stichting
mathematisch
centrum

MC

REKENAFDELING MR 131/72 FEBRUARY

J.W. DE BAKKER and W.P. DE ROEVER

# A CALCULUS FOR RECURSIVE PROGRAM SCHEMES

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat 49, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

## ABSTRACT

Scott's analysis of the semantics of recursive program schemes, leading to their characterization as minimal fixed points of continuous transformations, is presented. Tarski's axioms for a relation algebra are then combined with an induction rule due to Scott into a calculus for such schemes, in which properties such as equivalence, termination and correctness can be stated and proved formally. Various applications of the calculus are exhibited, including examples on while statements, a formal justification of Floyd's inductive assertion method, and an analysis of data structures - integers and trees - which can be characterized inductively.

CONTENTS

# 1. INTRODUCTION

## 1.1. General

The present paper is devoted to a study of the mathematical foundations of techniques for proving program correctness, in particular with reference to programs involving *recursion*.

We are concerned not so much with specific programs for solving individual problems, but with an analysis of techniques which are applicable to classes of programs. Technically speaking, we are interested in program schemes, and we derive results which hold for all interpretations of the components of these schemes.

Our paper has a number of predecessors. First of all, the unpublished notes by Dana Scott [26], partly describing joint work with the present first author. In these notes, a mathematical characterization of recursive program schemes as *minimal fixed points* of *continuous* transformations is presented; moreover, an induction rule for proving properties of such schemes is proposed. These notes formed the basis for the monograph by J.W. de Bakker: "Recursive procedures" [2] and the related paper [3]. In both these papers, Scott's theory was developed and then applied to examples on and further investigation of *equivalence* of programs.

The present paper extends the methods of the previous ones, in the sense that now various formulations of *correctness* and *termination* are also studied. The main new tool is the use of the calculus of relations in the sense of Tarski [31]. (Similar use of relations first appeared in unpublished work by Milner [21] and Park [24], who do not, however, use Tarski's axiom system.) Combination with Scott's induction rule then yields a system which has a richer power of expression than that of [2, 3].

The relevant results on the calculus of relations, formulated in the framework of relation algebras, are summarized in section 2. In section 3, the characterization of recursive program schemes as minimal fixed points of continuous transformations is presented. This section is essentially the same as the corresponding one in [3]. Next, in section

4 the ideas of sections 2 and 3 are combined into a calculus for recursive program schemes, and the correspondence of constructs in the calculus with various programming concepts is discussed. E.g., McCarthy's axioms for conditionals [20] are now derivable; moreover, a new operator is introduced which expresses that a program P yields, for input x, a result y which satisfies property p. This operator is applied in several examples in section 5, dealing with conditions satisfied upon termination of while statements. As another example, a question which at first sight appeared to be a tree-searching problem, is shown to be an instance of a much more general equivalence on schemes. Section 6 contains a formal justification of Floyd's inductive assertion method. A precise formulation of the method in our calculus, and a proof in which Scott's induction rule plays the main role, is given. In section 7, the calculus is applied to an investigation of two data structures which can be characterized inductively. First a set of axioms is given, expressing properties of the successor function S, which characterize the domain of non-negative integers. The main axiom is a formulation of the counterpart of mathematical induction in our framework. Various examples of properties of function over the non-negative integers are then discussed, including a proof of termination of (a generalization of) the factorial functions, and a formulation of Julia Robinson's "general recursion" [25]. Next, it is shown how a variation of the axioms for non-negative integers leads to a characterization of tree structures, where inductive arguments can be given in two directions: properties of the "father" nodes in terms of its "sons", and vice versa. Section 8 contains some conclusions, and an indication of possible extensions of our work.

## 1.2. Related work

The first proof technique for showing equivalence of recursive procedures was McCarthy's *recursion induction* [20]. (Study of this classical paper is recommended for the reader who is not familiar with the problem area dealt with in our paper.)

Characterization of recursive procedures as minimal fixed points and proof techniques based thereupon appear in some or other form in work by Bekić [4], Morris [22] and Park [23]. Related results in a different setting have been obtained by Blikle [6] and Leszczylowski [16].
A systematic development of the notion of continuity in the mathematical theory of computation has been given by Scott. For an introductory exposition see [27]; the mathematical background is to be found in [29], specific applications in [28] and [30].
Correctness proofs usually have as a starting point the method of Floyd [11], as elaborated in a number of papers by Manna (see e.g. [18]) and others.
Additional references may be found in De Bakker [2].

## 2. RELATIONS AND RELATION ALGEBRAS

In this section we summarize the basic notions on relations and relation algebras that we need in the sequel.

A *relation* R with respect to a set $\mathcal{D}$ is a subset of the cartesian product $\mathcal{D} \times \mathcal{D}$. For $(x,y) \in R$ we usually write xRy.

The following operations on relations will be used:

a. Binary operations

(i) *Composition*

$$R_1 \; ; \; R_2 \quad = \{(x,y) \mid \exists z[xR_1z \text{ and } zR_2y]\}$$

(ii) *Union*

$$R_1 \cup R_2 \quad = \{(x,y) \mid xR_1y \text{ or } xR_2y\}$$

(iii) *Intersection*

$$R_1 \cap R_2 \quad = \{(x,y) \mid xR_1y \text{ and } xR_2y\}$$

b. Unary operations

(i) *Conversion*

$$\breve{R} \quad = \{(x,y) \mid yRx\}$$

(ii) *Complementation*

$$\bar{R} \quad = \{(x,y) \mid (x,y) \notin R\}$$

c. Nullary operations

(i) The *empty* relation

$$\Omega \quad = \emptyset \text{ (the empty subset of } \mathcal{D} \times \mathcal{D})$$

(ii) The *identity* relation

$$E \quad = \{(x,x) \mid x \in \mathcal{D}\}$$

(iii) The *universal* relation

$$U \quad = \mathcal{D} \times \mathcal{D}$$

Clearly, $R_1 \subseteq R_2$ iff $R_1 \cup R_2 = R_2$ iff $R_1 \cap R_2 = R_1$.

Example 1. A relation R is an equivalence relation iff the following three conditions are satisfied:

1. $E \subseteq R$ (R is *reflexive*)

2. $R = \breve{R}$ (R is *symmetric*)

3. $R; R \subseteq R$ (R is *transitive*)

Example 2.

1. R is a *function* iff $\breve{R}; R \subseteq E$

2. R is *total*, i.e., $\forall x \exists y[xRy]$, iff $E \subseteq R; \breve{R}$.

Example 3.

$$\begin{aligned} R; \breve{R} \cap E &= \{(x,y) \mid xEy \text{ and } xR; \breve{R}y\} \\ &= \{(x,y) \mid x = y \text{ and } \exists z[xRz \text{ and } z\breve{R}y]\} \\ &= \{(x,x) \mid \exists y[xRy]\} \end{aligned}$$

Hence, $R;\breve{R} \cap E$ determines that subset of E which consists of all pairs (x,x) such that there exists some y with xRy. This indicates a correspondence with a predicate expressing the *termination* of a program for given input, an idea which will be pursued in section 4. Note also that $R; \breve{R} \cap E = R; U \cap E$.

A *concrete* relation algebra over a domain $\mathcal{D}$ is a family of subsets of $\mathcal{D} \times \mathcal{D}$, together with the above mentioned operations and closed with respect to these. In [31], Tarski has proposed what amounts to the notion of an *abstract* relation algebra. This is an algebraic structure over a given set, with operations as above, but the elements of which are not necessarily subsets of a pair set. Cf. the notion of a boolean algebra, which in a similar way is an abstraction of the algebra of subsets of a given set.

An abstract relation algebra $RA$ is a structure

$$\langle \mathcal{R},\ ;,\ \cup,\ \cap,\ \breve{}\ ,\ \bar{}\ ,\ \Omega,\ E,\ U\rangle$$

where

1. $\mathcal{R}$ is any set.

2a. $;$, $\cup$, $\cap$ are binary operations,

b. $\breve{}$ , $\bar{}$ are unary operations,

c. $\Omega$, $E$, $U$ are nullary operations.

3. $\langle \mathcal{R}, \cup, \cap, \bar{}\ , \Omega, U\rangle$ is a *boolean algebra* with 0-element $\Omega$ and 1-element $U$.

4. $;$, $\breve{}$ , $E$ satisfy the following five postulates:

$T_1$: $(R;S);T = R;(S;T)$

$T_2$: $\breve{\breve{R}} = R$

$T_3$: $\overset{\smile}{R;S} = \breve{S};\breve{R}$

$T_4$: $R;E = R$

$T_5$: If $(R;S) \cap T = \Omega$, then $(S;\breve{T}) \cap \breve{R} = \Omega$.

(For an alternative introduction of the notion of relation algebra, as a specialization of a lattice-ordered monoid, see Birkhoff [5], pp. 343, 344.)

In the sequel, we shall omit parentheses in our formulae, based on the associativity of the binary operations and on the convention that ";" has priority over "$\cap$", which has in turn priority over "$\cup$".

It is straightforward to verify that (the postulates of a boolean algebra and) $T_1$ to $T_5$ are satisfied in a concrete relation algebra. Consider for example $T_5$. Assume $R;\ S \cap T = \Omega$, and suppose that $S;\ \breve{T} \cap \breve{R} \neq \Omega$. Then there exist x, y, z such that $xSz$, $z\breve{T}y$, and $x\breve{R}y$. Hence, $xSz$, $yTz$ and $yRx$ hold, from which $y(R;S\cap T)z$ follows. Contradiction.

In [31], Tarski has investigated the converse problem: he has shown how to derive from the postulates of an abstract relation algebra a number of fundamental properties of a concrete relation algebra. These properties are collected in

Lemma 2.1

1. If $R \subseteq S$, then $\breve{R} \subseteq \breve{S}$, $R;T \subseteq S;T$ and $T;R \subseteq T;S$
2. $\Omega;R = R;\Omega = \Omega$
3. $E;R = R$
4. $\breve{\Omega} = \Omega$, $\breve{E} = E$, $\breve{U} = U$
5. $R;(S \cup T) = R;S \cup R;T$
   $(S \cup T);R = S;R \cup T;R$
6. $\overset{\smile}{R \cup S} = \breve{R} \cup \breve{S}$
   $\overset{\smile}{R \cap S} = \breve{R} \cap \breve{S}$
   $\breve{\bar{R}} = \bar{\breve{R}}$

Proof. See [31].

Lemma 2.1, part 1, expresses the *monotonicity* of "⌣" and ";". Together with the (obvious) monotonicity of "∪" and "∩", this will play an important part in the analysis of recursion, to be given in sections 3 and 4.

Caution: It is not true that *all* properties of a concrete relation algebra are derivable in an abstract relation algebra. For a counterexample see Lyndon [17].

In the sequel, we shall need various additional properties of abstract relation algebras. Their proofs will usually be based on

Lemma 2.2 $\quad R;S \cap T = R;(\breve{R};T \cap S) \cap T.$

Proof

$$\begin{aligned} R;S \cap T &= R;(U \cap S) \cap T \\ &= R;\{(\overline{\breve{R};T} \cup \breve{R};T) \cap S\} \cap T \\ &= \{R;(\overline{\breve{R};T} \cap S) \cap T\} \cup \{R;(\breve{R};T \cap S) \cap T\}. \end{aligned}$$

Also,

$$R;(\overline{\breve{R};T}\cap S) \cap T = \qquad (T_5)$$

$$(\overline{\breve{R};T}\cap S);\breve{T} \cap \breve{R} = \qquad (T_5,\ T_2)$$

$$\breve{T};R \cap \overline{\overline{\breve{R};T} \cap S} = \qquad (T_2,\ T_3,\ \text{lemma 2.1})$$

$$\breve{T};R \cap \overline{\breve{T};R} \cap \breve{S} =$$

$$\Omega\ .$$

Thus, $R;S \cap T = R;(\breve{R};T\cap S) \cap T$ follows.

The first applications of lemma 2.2 follow in the proof of lemma 2.3, in which a number of useful properties of relations and functions are derived formally. E.g., statement 2.3.4b. can be read as: If $R \subseteq S$, if $S$ is a function (hence $R$ is a function), and $R$ is total, then $R = S$.

Lemma 2.3

1. If $\breve{R};R \subseteq E$, then $R;(S\cap T) = R;S \cap R;T$

2. If $R \subseteq E$, then $\breve{R} = R$

3a. $R = (R;U\cap E);R$

b. $R;U = (R;U\cap E);U$

c. $R;U \cap E = R;\breve{R} \cap E$

4a. If $R \subseteq S$, $\breve{S};S \subseteq E$, then $R = (R;U\cap E);S$

b. If $R \subseteq S$, $\breve{S};S \subseteq E$, $E \subseteq \breve{R};R$ then $R = S$.

