsterdam (1972) 225-251.

Hoare, C.A.R.

(1969) "An axiomatic basis for computer programming," C.ACM, October (1969).

(1971) "Procedures and parameters: an axiomatic approach," Proceedings, Symposium on Semantics of Algorithmic Languages, Lecture Notes in Mathematics 188, Springer-Verlag, New York (1971) 102-116.

Hopcroft, J.E. and Ullman, J.D.

(1969) Formal Languages and Their Relation to Automata, Addison-Wesley, Reading, Mass. (1969).

Hotz, G.

(1974) "Strukturelle Verwandtschaften von semi-Thue-Systemen," Category Theory Applied to Computation and Control, Univ. of Mass.(1974). Springer Lecture Notes in Computer Science 25 (1975) 174-179.

Hotz, G. and Claus, V.

(1971) Automatentheorie und Formale Sprachen III, Bibliographisches Institut Mannheim (1971).

Hotz, G. and Walter, H.

(1969) "Automatentheorie und Formale Sprachen II," Mannheim (1969)

Ianov, I.I.

(1960) "The logical schemes of automata," English translation in Problems of Cybernetics 1, Pergamon Press (1960) 82-140.

Irons, E.T.

(1961) "A syntax directed compiler for ALGOL 60," C.ACM 4 (1961) 51-55.

(1963) "Towards more versatile mechanical translators," Proceedings, Symposium on Applied Mathematics 15, American Math. Society, Providence, R. I. (1963) 41-50.

Joshi, A.K.

(1973) See Levy and Joshi

Kalman, R.E.

(1969) "Introduction to the algebraic theory of linear dynamical systems," Lecture Notes in Operations Research and Math. Economics 11, Springer-Verlag (1969).

(1972) See Roucheleau, Wyman and Kalman.

Kalman, R.E., Falb, D.L. and Arbib, M.A.

(1967) Topics in Modern System Theory, McGraw-Hill, New York (1967).

Kalman, R.E. and Hautus, M.L.J.

(1971) "Realization of continuous time linear dynamical systems: Rigorous theory in the style of Schwartz," Proceedings, Conf. on Differential Equations, NRL Math. Research Center (1971) 14-23.

Kan, D.

(1958) "Adjoint functors," Trans. Amer. Math. Soc. 87 (1958) 294-329.

Karp, R.M.

(1959) "Some applications of logical syntax to digital computer programming, Harvard University Thesis (1959).

Karp, R.M. and Miller, R.E.

(1969) "Parallel program schemata," J. Comp. and Sys. Sci. 3 (1969) 147-195.

Kelly, G.M.

(1966) See Eilenberg and Kelly.

Kelly, G.M. and Mac Lane, S.

(1971) "Coherence in closed categories," J. Pure and Applied Algebra 1 (1971) 97-140.

Kelly, G.M. and Street, Ross

(1974) "Review of the elements of 2-categories," Category Seminar Sydney 1972/73, Lecture Notes in Mathematics 420, Springer-Verlag, (1974) 75-103.

Kildall, G.

(1973) "A unified approach to global program optimization," Proceedings, ACM Symposium on Principles of Programming Languages (1973) 194-206.

Knuth, D.E.

(1968) The Art of Computer Programming, Vol. I, Fundamental Algorithms, Addison-Wesley, Reading, Mass. (1968).

(1968a) "Semantics of context-free languages," Math. Sys. Th. 2 (1968) 127-145.

(1971) "Examples of formal semantics," Springer Lecture Notes in Mathematics 188 (Ed. E. Engeler) (1971) 212-235.

Kock, A.

(1970) "Monads on symmetric monoidal closed categories," Arch. Math.XXI (1970) 1-10.

Kotelly, J.C.

(1967) See Herman and Kotelly.

Kreowski, H.-J.

(1974) See Ehrig and Kreowski.

(1975) See Ehrig, Kreowski and Pfender.

Krohn, K. and Rhodes, J.

(1962) "Algebraic theory of automata," Symp. on the Math. Theory of Automata, Polytechnic Institute of Brooklyn (1962).

(1965) "Algebraic theory of machines, I: Prime decomposition for finite semigroups and machines," Trans. Amer. Math. Soc. 116 (1965) 450-464.