Proof

1. $\subseteq$: Clear

$\supseteq$: By lemma 2.2 and the assumption,

$$R;S \cap R;T = R;(\breve{R};R;T\cap S) \cap T \subseteq R;(T\cap S).$$

2. $R = R \cap E =$ (lemma 2.2) $R;(\breve{R};E \cap E) \cap E \subseteq R;\breve{R} \subseteq E;\breve{R} = \breve{R}$. Thus, $R \subseteq \breve{R}$ and from this $\breve{R} \subseteq \breve{\breve{R}} = R$, whence the result.

3a. $\breve{R} = \breve{R} \cap U =$ (lemma 2.2) $\breve{R};(R;U \cap E) \cap U = \breve{R};(R;U \cap E)$. Thus, by $T_3$, $R = (\overgroup{R;U \cap E});R = (R;U \cap E);R$, by part 2.

b. Clearly, $U;U = U$. Using this and replacing, in part 3a, $R$ by $R;U$, we obtain

$$R;U = (R;U;U \cap E);R;U \subseteq (R;U \cap E);U \subseteq R;U;U = R;U \ .$$

c. Direct from lemma 2.2.

4a. $\supseteq$: We have successively

$$R \subseteq S$$

$$\breve{S};R \subseteq \breve{S};S \subseteq E$$

$$\breve{S};R;\breve{R} \subseteq \breve{R}$$

$$R;\breve{R};S \subseteq R$$

$$(R;\breve{R} \cap E);S \subseteq R;\breve{R};S \subseteq R$$

The result now follows from part 3c.

$\subseteq$: Follows from part 3.

b. Immediate from part 4a.

## 3. RECURSIVE PROGRAM SCHEMES

A class of structures called *recursive program schemes* is introduced, and it is indicated how a program scheme can be interpreted as a relation over a given domain. This section follows closely [3].
In the composition of program schemes, the following classes of symbols are used

a. $\mathcal{A}$: The class of *elementary statement* symbols, with elements $A, A_1, A_2, \ldots$ .

b. $\mathcal{B}$: The class of *boolean* symbols, with elements $p, p_1, q, r, \ldots$ .

c. $\mathcal{C}$: The class of *constant* symbols, with the two elements $\Omega$ and E.

d. $\mathcal{P}$: The class of *procedure* symbols, with elements $P, P_1, P_2, \ldots$ .

The intended correspondence between these classes and their counterparts in an ALGOL-like programming language should be clear. In particular, $\Omega$ will correspond to the undefined statement (L: goto L, say), and E to the dummy statement.
Next, we give the definition of the class of statement schemes:

a. Each element of $\mathcal{A}$, $\mathcal{C}$ or $\mathcal{P}$ is a statement scheme.

b. If $S_1$, $S_2$ are statement schemes, and $p \in \mathcal{B}$, then $S_1;S_2$ and $(p \rightarrow S_1,S_2)$ are statement schemes.

The notation $(p \rightarrow S_1,S_2)$ is short for the ALGOL-notation if p then $S_1$ else $S_2$.
$S, S_1, \ldots, T, T_1, \ldots$ will stand for arbitrary statement schemes.
When we want to indicate that T possibly contains one or more occurrences of S, we write $T = T(S)$. The result of substituting $S_1$ for all occurences of S in T is then written as $T(S_1)$.
Besides the statement schemes we have *declaration schemes:* A declaration scheme is a pair $(P,T(P))$, with $P \in \mathcal{P}$, and $T(P)$ a statement scheme. Such a pair will usually be denoted in the sequel by the more familiar notation procedure $P;T(P)$.

A *program scheme* is again a pair (D,T), where D is a finite set of declaration schemes, and T is a statement scheme.

Examples of program schemes are

$$\begin{array}{l}(\underline{\text{procedure}}\ P;\ (p\rightarrow A;P,E)\\ \ P)\end{array}$$

or

$$\begin{array}{l}(\underline{\text{procedure}}\ P_1;\ (p\rightarrow A_1;P_1,E);\\ \ \underline{\text{procedure}}\ P_2;\ (q\rightarrow A_2;P_2,(r\rightarrow A_3;P_1,A_4));\\ \ P_1;\ (r\rightarrow P_2,\Omega);A_5).\end{array}$$

For the reader who is more accustomed to a functional notation, we give a definition scheme corresponding to the procedures $P_1$ and $P_2$ of the second example. Using the notation "$\Leftarrow$" for "is recursively defined by", we have

$$P_1(x) \Leftarrow (p(x) \rightarrow P_1(A_1(x)),x)$$

$$P_2(x) \Leftarrow (q(x) \rightarrow P_2(A_2(x)),(r(x) \rightarrow P_1(A_3(x)),A_4(x))).$$

Program schemes cannot be "executed" as such. First, a rule has to be given to attribute a meaning to the statement symbols, boolean symbols and constant symbols occurring in it. Secondly, we then must define how to provide a meaning for the various constructs in the program scheme, in terms of the meaning of their constituent symbols.

An *interpretation* I of a program scheme (D,T) consists of a pair $<\mathcal{D},h>$, where

a. $\mathcal{D}$ is any domain.

b. h maps symbols to relations or partial functions as follows:

1. To each statement symbol A occurring in (D,T), (i.e., either in T or in one of the elements of D) a relation $A^h \subseteq \mathcal{D} \times \mathcal{D}$ is assigned.
2. To each boolean symbol p occurring in (D,T) a *partial* function $p^h$: $\mathcal{D} \rightarrow \{0,1\}$ is assigned.

3. To the constant $\Omega$ the empty relation $\Omega^h \subseteq \mathcal{D} \times \mathcal{D}$ is assigned, and to E the identity relation $E^h = \{(x,x) \mid x \in \mathcal{D}\}$ is assigned.

Given an interpretation $I = \langle\mathcal{D},h\rangle$ of the symbols in a program scheme (D,T), we next define how to obtain the relation $(D,T)^h \subseteq \mathcal{D} \times \mathcal{D}$. For this definition, we need the definition of a "*computation sequence*' with respect to D and I":

Let $x_i \in \mathcal{D}$, $1 \leq i \leq n+1$, and let $S_i$, $1 \leq i \leq n$, be statement schemes. A computation sequence is a finite sequence

$$x_1\ S_1\ x_2\ S_2\ \ldots\ x_n\ S_n\ x_{n+1}$$

such that for each i, $1 \leq i \leq n$

a. If $S_i = E$, then $i = n$ and $x_{n+1} = x_n$ (i.e., $x_n\ E^h\ x_{n+1}$).

b. If $S_i = A;S$ then $\begin{cases} x_i\ A^h\ x_{i+1} \\ S_{i+1} = S\ . \end{cases}$

c. If $S_i = (p \rightarrow S',S'');S$ then $\begin{cases} x_{i+1} = x_i \\ S_{i+1} = S';S,\ \text{if } p^h(x_i) = 1 \\ S_{i+1} = S'';S,\ \text{if } p^h(x_i) = 0\ . \end{cases}$

d. If $S_i = P;S$ then $\begin{cases} x_{i+1} = x_i \\ S_{i+1} = T(P);S \text{ where } (P,T(P)) \in D. \end{cases}$

Observe that

1. Each computation sequence has E as its last statement scheme. This is a convention which simplifies the definition. In the sequel, we shall assume that conventional occurrences of E have been added, where necessary, without, however, explicitly indicating these occurrences. E.g., we shall write

$$x_1\ P\ x_2\ T(P)\ \ldots\ x_n\ E\ x_{n+1}$$

instead of

$$x_1 \; P;E \; x_2 \; T(P);E \; \ldots \; x_n \; E \; x_{n+1}.$$

2. Clause c corresponds to the usual meaning of conditionals, with 1(0) corresponding to <u>true</u> (<u>false</u>).

3. Clause d corresponds to the usual copy rule for procedures: In order to elaborate a procedure call P, replace P by its body T(P) and elaborate the result.

The following notion on computation sequences will be used below: Let $F \in \mathcal{A} \cup \mathcal{C} \cup \mathcal{P}$. We say that F occurs *executable* in a computation sequence iff it occurs in the form

$$\ldots \; x_i \; F;S \; x_{i+1} \; \ldots$$

Clearly, no computation sequence contains executable occurrences of $\Omega$. The notion of computation sequence is used as follows:
Given a program scheme (D,T) and an interpretation $I = \langle \mathcal{D},h \rangle$ of the symbols in D and T, we define the relation $(D,T)^h \subseteq \mathcal{D} \times \mathcal{D}$ by:
For each $x,y \in \mathcal{D}$,

$x(D,T)^h y$ iff there exists a computation sequence with respect to D and I: $x_1 \; S_1 \; x_2 \; S_2 \; \ldots \; x_n \; S_n \; x_{n+1}$ with $x_1 = x$, $S_1 = T$, and $x_{n+1} = y$.

In the sequel, we shall omit explicit mentioning of the set of declarations D, if no confusion can arise. We then write simply $T^h$ instead of $(D,T)^h$. Some simple consequences of its definition are:

a. $x(T_1;T_2)^h y$ iff there exists $z \in \mathcal{D}$ such that $x \; T_1^h \; z$ and $z \; T_2^h \; y$. Hence, the relation $(T_1;T_2)^h$ is indeed the composition of the relations $T_1^h$ and $T_2^h$.

b. $x(p \to T_1,T_2)^h y$ iff either $p^h(x) = 1$ and $x \; T_1 \; y$ or $p^h(x) = 0$ and $x \; T_2 \; y$.

For given $I = \langle \mathcal{D},h \rangle$, we say that $(T_1 \subseteq T_2)^I$ holds iff $T_1^h \subseteq T_2^h$ holds. If, for all I, $(T_1 \subseteq T_2)^I$ holds, we write $T_1 \subseteq T_2$. Similarly, equivalence of two program schemes under all interpretations is denoted by $T_1 = T_2$. We now apply the notions introduced so far in an analysis of the semantics of recursive procedures. The first result to be noted about procedures is the fixed point property, stated in

Lemma 3.1. If $(T_1,D_1)$ is a program scheme, and $(P,T(P)) \in D_1$, then

$$(3.1) \qquad P = T(P).$$

Proof. Follows easily from the definitions, in particular from the use of the copy rule in the definition of a computation sequence.

In general, (3.1) does not determine the procedure uniquely. E.g., consider the procedure $P_1$ determined by procedure $P_1$; $(p \to P_1,E)$. Each S of the form $(p \to A,E)$, for arbitrary A, satisfies $S = (p \to S,E)$, as can easily be seen from the equivalence $(p \to A,E) = (p \to (p \to A,E),E)$. We can express this fact also by saying that the transformation $T(X) = (p \to X,E)$ has an infinity of fixed points, i.e., for all S of the form $(p \to A,E)$, we have $T(S) = S$. This example illustrates the need for a further analysis of the semantics of recursive procedures, in order to determine which, among all possible fixed points, is the one we need.
The first step is to observe the monotonicity property of program schemes:

Lemma 3.2. For all interpretations I, if $(S_1 \subseteq S_2)^I$, then $(T(S_1) \subseteq T(S_2))^I$.

Proof. This amounts to a straightforward application of the definitions of computation sequence and of interpretation of schemes, and is omitted here.

The next lemma expresses the following familiar property of recursive procedures: If a recursive procedure terminates for a given argument, it terminates after a finite number of "inner calls" of the procedure. At the innermost level, P is not needed again, so it may be replaced by whatever statement we choose, without influencing the further execution. The reason for our choice of $\Omega$ for this statement will follow presently.

Lemma 3.3. If $I = \langle \mathcal{D},h\rangle$, and if, for $x,y \in \mathcal{D}$, $x\ P^h\ y$ holds, then there exists $i > 0$ such that $x\ T^i(\Omega)^h y$ holds.

Proof. (This proof is due to P. van Emde Boas, who observed that the original proof in [3] is incorrect.) The proof uses the relationship "to identify" between occurrences of symbols in a computation sequence: Let

$$x_1\ S_1\ x_2\ S_2\ \ldots\ x_i\ S_i\ x_{i+1}\ S_{i+1}\ \ldots\ x_n\ E\ x_{n+1}$$

be a computation sequence. We say that an occurrence of a symbol $F \in A \cup C \cup P$, where F occurs in some S contained in $S_i$, *directly identifies* the corresponding occurrence of F in S, contained in $S_{i+1}$, in each of the following three cases:

a. $S_i = A;S$ and $S_{i+1} = S$ .

b. $S_i = (p \to S,S');S''$ and $S_{i+1} = S;S''$, or

$S_i = (p \to S',S);S''$ and $S_{i+1} = S;S''$, or

$S_i = (p \to S',S'');S$ and either $S_{i+1} = S';S$ or $S_{i+1} = S'';S$ ,

c. $S_i = P;S$ and $S_{i+1} = T(P);S$.

The relationship *to identify* is then defined to be the reflexive and transitive closure of the relationship *to identify directly*.
We now introduce the following transformation on computation sequences: Let

(3.2) $$x_1\ S_1(P)\ x_2\ S_2(P)\ \ldots\ x_n\ E\ x_{n+1}$$

be a computation sequence.

Step 1. Consider all occurrences of P in (3.2) which are identified by an occurrence of P in $S_1(P)$.

Step 2. Mark all those considered occurrences of P which are executable.

Step 3. Replace all other considered occurrences of P by T(P).

Step 4. Replace all combinations

$$\ldots x_j\ P^*;S\ x_{j+1}\ T(P);S\ x_{j+2}\ \ldots$$

where $P^*$ is an occurrence of P, marked as a result of step 2, by

$$\ldots x_j\ T(P);S\ x_{j+2}\ \ldots$$

It can be verified that the result of applying this transformation to the computation sequence (3.2) is again a computation sequence which has at least one executable occurrence of P less than (3.2). In fact, at least the left-most executable occurrence of P in (3.2), if at all present, has been deleted. Observe that this transformation may be viewed as follows: The calls of P at the outermost level vanish and the calls at recursion depth 1 become outermost. By iterating the transformation upto the maximal recursion depth, all calls have become outermost and all recursion has vanished.
We use the transformation to obtain the proof of the lemma as follows: By assumption, there is a computation sequence

(3.3) $$x_1\ P\ x_2\ T(P)\ x_3\ \ldots\ x_n\ E\ x_{n+1}$$

with $x_1 = x$, $x_{n+1} = y$. Repeatedly applying the transformation to (3.3) yields eventually, for some integer $i > 0$, the computation sequence

(3.4) $$x_1\ T^i(P)\ x_3\ \ldots\ x_n\ E\ x_{n+1}$$

where (3.4) contains no executable occurrences of P.
Finally, we apply the following property of computation sequences:
Let, for $F \in \mathcal{A} \cup \mathcal{C} \cup \mathcal{P}$

(3.5) $$x_1\ S_1(F)\ x_2\ S_2(F)\ \ldots\ x_n\ E\ x_{n+1}$$

be a computation sequence. Consider all occurrences of F in (3.5) which are identified by some occurrence of F in $S_1(F)$. If none of the considered occurrences is executable, then the result of replacing all considered occurrences of F by some arbitrary S is again a computation sequence.

Applying this property to (3.4), taking $\Omega$ for S, yields the computation sequence

$$x_1\ T^i(\Omega)\ x_3 \ldots x_n\ E\ x_{n+1}\ .$$

Thus, $x\ T^i(\Omega)y$ holds, and the proof of the lemma is completed.

From lemma 3.3 we immediately obtain

Lemma 3.4

$$P \subseteq \bigcup_{i=0}^{\infty} T^i(\Omega).$$

We are now near to the desired characterization of recursive procedures. The last step is

Lemma 3.5. If $Q = T(Q)$, then $\bigcup_{i=0}^{\infty} T^i(\Omega) \subseteq Q$.

Proof. $\Omega \subseteq Q$ is clear. From this, by monotonicity (lemma 3.2), $T(\Omega) \subseteq T(Q) = Q$ follows. Repeating the argument, we have, for each $i > 0$, $T^i(\Omega) \subseteq Q$. Thus, $\bigcup_{i=0}^{\infty} T^i(\Omega) \subseteq Q$ is established.

The first main result of this section follows:

Theorem 3.1. If $(T_1,D_1)$ is a program scheme and $(P,T(P)) \in D_1$, then P is the *minimal fixed point* of T, i.e.,

$$P = \cap\{X: X = T(X)\}.$$

Moreover, $$P = \bigcup_{i=0}^{\infty} T^i(\Omega).$$

Proof. From lemma 3.1, 3.4 and 3.5.

We conclude this section with the introduction of the *continuity* property of the transformations T, which in fact contains their monotonicity as a special case:

Theorem 3.2

$$T(\bigcup_{i=0}^{\infty} S_i) = \bigcup_{i=0}^{\infty} T(S_i).$$

Proof. $\supseteq$ is a direct result of monotonicity: For each i, $T(S_i) \subseteq T(\bigcup_{i=0}^{\infty} S_i)$; hence, $\bigcup_{i=0}^{\infty} T(S_i) \subseteq T(\bigcup_{i=0}^{\infty} S_i)$. The proof of the reverse inclusion, which is omitted here, proceeds by an inductive argument on the complexity of the T concerned. E.g., if T(S) has the simple form T(S) = A;S, we have to verify whether $A; \bigcup_{i=0}^{\infty} S_i \subseteq \bigcup_{i=0}^{\infty} (A;S_i)$. From the definitions ot follows that this is indeed the case.

The fundamental property of continuity, as expressed by theorem 3.2, was first noted by Scott, and plays an essential role in much of his further work on the theory of computation, see e.g. [27, 28, 29, 30]. A direct consequence of theorem 3.2 is the fact that for each S,T

$$S(\bigcup_{i=0}^{\infty} T^i(\Omega)) = \bigcup_{i=0}^{\infty} S(T^i(\Omega)).$$

This will be used in the next section in the justification of the rule of inference of the formal system to be introduced there.

# 4. A CALCULUS FOR RECURSIVE PROGRAM SCHEMES

## 4.1. Introduction

The results obtained in the previous section on the semantics of recursive procedures, i.e., their characterization as minimal fixed points of continuous transformations, are now exploited in the development of a formal system in which properties of programs involving recursive procedures can be stated and proved formally. This system incorporates two main ideas. Firstly, it uses the axioms of a relation algebra, as given in section 2. Secondly, it features a rule of inference for recursion, which may be considered as a generalization of McCarthy's rule of recursion induction [20]. The justification of this rule is based on the continuity result (theorem 3.2).
Before we proceed with the precise definitions of the formal language and its axioms and rule of inference, to be given in the next subsection, we first discuss some points taken into account in these definitions, in particular dealing with the unification of the notions of sections 2 and 3 into one system, and with the way in which concepts from relation algebra are used to state and prove properties of programs.
In section 3, we introduce three ways of constructing schemes, starting from the elementary statements and the two constants: *composition, selection* and *recursion* (via declaration and call of recursive procedures).

a. *Composition.* As was remarked above, the first construct corresponds directly to composition of relations; hence, its incorporation into the formal system requires no special measures.

b. *Selection.* As to the conditionals, there is no immediate correspondence between the construct $(p\rightarrow S_1,S_2)$ and a concept from relation algebra. Therefore, we have to analyze this notion somewhat further: Remember that the boolean symbol p is interpreted as a partial function from the domain of interpretation $\mathcal{D}$ to $\{0,1\}$. Thus, we can partition $\mathcal{D}$ into three subsets:

$$\mathcal{D}_1 = \{x \mid p(x) = 1\}$$

$$\mathcal{D}_2 = \{x \mid p(x) = 0\}$$

$$\mathcal{D}_3 = \{x \mid p(x) \text{ is undefined}\}.$$

With this partition we associate a partition of the identity relation $E \subseteq \mathcal{D} \times \mathcal{D}$, $E = p \cup p' \cup p_\omega$, with

$$\begin{aligned} p &= \{(x,x) \mid p(x) = 1\} && = \mathcal{D}_1 \times \mathcal{D}_1 \\ p' &= \{(x,x) \mid p(x) = 0\} && = \mathcal{D}_2 \times \mathcal{D}_2 \\ p_\omega &= \{(x,x) \mid p(x) \text{ is undefined}\} && = \mathcal{D}_3 \times \mathcal{D}_3 \quad . \end{aligned}$$

(No confusion should be caused by the double use of the symbol p, once as a predicate, and once as a relation.) Using these three relations, we can now write for $(p\to S_1,S_2)$:

$$\begin{aligned} (p\to S_1,S_2) &= p;S_1 \cup p';S_2 \cup p_\omega;\Omega \\ &= p;S_1 \cup p';S_2 \end{aligned}$$

where we have used the fact that, if p is undefined, then the whole conditional $(p\to S_1,S_2)$ is undefined. Thus, if we introduce as counterpart of the partial function p the pair of relations $\langle p,p' \rangle$, characterized by

$$p \subseteq E, \; p' \subseteq E$$

$$p \cap p' = \Omega$$

then we can model the conditional in the calculus of relations. (Remark: This way of looking at predicates as pairs of subsets of the identity is usually attributed to Karp [13], cf. also Milner [21] and Park [24].)

As another application of this idea, we indicate how to phrase a formalism due to Hoare (see e.g. [12]) in our system. Hoare is interested in constructs of the form "p{Q}r", where Q is a program, and p and r are conditions satisfied upon entrance and exit of the program, respectively. Thus, p{Q}r can be formulated in predicate calculus as

$$\forall x,y[p(x) \wedge xQy \rightarrow r(y)]$$

which in turn can be transliterated into our formalism as

$$p;Q \subseteq Q;r$$

where, as always from now on, small letters stand for subsets of E.

c. *Recursion.* Let $(T_1,D_1)$ be a program scheme, and let $(P,T(P)) \in D_1$. As shown in theorem 3.1, we have for P: $P = \cap\{X:X = T(X)\}$. For this minimal fixed point of T(X) we now introduce a new notation, by means of the so-called μ-operator (μ-for minimal): we write

$$\mu X[T(X)] = \cap\{X:X = T(X)\}.$$

This notation emphasizes that the μ-operator is a *variable-binding* operator: all occurrences of X in μX[T(X)] are *bound* occurrences, and an occurrence of some Y in a formula is free iff it is not bound. As a first consequence of this, we need the rule of rewriting of bound variables: μX[T(X)] = μY[T(Y)], provided that Y does not occur free in T.

The reader should note that with the introduction of the μ-notation, the distinction between declaration and call is done away with. E.g., the program scheme

$$(\underline{\text{procedure}}\ P;(p\rightarrow A_1;P,A_2)$$
$$A_3;P;A_4;P)$$

in the new language is written as

$$A_3;\mu X[(p\rightarrow A_1;X,A_2)];A_4;\mu X[(p\rightarrow A_1;X,A_2)]$$

or, using the new notation for conditionals, as

$$A_3;\mu X[p;A_1;X\cup p';A_2];A_4;\mu X[p;A_1;X\cup p';A_2].$$

In our formal language, we shall allow as constructs for building up the transformations T, besides the three mentioned in section 3, also the operations of intersection and conversion. It can be verified that the monotonicity and continuity results of section 3 go through for this extended class of transformations. On the other hand, complementation is not a monotonic operation, and has to be excluded from the construction of the transformations T; hence, the restriction to *positive* terms in definition 4.1 below.

In [2, 3], we have discussed how to deal with *systems* of recursive procedures. E.g., we showed that if one has the two declarations

$$\underline{\text{procedure}}\ P_1;\ T_1(P_1,P_2)$$
$$\underline{\text{procedure}}\ P_2;\ T_2(P_1,P_2)$$

then

$$(P_1,P_2) = \cap\{(X,Y)\colon X = T_1(X,Y),\ Y = T_2(X,Y)\}.$$

Moreover, it was shown that

$$P_1 = \cap\{X\colon X = T_1(X,\cap\{Y\colon Y= T_2(X,Y)\})\}$$

and similarly for $P_2$. Using the μ-notation, this can be written as

$$P_1 = \mu X[T_1(X,\mu Y[T_2(X,Y)])]$$

and similarly for $P_2$. For the justification of these results we refer to [2, 3], where the monotonicity and continuity of the new type of transformation is also shown. (Note, however, that the section of [3] which is devoted to systems of recursive procedures will have to be modified along the lines of the proof of lemma 3.3. In particular, the transformation introduced in this proof has to be extended by dealing

simultaneously with all procedures of the system.)
This completes our preliminary discussion how to incorporate the constructs of composition, selection and recursion into the formal system to be given presently.
It will not have escaped the reader's attention that a number of operations in a relation algebra have no direct counterparts as a programming concept: This holds for the operations of intersection, conversion, complementation, and for the universal relation. Our use of these operations is indirect: They play a part in the statement and proof of certain properties of programs, of which we now give some examples:

a. *Termination.* As mentioned in section 2, the fact that a program P terminates for all input, i.e., $\forall x\ \exists y[xPy]$, can be phrased relationally as

$$E \subseteq P;\breve{P}\ .$$

b. *Determinism.* In general, we allow non-deterministic programs, i.e., programs which, for given input, may give more than one output. When we want to state that a program P <u>is</u> deterministic, this can be expressed as

$$\breve{P};P \subseteq E\ .$$

c. *Termination with a certain property.* One may be interested in the following question: Given a program P, a predicate p, and input x, is the following predicate true: $\exists y[xPy \wedge p(y)]$. This predicate will be abbreviated to: $P \circ p$, i.e., we have as definition

$$\forall x[(P\circ p)(x) \Longleftrightarrow \exists y[xPy \wedge p(y)]].$$

It can be verified that in a relation algebra we can define $P \circ p$ as

$$P \circ p = P;p;U \cap E\ .$$

Thus, we see an application of the intersection operation and of the universal relation in the statement of a certain property of P. The "o"-operation will be applied in particular in the examples of section 5.