Lallement, G.

(1969) "On the prime decomposition theorem for finite monoids," Math. Sys. Th. 5 (1969) 8-12.

Lambek, J.

(1968) "Deductive systems and categories, I," Math. Sys. Th. 2 (1968).

Landin, P.J.

(1969) See Burstall and Landin.

(1970) "A program machine symmetric automata theory," Machine Intelligence 5 (Eds. B. Meltzer and D. Michie), Edinburgh Univ. Press (1970).

Lawvere, F.W.

(1963) "Functional semantics of algebraic theories," Proceedings, Nat'l Acad. Sci. 50 (1963) 869-872.

(1964) "An elementary theory of the category of sets," Proceedings, Nat'l Acad. Sci. 52 (1964) 1506-1510.

(1969) "Diagonal arguments and Cartesian closed categories," Category Theory, Homology Theory, and Applications II, Lecture Notes in Math. 92, Springer-Verlag, New York (1969).

Lawvere, F.W. (cont'd)

(1969a) "Adjointness in foundations," Dialectica 23, (1969).

(1970) Equality in hyperdoctrines and comprehension schemas as an adjoint functor," in Applications of Categorical Algebra, Proceedings, Symp. Pure Math. XVII, American Mathematical Society (1970).

(1971) "Quantifiers and sheaves," Actes du Congres Int'l des Mathematiciens, T. 1, Nice 1970, Gauthier-Villars, Paris (1971) 329-334.

Levy, L.S. and Joshi, A.K.

(1973) "Some results in tree automata," Math. Sys. Th. 6 (1973) 334-342.

Lewis, C. H. and Rosen, B.K.

(1973) "Recursively defined data types: Part 1," Proceedings, ACM Symposium on Principles of Programming Languages (1973) 125-138; also IBM Research Report RC 4429.

(1974) "Recursively defined data types: Part 2," IBM Research Report RC 4713, February, 1974.

Lewis, P.M. and Stearns, R.E.

(1968) "Syntax directed transduction," J.ACM 15 (1968) 465-488.

Linton, F.E.J.

(1966) "Some aspects of equational categories," Proceedings, Conf. on Categorical Algebra, LaJolla, California 1965, Springer, New York (1966) 84-95.

(1969) "Relative functorial semantics: Adjointness results," Lecture Notes in Mathematics 99, Springer-Verlag (1969) 384-418.

Lipson, D.

(1970) See Birkhoff and Lipson.

Luckham, D.C., Park, D.M.R. and Paterson, M.S.

(1970) "On formalized computer programs," J. Comp. and Sys. Sci. (1970).

Mac Lane, S.

(1942) See Eilenberg and Mac Lane.

(1945) See Eilenberg and Mac Lane.

(1971) See Kelly and Mac Lane.

(1971a) Category Theory for the Working Mathematician, Springer-Verlag (1971).

(1971b) "One universe as a foundation for category theory," Reports of the Midwest Category Seminar III (Ed. by S. Mac Lane), Lecture Notes in Mathematics, 106 Springer-Verlag (1969) 192-200.

Mac Lane, S. and Birkhoff, G.

(1967) Algebra, MacMillan, New York (1967).

Magidor, M. and Moran, G.

(1969) "Finite automata over finite trees," Technical Report 30, Hebrew University, Jerusalem, Israel (1969).

Maibaum, T.S.E.

(1972) "The characterization of the derivation trees of context-free sets of terms as regular sets," Proceedings, 13th Annual Symp. on Switching and Automata Theory, College Park, Maryland (1972) 224-230.

Manes, E.G.

(1972) See Arbib and Manes.

(1974) See Arbib and Manes.

(1974a) See Arbib and Manes.

(1975) See Arbib and Manes.

(1975a) See Arbib and Manes.

(1975b) (Ed.) "Category theory applied to computation and control," Lecture Notes in Computer Science 25, Springer-Verlag, New York (1975).

(1976) Algebraic Theories, Springer-Verlag, New York, to appear.

Manna, Z.

(1969) "Correctness of programs," JCSS 3 (1969) 119-127.

Manna, Z. and Cadiou, J.M.