For our use of the operation of complementation, we refer to section 7.

After these introductory remarks, we are now ready for the precise definition of our formal system.

### 4.2. The calculus

We begin with the definition of the well-formed formulae of our formal language. As a starting point, we take the following classes of symbols:

a. The class of *relation variables*, divided into three subclasses:

(i) Possibly indexed capital letters, with the exception of the three elements of class b:

$A, A_1, \ldots, B, \ldots, X, Y, Z, \ldots .$

(ii) Possibly indexed small letters:

$p, p_1, \ldots, q, r, \ldots .$

(iii) Possibly indexed small primed letters:

$p', p'_1, \ldots, q', r', \ldots .$

b. The class of *relation constants*, consisting of the three symbols

$\Omega, E, U.$

From these classes of symbols, *terms* are formed by the operations of section 2, and by the $\mu$-operator which is restricted to apply only to *positive* terms.

Definition 4.1

1. Each relation variable or relation constant is a term.

2. If $\tau_1$ and $\tau_2$ are terms, then $\tau_1;\tau_2$, $\tau_1 \cup \tau_2$ and $\tau_1 \cap \tau_2$ are terms.

3. If $\tau$ is a term, then $\breve{\tau}$ and $\bar{\tau}$ are terms.

4. If $\tau$ is a term which is positive with respect to the variable X (def. 4.2) then $\mu X[\tau]$ is a term.

Definition 4.2

1. X is positive w.r.t. X.

2. If $\tau$ does not contain X free, then $\tau$ is positive w.r.t. X.

3. If $\tau_1,\tau_2$ are positive w.r.t. X, then $\tau_1;\tau_2$, $\tau_1 \cup \tau_2$ and $\tau_1 \cap \tau_2$ are positive w.r.t. X.

4. If $\tau$ is positive w.r.t. X, then $\breve{\tau}$ is positive w.r.t. X.

5. If $\tau$ is positive w.r.t. X and Y, then $\mu Y[\tau]$ is positive w.r.t. X.

As before, we write $\tau(X)$ instead of just $\tau$, when we want to indicate that we are especially interested in possible free occurrences of the variable X in $\tau$. The result of substituting a term $\tau_1$ for all free occurrences of X in $\tau$ is then written as $\tau(\tau_1)$.
The well-formed formulae of the language have the form of *assertions*, defined as follows:

Definition 4.3

1. An *atomic formula* is an expression of the form $\tau_1 \subseteq \tau_2$ or $\tau_1 = \tau_2$, where $\tau_1,\tau_2$ are terms.

2. A *formula* is a list of zero or more atomic formulae.

3. An *assertion* is an expression of the form $\Phi \vdash \psi$, with $\Phi$, $\psi$ formulae.

Examples of assertion are

1. $X \subseteq Y \vdash X;Z \subseteq Y;Z,\ Z;X \subseteq Z;Y,\ \breve{X} \subseteq \breve{Y},\ X \cup Z \subseteq Y \cup Z,\ X \cap Z \subseteq Y \cap Z$ .

2. $\vdash p \cap q = p;q$.

3. $\vdash \mu X[X] = \Omega$.

4. $\vdash \mu X[p;A;X \cup p'] = p;A;\mu X[p;A;X \cup p'] \cup p'$.

In the preceding sections, we have already covered most of the rules for interpreting assertions, which are now summarized in

Definition 4.4

An *interpretation* I of an assertion $\alpha$: $\Phi \vdash \psi$ consists of a pair $<\mathcal{D},h>$ such that

1. $\mathcal{D}$ is an arbitrary set.

2. h is a mapping from relation symbols to relations over $\mathcal{D}$ such that

   a. $\Omega^h = \emptyset$, $E^h = \{(x,x) \mid x \in \mathcal{D}\}$, $U^h = \mathcal{D} \times \mathcal{D}$.

   b. h assigns to each free variable A occurring in $\alpha$ an arbitrary relation $A^h \subseteq \mathcal{D} \times \mathcal{D}$;
   h assigns to each free variable p or p' in $\alpha$ a relation $p^h \subseteq E^h$ $(p'^h \subseteq E^h)$.

3. h is extended to terms as follows:

   a. $(\tau_1;\tau_2)^h = \tau_1^h;\tau_2^h$
   $(\tau_1 \cup \tau_2)^h = \tau_1^h \cup \tau_2^h$
   $(\tau_1 \cap \tau_2)^h = \tau_1^h \cap \tau_2^h$ .

   b. $\breve{\tau}^h = \breve{\tau^h}$
   $\bar{\tau}^h = \overline{\tau^h}$ .

   c. $\mu X[\tau]^h = \cap\{X^h \mid X^h \subseteq \mathcal{D} \times \mathcal{D}$ and $X^h = \tau^h\}$.

4 a. For each atomic formula $\tau_1 \subseteq \tau_2$, $(\tau_1 \subseteq \tau_2)^I$ holds iff $\tau_1^h \subseteq \tau_2^h$ holds; similarly for $\tau_1 = \tau_2$ .

   b. If $\Phi$ is a list of atomic formulae, $\Phi = \Phi_1, \Phi_2, \ldots, \Phi_n$, then $\Phi^I$ holds iff each of $\Phi_i^I$ holds, i = 1, 2, ..., n.

c. $(\Phi \vdash \psi)^I$ holds iff the implication $\Phi^I \supset \psi^I$ holds.

Definition 4.5

As assertion $\Phi \vdash \psi$ is called *valid* iff $(\Phi \vdash \psi)^I$ holds for all interpretations I.

Note that the examples after definition 4.3 are all valid assertions. We now give the axioms of the formal system. They are divided into three groups:

1. The axioms for a relation algebra (section 2), formulated as assertions. E.g., $T_5$ is now formulated as:

$$R;S \cap T = \Omega \vdash S;\breve{T} \cap \breve{R} = \Omega.$$

2. Two axioms corresponding to our treatment of predicates as disjoint subsets of the identity:

$$P_1 : \vdash p \subseteq E,\ p' \subseteq E$$

$$P_2 : \vdash p \cap p' = \Omega$$

3. An axiom for the μ-operator

$$M : \vdash \tau(\mu X[\tau(X)]) \subseteq \mu X[\tau(X)].$$

As rule of inference we have the so-called μ-induction rule:

$$I : \begin{array}{l} \Phi \vdash \psi(\Omega) \\ \Phi, \psi(X) \vdash \psi(\tau(X)) \\ \hline \Phi \vdash \psi(\mu X[\tau(X)]) \end{array}$$

where $\Phi$ is any formula which does not contain X free.

The validity of the axioms should be clear. In particular, *M* expresses one-half of the fixed point property of procedures. The other half, and the minimality property are derivable, as will be shown in lemma 4.1. As an explanation of the μ-induction rule *I*, we observe that it can be described informally as having the following inductive pattern: Suppose one wants to prove an assertion $\alpha(P)$ about a procedure $P = \mu X[\tau(X)]$. *I* then states the following: If one has shown that

a. $\alpha(\Omega)$ holds,

b. if $\alpha(X)$ holds, then $\alpha(\tau(X))$ holds,

one may infer that $\alpha(\mu X[\tau(X)]) = \alpha(P)$ holds. Consider as an example an assertion $\alpha(P)$ of the form $\vdash \tau_1(P) \subseteq \tau_2$. Since from the corresponding instances of a and b we have successively that $\vdash \tau_1(\Omega) \subseteq \tau_2$, $\vdash \tau_1(\tau(\Omega)) \subseteq \tau_2, \ldots, \vdash \tau_1(\tau^i(\Omega)) \subseteq \tau_2, \ldots$, each hold, we obtain that $\vdash \bigcup_{i=0}^{\infty} \tau_1(\tau^i(\Omega)) \subseteq \tau_2$ holds, Then, by continuity(theorem 3.2) $\vdash \tau_1(\bigcup_{i=0}^{\infty} \tau^i(\Omega)) \subseteq \tau_2$ follows, and the result $\vdash \tau_1(P) \subseteq \tau_2$ then is obtained from theorem 3.1. Justification of *I* in the general case is an easy extension of this example.

## 4.3. Basic lemmas

As a first application of the formal system, we prove lemma 4.1, which shows that monotonicity and the (minimal) fixed point property are derivable.

### Lemma 4.1

1. If $\tau(X,Y)$ is monotonic in X and Y, then $\mu X[\tau(X,Y)]$ is monotonic in Y:

$$\frac{\begin{array}{l} X_1 \subseteq X_2 \vdash \tau(X_1,Y) \subseteq \tau(X_2,Y) \\ Y_1 \subseteq Y_2 \vdash \tau(X,Y_1) \subseteq \tau(X,Y_2) \end{array}}{Y_1 \subseteq Y_2 \vdash \mu X[\tau(X,Y_1)] \subseteq \mu X[\tau(X,Y_2)].}$$

2. If $\tau(X)$ is positive w.r.t. X, then $\tau(X)$ is monotonic in X, i.e., then

$$X_1 \subseteq X_2 \vdash \tau(X_1) \subseteq \tau(X_2).$$

3. $\mu X[\tau(X)]$ is a fixed point of $\tau(X)$:

$$\vdash \mu X[\tau(X)] = \tau(\mu X[\tau(X)]).$$

4. $\mu X[\tau(X)]$ is the minimal fixed point of $\tau(X)$, or, even stronger:

$$\tau(Y) \subseteq Y \vdash \mu X[\tau(X)] \subseteq Y.$$

Proof

1. Assume the two premises. We apply *I* to show that the conclusion then follows. We use the following instance of *I*: For $\Phi$ we take $Y_1 \subseteq Y_2$, for $\psi(X)$: $X \subseteq \mu X[\tau(X,Y_2)]$, and for $\tau(X)$: $\tau(X,Y_1)$. Thus, we have to verify

$$Y_1 \subseteq Y_2 \vdash \Omega \subseteq \mu X[\tau(X,Y_2)]$$

which is clear, and

$$Y_1 \subseteq Y_2,\ X \subseteq \mu X[\tau(X,Y_2)] \vdash \tau(X,Y_1) \subseteq \mu X[\tau(X,Y_2)] \tag{4.1}$$

which established as follows: By the monotonicity of $\tau(X,Y)$ in X and Y we have

$$Y_1 \subseteq Y_2,\ X \subseteq \mu X[\tau(X,Y_2)] \vdash \tau(X,Y_1) \subseteq \tau(\mu X[\tau(X,Y_2)],Y_2). \tag{4.2}$$

By *M*,

$$\vdash \tau(\mu X[\tau(X,Y_2)],Y_2) \subseteq \mu X[\tau(X,Y_2)]. \tag{4.3}$$

Combining (4.2) and (4.3), (4.1) follows.

2. This follows by induction on the complexity of $\tau$, using lemma 2.1 and part 1.

3. $\subseteq$: We use $I$, with $\Phi$ empty, and for $\psi(X)$ we take $X \subseteq \tau(X)$. We then have to show that

$$\vdash \Omega \subseteq \tau(\Omega)$$

which is clear, and

$$X \subseteq \tau(X) \vdash \tau(X) \subseteq \tau(\tau(X))$$

which follows by the monotonicity result of part 2.

$\supseteq$: This is the same as axiom $M$.

4. The proof of this, which is a straightforward application of monotonicity and $I$, is left to the reader.

After thus having established that these basic properties of procedures are derivable in the formal system, we now show how the usual axioms for conditionals (McCarthy [20]) are derivable. This follows as a corollary from

Lemma 4.2

1. $\vdash \breve{p} = p$
2. $\vdash p;q = p \cap q$.

Proof

1. Follows by lemma 2.3, part 2, and axiom $P_1$.
2. $\subseteq$: Since $\vdash p \subseteq E$, $q \subseteq E$, by monotonicity we have that $\vdash p;q \subseteq q$, $p;q \subseteq p$. Thus, by properties of boolean algebra, $\vdash p;q \subseteq p \cap q$.

   $\supseteq$: By lemma 2.2, $\vdash p \cap q = p;(\breve{p};q \cap E) \cap q$. Hence, $\vdash p \cap q \subseteq p;(\breve{p};q \cap E) \subseteq p;\breve{p};q = p;p;q \subseteq p;q$ .

Corollary (cf. Engelfriet [10])

Using the notation $(p \rightarrow X,Y) = p;X \cup p';Y$ we have

$$\vdash (p\to(p\to X,Y),Z) = (p\to X,Z),$$

$$(p\to X,(p\to Y,Z)) = (p\to X,Z),$$

$$(p\to(q\to X,Y),(q\to V,W)) =$$

$$(q\to(p\to X,V),(p\to Y,W)) .$$

Proof. Immediately from lemma 4.2, using $P_1$ and $P_2$.

Remark: Note that another standard axiom:

$$\vdash (p\to X,Y);Z = (p\to X;Z,Y;Z)$$

follows from lemma 2.1, part 5.

In section 4.1 we already mentioned the "o"-operator, defined by

$$X \circ p = X;p;U \cap E.$$

The basic properties of this operator are collected in

Lemma 4.3

1. $\vdash (X;Y) \circ p = X \circ (Y\circ p)$
2. $\vdash (X\cup Y) \circ p = X\circ p \cup Y\circ p$
3. $\vdash X;p \subseteq (X\circ p);X$
4. $X;p \subseteq q;X \vdash X\circ p \subseteq q.$

Proof

1. By the definition of "o"-operation,

$$(X;Y) \circ p = X;Y;p;U \cap E$$

$$X \circ (Y\circ p) = X;(Y;p;U\cap E);U \cap E.$$

Since, by lemma 2.3, part 3b,

$$\vdash Y;p;U = (Y;p;U\cap E);U$$

the desired result follows.

2. Immediate from the definitions.

3. Applying lemma 2.3, part 3a, we obtain

$$\vdash X;p = (X;p;U\cap E);X;p \subseteq (X;p;U\cap E);X = (X\circ p);X.$$

4. Assume $X;p \subseteq q;X$. Then

$$\vdash X\circ p = X;p;U\cap E \subseteq q;X;U\cap E.$$

Using lemma 2.2, one easily verifies that, for any Y,

$$\vdash q;Y\cap E \subseteq q,$$

whence $\vdash X\circ p \subseteq q$ follows.

Observe that from parts 3 and 4 of lemma 4.3 we obtain that the following equality holds in all interpretations:

$$X\circ p = \cap\{q \mid X;p \subseteq q;X\}.$$

We conclude this subsection with a first discussion of another construct, viz. the so-called while statement <u>while</u> p <u>do</u> A. We shall use for this the abbreviation p * A. It is easily seen that we can define this statement in our formalism as

$$p*A = \mu X[(p\rightarrow A;X,E)]$$

or, alternatively,

$$p*A = \mu X[p;A;X\cup p'].$$

We shall also use

$$p_1 \wedge p_2 * A = \mu X[(p_1 \rightarrow (p_2 \rightarrow A;X,E),E)]$$

$$= \mu X[p_1;p_2;A;X \cup p_1;p_2' \cup p_1']$$

and

$$p_1 \vee p_2 * A = \mu X[(p_1 \rightarrow A;X,(p_2 \rightarrow A;X,E))]$$

$$= \mu X[(p_1 \cup p_1';p_2);A;X \cup p_1';p_2'].$$

As a first basic property of while statements we mention

$$\vdash \mu X[(p \rightarrow A_1;X,A_2)] = \mu X[(p \rightarrow A_1;X,E)];A_2$$

which is a special case of

<u>Lemma 4.4</u>

$$\vdash \mu X[A_1;X \cup A_2] = \mu X[A_1;X \cup E];A_2 \ .$$

<u>Proof</u>. Call the left-hand side $P_1$ and the right-hand side $P_2$.

a. $P_1 \subseteq P_2$. Left to the reader.

b. $P_2 \subseteq P_1$. By *I* it is sufficient to show $X;A_2 \subseteq P_1 \vdash (A_1;X \cup E);A_2{}^{c}\subseteq P_1$. By the fixed point property (lemma 4.1, part 3), $\vdash P_1 = A_1;P_1 \cup A_2$; hence, we have to show $X;A_2 \subseteq P_1 \vdash (A_1;X \cup E);A_2 \subseteq A_1;P_1 \cup A_2$ which follows by monotonicity (lemma 4.1, part 2).

Further properties of while statements are treated in section 5. Moreover, many other examples, in particular dealing with *equivalences* between statements, can be found in [2], in which also the *completeness* of a restricted part of the calculus (roughly, with only the operators ";", "∪", and the μ-operator restricted to regular procedures, i.e., procedures corresponding to flow diagrams) is shown in the following sense: For two such restricted terms, an assertion $\vdash \tau_1 = \tau_2$ is provable in the calculus iff it is valid, i.e., it holds in all interpretations.

## 4.4. A first example of program equivalence

The following problem, which at first sight appeared to be a problem of tree searching, was suggested to us as a candidate for application of our calculus by J.D. Alanen.

Suppose one wishes to perform a certain action A in all nodes of all trees of a forest (in the sense of Knuth [14], pp. 305-307). Let, for x any node, s(x) be interpreted as "has x a son?", and b(x) as "has x a brother?". Let S(x) be: "visit the first son of x", B(x): "visit the first brother of x", and F(x): "visit the father of x". The problem posed to us can then be formulated as:

Let

$$T_1 = \mu X[A;(s\rightarrow S;X;F,E);\ (b\rightarrow B;X,E)]$$

$$T_2 = \mu X[A;(s\rightarrow S;X;\ b*(B;X);F,E)].$$

Show that then

$$T_1 = T_2;\ b*(B;T_2).$$

We shall not exhibit the proof of this, since it soon appeared that this equivalence is only a special case of the following more general result:

Lemma 4.5. Define

(4.4) $$P_0 = \mu X[\tau_0(\tau_1(X),\tau_2(X))]$$

(4.5) $$P_1(Y) = \mu X[\tau_1(\tau_0(X,Y))]$$

(4.6) $$P_2 = \mu X[\tau_2(\tau_0(P_1(X),X))].$$

Then

(4.7) $$\vdash P_0 = \tau_0(P_1(P_2),P_2).$$

Remark. Observe that if we take

$$\tau_0(X,Y) = Y;X$$

$$\tau_1(X) = (b\to B;X,e)$$

$$\tau_2(X) = A;(s\to S;X;F,E)$$

then

$$P_0 = \mu X[A;(s\to S;X;F,E);\ (b\to B;X,E)] = T_1$$

$$P_1(Y) = \mu X[(b\to B;Y;X,E)] = b*(B;Y)$$

$$P_2 = \mu X[A;(s\to S;X;b*(B;X);F,E)] = T_2$$

and

$$\tau_0(P_1(P_2),P_2) = P_2;\ P_1(P_2) = P_2;\ b*(B;P_2);$$

hence, the tree searching result is indeed an instance of our general result, which we now prove:

Proof

a. $\subseteq$: From (4.5) and (4.6) respectively

$$\vdash P_1(P_2) = \tau_1(\tau_0(P_1(P_2),P_2))$$

$$\vdash P_2 = \tau_2(\tau_0(P_1(P_2),P_2)).$$

Hence,

$$\vdash \tau_0(\tau_1(\tau_0(P_1(P_2),P_2)),\ \tau_2(\tau_0(P_1(P_2),P_2))) = \tau_0(P_1(P_2),P_2)$$

from which, by (4.4) and the minimal fixed point property of $P_0$, we obtain

$$\vdash P_0 \subseteq \tau_0(P_1(P_2),P_2).$$

b. $\underline{\supseteq}$: From (4.4) it follows that

$$\vdash \tau_1(P_0) = \tau_1(\tau_0(\tau_1(P_0),\tau_2(P_0))).$$

Hence, using (4.5),

(4.8) $$\vdash P_1(\tau_2(P_0)) \subseteq \tau_1(P_0).$$

By the μ-induction rule *I*, in order to show $\vdash \tau_0(P_1(P_2),P_2) \subseteq P_0$, it is sufficient to show

$$\tau_0(P_1(X),X) \subseteq P_0 \vdash \tau_0(P_1(\tau_2(\tau_0(P_1(X),X))),$$
$$\tau_2(\tau_0(P_1(X),X))) \subseteq P_0$$

i.e., it is sufficient to show

$$\vdash \tau_0(P_1(\tau_2(P_0)),\tau_2(P_0)) \subseteq P_0$$

which follows from (4.4) and (4.8).

## 5. APPLICATIONS TO WHILE STATEMENTS

### 5.1. First example

Our first result shows how to reduce a certain procedure which itself does not have the form of a while statement to a combination of while statements. (Note that this is not possible in general, see e.g. [7, 1, 15] for a precise statement of this fact and relevant results.) We assert that

(5.1) $\vdash \mu X[(p_1 \to A_1;X,(p_2 \to A_2;X,E))] = p_1 * A_1;p_2 * (A_2;p_1 * A_1).$

We use two auxiliary results

(5.2) $\vdash \mu X[A;\tau(X)] = A;\mu X[\tau(A;X)]$

(5.3) $\vdash \mu X[\mu Y[\tau(X,Y)]] = \mu X[\tau(X,X)]$

the proofs of which are omitted.
We rewrite the left-hand side of (5.1):

$\vdash \mu X[(p_1 \to A_1;X,(p_2 \to A_2;X,E))] =$ (by (5.3))

$\mu Y[\mu X[(p_1 \to A_1;X,(p_2 \to A_2;Y,E))]] =$ (by lemma 4.4)

$\mu Y[\mu X[(p_1 \to A_1;X,E)];(p_2 \to A_2;Y,E)] =$ (def.)

$\mu Y[p_1 * A_1;(p_2 \to A_2;Y,E)] =$ (by (5.2))

$p_1 * A_1;\ \mu Y[(p_2 \to A_2;p_1 * A_1;Y,E)] =$ (def.)

$p_1 * A_1;\ p_2 * (A_2;p_1 * A_1)$ .

Thus, the proof of (5.1) is completed.

Remark: (5.1) is easily generalized to the following result:

Let

$$\alpha_0 = E$$

$$\alpha_i = \alpha_{i-1};\ p_i*(A_i;\alpha_{i-1}), \qquad i = 1,\ 2,\ \dots,\ n.$$

Then

$$\vdash \mu X[(p_1\to A_1;X,(p_2\to A_2;X,\dots,(p_n\to A_n;X,E)\dots))] = \alpha_n \ .$$

## 5.2. Second example

Our second example is taken from Dijkstra [9], p. 31, 32. Consider the following two programs, which determine whether the elements of two arrays x[1:n], y[1:n] are equal:

$P_1$ is

```
    i:= 1; equal:= true;
    while i ≤ n do begin equal:= equal ∧ x[i] = y[i];
                         i:= i+1
                   end
```

$P_2$ is

```
    i:= 1; equal:= true;
    while i ≤ n ∧ equal do
                   begin equal:= equal ∧ x[i] = y[i];
                         i:= i+1
                   end
```

Our aim is to show the following two assertions:

If $P_1$ terminates with the value true for equal, then so does $P_2$;
If $P_1$ terminates with the value false for equal, then so does $P_2$.

We first have to formulate this as an assertion in our calculus.

```
Let      A correspond to begin equal:= equal ∧ x[i] = y[i];
                              i:= i+1
                              end,
```

$\langle p_1,p_1'\rangle$ correspond to $i \le n$

$\langle p_2,p_2'\rangle$ correspond to equal

We observe that

1. equal is a *total* predicate, expressed by

$$p_2 \cup p_2' = E.$$

2. If equal is true after execution of A, then it was also true before execution of A. This can be expressed by

$$A;p_2 \subseteq p_2;A.$$

Remembering, that we have as interpretation for the "o"-operator: $(X \circ p)(x) \longleftrightarrow \exists y[xXy \wedge p(y)]$ and using the notation for while statements of section 4.3, we see that we can formulate our assertion as

(5.4) $$p_2 \cup p_2' = E,\ A;p_2 \subseteq p_2;A \vdash (p_1 * A) \circ p_2 \subseteq (p_1 \wedge p_2 * A) \circ p_2\ ,$$
$$(p_1 * A) \circ p_2' \subseteq (p_1 \wedge p_2 * A) \circ p_2'\ ,$$

which we now proceed to derive, using a sequence of auxiliary results:

1. $$A;p \subseteq p;A \vdash A \circ p \subseteq p.$$

Proof. By lemma 4.3.4.