(1972) "Recursive definitions of partial functions and their computations," in Proceedings, ACM Conf. on Proving Assertions about Programs, (1972) 58-65.

Manna, Z., Ness, S. and Vuillemin, J.

(1972) "Inductive methods for proving properties of programs," in Proceedings, ACM Conference on Proving Assertions about Programs, (1972) 27-50.

Manna, Z. and Pnueli, A.

(1970) "Formalization of properties of functional programs," J.ACM 17 (1970) 555-569.

Manna, Z. and Vuillemin, J.

(1972) "Fixpoint approach to the theory of computation," C.ACM 15 (1972) 528-536.

Marinescu, M.

(1969) See Baiann and Marinescu.

Markowsky, G.

(1974) "Chain-complete posets and directed sets, with applications," IBM Research Report RC 5024, August (1974).

Martin, D.F. and Cleveland, J.C.

(1974) Class Notes, UCLA Computer Science Department (1974).

McCarthy, J.

(1963) "A basis for a mathematical theory of computation," Computer Programming and Formal Languages (Eds. P. Braffort and D. Hirschberg) North-Holland, Amsterdam (1963) 33-69.

(1963a) "Towards a mathematical science of computation," Proceedings, 1962 IFIP Congress, North-Holland, Amsterdam (1963).

McCulloch, W.S. and Pitts, W.H.

(1943) "A logical calculus of the ideas imminent in nervous activity," Bull. Math. Biophys. 5 (1943) 115-33.

McNaughton, R.

(1973) "Algebraic decision procedures for local testability," to appear in Math. Sys. Th.

Medvedev, Yu T.

(1956) "On the class of events representable in a finite automaton," Avtomaty, Moscow, Ixdatel'stvo Inostrannoi Literary (1956) 385-401. Translated in Sequential Machines: Selected Papers (Ed. E.F. Moore) Addison-Wesley, Reading, Mass. (1964).

Meertens, L.G.L. th.

(1972) See de Bakker and Meertens.

Mezei, J. and Wright, J.B.

(1967) "Algebraic automata and context-free sets," Information and Control 11 (1967) 3-29.

Miller, R.E.

(1969) See Karp and Miller.

Miller, R.E. and Thatcher, J.W.

(1972) (Eds.) Complexity of Computer Computations, Plenum Press (1972).

Milner, R.

(1971) "An algebraic definition of simulation between programs," Stanford A.I. Memo AIM-142 and Computer Science Department Report CS 205, (1971).

(1972) "Implementation and applications of Scott's logic for computable functions," in Proceedings, ACM Conference on Proving Assertions about Programs (1972) 1-6.

Minsky, M.

(1968) Semantic Information Processing, MIT Press, Cambridge, Mass. (1968).

Mitchell, B.

(1965) Theory of Categories, Academic Press, New York (1965).

Moore, E.G.

(1956) "Gedanken experiments on sequential machines," Automata Studies, Princeton Univ. Press (1956) 129-153.

Moran, G.

(1969) See Magidor and Moran

Morris, F.L.

(1972) "Correctness of translations of programming languages -- an algebraic approach," Stanford Artificial Intelligence Project Memo AIM-174 (1972).

(1973) "Advice on structuring compilers and proving them correct," in Proceedings, SIGACT/SIGPLAN Symposium on Principles of Programming Language, Boston (1973).

Morris, J.H. Jr.

(1971) "Another recursion induction principle," C.ACM 14 (1971) 351-354.

Myhill, J.

(1957) "Finite automata and the representation of events," WADC Tech. Report (1957) Wright-Patterson AFB, Ohio 57-624.

Naur, P.

(1966) "Proof of algorithms by general snapshots," Bit 6 (1966) 310-316.

Nerode, A.

(1958) "Linear automaton transformations," Proceedings, Amer. Math. Soc. 9 (1958) 541-544.

(1959) "Some stone spaces and recursion theory," Duke Math. J. 26 (1959) 397-406.

Ness, S.

(1972) See Manna, Ness and Vuillemin.

Nivat, M.

(1968) "Transductions des langages de Chomsky," Annales Institut Fourier (Grenoble) 18 (1968) 339-456.

(1972) "Languages algebraic sur le magna libre et semantique des schemas de programme," in Automata, Languages and Programming (M. Nivat, ed.) North Holland (1972) 293-308.

Pareigis, B.

(1970) Categories and Functors, Academic Press, New York (1970).

Park, D.M.R.

(1969) "Fixpoint induction and proofs of program properties," Machine Intelligence 5 (Eds. B. Meltzer and D. Michie) Edinburgh Univ. Press (1969) 59-78.

(1970) See Luckham, Park and Paterson.

(1972) See Hitchcock and Park.

Paterson, M.S.

(1968) "Program schemata," Machine Intelligence 3, American Elsevier (1968)

(1970) See Luckham, Park and Paterson.

Paz, A.

(1971) Introduction to Probabilistic Automata, Academic Press (1971).

Peirce, C.S.

(1931) Collected Papers, Harvard University Press, Cambridge, Mass. Six Volumes (1931-1936).

Perles, E.

(1961) See Bar-Hillel, Perles and Shamir.

Perrot, J-F.

(1970) "On the relationship between finite automata, finite monoids and prefix codes," Proceedings 2nd ACM Symposium on Theory of Computing (1970) 217-220.

(1971) "Groups and automata," in Theory of Machines and Computations (Eds. Z. Kohavi and A. Paz) Academic Press (1971) 287-293.

Peters, S. and Ritchie, R.W.

(1967) "On the generative power of transformational grammars," unpublished manuscript (1967).

Petrone, L.

(1965) "Syntactic mappings of context-free languages," Proceedings, IFIP Congress, 2 (1965) 590-591.

(1968) "Syntax directed mapping of context-free languages," Proceedings, 9th Ann. Symp. on Switching and Automata Theory, October (1968).

Pfender, M.

(1972) See Ehrig and Pfender.

(1973) See Ehrig, Pfender and Schneider.

(1973a) See Ehrig, Pfender and Schneider.

(1975) See Ehrig, Kreowski and Pfender.

Pierce, R.S.

(1968) Introduction to the Theory of Abstract Algebras, Holt, Rinehart and Winston (1968).

(1970) "Classification problems," Math. Sys. Th. 4 (1970) 65-80.

Pitts, W.H.

(1943) See McCulloch and Pitts.

Pnueli, A.

(1970) See Manna and Pnueli.

Post, E.

(1943) "Formal reduction of the general combinatorial decision problem," Amer. J. Math. 65 (1943) 197-215.

(1947) "Recursive unsolvability of a problem of Thue," J. Symbolic Logic 12 (1947) 1-11.

Prather, R.E.

(1970) "On categories of infinite automata," Math. Sys. Th. 4 (1970) 295-305.

Rabin, M.O.

(1967) "Mathematical theory of automata," Proceedings, Symp. on Applied Mathematics 19 (1967) 153-175.

Rabin, M.O. and Scott, D.

(1959) "Finite automata and their decision problems," IBM J. Res. Dev. 3 (1959) 114-125.

Reynolds, J.C.

(1972) "Definitional interpreters for higher-order programming languages," Proceedings, ACM National Conference (1972) 717-740.

(1974) "Topics in programming languages," Unpublished course notes, Systems and Information Sciences Dept., Syracuse University (1974).

(1975) "Semantics of the lattice of flow diagrams," Manuscript, July 1975. To appear JACM.

Rhodes, J.

(1962) See Krohn and Rhodes.

(1965) See Krohn and Rhodes.

Rice, H.G.

(1962) See Ginsburg and Rice.

Riguet, J.

(1953) "Sur les rapports entre les concepts de machine de multipole et de structure algebrique," C.R. Acad. Sci. Paris 237 (1953) 425-427.

(1962) "Programmation et theorie des categories," Symposium on Symbolic Languages in Data Processing. Rome, Gordon and Breach (1962) 83-98.

Ritchie, R.W.

(1967) See Peters and Ritchie.

Robinson, A.

(1963) Introduction to Model Theory and the Metamathematics of Algebra. North-Holland (1963).

Rose, G.F.

(1964) "An extension of ALGOL-like languages," C.ACM 7 (1964) 52-61; Also, SDC TM-738/003/00 May (1963).

Rosen, B.K.