2. $$A;p \subseteq p;A \vdash (q * A) \circ p \subseteq p.$$

Proof. By *I*, it is sufficient to show that

$$A;p \subseteq p;A,\ X \circ p \subseteq p \vdash (q;A;X \cup q') \circ p \subseteq p.$$

Clearly, $\vdash q' \circ p \subseteq p$. Also,

$$\vdash (q;A;X) \circ p = (q;A) \circ (X \circ p) \subseteq (q;A) \circ p = q \circ (A \circ p) \subseteq q \circ p \subseteq p$$

where we have applied lemma 4.3.1, the assumption, lemma 4.3.1, and auxiliary result 1.

The proof is then completed by applying lemma 4.3.2.

3. $$q \cup q' = E \vdash p \wedge q * A;\ p * A = p * A.$$

Proof. Left to the reader. If necessary, he may invoke the completeness theorem of [2].

4. $$p_2 \cup p_2' = E,\ A;p_2 \subseteq p_2;A \vdash (p_1 * A) \circ p_2 \subseteq (p_1 \wedge p_2 * A) \circ p_2.$$

Proof. By result 2, $\vdash (p_1 * A) \circ p_2 \subseteq p_2$. Hence,

$$\vdash (p_1 \wedge p_2 * A) \circ ((p_1 * A) \circ p_2) \subseteq (p_1 \wedge p_2 * A) \circ p_2,$$

and the result follows from 3.

5. $$p_2 \cup p_2' = E \vdash (p_1 * A) \circ p_2' \subseteq (p_1 \wedge p_2 * A) \circ p_2' .$$

Proof. Let $P = p_1 \wedge p_2 * A = p_1;p_2;A;P \cup p_1;p_2' \cup p_1'$ .
By *I*, it is sufficient to show that

$$p_2 \cup p_2' = E,\ X \circ p_2' \subseteq P \circ p_2' \vdash (p_1;A;X \cup p_1') \circ p_2' \subseteq$$
$$(p_1;p_2;A;P \cup p_1;p_2' \cup p_1') \circ p_2' .$$

Assume the two hypotheses. We then have, applying lemma 4.3:

$$\vdash (p_1;A;X \cup p_1') \circ p_2' = (p_1;A;X) \circ p_2' \cup (p_1' \circ p_2') =$$
$$(p_1;A) \circ (X \circ p_2') \cup (p_1' \circ p_2') \subseteq (p_1;A) \circ (P \circ p_2') \cup (p_1' \circ p_2') =$$
$$(p_1;(p_2 \cup p_2');A;P) \circ p_2' \cup (p_1' \circ p_2') =$$
$$(p_1;p_2;A;P) \circ p_2' \cup (p_1;p_2';A;P) \circ p_2' \cup (p_1' \circ p_2').$$

The desired result then follows, since it is easily verified that

$$\vdash (p_1;p_2';A;P) \circ p_2' \subseteq p_1;p_2' = (p_1;p_2') \circ p_2' .$$

The proof of (5.4) now immediately follows from results 4 and 5.

## 5.3. Third example

We introduce the following notion: Let A be a statement, and suppose we know that p is true upon entrance of A. We are interested in the least predicate which is satisfied upon exit from A, denoted by $\varepsilon_A(p)$: Thus, we have as definition

$$\varepsilon_A(p) = \cap\{q \mid p;A \subseteq A;q\}.$$

From this definition, we see that

$$\varepsilon_A(p) = \cap\{q \mid \breve{A};p \subseteq q;\breve{A}\}.$$

Comparison with the result (section 4.3)

$$X\circ p = \cap\{q \mid X;p \subseteq q;X\}$$

yields: $\varepsilon_A(p) = \breve{A}\circ p$. Thus, using lemma 4.3, we obtain

$$\varepsilon_{A_1;A_2}(p) = \varepsilon_{A_2}(\varepsilon_{A_1}(p))$$

$$\varepsilon_{A_1\cup A_2}(p) = \varepsilon_{A_1}(p) \cup \varepsilon_{A_2}(p).$$

We now consider the problem of expressing $\varepsilon_{q*A}(p)$ in terms of $\varepsilon_A(p)$. We assert that

$$\varepsilon_{q*A}(p) = q' \cap \mu X[p \cup \varepsilon_A(q\cap X)].$$

This formula can be understood intuitively as follows: First of all, upon termination of q * A, we are certain that q' holds. Moreover, we know that A has been executed zero or more, say n times, resulting in a termination condition $\delta^{(n)}$, for each $n \geq 0$; thus, we write

$$\varepsilon_{q*A}(p) = q' \cap (\delta^{(0)}\cup\delta^{(1)}\cup\delta^{(2)}\cup\ldots)$$

where we have

1. $\delta^{(0)} = p$: If A is not performed at all, then p is still true.
2. $\delta^{(1)} = \varepsilon_A(p\cap q)$: If A is performed once, then, upon entrance of A, q was true; hence, the result $\varepsilon_A(p\cap q)$.
3. $\delta^{(2)} = \varepsilon_A(q\cap\varepsilon_A(p\cap q))$: Upon entrance of the second execution of A, the condition satisfied upon exit from the first execution of A is true; moreover, q is true again.

etc.

Thus, we see that

$$\delta^{(0)}\cup\delta^{(1)}\cup\delta^{(2)}\cup \dots = p \cup \varepsilon_A(p\cap q) \cup \varepsilon_A(q\cap\varepsilon_A(p\cap q)) \cup \dots .$$

from which, using theorem 3.1, we infer that

$$\bigcup_{i=0}^{\infty} \delta^{(i)} = \mu X[p \cup \varepsilon_A(q\cap X)].$$

Using this, the assertion we have to prove can be formulated as

$$(5.5) \qquad \vdash \overset{\smile}{\mu X[q;A;X\cup q']}\circ p = q' \cap \mu X[p\cup\breve{A}\circ(q\cap X)].$$

The proof of this uses two auxiliary results.

$$1. \qquad \vdash \mu X[A;X\cup E]\circ p = \mu X[A\circ X\cup p].$$

$$2. \qquad \vdash \overset{\smile}{\mu X[A;X\cup E]} = \mu X[\breve{A};X\cup E].$$

Result 1 follows by a straightforward application of *I*.
Result 2 can be derived from

$$3. \qquad \vdash \overset{\smile}{\mu X[\tau(X)]} = \mu X[\overset{\smile}{\tau(\breve{X})}]$$

and

4. $\vdash \mu X[A;X\cup E] = \mu X[X;A\cup E]$

the proofs of which are left to the reader.

The proof of (5.5) can now be given as follows:

$$\vdash \mu\overparen{X[q;A;X\cup q']} \circ p \quad = \qquad \text{(lemma 4.4)}$$

$$\overset{\smile}{\mu X[q;A;X\cup E]};q' \circ p \quad =$$

$$(q';\ \mu\overparen{X[q;A;X\cup E]}) \circ p = \qquad \text{(result 2)}$$

$$(q';\ \mu X[\overline{q;A};X\cup E]) \circ p = \qquad \text{(result 1)}$$

$$q' \circ \mu X[\overline{q;A}\circ X\cup p] \quad =$$

$$q' \circ \mu X[\breve{A}\circ(q\circ X)\cup p] \quad =$$

$$q' \cap \mu X[\breve{A}\circ(q\cap X)\cup p]$$

where the justification of the last step is again left to the reader.

## 6. FLOYD'S INDUCTIVE ASSERTION METHOD

In [11], Floyd has proposed a method for proving correctness of programs, which is usually called the "inductive assertion method". We first give an informal description of the method, and then a formal justification of it in terms of our calculus. (For another proof which uses category-theoretical techniques, see Burstall [8].)

Let P be a program in the form of a flow diagram. Associate with appropriate points in the program "assertions", i.e., predicates expressing properties of one or more of the variables manipulated by P. (Note that the term "assertion" here has a different meaning from def. 4.3.) This association may be visualized by labelling the corresponding edges of the graph representing P with the assertions concerned. Let $p_1, p_2, \ldots, p_n$ be a set of assertions for P, where $p_1$ labels its entrance and $p_n$ its exit. The $p_1, p_2, \ldots, p_n$ are said to have been "verified", if the following condition has been proved:

For each "execution path", leading from entrance to exit, the following holds: Let $p_1 = p_{i_1}, p_{i_2}, \ldots, p_{i_m} = p_n$ be the assertions encountered on this path. For each s, s = 1, 2, ..., m-1, let S be the statement executed between the points labelled by $p_{i_s}$ and $p_{i_{s+1}}$ (S may be (a subset of) the dummy statement E, cf. our treatment of conditionals). Then, if $p_{i_s}$ is true upon entrance of S, upon exit from S, if at all, $p_{i_{s+1}}$ is true.

Floyd's theorem now states that, if $p_1, p_2, \ldots, p_n$ is a verified set of assertions for P, and if P is entered with $p_1$ true, then upon exit from P, if at all, $p_n$ is true.

Our formulation and proof of this theorem yields a slight extension of Floyd's result in the sense that it also covers the case of recursive procedures which are not representable as flow diagrams.

The notion of a verified set of assertions is not defined directly, but it occurs implicitly in our definition of the set of "inductive assertion patterns" for a term $\tau$:

Definition 6.1

1. An *inductive assertion pattern* (i.a.p.) is a list of inclusions of the form

$$p_1;\tau_1 \subseteq \tau_1;q_1,\ p_2;\tau_2 \subseteq \tau_2;q_2,\ \ldots,\ p_n;\tau_n \subseteq \tau_n;q_n.$$

2. An i.a.p. of this form is called *fully expanded* with respect to a variable X, iff, for each i, i = 1, 2, ..., n, either $\tau_i$ contains no free occurrences of X, or $\tau_i = X$.

3. The set of all inductive assertion patterns for a term $\tau$ with entrance assertion p and exit assertion q, denoted by $A(p,\tau,q)$ is defined inductively as follows:

   a. The one-element list

   $$p;\tau \subseteq \tau;q$$

   is an element of $A(p,\tau,q)$.

   b. If $\tau$ has the form $\tau = \tau_1 \cup \tau_2$, if $\alpha_1 \in A(p_1,\tau_1,q_1)$, $\alpha_2 \in A(p_2,\tau_2,q_2)$, then the list

   $$p \subseteq p_1,\ p \subseteq p_2,\ \alpha_1,\ \alpha_2,\ q_1 \subseteq q,\ q_2 \subseteq q$$

   is an element of $A(p,\tau,q)$.

   c. If $\tau$ has the form $\tau = \tau_1;\tau_2$, if $\alpha_1 \in A(p_1,\tau_1,q_1)$, $\alpha_2 \in A(p_2,\tau_2,q_2)$, then the list

   $$p \subseteq p_1,\ \alpha_1,\ q_1 \subseteq p_2,\ \alpha_2,\ q_2 \subseteq q$$

   is an element of $A(p,\tau,q)$.

   d. If $\tau$ has the form $\tau = \mu X[\tau_1]$, if $\alpha_1 \in A(p_1,\tau_1,q_1)$ and if $\alpha_1$ is fully expanded w.r.t. X, then the list $\alpha$ is an element of $A(p,\tau,q)$, where $\alpha$ is constructed as follows:
   Since $\alpha_1$ is fully expanded w.r.t. X, it can be written as $\alpha_1 = \alpha',\xi$, where $\alpha',\xi$ are i.a.p.'s such that

(i) None of the elements in $\alpha'$ contains free occurrences of X.

(ii) $\xi$ is of the form

$$\xi: p_{i_1};X \subseteq X;q_{i_1}, p_{i_2};X \subseteq X;q_{i_2}, \ldots, p_{i_m};X \subseteq X;q_{i_m}$$

with $1 \leq i_s \leq n$ for $s = 1, 2, \ldots, m$.

Let $\eta_1,\eta_2,\pi$ be i.a.p.'s of the form (see comment below)

$$\eta_1: p_{i_1} \subseteq p_1, p_{i_2} \subseteq p_1, \ldots, p_{i_m} \subseteq p_1$$

$$\eta_2: q_1 \subseteq q_{i_1}, q_1 \subseteq q_{i_2}, \ldots, q_1 \subseteq q_{i_m}$$

$$\pi : p \subseteq p_1, q_1 \subseteq q.$$

Then $\alpha$ is the list

$$\alpha : \alpha', \eta_1, \eta_2, \pi.$$

(The definitions of $\eta_1$ and $\eta_2$ may be understood intuitively as follows: Consider an occurrence of X in $\tau_1$, with entrance assertion $p_{i_s}$ and exit assertion $q_{i_s}$, i.e., $p_{i_s};X \subseteq X;q_{i_s}$ is an element of $\xi$. "Execution" of this X may be seen as a jump back to the beginning of $\tau_1$, which has entrance assertion $p_1$; hence, the requirement $p_{i_s} \subseteq p_1$. Similarly, completion of the execution of $\tau_1$, with exit assertion $q_1$, implies completion of the "execution" of X; hence, $q_1 \subseteq q_{i_s}$ is required.)

Example (see also figure 1)

Consider the term $\tau = \mu X[A_1;X;A_2 \cup A_3;A_4]$.

A possible i.a.p. for this term is the following list of inclusions (where we have indicated by a, b, c, d the corresponding clause of definition 6.1, part 3):

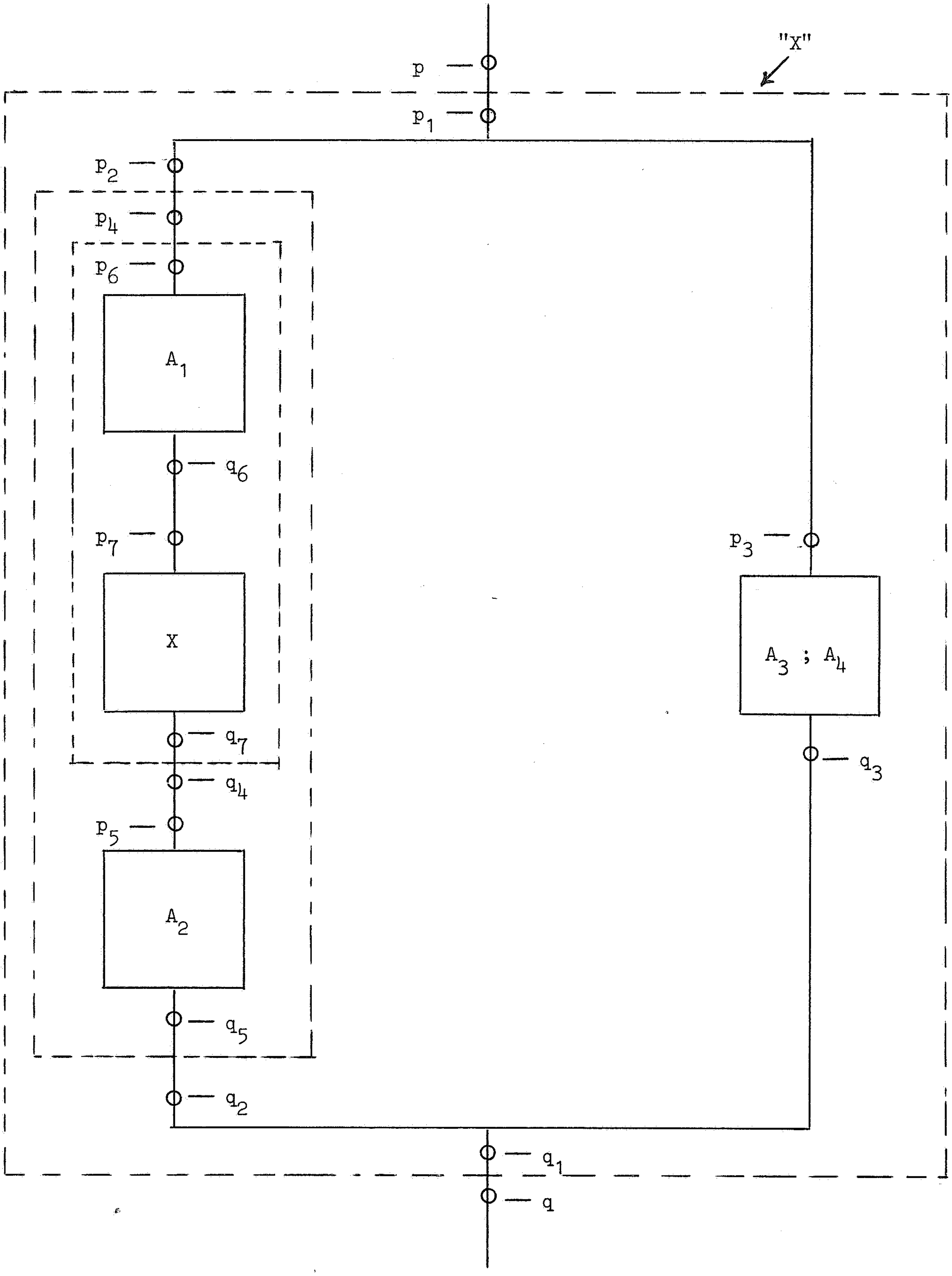

Figure 1

(d,π) $p \subseteq p_1,\ q_1 \subseteq q$ ,

(b) $p_1 \subseteq p_2,\ p_1 \subseteq p_3,\ q_2 \subseteq q_1,\ q_3 \subseteq q_1,$

(c) $p_2 \subseteq p_4,\ q_4 \subseteq p_5,\ q_5 \subseteq q_2,$

(c) $p_4 \subseteq p_6,\ q_6 \subseteq p_7,\ q_7 \subseteq q_4,$

(a) $p_6;A_1 \subseteq A_1;q_6,\ p_5;A_2 \subseteq A_2;q_5,$

(d,$\eta_1$) $p_7 \subseteq p_1,$

(d,$\eta_2$) $q_1 \subseteq q_7,$

(a) $p_3;A_3;A_4 \subseteq A_3;A_4;q_3$ .

Theorem 6.1. Let $A(p,\tau,q)$ be the set of inductive assertion patterns for $\tau$ with entrance condition p and exit condition q, and let $\alpha \in A(p,\tau,q)$. Then

$$\alpha \vdash p;\tau \subseteq \tau;q\ .$$

Proof. The proof proceeds by induction on the complexity of the term $\tau$.

a. If $\alpha$ is defined by def. 6.1, clause 3a, we have nothing to prove.

b. Let $\tau = \tau_1 \cup \tau_2$, and let $\alpha$, $\alpha_1$, $\alpha_2$ be as in def. 6.1, clause 3b. Then, by the induction hypothesis,

$$\alpha_1 \vdash p_1;\tau_1 \subseteq \tau_1;q_1$$

$$\alpha_2 \vdash p_2;\tau_2 \subseteq \tau_2;q_2\ .$$

Hence,

$$p \subseteq p_1,\ p \subseteq p_2,\ \alpha_1,\ \alpha_2,\ q_1 \subseteq q,\ q_2 \subseteq q$$

$$\vdash p;(\tau_1 \cup \tau_2) = p;\tau_1 \cup p;\tau_2 \subseteq p_1;\tau_1 \cup p_2;\tau_2 \subseteq \tau_1;q_1 \cup \tau_2;q_2 \subseteq$$

$$\tau_1;q \cup \tau_2;q = (\tau_1 \cup \tau_2);q\ .$$

Thus

$$\alpha \vdash p;\tau \subseteq \tau;q$$

follows.

c. Similar to case b.

d. Let $\tau = \mu X[\tau_1]$, and let $\alpha$, $\alpha_1$, $\alpha'$, $\xi$, $\eta_1$, $\eta_2$, $\pi$ be as in def 6.1, clause 3d.

By the induction hypothesis,

$$\alpha_1 \vdash p_1;\tau_1 \subseteq \tau_1;q_1$$

i.e.,

$$\alpha',\xi \vdash p_1;\tau_1 \subseteq \tau_1;q_1 \ .$$

From this we immediately obtain

$$\alpha',\xi,\eta_1,\eta_2 \vdash p_1;\tau_1 \subseteq \tau_1;q_1,\ p_{i_1};\tau_1 \subseteq \tau_1;q_{i_1},\ \ldots,\ p_{i_m};\tau_1 \subseteq \tau_1;q_{i_m} \ .$$

Hence, observing that $\alpha'$ does not contain X free, and applying the $\mu$-induction rule, we infer

$$\alpha',\eta_1,\eta_2 \vdash p_1;\mu X[\tau_1] \subseteq \mu X[\tau_1];q_1,\ p_{i_1};\mu X[\tau_1] \subseteq \mu X[\tau_1];q_{i_1},\ \ldots,$$
$$p_{i_m};\mu X[\tau_1] \subseteq \mu X[\tau_1];q_{i_m} \ .$$

Hence,

$$\alpha',\eta_1,\eta_2 \vdash p_1;\mu X[\tau_1] \subseteq \mu X[\tau_1];q_1$$

follows. From this

$$\alpha',\eta_1,\eta_2,\pi \vdash p;\mu X[\tau_1] \subseteq \mu X[\tau_1];q$$

is immediate; therefore, we have completed the proof of

$$\alpha \vdash p;\tau \subseteq \tau;q$$

in this case and, thus, of theorem 6.1.

## 7. RECURSION AND INDUCTION: INTEGERS AND TREES

The considerations of the preceding sections have all been of a rather general nature, in the sense that the underlying domains of interpretation were completely arbitrary. We now want to specialize our general theory to provide characterizations of two special domains, viz. those of *natural numbers* and *trees*. In other words, we look for some special constants, and axioms characterizing these, which we add to the general system of section 4.2. In our paper [2], we did this for natural numbers by the introduction of the constants S, M, $A_0$, $p_0$ and $p'_0$, with intended interpretation over the domain of natural numbers $N$:

$$S = \{(x,x+1) \mid x \in N\}$$

$$M = \{(x,x-1) \mid x \in N \backslash \{0\}\}$$

$$A_0 = \{(x,0) \mid x \in N\}$$

$$p_0 = \{(0,0)\}$$

$$p'_0 = \{(x,x) \mid x \in N \backslash \{0\}\}.$$

These constants were then characterized by the following axioms (based on unpublished work by Scott):

$$S_1\text{: } S;M = E$$

$$S_2\text{: } A_0;M = \Omega$$

$$S_3\text{: } S;A_0 = A_0$$

$$S_4\text{: } \mu X[(p_0 \to A_0, M;X;S)] = E.$$

Observe that axiom $S_4$ can be formulated as: Let the function f(x) be recursively defined by (denoted by "$\Leftarrow$")

$$f(x) \Leftarrow (x=0 \to 0,\ f(x-1)+1)$$

then, for all non-negative integers x:

$$f(x) = x.$$

In [2], we showed that $S_4$ gives us the notion of mathematical induction, in the sense that it can be used to prove the following rule of inference

$$\frac{\begin{array}{c}\vdash A_0;F \subseteq A_0;G \\ X;F \subseteq X;G \vdash X;S;F \subseteq X;S;G\end{array}}{\vdash F \subseteq G}$$

The proof of this follows easily from $S_4$, using the μ-induction rule *I*. As an example of applying $S_1$ to $S_4$, we showed in [2] that McCarthy's result on the 91-function (see e.g. [19]) is derivable, i.e., we gave a formulation and proof of the following result: If

$$f(x) \Leftarrow (x>100 \rightarrow x-10,\ f(f(x+11)))$$

then

$$f(x) = (x>100 \rightarrow x-10,\ 91).$$

Using the additional formalism of the calculus of relations, it appears that <u>one</u> constant, viz. the successor function S, suffices, since for the other constants we can write

$$M = \breve{S}$$

$$p_0' = \breve{S};S$$

$$p_0 = \overline{\breve{S};S} \cap E \quad \text{(remember that } \bar{\ } \text{ denotes complementation)}$$

$$A_0 = \mu X[(\overline{\breve{S};S} \cap E) \cup \breve{S};X].$$

As new axioms we now introduce

$$I_1\colon \ \vdash \breve{S};S \subseteq E$$

$$I_2\colon \ \vdash S;\breve{S} \subseteq E$$

$$I_3\colon \ \vdash E \subseteq S;\breve{S}$$

$$I_4\colon \ \vdash E \subseteq \mu X[(\overline{\breve{S};S} \cap E) \cup \breve{S};X;S].$$

As first consequences of $I_1$ to $I_4$ we have

(7.1) $\qquad \vdash S;\breve{S} = E$

which follows from $I_2$ and $I_3$ , and

(7.2) $\qquad \vdash E = \mu X[(\overline{\breve{S};S} \cap E) \cup \breve{S};X;S]$

which follows by $I_4$ and the fact that, since, by $I_1$, $\vdash (\overline{\breve{S};S} \cap E) \cup \breve{S};E;S \subseteq E$, lemma 4.1.4 yields $\vdash \mu X[(\overline{\breve{S};S} \cap E) \cup \breve{S};X;S] \subseteq E$.