(1973) See Lewis and Rosen

(1973a) "Tree-manipulating systems and Church-Rosser theorems," JACM 20 (1973) 160-187.

(1974) See Lewis and Rosen.

(1975) "Deriving graphs from graphs by applying a production," Acta Informatica 4 (1975) 337-358.

(1975a) "Correctness of parallel programs," Theoretical Computer Science, to appear, 1975.

Rosenberg, A.L. and Thatcher, J.W.

(1975) "What is a multilevel array," IBM J. Research and Development 19 (1975) 163-169.

Rosenbloom, P.

(1950) Elements of Mathematical Logic, Dover Publications (1950).

Rota, G.C.

(1969) "Baxter algebras and combinatorial identities I,II," Bull. Amer. Math. Soc. 75 (1969) 325-334.

Roucheleau, Y.

(1972) "Finite-dimensional, constant, discrete-time linear dynamical systems over some classes of commutative rings," Ph.D Dissertation, Operations Research Dept., Stanford University (1972).

Roucheleau, Y., Wyman, B.F., and Kalman, R.E.

(1972) "Algebraic structure of linear dynamical systems, III. Realization theory over a commutative ring," Proceedings, Nat'l Acad. of Sciences 69 (1972) 3404-3406.

Rounds, W.C.

(1968) "Trees, transducers, and transformations," Ph.D. Dissertation, Stanford University, August (1968).

(1969) "Context-free grammars on trees," Proceedings, ACM Symp. on Theory of Computation, Marina del Rey (1969) 143. Later as "Mappings and grammars on trees," Math. Sys. Theory 4 (1970) 257-287.

Rutledge, J.D.

(1964) "On Ianov's program schemata," J.ACM 11 (1964) 1-9.

Schmidt, E.M.

(1975) See Engelfriet and Schmidt.

Schneider, H.J.

(1973) See Ehrig, Pfender and Schneider.

(1973a) See Ehrig, Pfender and Schneider.

Schnorr, C.P.

(1969) "Transformational classes of grammars," Information and Control 14 (1969) 252-277.

Schoenfield, J.R.

(1967) Mathematical Logic, Addison-Wesley, Reading, Mass. (1967).

Schutzenberger, M.

(1961) "On the definition of a family of automata," Information and Control 4 (1961) 245-270.

(1963) See Chomsky and Schutzenberger.

(1965) "On finite monoids having only trivial subgroups," Information and Control 8 (1965) 190-194.

(1969) See Eilenberg and Schutzenberger.

Scott, D.

(1959) See Rabin and Scott.

(1970) "Outline of a mathematical theory of computation," Proceedings, 4th Princeton Conf. on Inform. Science and Systems (1970).

(1971) "The lattice of flow diagrams," Symposium on Semantics of Algorithmic Languages, (Ed. E. Engeler), Lecture Notes in Math. 188 Springer-Verlag, Berlin (1971).

(1972) "Lattice theory, data types and semantics," Formal Semantics of Programming Languages, (Ed. R. Rustin), Prentice-Hall, New Jersey (1972) 65-106.

(1972a) "Continuous lattices," Proceedings, 1971 Dalhoisie Conf., Springer Lecture Note Series, 274, Springer-Verlag (1972) 97-136.

(1972b) "Mathematical concepts in programming language semantics," Proceedings, AFIPS Conference 40 (1972) 225-234.

(1972c) "Data types as lattices," unpublished notes, Amsterdam (1972).

(1973) "Lattice-theoretic models for the λ-calculus," to appear.

Scott, D. and de Bakker, J.W.

(1969) "A theory of programs," unpublished notes, IBM Seminar, Vienna (1969).

Scott, D. and Strachey, C.

(1971) "Toward a mathematical semantics for computer languages," Proceedings, Symp. on Computers and Automata Microwave Res. Inst. 21, Polytechnic Institute of Brooklyn (1971).

Semandeni, Z.

(1972) "On logical educational materials, categories, and automata," Queen's Univ. at Kingston, Ontario, Canada, Dept. of Math., Reprint No. 1972-38 (1972).

Shamir, E.

(1961) See Bar-Hillel, Perles and Shamir.

(1968) "Algebraic, rational and context-free power series in non-commuting variables," Algebraic Theory of Machines, Languages, and Semigroups (Ed. M.A. Arbib), Academic Press, New York (1968) 329-341.

Shepard, C.D.

(1969) "Languages in general algebras," Proceedings, ACM Symp. Theory of Computing (1969) 155-163.

Sipusic, T.L.

(1972) "Two automata decomposition theorems generalized to machines in a closed monoidal category," Institute for Computer Research, Univ. of Chicago, Quarterly Report No. 35, November (1972).

Srinivasan, C.V.

(1962) "State diagram of linear sequential machines," J.Franklin Inst. 273 (1962) 383-418.

Starke, P.H.

(1972) Abstrakte Automaten, VEB-Verlag, Berlin 1969; English translation: Abstract Automata, Elsevier, North Holland (1972).

(1972a) "Allgemeine Probleme und Methoden in der Automatentheorie," EIK 8 (1972) 489-517.

Stearns, R.E.

(1966). See Hartmanis and Stearns.

(1968) See Lewis and Stearns.

Strachey, C.

(1971) See Scott and Strachey.

Strecker, C.E.

(1973) See Herrlich and Strecker.

Street, Ross

(1974) See Kelly and Ross.

Strong, H.R.

(1968) "Algebraically generalized recursive function theory," IBM J. Res. Devel. 12 (1968) 465-475.

(1973) See Walker and Strong.

Tarski, A.

(1955) "A lattice theoretical fixpoint theorem and its applications," Pacific J. Math. 5 (1955) 285-390.

Thatcher, J.W.

(1963) "Notes on mathematical automata theory," University of Michigan Technical Note, December, 1963.

Thatcher, J.W. (cont'd)

(1967) "Characterizing derivation trees of context-free grammars through a generalization of finite automata theory," J. Comp. and Sys. Sci. 1 (1967) 317-322.

(1969) "Transformations and translations from the point of view of generalized finite automata theory," Proceedings, ACM Symp. Th. Comp., Marina del Rey (1969) 129-142.

(1970) "Generalized sequential machine maps," J. Comp. and Sys. Sci. 4 (1970).

(1972) See Thatcher and Miller.

(1973) "Tree automata - an informal survey," Currents in Computing, (Ed. A.V. Aho), Prentice-Hall, New Jersey (1973).

(1974) See Burstall and Thatcher.

(1974a) See Goguen, Thatcher, Wagner and Wright (1974).

(1974b) See Goguen, Thatcher, Wagner and Wright (1974a).

(1975) See Rosenberg and Thatcher.

(1975a) See Goguen, Thatcher, Wagner and Wright (1975).

(1975b) See Goguen, Thatcher, Wagner and Wright (1975a).

(1975c) See Goguen, Thatcher, Wagner and Wright (1975b).

(1975d) See Goguen, Thatcher, Wagner and Wright (1975c).

(1975e) See Goguen, Thatcher, Wagner and Wright (1975d).

Thatcher, J.W. and Wright, J.B.

(1968) "Generalized finite automata with an application to a decision problem of second-order logic," Math. Sys. Th. 2 (1968) 57-81.

Thue, A.

(1914) "Probleme uber vanderungen vin zeichenreihen nach gegebenen Regeln," Skrifter utgit av Videnskapsselskapet i Kristiania, I. Matematisk naturvidenskabelig klasse, No. 10, (1914) 34.

Turing, A.M.

(1949) "Checking a large routine," Report of a Conference on High Speed Automatic Calculating Machines, University Mathematical Laboratory, Cambridge, England, June (1949) 67-69.

Ullman, J.D.

(1968) See Aho and Ullman.

(1969) See Hopcroft and Ullman.

von Wijngaarden, A.

(1969) "Report on the algorithmic language, ALGOL 68," Numer. Math. 14 (1969) 79-218.

von Neumann, John

(1963) Collected Works 5, Macmillan, New York (1963).

Vuillemin, J.

(1972) See Manna, Ness and Vuillemin.

(1972a) See Manna and Vuillemin.

(1973) "Correct and optimal implementations of recursion in a simple programming language," Proceedings, Fifth Annual Symposium on Theory of Computing (1973) 224-239.

Wagner, E.