It is not difficult to show that $S_1$ to $S_4$ follow from $I_1$ to $I_4$:

1. $S_1$ follows by (7.1).
2. Proof of $S_2$. We have to show

$$\vdash \mu X[(\overline{\breve{S};S} \cap E) \cup \breve{S};X];\ \breve{S} \subseteq \Omega.$$

The proof of this is direct by the μ-induction rule, provided we have shown that

$$\vdash (\overline{\breve{S};S} \cap E);\breve{S} \subseteq \Omega.$$

Since S is a function ($I_1$), application of lemma 2.3.1 yields

$$\vdash (\overline{\breve{S};S} \cap E);\breve{S} = \overline{\breve{S};S;\breve{S}} \cap \breve{S}$$

and the desired result then follows by applying (Tarski's axiom) $T_5$.

3. Proof of $S_3$. Left to the reader.

4. Proof of $S_4$. It can be verified that $\vdash p_0;A_0 = p_0$, and $\vdash p_0';M = \breve{S}$. The result then follows by (7.2).

We now illustrate the new axiom system by some further examples.

*Example 1.* "All natural numbers are different", i.e, writing $S^i$ for $\underbrace{S;S;\ldots;S}_{i \text{ times}}$ we have

$$\vdash A_0;S^i \cap A_0;S^j = \Omega \qquad (i \neq j).$$

Proof

1. We first show that, for each $k \geq 1$,

$$\vdash S^k \cap E = \Omega.$$

By (7.2) and the μ-induction rule, it is sufficient to show that

$$S^k \cap X \subseteq \Omega \vdash S^k \cap \{(\overline{\breve{S};S} \cap E) \cup \breve{S};X;S\} \subseteq \Omega.$$

(i) $\qquad \vdash S^k \cap \overline{\breve{S};S} \cap E = \Omega.$

We have

$$\vdash S^k \cap \overline{\breve{S};S} \cap E = S^k;p_0' \cap p_0.$$

Since it is easily derived, using lemma 2.2, that, in general,

$$\vdash X;p' \cap p = \Omega,$$

the result follows.

(ii) $$S^k \cap X \subseteq \Omega \vdash S^k \cap \breve{S};X;S \subseteq \Omega.$$

By lemma 2.2,

$$\vdash \breve{S};X;S \cap S^k = \breve{S};(S;S^k \cap X;S) \cap S^k = \qquad \text{(lemma 2.3.1, } I_2\text{)}$$

$$\breve{S};(S^k \cap X);S \cap S^k = \Omega.$$

2. By part 1, for $i \neq j$,

$$\vdash S^i \cap S^j = \Omega.$$

3. $$\vdash \breve{A}_0;A_0 \subseteq E \qquad \text{(in fact, } \vdash \breve{A}_0;A_0 = p_0\text{).}$$

The proof of this is omitted.

4. $$\vdash A_0;S^i \cap A_0;S^j = \qquad \text{(lemma 2.3.1 and part 3)}$$

$$A_0;(S^i \cap S^j) = \qquad \text{(part 2)}$$

$$\Omega$$

*Example 2.* Termination of the factorial function.

It is not difficult to verify that the factorial function f(x), defined by

$$f(x) \Leftarrow (x=0 \rightarrow 1,\ x*f(x-1))$$

can be viewed as an instance of the general procedure

$$F = \mu X[(p_0 \rightarrow A_1,\ \breve{S};X;A_2)]$$

for suitable choice of $A_1$, $A_2$. We now assert that, if $A_1$, $A_2$ are total, then $F$ is total, i.e., that

$$E \subseteq A_1;\breve{A}_1,\ E \subseteq A_2;\breve{A}_2 \vdash E \subseteq F;\breve{F}.$$

Using (7.2) and the μ-induction rule, it is sufficient to show that

$$E \subseteq A_1;\breve{A}_1,\ E \subseteq A_2;\breve{A}_2,\ Y \subseteq F;\breve{F}$$

$$\vdash p_0 \cup \breve{S};Y;S \subseteq (p_0;A_1 \cup \breve{S};F;A_2);\ (\breve{A}_1;p_0 \cup \breve{A}_2;\breve{F};S)$$

which is easy to verify.

*Example 3.* General recursion.

Following Julia Robinson [25] (see also Wright [32]) we say that X is defined by general recursion from $A_0$, S, H and K, where $A_0$ and S are the relations introduced above and H and K are arbitrary relations, if

1. $A_0;X = H$

2. $S;X = X;K.$

We assert that, if X satisfies 1 and 2, then

$$\vdash X = \mu Y[p_0;H \cup S;Y;K].$$

The proof of this follows directly from 1 and 2, using $I_4$ and the μ-induction rule.

This completes our illustrations of the treatment of natural numbers in our calculus, as characterized by $I_1$ to $I_4$. Another look at these axioms suggests that one consider possible combinations of axioms from the following extended set:

$J_1$: $\vdash \breve{S};S \subseteq E$ (= $I_1$)

$J_2$: $\vdash S;\breve{S} \subseteq E$ (= $I_2$)

$$J_3\colon \vdash E \subseteq \breve{S};S$$

$$J_4\colon \vdash E \subseteq S;\breve{S} \qquad (= I_3)$$

$$J_5\colon \vdash E \subseteq \mu X[(\overline{\breve{S};S}\cap E) \cup \breve{S};X;S] \qquad (= I_4)$$

$$J_5\colon \vdash E \subseteq \mu X[(\overline{S;\breve{S}}\cap E) \cup S;X;\breve{S}].$$

We assert that the set $\{J_2, J_5, J_6\}$ characterizes tree structures: Let, for nodes x and y in a tree, xSy hold iff x is the father of y. Clearly, then, $\breve{S}$ is a function, as expressed by $J_2$. Observe that neither S is a function ($J_1$ does not hold), nor are S or $\breve{S}$ total: for terminal nodes in the tree, S is undefined, and for the root of the tree, $\breve{S}$ is undefined. $J_5$ then expresses that, for each node x, if we go up in the tree until we are at the root, and then down again the same number of times, we _may_ arrive at the same node x. $J_6$ asserts a similar property for going down in the tree until a terminal node is met, and then upwards again. Observe that, in combination with $J_2$, one sees that one _must_ arrive at the same node.

A further analysis of the tree structures will also need, besides the father-son relationship, introduction of the brother-relationship. This idea is not pursued further in this paper. Also, we do not deal with other combinations of $J_1$ to $J_6$ which may be of interest. (Clearly, there are also some uninteresting ones, e.g., if we assume $J_3$, then $J_5$ reduces to $E \subseteq \Omega$.)

For a treatment of a very similar axiom system for symbol manipulation we refer to [3]. A characterization of list structures and proofs of properties based thereupon are currently studied by the second author.

## 8. CONCLUSIONS

We have presented a calculus for recursive program schemes in which properties of such schemes can be formulated and proved. The calculus has been illustrated by both specific examples on while statements in which the "o" operator was a useful tool, by an investigation of the foundation of Floyd's inductive assertion method, and by a characterization of two special domains: integers and trees.

Of course, we are aware of the fact that for a number of problems in the theory of programs our calculus is not the most convenient vehicle. E.g., for proving properties of individual programs, say a specific sorting algorithm, a less formal approach may well be preferable.

Our main justification for the development and use of the calculus is the framework it provides for the investigation of properties of programs (the proof of) which would not easily or concisely be expressible, or would not arise naturally otherwise. We hope that our applications of the calculus have served their purpose in convincing the reader of its usefulness in these respects.

An important restriction in our calculus is its limitation to *monadic* schemes only. However, a number of ideas, developed by W.P. de Roever, are available on extensions to *polyadic* functions, using projection functions and an axiomatic characterization of these. E.g., a first version of a proof of the correctness of the well-known recursive solution of the Towers of Hanoi problem has been derived. A publication on this extension is in preparation.

## BIBLIOGRAPHY

[ 1] Ashcroft, E. & Z. Manna, The translation of "goto" programs to "while" programs, Proc. IFIP Congress 71, Booklet TA-2, pp. 147-152 (1971).

[ 2] De Bakker, J.W., Recursive Procedures, Mathematical Centre Tracts 24, Mathematical Centre, Amsterdam (1971).

[ 3] De Bakker, J.W., Recursion, induction and symbol manipulation, in Proc. MC-25 Informatica Symposium, Mathematical Centre Tracts 37, Mathematical Centre, Amsterdam (1971).

[ 4] Bekić, H., Definable operations in general algebra, and the theory of automata and flow charts, to appear.

[ 5] Birkhoff, G., Lattice Theory, Third edition, American Math. Soc., Providence (1967).

[ 6] Blikle, A., Equations in a space of languages, Report 43, Computation Centre Polish Academy of Sciences (1971).

[ 7] Böhm, C. & G. Jacopini, Flow diagrams, Turing machines, and languages with only two formation rules, Comm. ACM, 9, pp. 366-372 (1966).

[ 8] Burstall, R.M., An algebraic description of programs with assertion, verification and simulation, in Proc. of the Conference on Proving assertions about programs, ACM (1972).

[ 9] Dijkstra, E.W., Notes on structured programming, T.H. Report 70-WSK-03, Technological University Eindhoven, Second Ed. (1970).

[10] Engelfriet, J., Abstract program schemes, unpublished memorandum, Technological University Twente (1971).

[11] Floyd, R.W., Assigning meanings to programs, in Proc. of a Symposium in Applied Mathematics, Vol. 19, Mathematical Aspects of Computer Science, pp. 19-32 (ed. J.T. Schwartz), American Math. Soc., Providence (1967).

[12] Hoare, C.A.R., An axiomatic basis for computer programming, Comm. ACM 12, pp. 576-583 (1969).

[13] Karp, R.M., Some applications of logical syntax to digital computer programming, Thesis Harvard University (1959).

[14] Knuth, D.E., The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison Wesley, Reading (1968).

[15] Knuth, D.E. & R.W. Floyd, Notes on avoiding "goto" statements, Information Processing Letters, 1, pp. 23-31 (1971).

[16] Leszczylowski, J., A theorem on resolving equations in the space of languages, Bull. Acad. Polon. Sci., Sér. Sci. Math. Astron. Phys., to appear.

[17] Lyndon, R.C., The representation of relational algebras. Ann. of Math., 51, pp. 707-729 (1950).

[18] Manna, Z., The correctness of programs, J. Comp. Syst. Sci., 3, pp. 119-127 (1969).

[19] Manna, Z. & A. Pnueli, Formalization of properties of functional programs, J. ACM, 17, pp. 555-569 (1970).

[20] McCarthy, J., A basis for a mathematical theory of computation, in Computer Programming and Formal Systems, pp. 33-70 (eds. P. Braffort and D. Hirschberg), North-Holland, Amsterdam (1963).

[21] Milner, R., Algebraic theory of computable polyadic functions, Comp. Science Memorandum no. 12, University College of Swansea (1970).

[22] Morris Jr., J.H., Another recusion induction principle, Comm. ACM, 14, pp. 351-354 (1971)

[23] Park, D., Fixpoint induction and proofs of program semantics, in Machine Intelligence, Vol. 5, pp. 59-78 (eds. B. Meltzer and D. Michie), Edinburgh University Press, Edinburgh (1970).

[24] Park, D., Notes on a formalism for reasoning about schemes, unpublished notes, University of Warwick (1970).

[25] Robinson, Julia, Recursive functions of one variable, Proc. AMS, 19, pp. 815-820 (1968).

[26] Scott, D. & J.W. de Bakker, A theory of programs, unpublished notes, IBM Seminar, Vienna (1969).

[27] Scott, D. Outline of a mathematical theory of computation, Proc. of the Fourth Annual Princeton Conference on Information Sciences and Systems, pp. 169-176, Princeton (1970).

[28] Scott, D., The lattice of flow diagrams, in Symposium on Semantics of Algorithmic Languages, Lecture Notes in Mathematics, Vol. 188, pp. 311-364 (ed. E. Engeler), Springer-Verlag, Berlin (1971).

[29] Scott, D., Continuous lattices, in Proc. Dalhousie Conference, Springer Lecture Notes, to appear.

[30] Scott, D. & C. Strachey, Toward a mathematical semantics for computer languages, in Proc. of the Symposium on Computers and Automata, Microwave Research Institute Symposia Series Vol. 21, Polytechnic Institute of Brooklyn, to appear.

[31] Tarski, A., On the calculus of relations, J. Symbolic Logic, 5, pp. 85-97 (1941).

[32] Wright, J.B., Characterization of recursively enumerable sets, Report RC 3419, IBM Research Center, Yorktown Heights (1971).