(1969) "Uniformly reflexive structures: On the nature of Godelizations and relative computability," Trans. Amer. Math. Soc. 144 (1969) 1-41.

(1971) "Languages for defining sets in arbitrary algebras," Proceedings, 12th IEEE Symp. on Switching and Automata Th., East Lansing, Mich. (1971).

(1971a) "An algebraic theory of recursive definitions and recursive languages," Proceedings, 3rd Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio (1971).

(1973) "From algebras to programming languages," Proceedings, 5th ACM Symp. on Theory of Computing, Austin, Texas (1973).

(1974) See Goguen, Thatcher, Wagner and Wright.

(1974a) See Goguen, Thatcher, Wagner and Wright.

(1975) See Goguen, Thatcher, Wagner and Wright.

(1975a) See Goguen, Thatcher, Wagner and Wright.

(1975b) See Goguen, Thatcher, Wagner and Wright.

(1975c) See Goguen, Thatcher, Wagner and Wright.

(1975d) See Goguen, Thatcher, Wagner and Wright.

Walker, S.A. and Strong, H.R.

(1973) "Characterization of flowchartable recursions," JCSS 7 (1973) 404-447.

Walter, H.

(1969) See Hotz and Walter.

(1970) "Verallgemeinerte Pullbackkonstruktionen bei Semi-Thue-Systemen und Grammatiken," EIK 6 (1970) 239-254.

(1975) "Einige topologische Aspekte der syntaktischen analyse und ubersetzung bei Chomsky-Grammatiken," J.Comp. Sys. Sci., to appear (1975).

Wand, M.

(1971) "Theories, pretheories, and finite state transformations on trees," MIT, Artificial Intell. Lab. Memo. No. 216 (1971).

(1972) "A concrete approach to abstract recursive definitions," MIT, Artificial Intell. Lab. Memo. No. 262 (1972).

Wand, M. (Cont'd)

(1972a) "Closure under program schemes and reflective subcategories," MIT, Artificial Intell. Lab. Memo. (1972) .

(1972b) "Uninterpreted schemes are natural transformations," MIT, Artificial Intell. Lab. (1972).

(1972c) "Models of higher type," MIT, Artificial Intell. Lab. (1972).

(1974) "An algebraic formulation of the Chomsky hierarchy," Category Theory Applied to Computation and Control, Univ. of Mass., 1974. Springer Lecture Notes in Computer Science 25 (1975) 209-213.

(1974a) "On the recursive specification of data types," Category Theory Applied to Computation and Control, Univ. of Mass., 1974. Springer Lecture Notes in Computer Science 25 (1975) 214-217.

Warshall, S.

(1962) "A theorem on Boolean matricies," JACM 9 (1962) 11-12.

Weber, H.

(1966) See Wirth and Weber.

Wegner, P.

(1972) "A view of computer science education," Amer. Math. Monthly, February (1972) 168-172.

(1974) "Modification of Aho and Ullman's correctness proof of Warshall's algorithm," SIGACT News January (1974) 32-35.

Willsky, A.S.

(1973) See Brockett and Willsky.

(1974) See Brockett and Willsky.

Windeknecht, T.G.

(1967) "Mathematical systems theory: Casuality," Math. Sys. Th. 1 (1967).

Wirth, N.

(1973) Systematic Programming: An Introduction, Prentice-Hall, Englewood Cliffs, New Jersey (1973).

Wirth, N. and Weber, H.

(1966) "EULER: A generalization of ALGOL and its formal definitions: Part I," C.ACM 9 (1966) 13-23.

Wiweger, A.

(1973) "On coproducts of automata," Bull. Acad. Polonaise Sci. 21 (1973) 753-758.

Wright, J.B.

(1960) See Buchi and Wright.

(1962) See Burks and Wright.

(1965) See Thatcher and Wright.

(1967) See Eilenberg and Wright.

(1967a) See Mezei and Wright.

(1968) See Thatcher and Wright.

(1974) See Goguen, Thatcher, Wagner and Wright.

(1974a) See Goguen, Thatcher, Wagner and Wright.

(1975) See Goguen, Thatcher, Wagner and Wright.

(1975a) See Goguen, Thatcher, Wagner and Wright.

(1975b) See Goguen, Thatcher, Wagner and Wright.

(1975c) See Goguen, Thatcher, Wagner and Wright.

(1975d) See Goguen, Thatcher, Wagner and Wright.

Wyman, D.F.

(1972) See Roucheleau, Wyman and Kalman.

Yacobellis, R.

(1972) See Goguen and Yacobellis.

(1973) "Generalized Krohn-Rhodes theory," Ph.D. Dissertation, Univ. of Chicago, Committee on Information Sciences (1973).

Younger, D.H.

(1967) Recognition and parsing of context-free languages in time n^3. Information and Control 10, 2 (1967) 189-208.

Zadeh, L. and Desoer, C.A.

(1963) Linear Systems Theory, McGraw-Hill, New York (1963).

Zalcstein, Y.

(1970) See Give'on and Zalcstein.

(1972) "Locally testable languages," J. Comp. and Sys. Sci. 6 (1972) 151-167.

(1973) "Syntactic semigroups of some classes of star-free languages," Automata, Languages and Programming (Ed. M. Nivat), North-Holland (1973).

(1973a) "On the semigroups of linear sequential machines," J. Comp. and Sys. Sci. 2 (1973) 25-28.

Zeiger, H.P.

(1967) "Ho's algorithm, commutative diagrams, and the uniqueness of minimal linear systems," Information and Control 11 (1967) 71-79.

(1969) See Arbib and Zeiger.

Zeigler, B.

(1971) "A note on series-parallel irreducibility," Math. Sys. Th. 5 1-3.

Zisman, M.

(1967) See Gabriel and Zisman.

INDEX OF DEFINITIONS, CONCEPTS AND EXAMPLES

INDEX OF SYMBOLS AND NOTATION WITH FIRST OCCURRENCE

Logical Symbols

$\wedge$	and (16)
$\perp$	bottom (37)
$\perp$	false (16)
$\forall$	for all (16)
===>	implication (16)
<===>	logical equivalence (16)
$\neg$	not (16)
$\vee$	or (16)
$\exists$	there exists (16)
$\exists!$	there exists a unique (16)
$\top$	top (37)
$\top$	true (16)

Sets and Posets

#	cardinality (23)
$\cdot$	composition (23)
$^{\vee}$	converse (23)
$\coprod$	coproduct (28)
$\upharpoonright$	corestriction (22)
-	difference (18)
$\emptyset$	empty set (18)
λ	empty function (26)
$\underset{\sim}{P}_F$	finite subsets (18)
1_S	identity on S (20)
$\cap$	intersection (18)
$\bigcap$	intersection (28)
$^{-1}$	inverse (26)
$\sqcup$	least upper bound (104)
$\in$	membership (17)
$\notin$	nonmembership (17)
ω	nonnegative integers (19)
$\underline{\leq}$	order relation on poset (103)
$\underset{\sim}{P}$	powerset (18)
X	product (18)
$\prod$	product (28)
/	quotient (29)
*	reflexive transitive closure (30)
$\upharpoonright$	restriction (22)
$\subseteq$	subset (18)
$^{+}$	transitive closure (30)
$\cup$	union (18)
$\bigcup$	union (28)

Miscellaneous

+	addition (14)
=	equals (16)
/	division (16)
$\bigwedge$	greatest lower bound (38)
$\bigvee$	least upper bound (38)
$\leq$	order (37)
$\geq$	order (38)
$\sum$	sum (16)
____	boldface (5)
- - -	italic (5)
$\sim\sim\sim$	script (5)
====	special (5)
$\square$	end of proof (16)

Categories - general

$\circ$	composition (40)
$\circledast$	free category (75)
1_A	identity (41)
$\underline{\underline{C}}$	morphisms (39)
$\lvert\underline{\underline{C}}\rvert$	objects (39)
^{op}	opposite (63)
∂_0	source (39, 72)
∂_1	target (39, 72)

Arrow notations

$\rightarrow$	(19)
$\mapsto$	(20)
$>\!\!\rightarrow$	(22)
$\twoheadrightarrow$	(22)
$\hookrightarrow$	(22)
$\overset{\sim}{=}$	(22)
$\Longrightarrow$	(82)
$\multimap$	(103)

372/ 36